Sun'iy o'rganish asoslari

Machine learning

Avazjon Maraximov va Tohir Akramov

September 3, 2020

Ushbu kitobni ulug' o'zbek matematigi Qori
Niyoziy xotirasiga bag'ishladik

Mundarija

So'zboshi

Inson miyasi murakkab va qiziqarli organdir. Uning imkoniyatlari ko'z bilan ko'riladigan narsalardan ancha yuqori turadi. Murakkab fiziologik, psixologik va hissiy funksiyalar inson miyasi nimaga qodirligi haqida gap ketganda aysbergning uchini hosil qiladi xolos. Miyaning juda hayratlanarli tabiati ilm-fanga ilhom bag'ishlaydi.

Inson tabiatdan andoza olishda g'ayrioddiy moyillikka ega. Insoniyat qushlarning uchishini ko'rdi va o'zi ham uchadigan narsalarga ega bo'lishni xohladi. Ucha oluvchi ob'ektlar – samolyotlar bu kuzatishning natijasidir. Bunga o'xshash har bir yangilikning markazida tabiat turadi.

Ilm-fan barcha cheklovlarni yengib o'tdi va inson miyasiga taqlid qilishgacha yetib keldi. Ko'pgina tadqiqotlar odamning miyasi qanday ishlashini va juda ko'p ma'lumotlarni osonlikcha saqlash, sharhlash va boshqarishni tushunishga kirishdi. Shu ma'noda, Sun'iy neyron to'rlar tushunchasi miyamizning biologik neyron to'rlarining kichkina, ammo aniq tasviridir.

Endi bizda miyaning ishlashiga, ba'zi funksiyalarini bajarishga taqlid qiladigan Sun'iy aql mashinalari mavjud. Sun'iy aql bizga ob'ektlarni tasniflash, biz bilan muloqot qilish, kelajakni ko'rish va turli o'yinlarni bizdan ko'ra yaxshiroq o'ynashga qodir bo'lgan mashinalarni berdi.

Bu kitob Sun'iy o'rganish va Sun'iy Neyron To'rlari (SNT) kursiga bag'ishlanadi. Namunaviy dasturlar Python algoritmik tilida keltiriladi.

O'quv qo'llanmaning tarkibi

Neyron to'rlari yoki aniqrog'i Sun'iy Neyron to'rlari ko'plab fanlarga asoslangan texnologiyani anglatadi: neyroshunoslik, matematika, statistika, fizika, informatika va muhandislik. Neyron to'rlari muhim xususiyat tufayli modellashtirish, vaqt ketma-ketligini tahlil qilish, qiyofani tanish, signallarni qayta ishlash va boshqarish kabi turli sohalarda qo'llaniladi.

Ushbu qo'llanma mavzuning ko'p qirrali xususiyatini tan olgan holda, neyron to'rlarining keng qamrovli asosini beradi. Unda keltirilgan boblar namunaviy misollar, kompyuterga yo'naltirilgan eksperimentlar, masala yechish

namunalari va bibliografiya bilan ta'minlangan.

Ushbu o'quv qo'llanma to'rt qismdan iborat bo'lib, ular quyidagicha tartiblangan:

1-bob Sun'iy o'rganishga kirish. Ushbu bob asosan, sifat jihatidan, neyron tarmoqlari, ularning xususiyatlari, tarkibi va sun'iy intellekt bilan qanday bog'liqligini tasvirlaydi. Ushbu bob ba'zi tarixiy qaydnomalar bilan yakunlanadi.

2-bobda Sun'iy o'rganishning matematik asoslari o'rnatiladi. Unga tegishli bo'lgan barcha tushunchalar, ya'ni Chiziqli algebradan tortib Ehtimollar nazariyasi, Hisoblashlar, Optimallashtirish va Sun'iy o'rganishgacha olingan matematik tushunchalar chuqur o'rganish uchun zarur bo'lgan matematik asosni yaratish uchun batafsil muhokama qilinadi. Ushbu tushunchalar ularni Sun'iy o'rganish va chuqur o'rganish sohalarida foydalanishga e'tibor qaratilishi bilan izohlanadi.

3-bobda Sun'iy Neyron To'rlari haqida muhokama qilamiz. Ushbu bob Sun'iy o'rganish nima ekanligi haqida bahs qiladi va uning yillar davomida rivojlanishini muhokama qiladi. Sun'iy Neyron To'rlarining asosiy qurilish bloklari, bir nechta o'rganish usullari, masalan, pertseptron-o'rganish qoidasi va ortga tarqalish usullari batafsil muhokama qilinadi.

4-bobda biz qo'lda yozilgan raqamlarni tanishni o'rganuvchi neyron to'rini amalga oshiruvchi kompyuter dasturini yozamiz. Dastur bir nechta satrdan iborat bo'lib, maxsus neyron to'r kutubxonalaridan foydalanmaydi. Ammo ushbu qisqa dastur inson aralashuvisiz 96 foizdan yuqori aniqlikdagi raqamlarni taniy oladi. Bundan tashqari, bu chuqur o'rganish kabi ilg'or uslublarni ishlab chiqishning ajoyib usuli. Shunday qilib, bob davomida biz qo'l yozuvi tanib olish muammosiga bir necha bor qaytamiz.

Tashakkurnoma

Ushbu o'quv qo'llanma yozilish vaqtidan boshlab o'quvchi qo'liga yetib borguncha bo'lgan vaqt oralig'ida bizlarga ko'mak bergan O'zbekiston Milliy Universiteti professor-o'qituvchilariga samimiy minnatdorchilik bildiramiz.

Mualliflar
https://www.nuu.uz/uzl

1

Sun'iy o'rganish: Kirish

1.1 Sun'iy o'rganish nima?

Informatika (inglizcha "Computer science") bu axborotni avtomatik qayta ishlash bilan bog'liq bo'lgan ilmiy, texnik va ishlab chiqarish faoliyat sohasi bo'ib, kompyuterlar, robotlar va boshqa avtomatik qurilmalar tomonidan amalga oshiriladi. Informatikaning rivojlanishi ko'pincha elektron hisoblash mashinalari - kompyuterlar rivoji bilan chalkashtirib yuboriladi. To'g'ri, ilk kompyuter (ENIAC 1946) yaratilgandan boshlab hozirgi kunimizgacha kompyuterlar tobora kuchayib keldi. Biroq, bu taraqqiyot ma'lum bir sohada Informatika muammolarini hal qilishga har doim ham imkon beravermadi. Natijada, Informatika muammolarini yechishda nafaqat kompyuterlarni, balki dasturiy ta'minotni ("software") ham quvvatlash kerak, degan fikrga kelindi. O'z navbatida, dasturiy ta'minotni qurish bir necha yondashuvlarga asoslangan. Ulardan eng ko'p ishlatiladigani ikkita: (i) algoritmik yondashuv va (ii) bilimga asoslangan yondashuv.

1. *Algoritmik yondashuv*da muayyan masalani hal etish jarayoni tizimli ravishda, ketma-ket yozilishi talab qilinadi. Bu esa biror dasturlash tilida transkripsiya qilishdan oldin amalga oshiriladi. Muammo murakkab bo'lsa, u juda qiyin yoki imkonsiz qadam bo'lishi ham mumkin. Boshqa tomondan, kompyuter bu dasturdagi harflar bo'yicha barcha ko'rsatmalarni bajaruvchi mantiqiy mashina. Qachonki dasturdagi barcha amallar algoritmik tarzda tizimli yozib berilgan bo'lsa, kompyuter qurilmasi uchun bu ayni muddaodir. Afsuski, har doim ham bunday bo'lavermaydi.

2. *Bilimga asoslangan yondashuv* - bu Sun'iy Intellekt, inglizcha "Artificial Intelligence" yoki qisqacha AI. Bunda biror muammoni hal qilish avvaldan (soha eksperti tomonidan) o'rnatilgan qoidalar to'plamiga havola etiladi. Mutaxassis tomonidan ko'zda tutilmagan holatlar tegishli ravishda ko'rib chiqilmaydi. Aslida, ushbu yondashuv qoidalar ko'rinishida, modellashtirish mumkin bo'lgan "aniq" sohalar, masalan, elektronika, mexanika, fizika va h.k. bilan cheklangan. Shu sababli bilimga asoslangan yondashuv asosan, bilimlarni aniq shaklda saqlashning qulay bir usuli hisoblanadi xolos.

Yuqoridagi ikkita yondashuv barcha mavjud muammolarni yechish uchun yetarli emas. Aniqrog'i, algoritmik va bilimga asoslangan yondashuvlar yordamida uzoq vaqtdan beri o'rganib kelinayotgan shaklni aniqlash (rasmlar yoki signallar), diagnostika, motor-boshqaruv, avtomatik tarjima, tilni tushunish sohalarida yetarlicha muvaffaqiyat qozonilmadi. Vaholanki, nisbatan sodda tirik mavjudotlar ushbu operatsiyalarning ba'zilarini qiyinchiliksiz bajarishga qodir. Bunga ishonch hosil qilish uchun chivinning parvozini

kuzatish va uni tutishga urinib ko'rishning o'zi yetarli. Ko'rshapalakning sonor harakati haqida eslatib o'tishning o'zi kifoya.

Shunday qilib, Informatikada uchinchi bir yondashuvga ehtiyoj sezamiz. Bu yondashuvda miyaning ma'lumotlarni qayta ishlash mexanizmidan andoza olishga harakat qilinadi. Asosiy g'oya shundan iboratki, ongli xatti-harakatlar aqliy mexanizmlar majmuiga asoslanadi. Va mana shu bosh g'oya Sun'iy o'rganishning rivojlanishiga asos bo'ladi.

1.1.1 Sun'iy o'rganish

Mantiqan olib qaraganda, Sun'iy o'rganish - bu Sun'iy Intellektning tarkibiy bir qismi. Shunday ekan, keling avval "Sun'iy Intellekt nima?" degan savolga aniq-konkret bir javob berib ketaylik.

Andrew Moore, Carnegie Mellon Universiteti Kompyuter ilmlari kafedrasining mudiri, buni quyidagicha ta'riflaydi: "Sun'iy intellekt bu fan, shu bilan birga kompyuter muhandisligi bo'lib, uning yordamida yaqin vaqtgacha biz inson zakovatini talab qiladi deb o'ylagan ishlarni kompyuterda bajariladi".

"Sun'iy Intellekt nima?" degan savol yanada umumiy bo'lgan "Intellekt nima?" degan savol javobiga bog'liqdir.

Avvalgi savolga javob berish biz uchun qiyinchilik tug'diradi. Uning javobiga yaqinroq borish uchun Sun'iy Intellektni qismlarga ajratib olamiz: (i) zaif Sun'iy Intellekt va (ii) kuchli Sun'iy Intellekt.

Zaif Sun'iy Intellektga xos xususiyatlar quyidagilarda aks etadi:

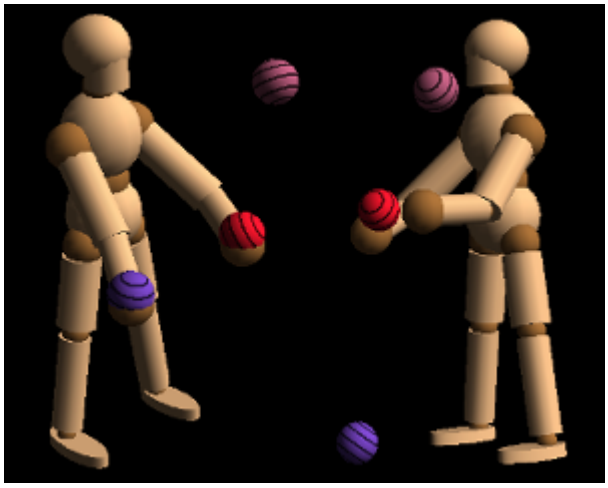
- amaliy dastur muammolari bilan shug'ullanadi
- muayyan sohalarda inson tafakkurini qo'llab-quvvatlaydi
- soha-ostidagina o'rgana oladi
- mas'uliyatsiz, ya'ni daxldorlik mavjud emas

Kuchli Sun'iy Intellekt quyidagi sifatlari bilan namoyon bo'ladi:

- "umumiy intellekt" (aql, mantiqiy fikrlash, strategiyadan foydalanish, jumboqlarni hal qilish va noaniqlik asosida hukm chiqarish)
- inson intellekti bilan tenglasha oladigan, ammo bir xil bo'lmasligi kerak, boshqacha bo'lishi mumkin
- rejalar tuza olish
- atrofni o'rganishga qodir

- muloqot qobiliyati, tabiiy til
- daxldorlik (bahsli masala)
- sezgirlik, hissiyotlar (bahsli masala)
- o'z-o'zini anglash (bahsli masala)

Mana, Sun'iy Intellekt, shuningdek zaif va kuchli Sun'iy Intellekt haqida bilib oldik. Sun'iy o'rganish haqida nima deyish mumkin?



Rasm 1.1: Shunchaki illyustrative rasm.

Keling, IBM kashfiyotchisi Arthur Samuek tomonidan berilgan juda "eski" ta'rifni keltiraylik:

"Sun'iy o'rganish bu aniq dasturlashlarsiz kompyuterga o'rganish imkoniyatini beruvchi fan sohasi."

Yaxshi ta'rif, ammo ko'plab savollar ochiq qolgan – matematik jihatdan aniq ta'rif emas. Qariyb 40 yil o'tgach, 1998 yilda Tom Mitchell "to'g'ri o'rganish masalasi"ni quyidagicha shakllantirdi:

"To'g'ri qo'yilgan O'rganish Masalasi: kompyuter dasturi E tajriba asosida T vazifani P yaxshilanish ko'rsatkichi bilan o'rganadi deyiladi, agar uning T vazifa bo'yicha P ko'rsatkichi E tajriba bilan yaxshilanib borsa."

Eslatma: Agar quyidagi shartlar bajarilsa, matematik masala to'g'ri qo'yilgan deyiladi (yaxshi qo'yilgan):

- masalaning yechimi mavjud (mavjudlik)
- ushbu yechim to'liq aniqlangan (yagonalik)

- yechim boshlang'ich shartlar bilan uzluksiz o'zgaradi (stabil-barqarorlik)

Shunday qilib, Sun'iy o'rganish shuni bildiradiki, bunda algoritm (mashina) avtomatik ravishda o'rganishini amalga oshiradi. Bu shuni anglatadiki, algoritm avtomatik ravishda berilgan ma'lumotlardan kerakli bilimlarni olishga qodir. Maqsad – yangi, avval ko'rilmagan ma'lumotlar uchun yechimni bashorat qilishdir. Uni qo'yishning yana bir usuli bor: an'anaviy evristik qarorlar qabul qilish algoritmlarida dasturchi qarorlar qabul qiluvchi qoidalarni belgilaydi. Sun'iy o'rganish bilan, ushbu ish dastur tomonidan mustaqil ravishda, inson aralashuvisiz amalga oshiriladi.

Masalan, yaroqli elektron pochta xabarlari va kiruvchi spamlarni ajratish uchun dastur yozmoqchimiz deylik. Biz oddiy qoidalar to'plamini yozishga urinib ko'rishimiz mumkin, masalan ma'lum bir xususiyatlarni o'z ichiga olgan xabarlarni belgilash (masalan, "viagra" yoki aniq soxta sarlavhalar). Biroq, qaysi matnning haqiqiyligini aniq ajratish uchun qoidalarni yozish aslida juda qiyin bo'lishi mumkin, natijada ko'plab o'tkazib yuborilgan spam-xabarlarda yoki, eng yomoni, ko'plab yo'qolgan elektron pochta xabarlarida. Eng yomoni, spamerlar ushbu strategiyalarni (masalan, "vi@gr@" deb yozish) aldash maqsadida spamni yuborish usulini faol ravishda o'zgartiradilar. Samarali qoidalarni yozish va ularni zamonaviy qilish - tezda hal qilib bo'lmaydigan vazifaga aylanadi. Yaxshiyamki, sun'iy o'rganish muammoni hal qildi. Zamonaviy spam-filtrlar misollardan "o'rganilgan": biz o'quv algoritmini "elektron pochta" yoki "spam" (kiruvchi elektron pochta) sifatida qo'lda belgilagan elektron pochta xabarlari bilan ta'minlaymiz va algoritmlar ularni avtomatik ravishda ajratishni o'rganadi.

Sun'iy o'rganish fazalari

Guvohi bo'lib turganingizdek, Suniy o'rganish keng qamrovli va qiziqarli soha hisoblanib, uni ifodalashning bir necha yo'li mavjud:

1. Sun'iy Intellekt nuqtai-nazaridan: O'rganish inson bilimlari va aql-idrokining markaziy qismidir. Shu bilan birga u aqlli mashinalarni qurish uchun ham zarurdir. Sun'iy Intellektdagi ko'p yillik harakatlar shuni ko'rsatdiki, barcha qoidalarni dasturlash orqali aqlli kompyuterlarni yaratishga harakat qilish samarasiz; avtomatik o'rganish juda muhimdir. Masalan, biz odamlar biror tilni tushunish qobiliyatiga ega bo'lmasak, uni o'rganamiz. Ammo, kompyuterni tilni o'rganish uchun emas, balki uni dasturlash uchun harakat qildirish mantiqan to'gri bo'ladi.

2. Dasturiy ta'minot muhandisligi nuqtai-nazaridan: Sun'iy o'rganish bizga

an'anaviy usulda kod yozishdan osonroq bo'lishi mumkin bo'lgan misollar yordamida kompyuterlarni dasturlashimizga imkon beradi.

3. Statistika nuqtai-nazaridan: Sun'iy o'rganish - bu informatika va statistikaning birlashishi, ya'ni bunda hisoblash texnikasi statistik muammolarga nisbatan qo'llaniladi. Sun'iy o'rganish odatiy statistika muammolaridan tashqari ko'pgina kontekstlarda juda ko'p muammolarga nisbatan qo'llanilgan. Sun'iy o'rganish ko'pincha statistikaga qaraganda farqli bo'lgan maqsad bilan ishlab chiqiladi (masalan, tezligi aniqlikdan ko'ra muhimroq).

Ko'pincha, Sun'iy o'rganish usullari ikki fazaga ajratiladi:

1. **O'qitish**: O'quv ma'lumotlari ("training data") to'plamidan foydalanib biror model o'qitiladi.

2. **Qo'llash**: O'qitilgan Model ba'zi yangi test ma'lumotlari ("test data") to'g'risida qaror qabul qilish uchun ishlatiladi.

Masalan, spamni filtrlash holatida o'qitish ma'lumotlari jamg'arma yoki spam deb belgilangan elektron pochta xabarlarini tashkil qiladi va biz olgan har bir yangi elektron pochta xabarlari (va tasniflash uchun) sinov ma'lumotlari hisoblanadi. Ammo, sun'iy o'qitishning boshqa usullari ham mavjud.

1.1.2 Sun'iy o'rganish turlari

Sun'iy o'rganishning ba'zi asosiy turlari:

1. **Boshqaruvli o'rganish (Supervised learning)**, unda o'qitish ma'lumotlari to'g'ri javoblar, masalan, "spam" yoki "ham" (spam-bo'lmagan) deb ko'rsatilgan. Boshqaruvli o'rganishning ikkita eng keng tarqalgan turlari bu tasniflash/klassifikatsiyalash (natijalar diskret yorliqlar, spam-filtrlashda bo'lgani kabi) va regressiya (natijalar haqiqiy qiymatli bo'lgan hollarda).

2. **Boshqaruvsiz o'rganish (Unsupervised learning)**, unda biz tahlil qilishimiz va ularning ichidagi shakllarni topishimiz kerak bo'lgan yorliqsiz ma'lumotlar to'plami ("unlabeled data") berilgan bo'ladi. Eng muhim ikkita misol - o'lchamlarni qisqartirish va klasterlash.

3. **Mustahkam o'rganish (Reinforcement learning)**, bunda agent (masalan, robot yoki boshqaruvchi) avvalgi harakatlar natijalarini hisobga olgan holda eng maqbul harakatlarni o'rganishga intiladi.

1.2 Tarixiy bosqichlar va xozirgi holat

Sun'iy Neyron To'rlar 1940 yillarning boshlarida juda katta umidvorlik bilan boshlandi. Biz ushbu yillar davomida soha qanday rivojlangani va davrlar davomida qanday qiyinchiliklarga duch kelgani to'g'risida bilish uchun Sun'iy Neyron To'rlari hamjamiyatidagi muhim voqealar xronologiyasidan o'tamiz.

- Ikki elektrotexnika muhandisi Warren McCullogh va Walter Pitts 1943 yilda neyron to'rlari bilan bog'liq "Asab faoliyatda paydo bo'luvchi g'oyalarning mantiqiy hisob-kitobi" deb nomlangan maqolani nashr etdilar; original maqolani ushbu havoladan olishingiz mumkin: A Logical Calculus of the Ideas Immanent in Nervous Activity. Ularning neyronlari ikkilamchi chiqish holatiga ega va neyronlarga kirishning ikki turi mavjud: qo'zg'atuvchi kirishlar va inhibitiv kirishlar. Neyronga barcha qo'zg'aluvchan kirishlar teng musbat vaznlarga ega. Agar neyronga kiritilgan barcha kirishlar qo'zg'aladigan bo'lsa va umumiy kirish $\sum_i \omega_i x_i > 0$ bo'lsa, neyron 1 ni beradi. Agar biron bir inhibitiv kirish faol bo'lsa yoki $\sum_i \omega_i x_i \leq 0$ bo'lsa, chiqish 0 bo'ladi. Ushbu mantiqdan foydalanib, barcha "Boolean" mantiqiy funksiyalar bir yoki bir nechta shunday neyronlar tomonidan amalga oshirilishi mumkin. Ushbu tarmoqlarning kamchiliklari shundaki, ular vaznni mashq qilish orqali o'rganish imkoniga ega emas. Vaznni qo'lda aniqlash va neyronlarni kerakli hisoblashga erishish uchun birlashtirish kerak.

- Keyingi katta narsa 1957 yilda Frank Rosenblatt tomonidan ixtiro qilingan **Perceptron**. U hamkorlari Alexander Stieber va Robert H. Shatz bilan birgalikda o'zlarining ixtirolarini "Percepttron - Avtomatikni anglash va tanib olish" deb nomlangan hisobotda hujjatlashtirdilar: The Perceptron – A Perceiving and Recognizing Automaton. Perceptron ikkilik tasniflash vazifalari asosida qurilgan. Perceptronni o'rganish qoidasi bilan neyronlarning vaznlari va biasni o'rganish mumkin. Vaznlar ham musbat, ham manfiy bo'lishi mumkin. Frank Rosenblatt Perceptron modelining imkoniyatlari haqida qat'iy talablar qo'ydi. Afsuski, ularning hammasi ham haqiqatga aylanmadi.

- Marvin Minsky va Seymour A. Papert 1969 yilda *Perceptronlar: Hisoblash Geometriyasiga kirish* nomli kitob yozdi ("Perceptrons: An Introduction to Computational Geometry", MIT Press). Bu kitobda Perceptronnining o'rganish algoritmi chaklanganligi ko'rsatib berildi: masalan, XOR Boolean funksiyasini Perceptron yordamida rivojlantirish kabi oddiy vazifalar ham imkonsiz ekanligi ko'rsatilgan. Sun'iy Neyron To'rlari hamjamiyatining yaxshiroq qismi, Minski va Papert tomonidan

berilgan cheklashlar barcha neyron to'rlariga taalluqli ekanligini va shuning uchun sun'iy neyron to'rlarida tadqiqotlar deyarli o'n yillarga, 1980 yillargacha to'xtatilganligini tushundi.

- 1980 yillarda sun'iy neyron to'rlariga bo'lgan qiziqish, birinchi navbatda, ko'p qatlamli muammolarni o'rganishda ortga tarqalish usuli va neyron to'rlarining nochiziqli tasnifni yechish qobiliyati tufayli, Geoffrey Hilton, David Rumelhart, Ronald Williams va boshqalar tomonidan jonlantirildi.

- 1990-yillarda V. Vapnik va C. Cortes tomonidan ixtiro qilingan qo'llab-quvvatlovchi vektorli mashinalar (support vector machines, SVM) katta muammolarga duch kelmaganligi sababli ommalashib ketdi.

- Sun'iy neyron to'rlari 2006 yilda Geoffrey Hinton va boshqalar nazoratsiz tayyorgarlikdan oldingi va chuqur ishonuvchan tarmoqlar g'oyasini kiritganlarida chuqur o'rganish nomi o'zgartirildi. Ularning chuqur ishonuvchan to'rlaridagi ishlari "Chuqur ishonch tarmoqlarni tezkor o'rganish algoritmi" nomli maqolada chop etildi. Maqola A Fast Learning Algorithm for Deep Belief Nets da joylashgan.

- ImageNet, etiketli rasmlarning katta to'plami, 2010 yilda Stenfordda bir ilmiy guruh tomonidan yaratilgan va chiqarilgan.

- 2012 yilda Alex Krizhevsky, Ilya Sutskever va Geoffrey Hinton ImageNet tanlovida 16 foiz xatolikka erishganligi uchun g'olib bo'lishgan, dastlabki ikki yil ichida eng yaxshi modellar 28 foiz va 26 foiz xatolarga duch kelishgan. Bu katta yutuq marjasi edi. Ushbu yechimni amalga oshirish bugungi kunda har qanday chuqur o'rganishni amalga oshirishda odatiy bo'lgan chuqur o'rganishning bir necha jihatlariga ega.

  - Modelni o'rgatish uchun grafik ishlov berish bloklari (Graphical processing units, GPUs) ishlatilgan. GPUlar matritsali operatsiyalarni bajarishda juda yaxshi va ular juda tez ishlaydi, chunki parallel hisoblashlarni bajarish uchun minglab yadro mavjud.
  - "Dropout" ortiqcha ishlarni qisqartirish uchun tartibga solish usuli sifatida ishlatilgan.
  - Yashirin qatlamlarni faollashtirish funksiyalari sifatida to'g'rilangan chiziqli birliklar (Rectified Linear Units, ReLUs) ishlatilgan.

2

Sun'iy o'rganish: Matematik asoslari

Har qanday Sun'iy o'rganish algoritmining markazida Matematika turadi. Matematikaning asosiy konsepsiyalarini puxta tushunish biror Sun'iy o'rganish masalasiga tegishli algoritmlarni tog'ri tanlashga imkon beradi. Bundan tashqari, u Sun'iy o'rganish modellarini mukammallashtirish va kerakli algoritmni qurishdagi ehtimoliy xatoliklarni aniqlashga ko'mak beradi. Matematika fan sifatida juda keng, ammo Sun'iy o'rganish bo'yicha mutaxassislar va/yoki havaskorlar bilishi kerak bo'lgan bir nechta zaruriy mavzular mavjud. Bu mavzularni bilmasdan turib Sun'iy o'rganishdek ajoyib sohadan yetarlicha foyda olish mmkin emas. Quyida matematikaning kerakli bo'limlari, shuningdek ularning Sun'iy o'rganish sohasidagi ahamiyati (imkon qadar) tartib bilan yoritilgan. Biz ushbu bobda har bir bo'limga tegishli tushunchalarni muhokama qilamiz. Mana o'sha bo'limlar:

- Chiziqli Algebra
- Ehtimollar Nazariyasi va Statistika
- Differensial va integral hisob
- Sun'iy o'rganish algoritmlarini optimallashtirish va shakllantirish.

2.1 Chiziqli Algebra

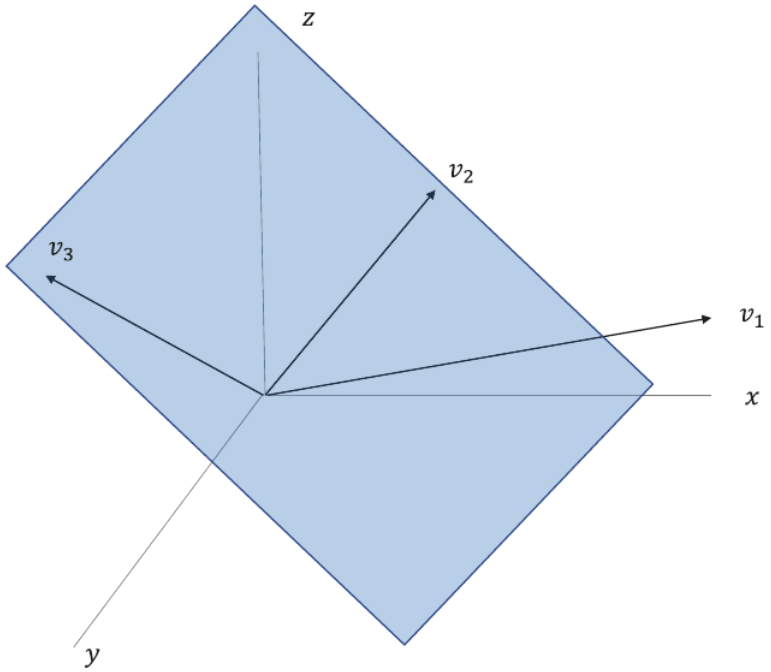
Chiziqli algebra - bu matematika bo'limlaridan biri bo'lib, bunda vektorlar va ularni vektorlar fazosida akslantirish kabi masalalar qaraladi. Qayd etish lozimki, Sun'iy o'rganish jarayonida biz ko'p o'lchamli ma'lumotlar bazasi va ularni boshqarish bilan shug'ullanamiz. Shunday ekan, deyarli barcha Sun'iy o'rganish algoritmlarida chiziqli algebra hal qiluvchi rol o'ynaydi. 2.1-rasmda uch o'lchamli ($3D$) vektorlar fazosi tasvirlangan bo'lib, unda $\mathbf{v}_1$, $\mathbf{v}_2$ va $\mathbf{v}_3$ - ixtiyoriy vektorlar va P - ikki o'lchovli ($2D$) tekislik.

2.1.1 Vektor kattalik

Mazkur mavzuni boshlashdan oldin bir konsepsiyani, ya'ni Yevklid masofani ta'riflashimiz kerak bo'ladi. Agar ikki-o'lchamli ($2D$) koordinatalar sistemasi bo'lsa va unda ikki nuqta $p_1 := (x_1, y_1)$ va $p_2 := (x_2, y_2)$ berilgan bo'lsa, nuqtalar orasidagi fazoviy masofa

$$d(p_1, p_2) := \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

ifoda bilan aniqlanishi mumkin. Bu masofa **Yevklid masofasi** deb ataladi. Agar fazoni ta'riflashda Yevklid masofasidan foydalansak, Yevklid fazosiga



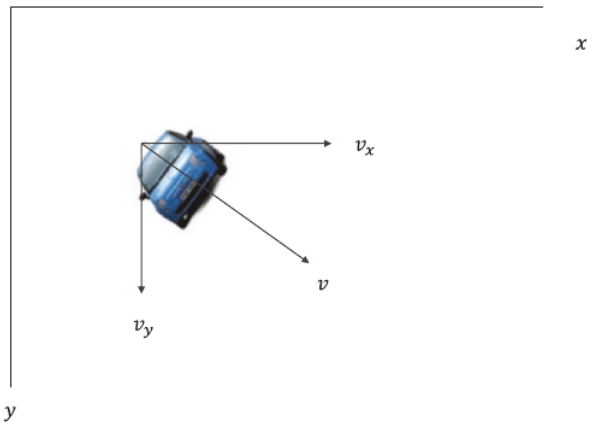
Rasm 2.1: Ixtiyoriy 3 ta vektor va ikki-o'lchamli ($2D$) vektor tekislikning uch-o'lchamli ($3D$) vektorlar fazosida joylashishi.

duch kelamiz. Yevklid fazosi eng keng tarqalgan tur bo'lib, u bizning fazoviy sezgilarimizga mos keladi. Ushbu kitobda biz faqat Yevklid fazosidan foydalanib ish ko'ramiz.

Keling endi e'tiborimizni vektorlarga qarataylik: n-o'lchamli vektor $\mathbf{x}$ bu $(x_1, \ldots, x_n)$ ko'rinishdagi kattalik bo'lib, har bir x_i element vektorning i-komponentasi deyiladi. Ma'lumki, n-o'lchamli vektorni n-o'lchamli fazoda nuqta sifatida tasvirlash juda osondir. Ushbu fazo (to'liq aniqlangan bo'lsa) **vektorlar fazosi** deb nomlanadi. Misol uchun, 2.2-rasmda v_x va v_y tezlik komponentalari bilan $2D$ fazoda – tekislikda harakatlanuvchi avtomobilning tezligi mos ravishda x va y yo'nalishlarda tasvirlangan.

Dasturlash sohasida vektor tushunchasi bir oz boshqacha ifoda etiladi. Unga ko'ra, uzluksiz yoki diskret sonlarning massivi vektor, vektorlardan tashkil topgan fazo esa vektor fazosi deyiladi. Vektorlar fazosining o'lchamlari chekli yoki cheksiz bo'lishi mumkin, ammo ko'pchilik Sun'iy o'rganish yoki Ma'lumot ilmlari ("Data-science") bilan bog'liq masalalarda aniq uzunlikdagi vektorlar bilan ish ko'riladi.

Yana bir bor eslatib o'tamizki, Sun'iy o'rganishda biz ko'p o'lchamli ma'lumotlar qatori bilan ishlaymiz va shuning uchun vektorlar juda muhimdir.



Rasm 2.2: Tezlik komponentalari v_x va v_y bo'lgan va ikki o'lchamli vektor fazo – xy tekislikda harakatlanayotgan avtomobil.

Aytaylik, biz biror hududdagi uy-joy narxini uyning maydoni, yashash xonalari soni, vannaxonalar soni va ushbu hududdagi aholi zichligiga qarab taxmin qilmoqchimiz. Bu xususiyatlarning barchasi uy-joy narxini bashorat qilish masalasi uchun kiritma vektorni ("input vector") hosil qiladi.

Quyida biz Python algoritmik tili yordamida vektor hosil qilishni ko'rsatamiz. Alohida qayd etish lozimki, NumPy - Python algoritmik tilining kutubxona (paket)laridan hisoblanadi. Aniqrog'i, NumPy bu ilmiy hisoblash uchun paket bo'lib, unda n-o'lchamli massiv ob'ekti ishlatiladi. Shuningdek, NumPy - Sun'iy o'rganish bo'g'inining asosidir. Chunki, NumPy Sun'iy o'rganishda juda ko'p ishlatiladigan ma'lumot tuzilmalari (vektorlar, matritsalar va tenzorlar) bilan samarali ishlashga imkon beradi. NumPy ushbu kitobning diqqat markazida bo'lmasa ham, ushbu bobda tez-tez namoyish etiladi. Shunday qilib, biz mazkur bobda Sun'iy o'rganish bo'yicha ishlashimiz mumkin bo'lgan eng keng tarqalgan NumPy operatsiyalarini ko'rib chiqamiz.

```
# Masalaning qo'yilishi: vector hosil qilishni ko'rsating
# Yechim: Bir o'lchovli qator yaratish uchun NumPy-dan
   foydalanamiz
# Kutubxonani yuklash
import numpy as np

# Satr ko'rinishida vektor hosil qilish
vector_row = np.array([1, 2, 3])

# Ustun ko'rinishida vektor hosil qilish
vector_column = np.array([[1],
                          [2],
```

```
12                                  [3]])
```

Listing 2.1: Python yordamida vektor olish

Izoh: NumPy-ning asosiy ma'lumotlar tarkibi ko'p o'lchamli massivdir ("array"). Vektor yaratish uchun biz shunchaki bir o'lchamli massiv yaratamiz. Huddi vektorlar singari, bu massivlar gorizontal (ya'ni satrlar) yoki vertikal ko'rinishda (ya'ni ustunlar) berilishi mumkin.

2.1.2 Skalyar kattalik

Skalyar bu birgina son qiymati bilan ifodalanuvchi kattalik bo'lib, bir o'lchamli vektor deb faraz qilsa ham bo'ladi. Misol sifatida bolaning bo'yi, apelsinning massasi, soat ko'rsatkichi va boshqalarni keltirish mumkin. n-o'lchamli vektor n ta skalyar qiymatlar ketma-ketligi sifatida beriladi.

Chiziqli algebrada haqiqiy sonlar yoki maydonning boshqa elementlari skalyar deb ataladi. Umuman olganda, ixtiyoriy vektor maydonni haqiqiy sonlar o'rniga biror bir maydon yordamida, xususan kompleks sonlar yordamida aniqlash mumkin. U holda bu vektor fazosining skalyarlari unga tegishli bo'lgan vektor maydonning komponentalari bo'ladi. Tushunish oson bo'lishi uchun yana 2.2-rasmga qaytamiz: avtomobilning tezlik maydoni $\mathbf{v}$ vektor hisoblanib, v_x va v_y tezlik komponentalari esa skalyar kattaliklardir.

2.1.3 Matritsa

Matritsa - bu qatorlar va ustunlarga joylashtirilgan ikki o'lchamli massiv. Matritsaning o'lchami uning satr uzunligi va ustun uzunligi bilan belgilanadi. Agar A matritsa m satr va n ustunga ega bo'lsa, u $m \times n$ elementga ega bo'lgan to'rtburchak ob'ekt sifatida tasvirlanadi va $A_{m\times n}$ deb belgilanadi:

$$A_{m\times n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

```
1  # Masalaning qo'yilishi: matritsa hosil qilishni ko'rsating
2  # Yechim: Ikki o'lchovli (2D) massiv yaratish uchun NumPy-dan
       foydalanamiz
3  # Kutubxonani yuklash
4  import numpy as np
5  # Matritsa yaratish
6  A = np.array([[1, 2],
7                [1, 2],
```

```
8                 [1, 2]])
```

Listing 2.2: Python yordamida matritsa hosil qilish

Izoh: Matritsani yaratish uchun biz $2D$ *massivdan foydalana olamiz. Yuqoridagi misolda matritsaning uchta qator* ($m = 3$) *va ikkita ustuni* ($n = 2$) *mavjud.*

2.1.4 Tenzor

Tenzor - bu ko'p o'lchamli massivdir. Aslida, vektor va matritsani mos ravishda $1D$ va $2D$ tenzorlar deb qarash mumkin. Sun'iy o'rganishda tenzorlar asosan ma'lumotlarni saqlash va qayta ishlash uchun ishlatiladi. Masalan, RGB-rangli tasvir ("Red, Green, Blue" ya'ni, R-qizil, G-yashil va B-moviy rang) 3D tenzor shaklida saqlanadi. Bunda bitta o'lchov bo'ylab biz gorizontal o'qqa, boshqa o'lchov bo'ylab vertikal o'qqa egamiz va uchinchi o'lcham RGB rang kanaliga mos keladi.

2.1.5 Matritsalar ustida amallar

Ko'p hollarda Sun'iy o'rganish protseslari matritsalar ustida amallar orqali amalga oshiriladi, masalan, ko'paytirish, qo'shish, ayirish, akslantirish va boshqalar. Shunday qilib, matritsalar utida amallarning asosiylari bilan tanishib chiqamiz.

Biror $m \times n$ matritsani yonma-yon joylashtirilgan o'lchami m bo'lgan ustun shaklidagi n ta vektorlarni o'z ichiga olgan matritsa deb qarash mumkin. Biz matritsani quyidagicha ifodalaymiz

$$A_{m\times n} \in \mathbb{R}^{m\times n}.$$

Matritsalarni qo'shish

Ikkita A va B matritsalarning yig'indisi deganda ularning mos elementlarining yig'indisi tushuniladi. Bu yig'indi amalga oshishi uchun qo'shiluvchi matritsalarning o'lchamlari bir-biriga mos kelishi kerak. Agar C matritsa A va B matritsalar yig'indisi bo'lsa, u holda

$$C_{ij} = A_{ij} + B_{ij},$$

bu yerda $\forall i \in \{1, 2, \dots m\}, \forall j \in \{1, 2, \dots n\}$.

Misol: $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$ bo'lsa, u holda

$$A + B = \begin{pmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}.$$

Matritsalarni ayirish

Ikkita A va B matritsalarining ayirmasi deganda ularning mos elementlari ayirmasi tushuniladi. Bu ayirma kamayuvchi va ayriluvchi matritsalarning o'lchamlari mos kelganda to'g'ridan-to'g'ri amalga oshiriladi.

Agar C matritsa $A - B$ ni ifodalaydigan matritsa bo'lsa, u holda

$$C_{ij} = A_{ij} - B_{ij},$$

bu yerda $\forall i \in \{1, 2, \dots m\}, \forall j \in \{1, 2, \dots n\}$.

Misol: $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$ bo'lsa, u holda

$$A - B = \begin{pmatrix} 1-5 & 2-6 \\ 3-7 & 4-8 \end{pmatrix} = \begin{pmatrix} -4 & -4 \\ -4 & -4 \end{pmatrix}.$$

```
# Masalaning qo'yilishi: ikkita matritsani qo'shish va ayrish
# Yechim: NumPy-dan foydalanib ikkita matritsa qo'shiladi va
    ayriladi

# Kutubxonani yuklash
import numpy as np

# Matritsa A ni yaratish
matrix_A = np.array([[1, 1, 1],
                     [1, 1, 1],
                     [1, 1, 2]])

# Matritsa B ni yaratish
matrix_B = np.array([[1, 3, 1],
                     [1, 3, 1],
                     [1, 3, 8]])

# A + B:
np.add(matrix_A, matrix_B)

# A - B:
np.subtract(matrix_A, matrix_B)

#-------------------------------------------------------------
# Yuqoridagilarning muqobilida, +/- operatorlarni qo'llash
    ham mumkin:
#-------------------------------------------------------------

# A + B:
matrix_A + matrix_B

```

```
# A - B:
matrix_A - matrix_B
```

Listing 2.3: Python yordamida ikkita matritsani qo'shish va ayrish

Matritsalarni ko'paytirish

Ikkita $A \in R^{m\times n}$ va $B \in R^{p\times q}$ matritsalar bir-biriga ko'paytirilishi uchun $n = p$ tenglik bajarilishi kerak. Olingan matritsa $C \in R^{m\times q}$ elementlari quyidagicha ifodalanishi mumkin:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj},$$

bu yerda $\forall i \in \{1, 2, \dots m\}, \forall j \in \{1, 2, \dots q\}$.

Misol: $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ va $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$ berilgan bo'lsa, c_{ij} matritsa elementlarini quyidagicha aniqlaymiz:

- $c_{11} = \begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 5 \\ 7 \end{pmatrix} = 1 \times 5 + 2 \times 7 = 19$
- $c_{12} = \begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 6 \\ 8 \end{pmatrix} = 1 \times 6 + 2 \times 8 = 22$
- $c_{21} = \begin{pmatrix} 3 & 4 \end{pmatrix} \begin{pmatrix} 5 \\ 7 \end{pmatrix} = 3 \times 5 + 4 \times 7 = 43$
- $c_{22} = \begin{pmatrix} 3 & 4 \end{pmatrix} \begin{pmatrix} 6 \\ 8 \end{pmatrix} = 3 \times 6 + 4 \times 8 = 50.$

$$C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}.$$

```
# Masalaning qo'yilishi: ikkita matritsani ko'paytirish
# Yechim: NumPy-da "dot" buyrug'idan foydalanish mumkin.
# dot inglizcha "dot-product" so'zidan olingan

# Kutubxonani yuklash
import numpy as np

# Matritsa A ni yaratish
matrix_A = np.array([[1, 2],
                     [3, 4]])

```

```
# Matritsa B ni yaratish
matrix_B = np.array([[5, 6],
                     [7, 8]])

# Ikkita matritsani ko'paytirish
np.dot(matrix_A, matrix_B)

#------------------------------------------------------------
# Python 3.5+ versiyalar uchun dot-ning muqobilida, @
   operatorini qo'llash ham mumkin:
#------------------------------------------------------------

# Ikkita matritsani ko'paytirish
matrix_A @ matrix_B

#------------------------------------------------------------
# Ikki matritsaning mos elementlarini ko'paytirish uchun *
   operatori qo'llaniladi:
#------------------------------------------------------------

# Ikki matritsaning mos elementlarini ko'paytirish
matrix_A * matrix_B
```

Listing 2.4: Python yordamida ikkita matritsani ko'paytirish

Matritsalarni transponirlash/akslantirish

$A \in \mathbb{R}^{m\times n}$ matritsaning transponiri odatda $A^T \in \mathbb{R}^{n\times m}$ ko'rinishda ifodalanadi va ustun vektorlarini qator vektorlari sifatida o'tkazish orqali olinadi.

$$a'_{ji} = a_{ij},$$

bu yerda $\forall i \in \{1, 2, \dots m\}$, $\forall j \in \{1, 2, \dots n\}$; $a'_{ji} \in A^T$ va $a_{ij} \in A$. Masalan, $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ berilgan bo'lsa, u holda $A^T = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$.

```
# Masalaning qo'yilishi: matritsaning transponirini toping
# Yechim: T metoddan foydalanamiz

# Kutubxonani yuklash
import numpy as np

# Matritsani yaratish
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

```

```
12
13 # matritsaning transpozitsiyasi
14 matrix.T
```

Listing 2.5: Python-da matritsaning transponirini topish

Ikkita A va B matritsalar ko'paymasining transponiri A va B matritsalar transponirining teskari tartibdagi ko'paytmasiga teng, ya'ni $(AB)^T = B^T A^T$.

Isbotlaymiz: bizga $A = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$ va $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$ berilgan bo'lsa, u holda matritsalarning ko'paytmasi $(AB) = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 95 & 132 \\ 301 & 400 \end{pmatrix}$. Demak, $(AB)^T = \begin{pmatrix} 95 & 301 \\ 132 & 400 \end{pmatrix}$.

Boshqa tomondan, $A^T = \begin{pmatrix} 19 & 43 \\ 22 & 50 \end{pmatrix}$ va $B^T = \begin{pmatrix} 5 & 7 \\ 6 & 8 \end{pmatrix}$,

$$B^T A^T = \begin{pmatrix} 5 & 7 \\ 6 & 8 \end{pmatrix} \begin{pmatrix} 19 & 43 \\ 22 & 50 \end{pmatrix} = \begin{pmatrix} 95 & 301 \\ 132 & 400 \end{pmatrix}.$$

Ko'rinib turiptiki $(AB)^T = B^T A^T$ o'rinli ekan.

```
1  # Masalaning qo'yilishi: Yuqoridagi tenglikni Python-da
       tekshiring
2  # Yechim:
3
4  # Kutubxonani yuklash
5  import numpy as np
6
7  # A matritsa
8  matrix_A = np.array([[19, 22],
9                       [43, 50]])
10
11 # B matritsa
12 matrix_B = np.array([[5, 6],
13                       [7, 8]])
14
15 # Yangi belgilashlar kiritamiz:
16
17 # x = (AB)^T:
18 x = (matrix_A * matrix_B).T
19
20 # y = B^T * A^T
21 y = matrix_B.T * matrix_A.T
22
23
24 # x va y matritsalarning har bir matritsa elementlari mos
       ravishda tengligini tekshiramiz:
```

```
print ( x.all() == y.all() )

# Dastur amalga oshirganingizdan so'ng "True" ya'ni to'g'ri
    degan yozuv chiqadi
```

Ikki vektorning skalyar ko'paytmasi

Har qanday n-o'lchamli vektorni $\mathbf{v} \in \mathbb{R}^{n\times 1}$ matritsa shaklida ko'rsatish mumkin. Ikkita n-o'lchamli vektorlar, $\mathbf{v}_1 \in \mathbb{R}^{n\times 1}$ va $\mathbf{v}_2 \in \mathbb{R}^{n\times 1}$, berilgan bo'lsin

$$\mathbf{v}_1 = \begin{pmatrix} v_{11} \\ v_{12} \\ \vdots \\ v_{1n} \end{pmatrix}, \mathbf{v}_2 = \begin{pmatrix} v_{21} \\ v_{22} \\ \vdots \\ v_{2n} \end{pmatrix}.$$

Ikki vektorning skalyar ko'paytmasi (inglizchada "dot-product") bu vektorlar mos komponentalari ko'paytmasining yig'indisidir:

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = \mathbf{v}_1^T \mathbf{v}_2 = \mathbf{v}_2^T \mathbf{v}_1 = v_{11}v_{21} + v_{12}v_{22} + \ldots + v_{1n}v_{2n} = \sum_{k=1}^{n} v_{1k}v_{2k}.$$

Misol uchun, $\mathbf{v}_1 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \mathbf{v}_2 = \begin{pmatrix} 3 \\ 5 \\ -1 \end{pmatrix}$ berilgan bo'lsin. Bu vektorlarning skalyar ko'paytmasi: $\mathbf{v}_1 \cdot \mathbf{v}_2 = \mathbf{v}_1^T \mathbf{v}_2 = 1 \times 3 + 2 \times 5 - 3 \times 1 = 10$.

```
# Masalaning qo'yilishi: Python-da ikki vector uchun "dot-
    product"ni toping
# Yechim:

# Kutubxonani yuklash
import numpy as np

# Ikkita vektor hosil qilamiz
vector_a = np.array([1,2,3])
vector_b = np.array([4,5,6])

# dot product ni hisoblash
np.dot(vector_a, vector_b)

#-------------------------------------------------------------
# Python 3.5+ versiyalar uchun dot-ning muqobilida, @
    operatorini qo'llash ham mumkin:
#-------------------------------------------------------------

# "dot-product"ni @ bilan hisoblash
```

```
vector_a @ vector_b

# Har ikki usulda ham yechim 32
```

Listing 2.6: Ikki vektorning skalyar ko'paytmasi.

Matritsaning vektorga ta'siri

Matritsani vektorga ko'paytirganda, natijada boshqa bir vektor paydo bo'ladi. Aytaylik, $A \in \mathbb{R}^{m\times n}$ matritsa biror vektor $\mathbf{x} \in \mathbb{R}^{n\times 1}$ ga ko'paytirilayotgan bo'lsin. Natijada $\mathbf{b} \in \mathbb{R}^{m\times 1}$ vektor hosil bo'ladi.

$$A = \begin{pmatrix} c_1^{(1)} & c_1^{(2)} & \dots & c_1^{(n)} \\ c_2^{(1)} & c_2^{(2)} & \dots & c_2^{(n)} \\ \vdots & \vdots & \vdots & \vdots \\ c_m^{(1)} & c_m^{(2)} & \dots & c_m^{(n)} \end{pmatrix}, \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}.$$

Bu yerda A matritsa n ta ustundan iborat bo'lgan vektorlar $\mathbf{c}^{(i)} \in \mathbb{R}^{m\times 1}$ $(\forall i \in \{1, 2, 3, \dots n\})$ orqali tasvirlandi, ya'ni $A = \left(\mathbf{c}^{(1)}\mathbf{c}^{(2)}\mathbf{c}^{(3)} \dots \mathbf{c}^{(n)}\right)$.

$$\mathbf{b} = A\mathbf{x} = \left(\mathbf{c}^{(1)}\mathbf{c}^{(2)} \dots \mathbf{c}^{(n)}\right) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = x_1\mathbf{c}^{(1)} + x_2\mathbf{c}^{(2)} + \dots + x_n\mathbf{c}^{(n)}.$$

Ko'rinib turibdiki, $\mathbf{b}$ vektor A matritsa ustun vektorlarining chiziqli kombinatsiyasidan boshqa narsa emas, va $\mathbf{x}$ vektorning komponentalari chiziqli koeffitsientlardir.

Qayd etish kerakki, $\mathbf{b}$ vektor A ning ustun vektorlari bilan bir xil o'lchamga ega. Bu matematikaning go'zalliklaridan biri.

Keling bir misol ko'raylik.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} 2 \\ 2 \\ 3 \end{pmatrix}, \ \mathbf{b} = A\mathbf{x} = 2\begin{pmatrix} 1 \\ 4 \end{pmatrix} + 2\begin{pmatrix} 2 \\ 5 \end{pmatrix} + 3\begin{pmatrix} 3 \\ 6 \end{pmatrix} = \begin{pmatrix} 15 \\ 36 \end{pmatrix}.$$

```
# Masalaning qo'yilishi: Python-da matritsani vectorga ko'
    paytiring
# Yechim:

# Kutubxonani yuklash
import numpy as np

```

```
# Matritsani yaratish
matrix_A = np.array([[1, 2, 3],
                     [4, 5, 6]])

# Vektorni yaratish
vector_x = np.array([[2],
                     [2],
                     [3]])

# Matritsani vectorga ko'paytirish
print np.dot(matrix_A, vector_x)

#------------------------------------------------------------
# Python 3.5+ versiyalar uchun dot-ning muqobilida, @
   operatorini qo'llash ham mumkin:
#------------------------------------------------------------

# "dot-product"ni @ bilan hisoblash
print (matrix_A @ vector_x)

# Har ikki usulda ham yechim:
[[15]
 [36]]
```

Listing 2.7: Matritsaning vektorga ta'siri

Vektorlarning chiziqli bog'liqmasligi

Biror vektor boshqa bir vektorlarning chiziqli kombinatsiyasi sifatida ifodalanishi mumkin bo'lsa, bu vektor berilgan vektorlarga chiziqli bog'liq deyiladi.

Oson tushunilishi uchun bir misol keltiramiz: ixtiyoriy $\mathbf{v}_1$, $\mathbf{v}_2$ va $\mathbf{v}_3$ vektorlar uchun $\mathbf{v}_1 = 5\mathbf{v}_2 + 7\mathbf{v}_3$ o'rinli bo'lsa, u holda $\mathbf{v}_1$, $\mathbf{v}_2$ va $\mathbf{v}_3$ chiziqli mustaqil bo'la olmaydi, chunki ulardan kamida bittasi qolgan ikkita vektorning yig'indisi sifatida ifodalanishi mumkin. Umuman olganda, n ta vektordan iborat to'plam $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \ldots, \mathbf{v}_n \in \mathbb{R}^{m\times 1}$ chiziqli mustaqil (yoinki, bog'liqmas) deyiladi, faqat va faqat $a_i = 0\ \forall i \in \{1, 2, \ldots n\}$ bo'lsagina $a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + a_3\mathbf{v}_3 + \ldots + a_n\mathbf{v}_n = 0$ bajarilsa.

Agar $a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + a_3\mathbf{v}_3 + \ldots + a_n\mathbf{v}_n = 0$ shart bajarilsa, lekin hamma a_i uchun ham bo'lmasa, u holda vektorlar chiziqli mustaqil emas, ya'ni ular chiziqli bog'liq hisoblanaadi.

Berilgan vektorlar to'plami uchun, ularning chiziqli bog'liq yoki bog'liqmasligini tekshirish uchun quyidagi usuldan foydalanish mumkin.

$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + a_3\mathbf{v}_3 + \ldots + a_n\mathbf{v}_n = 0$ tenglamani quyidagicha yozamiz:

$$\begin{pmatrix}\mathbf{v}_1 & \mathbf{v}_2 & \ldots & \mathbf{v}_n\end{pmatrix}\begin{pmatrix}a_1\\a_2\\\vdots\\a_n\end{pmatrix} = 0,$$

bu yerda $\mathbf{v}_i \in \mathbb{R}^{m\times 1}\ \forall i \in \{1, 2, \ldots, n\}$, $\begin{pmatrix}a_1\\a_2\\\vdots\\a_n\end{pmatrix} \in \mathbb{R}^{n\times 1}$.

Bu tenglamani $\begin{pmatrix}a_1 & a_2 & \ldots & a_n\end{pmatrix}^T$ ga nisbatan yechishimiz davomida yechimlar ichida atiga bitta nol vektor bo'lsa ham, $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \ldots, \mathbf{v}_n$ vektorlar to'plamini chiziqli mustaqil bo'ladi, deb aytish mumkin.

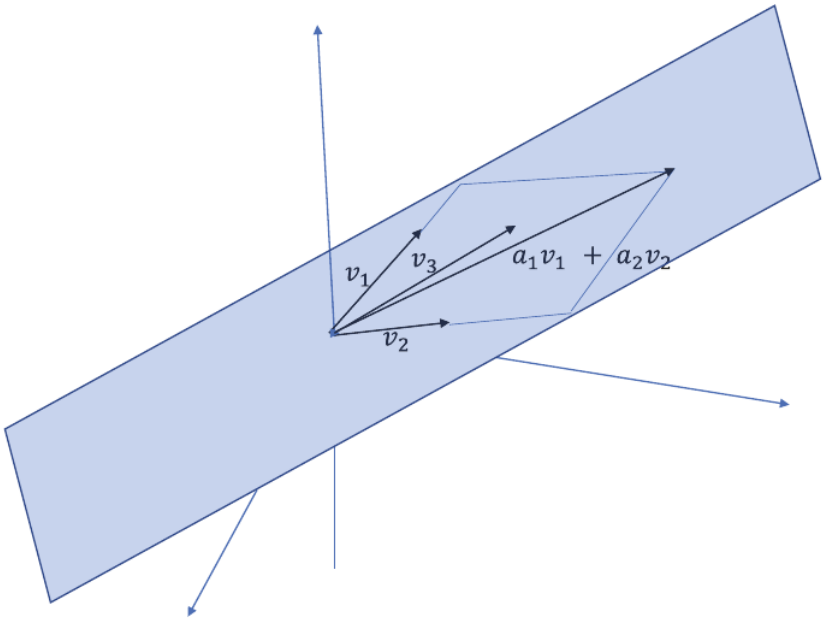
Agar n ta vektorlar to'plami $\mathbf{v}_i \in \mathbb{R}^{n\times 1}$ chiziqli mustaqil bo'lsa, bu vektorlar n o'lchovli butun fazoni egallaydi. Boshqacha qilib aytganda, n ta vektorning chiziqli kombinatsiyasini olib, n o'lchovli fazoda barcha mumkin bo'lgan vektorlarni hosil qilish mumkin. Agar n ta vektor chiziqli bog'liq bo'lsa, ular n o'lchovli fazoda faqat qism-fazoni ifodalay oladi.

Ushbu haqiqatni ko'rsatish uchun, 2.3-rasmda tasvirlanganidek uch o'lchovli fazoda vektorlarni olaylik. Agar biror vektor $\mathbf{v}_1 = \begin{pmatrix}1 & 2 & 3\end{pmatrix}^T$ berilgan bo'lsa, uning yordamida, uch o'lchovli fazoda, faqat bitta o'lchamni ko'rsatish mumkin xolos. Chunki, bu vektor orqali hosil qilingan barcha vektorlar $\mathbf{v}_1$ yo'nalishi bilan bir xil bo'lgan va faqat biror skalyar ko'paytuvchi yordamida aniqlanadi. Boshqacha qilib aytganda, har bir vektor $a_1\mathbf{v}_1$ shaklida bo'ladi.

Endi ikkinchi bir vektor $\mathbf{v}_2 = \begin{pmatrix}5 & 9 & 7\end{pmatrix}^T$ kiritaylik, uning yo'nalishi $\mathbf{v}_1$ vektor yo'nalishi bilan bir xil emas. Shunday qilib, ikki vektorning shpuri $Sp(\mathbf{v}_1, \mathbf{v}_2)$ (ya'ni $\mathbf{v}_1$ va $\mathbf{v}_2$ vektorlarning mumkin bo'lgan barcha chiziqli kombinatsiyalari to'plami) $\mathbf{v}_1$ va $\mathbf{v}_2$ ning chiziqli kombinatsiyasidan boshqa narsa emas. Ushbu ikki vektor yordamida biz ikkala vektor tekisligida yotadigan har qanday vektor $a\mathbf{v}_1 + b\mathbf{v}_2$ hosil qilishimiz mumkin. Asosan, biz $3D$ fazoda $2D$ qism-fazoni ajratamiz. Xuddi shu holat 2.3-rasmdagi diagrammada ko'rsatilgan.

Keling, uchinchi bir vektor $\mathbf{v}_3 = \begin{pmatrix}4 & 8 & 1\end{pmatrix}^T$ ni ham kiritaylik. Endi biz $Sp(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ ko'rib chiqsak, uch o'lchovli tekislikda har qanday vektorni hosil qilishimiz mumkin: istalgan uch o'lchovli vektor olinadi va oldingi uch vektorning chiziqli birikmasi sifatida ifodalanadi.

Ushbu uch vektor $3D$ fazo uchun *bazis vektorlar*ni tashkil qiladi. Har qanday chiziqli bog'liqmas uchta vektor $3D$ fazo uchun bazis bo'lib xizmat



Rasm 2.3: Ikki o'lchovli qism-fazo uch o'lchovli ($3D$) vektor fazoda $\mathbf{v}_1$ va $\mathbf{v}_2$ vektorlar orqali ifodalangan.

qilishi mumkin. Huddi shu tasdiqni har qanday n-o'lchovli fazo uchun ham umumlashtirish mumkin.

Agar biz biror $\mathbf{v}_3$ vektorni tanlab olgan bo'lsak va bu vektor boshqa ikkita $\mathbf{v}_1$ va $\mathbf{v}_2$ vektorlarning chiziqli kombinatsiyasi bo'lsa, u holda $3D$ fazoni berilgan vektorlar bilan to'lig'icha tasvirlab bo'lmaydi. Biz $\mathbf{v}_1$ va $\mathbf{v}_2$ orqali tasvirlangan ikki o'lchovli qism-fazo bilan chegaralanamiz xolos.

Matritsaning rangi

Chiziqli algebrada eng muhim tushunchalardan biri bu matritsa rangidir. Matritsaning rangi bu chiziqli bog'liqmas ustun yoki satr vektorlar sonidir. Bog'liqmas ustun vektorlar soni har doim matritsadagi bog'liqmas satr vektorlar soniga teng bo'ladi.

Misol sifatida quyidagi matritsani qaraymiz:

$$A = \begin{pmatrix} 1 & 3 & 4 \\ 2 & 5 & 7 \\ 3 & 7 & 10 \end{pmatrix}.$$

$\begin{pmatrix}1\\2\\3\end{pmatrix}$ va $\begin{pmatrix}3\\5\\7\end{pmatrix}$ ustun vektorlar chiziqli bog'liqmas. Biroq, $\begin{pmatrix}4\\7\\10\end{pmatrix}$ unday emas, chunki bu vektorni avvalgi ikkita ustun vektorning chiziqli kombinatiyasidir. Ya'ni, $\begin{pmatrix}4\\7\\10\end{pmatrix} = \begin{pmatrix}1\\2\\3\end{pmatrix} + \begin{pmatrix}3\\5\\7\end{pmatrix}$. Bundan kelib chiqadiki, berilgan A matritsaning rangi 2 ga teng va bu matritsa ikkita chiziqli bog'liqmas ustun vektorga ega.

Matritsaning rangi 2 ga teng bo'lganligi sababli, matritsaning ustun vektorlari uch o'lchovli vektor fazosida faqat ikki o'lchovli qism-fazoni tasvirlashi mumkin. Bu yerda ikki o'lchovli qism-fazo $\begin{pmatrix}1\\2\\3\end{pmatrix}$ va $\begin{pmatrix}3\\5\\7\end{pmatrix}$ vektorlar chiziqli kombinatsiyasi orqali hosil bo'ladi.

```
# Masalaning qo'yilishi: Python-da matritsaning rangini
    toping
# Yechim: NumPy-dan chiziqli algebra bo'limiga tegishli
    matrix_rank metodidan foydalanamiz

# Kutubxonani yuklash
import numpy as np

# Create matrix
matrix = np.array([[1, 1, 1],
                   [1, 1, 10],
                   [1, 1, 15]])

# Matritsaning rangi
np.linalg.matrix_rank(matrix)

# Javobi: 2

#-------------------------------------------------------------
# Izoh: Matritsaning rangi - mana shu matritsa ustun yoki
    satrlari orqali tasvirlangan vektor fazosi o'lchamlari.
    Matritsaning rangini topish, demak, NumPy-da matrix_rank
    orqali amalga oshiriladi.
#-------------------------------------------------------------
```

Listing 2.8: Matritsaning rangini topish

Muhim eslatmalar:

- Kvadratik matritsa $A \in \mathbb{R}^{n\times n}$ "to'liq rang"ga ega deyiladi, agar A matritsaning rangi n bo'lsa. Rangi n bo'lgan kvadrat matritsa shunday matritsaki, uning barcha n ustun vektorlari va/yoki n satr vektorlari

chiziqli mustaqil bo'ladi. Shu sababli A matritsasining n ustun vektorlarning chiziqli kombinatsiyasi orqali butun n-o'lchovli fazoni tasvirlash mumkin bo'ladi.

- Agar kvadrat matritsa $A \in \mathbb{R}^{n\times n}$ "to'liq rang"ga ega bo'lmasa, u singulyar matritsa ("singular matrix") hisoblanadi; ya'ni uning barcha ustun vektorlari yoki satr vektorlari chiziqli mustaqil emas. Yakkalangan matritsaning teskarisi aniqlanmagan va nol determinantiga ega.

```
# Masalaning qo'yilishi: Python-da singulyar matritsa ("singular matrix")ning rangini toping
# Yechim: NumPy-dan chiziqli algebra bo'limiga tegishli matrix_rank metodidan foydalanamiz

# Kutubxonani yuklash
import numpy as np

# Create matrix
matrix = np.array ([[1,   8,   50],
          [8,   64,  400],
          [50,  400, 2500]])

# Matritsaning rangi
np.linalg.matrix_rank(matrix)

# Javobi: 1

#-------------------------------------------------------------
# Izoh: har bir satr vektor birinchi satr vektorning chiziqli kombinatsiyasi:
# 2-satr = 8 * 1-satr;
# 3-satr = 50 * 1-satr.
# demak, faqat bitta mustaqil/bog'liqmas satr vektor bor xolos, shuning uchun Rank(matrix) = 1.
#-------------------------------------------------------------
```

Listing 2.9: Yakkalangan Matritsaga misol

Birlik matritsa yoki Operator

$I \in \mathbb{R}^{n\times n}$ matritsa *birlik matritsa* yoki operator deyiladi, agar ixtiyoriy vektor yoki matritsa unga ko'paytirilganda o'zgarishsiz qolsa. 3×3 birlik matritsa quyidagicha berilgan bo'lsin

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \in \mathbb{R}^{3\times 3}.$$

Bu matritsaning $\mathbf{v} = \begin{pmatrix} 2 & 3 & 4 \end{pmatrix}^T$ vektor bilan ko'paytmasini qaraymiz:

$$I\mathbf{v} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} = \mathbf{v}.$$

Huddi shu ma'noda, I birlik matritsaning biror $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ matritsaga ko'paytmasini tekshiramiz va shunga amin bo'lamizki, har ikkala AI va IA matritsalar ko'paytmasi A matritsasiga tengdir. Demak, ikkita ko'paytuvchi matritsalardan biri birlik matritsa bo'lganida matritsalar ko'paytmasi kommutativdir.

```
# Masalaning qo'yilishi: Python-da birlik matritsa yarating
# Yechim: NumPy-da numpy.identity metodidan foydalanamiz

# Kutubxonani yuklash
import numpy as np

# Birlik matritsa olish uchun numpy.identity(n) buyrug'idan
   foydalaniladi: bu yerda n parametr -- n x n matritsaning o
   'lchami

# 2x2 birlik matritsa:
b = np.identity(2)
print("Matrix b : \n", b)


#-----Javobi-----------------------
# Matrix b :
 [[1. 0.]
 [0. 1.]]
#----------------------------

# 3x3 birlik matritsa:
a = np.identity(3)
print("\nMatrix a : \n", a)

#-----Javobi-----------------------
Matrix a :
 [[1. 0. 0.]
  [0. 1. 0.]
  [0. 0. 1.]]
#----------------------------
```

Listing 2.10: Birlik matritsa bilan ishlash

Matritsaning determinanti

A kvadratik matritsaning determinanti qandaydir son qiymati hisoblanib, u $det(A)$ kabi belgilanadi. Buni bir necha yo'l bilan talqin qilish mumkin. $A \in \mathbb{R}^{n\times n}$ matritsa uchun determinant n o'lchovli hajmni bildiradi va bu hajm matritsaning n satr vektorlari bilan o'ralgan. Determinant nolga teng bo'lmasligi uchun, A matritsaning barcha ustun yoki satr vektorlari chiziqli mustaqil/bog'liqmas bo'lishi kerak. Agar n satr vektorlari yoki ustun vektorlari chiziqli mustaqil bo'lmasa, ular butun n o'lchovli fazoni egallay olmaydi, balki n dan kichik o'lchamdagi qism-fazoni egallay va shuning uchun n o'lchovli hajm nolga teng. $A \in \mathbb{R}^{2\times 2}$ matritsa uchun determinant quyidagicha ifodalanadi:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \in \mathbb{R}^{2\times 2},$$

$$det(A) = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}.$$

Huddi shunday, biror $B \in \mathbb{R}^{3\times 3}$ matritsaning determinanti quyidagicha topiladi:

$$B = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \in \mathbb{R}^{3\times 3},$$

$$det(B) = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}.$$

Misol uchun $A = \begin{pmatrix} 6 & 1 & 1 \\ 4 & -2 & 5 \\ 2 & 8 & 7 \end{pmatrix}$ matritsa determinantini hisoblab topamiz:

$$det(A) = 6 \times \begin{vmatrix} -2 & 5 \\ 8 & 7 \end{vmatrix} - 1 \times \begin{vmatrix} 4 & 5 \\ 2 & 7 \end{vmatrix} + 1 \times \begin{vmatrix} 4 & -2 \\ 2 & 8 \end{vmatrix} =$$
$$= 6(-14 - 40) - 1(28 - 10) + 1(32 + 4) = -306.$$

```
# Masalaning qo'yilishi: Python-da matritsaning
    determinantini toping
# Yechim: NumPy-dan chiziqli algebra bo'limiga tegishli det
    metodidan foydalanamiz

# Kutubxonani yuklash
import numpy as np

# Matritsa tuzish
A = np.array([[1, 2, 3],
```

```
                [2, 4, 6],
                [3, 8, 9]])

# A matritsa determinanti
np.linalg.det(A)

# Javobi: 0.0
# Nima uchun det(A)=0 ekanligini tushutiring

# Boshqa bir misol:

# Matritsa tuzish
B = np.array([[6, 1, 1],
                [4, -2, 5],
                [2, 8, 7]])

# B matritsa determinanti
np.linalg.det(B)

# Javobi: -306.0
```

Listing 2.11: Matritsaning determinantini topish

Determinantning talqini

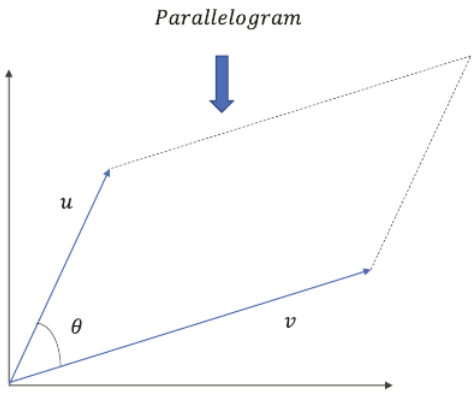
Yuqorida aytib o'tilganidek, matritsa determinantining absolyut qiymati satr vektorlari bilan o'ralgan hajmni ifodalaydi.

Eng odiy holda, $A \in \mathbb{R}^{2\times 2}$ matritsa determinanti parallelogram yuzasini ifodalab, uning tomonlari matritsadagi ikkita satr vektor orqali berilgan hisoblanadi. Bu tasdiqni mustahkamlash uchun $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ matritsaning determinantini topaylik. $det(A)$ ning qiymati tomonlari $\mathbf{u} = \begin{pmatrix} a & b \end{pmatrix}^T$ va $\mathbf{v} = \begin{pmatrix} c & d \end{pmatrix}^T$ vektorlar orqali berilgan parallelogram yuzasiga tengdir.

Haqiqatan ham, parallelogram yuzi $S = \mid \mathbf{u} \mid \times \mid \mathbf{v} \mid \times \sin\theta$ bu yerda θ - $\mathbf{u}$ va $\mathbf{v}$ orasidagi burchak (2.4-rasmga qarang).

$$S = \sqrt{a^2+b^2} \cdot \sqrt{c^2+d^2} \frac{(ad-bc)}{\sqrt{a^2+b^2} \cdot \sqrt{c^2+d^2}} = (ad-bc) = det(A).$$

Huddi shunday, $B \in \mathbb{R}^{3\times 3}$ matritsaning determinanti parallelepipedning hajmini ifodalab, uning yoqlari matritsadagi uchta satr vektor orqali berilgan hisoblanadi.



Rasm 2.4: Ikki vektor orqali qurilgan parallelogram.

Teskari matritsa

Biror $A \in \mathbb{R}^{n\times n}$ kvadrat matritsaning teskarisi A^{-1} kabi belgilanib, A ga ko'paytirilganda birlik matritsa $I \in \mathbb{R}^{n\times n}$ hosil bo'ladi:

$$AA^{-1} = A^{-1}A = I.$$

Qayd etish kerakki, A kvadrat matritsaning hammasi ham teskarisiga ega emas. A ga teskari matritsani topish formulasi quyidagicha:

$$A^{-1} = \frac{A_{qo'shma}}{det(A)} = \frac{(A\ ning\ algebraik\ to'ldiruvchisi)^T}{det(A)}.$$

Agar $A \in \mathbb{R}^{n\times n}$ kvadrat matritsa singulyar bo'lsa, ya'ni, agar A ning mustaqil ustun yoki satr vektorlari bo'lmasa, A ning teskarisi mavjud emas. Buning sababi singulyar matritsa uchun $det(A) = 0$ va shuning uchun teskarisi aniqlanmagan bo'ladi.

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

matritsaning elementlari a_{ij} orqali ifodalangan deb olaylik, bunda i va j matritsa elementining mos ravishda satr va ustun tartib raqamlari. Shunda, matritsa elementining algebraik to'ldiruvchisi mazkur ko'rinishga ega bo'ladi:

$$a_{ij} = (-1)^{i+j} d_{ij},$$

bu yerda d_{ij} A matritsaning i-satr va j-ustunini olib tashlashdan keyin paydo bo'lgan matritsaning determinanti.

Misol sifatida a element uchun algebraik to'ldiruvchini topamiz: $a = (-1)^{1+1}\begin{vmatrix} e & f \\ h & i \end{vmatrix} = ei - fh$; b element uchun esa: $b = (-1)^{1+2}\begin{vmatrix} d & f \\ g & i \end{vmatrix} = -(di - fg)$.

Matritsaning algebraik to'ldiruvchisi shakllantirilgandan so'ng, uning transponirlangani bizga $A_{qo'shma}$ beradi. O'z navbatida $A_{qo'shma}$ ning $det(A)$ ga nisbatidan A^{-1} topiladi.

Yuqoridagilarni bilgan holda, $A = \begin{pmatrix} 4 & 3 \\ 3 & 2 \end{pmatrix}$ matritsaning teskarisini topamiz:

- A ning algebraik to'ldiruvchisi $= \begin{pmatrix} 1(2) & -1(3) \\ -1(3) & 1(4) \end{pmatrix} = \begin{pmatrix} 2 & -3 \\ -3 & 4 \end{pmatrix}$
- $det(A) = \begin{vmatrix} 4 & 3 \\ 3 & 2 \end{vmatrix} = 8 - 9 = -1$
- $A^{-1} = \frac{\begin{pmatrix} 2 & -3 \\ -3 & 4 \end{pmatrix}^T}{-1} = \begin{pmatrix} -2 & 3 \\ 3 & -4 \end{pmatrix}$.

Matritsani teskarisi uchun bir nechta qoidalar:

- $(AB)^{-1} = B^{-1}A^{-1}$;
- $I^{-1} = I$, bu yerda I birlik matritsa.

```
# Masalaning qo'yilishi: Python-da matritsaning teskarisini
    toping
# Yechim: NumPy-dan chiziqli algebra bo'limiga tegishli inv
    metodidan foydalanamiz

# Kutubxonani yuklash
import numpy as np

# matritsani qurish
matrix = np.array([[1, 4],
                   [2, 5]])

# Matritsaning teskarisi
np.linalg.inv(matrix)

# Natija:
[[-1.66666667,  1.33333333],
[ 0.66666667, -0.33333333]]

#------------------------------------------------------------
```

```
# Izoh: NumPy-da, agar A ^ -1 mavjud bo'lsa, uni hisoblash
   uchun linalg.inv-dan foydalanish mumkin. Buni amalda ko'
   rish uchun biz matritsani teskarisiga ko'paytiramiz va
   natijada birlik matritsa bo'ladi:
#------------------------------------------------------------

# Matritsani uning teskarisiga ko'paytirish:
matrix @ np.linalg.inv(matrix)

# Natija:
[[ 1.,  0.],
[ 0.,  1.]]
# ----------------------------------------
# Yana bir misol:
# matritsani qurish
A = np.array([[4, 3],
              [3, 2]])

# Natija:
[[-2.  3.]
 [ 3. -4.]]

# Matritsani uning teskarisiga ko'paytirish:
A @ np.linalg.inv(A)

# Natija:
[[1. 0.]
 [0. 1.]]
```

Listing 2.12: Matritsaning teskarisini topish

Vektorning normasi

Vektor normasi uning kattaligining o'lchovidir. Bunday normalarning bir necha turlari mavjud. Eng keng foydalaniladigani - Yevklid normasidir va u ba'zan l^2 sifatida ham yoziladi.

$\mathbf{x} \in \mathbb{R}^{n\times 1}$ vektor uchun l^2 norma quyidagicha topiladi:

$$\|\mathbf{x}\|_2 = \left(|x_1|^2 + |x_2|^2 + \ldots + |x_n|^2\right)^{1/2} = (\mathbf{x}.\mathbf{x})^{1/2} = (\mathbf{x}^T\mathbf{x})^{1/2}.$$

Shunga o'xshash, l^1 norma vektor komponentlarining absolyut qiymatlari yig'indisidir:

$$\|\mathbf{x}\|_1 = |x_1| + |x_2| + \ldots + |x_n|.$$

Umuman olganda, vektorning l^p normasini quyidagicha aniqlash mumkin, bu yerda $1 < p < \infty$:

$$\|\mathbf{x}\|_p = \left(|x_1|^p + |x_2|^p + \ldots + |x_n|^p\right)^{1/p}.$$

Agar $p \to \infty$ bo'lsa, norma "Supremum normasi" deb nomlanadi va quyidagicha aniqlanadi:

$$\lim_{p\to\infty} \|\mathbf{x}\|_p = \lim_{p\to\infty} (|x_1|^p + |x_2|^p + \ldots + |x_n|^p)^{1/p} = max(x_1, x_2, \ldots, x_n).$$

Odatda, Sun'iy o'rganish masalalarida l^2 va l^1 normalari bir necha maqsadlarda ishlatiladi. Masalan, chiziqli regressiyada foydalanadigan eng kichik kvadratik qiymat funksiyasi ("cost function") xatolik vektorining l^2 normasidir. Shunga o'xshab, ba'zan model o'quv ma'lumotlariga ("training data") yaxshi mos kelmaydi va yangi ma'lumotlarni umumlashtira olmaydi va biz ko'pincha o'z modelimiz uchun *regulyarizatsiya* qilishga majbur bo'lamiz. Regulyarizatsiya uchun parametr vektorining l^2 normasi ishlatilsa, u odatda *Ridge Regularizatsiya*si deb nomlanadi, uning o'rniga l^1 norma ishlatilganda esa *Lasso Regularizatsiya* deb nomlanadi.

```
# Masalaning qo'yilishi: Python-da vektorning l^1 normasini
    toping
# Yechim:

# Vectorning l^1 normasi

# Vektorning l^1 normasi vektor komponentalari absolyut
    qiymatlarining yig'indisidir.

# l^1 norma NumPy-da norm() funksiya orqali topiladi: norma
    tartibini ko'rsatuvchi parametr sifatida 1 olinadi.

# Kutubxonani yuklash
import numpy as np

# NumPy-da array funksiyani yuklash
from numpy import array

# NumPy-da norm funksiyani yuklash
from numpy.linalg import norm

# Vektorni qurish
a = array([1, 2, 3])
print(a)

# a vektorning l^1 normasini topish:
l1 = norm(a, 1)
print(l1)

#-------------------------------------------------------------
# Izoh: Dastlab 1 x 3 vektorni qurib oldik, keyin uning l^1
    normasini hisobladik. Dasturni ishga tushirib, avval
```

```
   qurilgan vektor ekranga chop etiladi, so'ngra uning l^1
   normasi:
#------------------------------------------------------------

[1 2 3]
6.0

#------------------------------------------------------------
# Izoh: The l^1 norm is often used when fitting machine
   learning algorithms as a regularization method, e.g. a
   method to keep the coefficients of the model small, and in
    turn, the model less complex.
#------------------------------------------------------------
```

Listing 2.13: Python-da vektor normalari bilan ishlash

```
# Masalaning qo'yilishi: Python-da vektorning l^2 normasini
   toping
# Yechim:

# Vektorning uzunligini uning l^2 normasi orqali hisoblash
   mumkin.

# l^2 norma "Yevklid normasi" deb ham ataladi. Bunga sabab, l
   ^2 normani topishda Yevklid fazosidagi masofa hisoblanadi
   va bu musbat masofa bo'ladi.

# l^2 norma NumPy-da norm() funksiya orqali topiladi: norma
   tartibini bildiruvchi parametr o'rniga hech narsa
   yozilmaydi ("norm() function with default parameters").

# Kutubxonani yuklash
import numpy as np

# NumPy-da array funksiyani yuklash
from numpy import array

# NumPy-da norm funksiyani yuklash
from numpy.linalg import norm

# Vektorni qurish
a = array([1, 2, 3])
print(a)

# a vektorning l^2 normasini topish:
l2 = norm(a)
print(l2)

#------------------------------------------------------------
```

```
# Izoh: Dastlab 1 x 3 vektorni qurib oldik , keyin uning l^2 normasini hisobladik. Dasturni ishga tushirib, avval qurilgan vektor ekranga chop etiladi, so'ngra uning l^2 normasi:
#------------------------------------------------------------
[1 2 3]
3.74165738677
#------------------------------------------------------------
# Izoh: Like the l^1 norm, the l^2 norm is often used when fitting machine learning algorithms as a regularization method, e.g. a method to keep the coefficients of the model small and, in turn, the model less complex. By far, the l^2 norm is more commonly used than other vector norms in machine learning.
#------------------------------------------------------------
```

Listing 2.14: Python-da vektor normalari bilan ishlash

```
# Masalaning qo'yilishi: Python-da vektorning Supremum normasini (yoki Maximum normasi) toping
# Yechim:

# Vektorning Maximum normasi deganda l^{cheksizlik} (l^inf, inf -- infinity) tushuniladi.
# Vektorning maximum normasi - bu vektor komponentalari ichida absolyut qiymati eng katta, ya'ni maximum qiymatlisidir. Shuning uchun ham bu norma shunday ataladi.


# l^inf norma NumPy-da norm() funksiya orqali topiladi: norma tartibini bildiruvchi parametr o'rniga inf qo'yib topiladi

# Kutubxonani yuklash
import numpy as np

# NumPy-dan infinity-ni yuklash
from numpy import inf

# NumPy-dan array funksiyani yuklash
from numpy import array

# NumPy-da norm funksiyani yuklash
from numpy.linalg import norm

# Vektorni qurish
a = array([1, -4, 3])
print(a)
# a vektorning l^inf normasini topish
maxnorm = norm(a, inf)
```

```
print(maxnorm)

#------------------------------------------------------------
# Izoh: Dastlab 1 x 3 vektorni qurib oldik, keyin uning l^inf
    normasini hisobladik. Dasturni ishga tushirib, avval
   qurilgan vektor ekranga chop etiladi, so'ngra uning l^inf
   normasi:
#------------------------------------------------------------
[1 -4 3]
4.0
#------------------------------------------------------------
# Izoh: Max norm is also used as a regularization in machine
   learning, such as on neural network weights, called max
   norm regularization.
#------------------------------------------------------------
```

Listing 2.15: Python-da vektor normalari bilan ishlash

Psevdo-teskari matritsa

$A\mathbf{x} = \mathbf{b}$ tenglama berilgan bo'lsin, unda $A \in \mathbb{R}^{n\times n}$ va $\mathbf{b} \in \mathbb{R}^{n\times 1}$ berilgan va biz tenglamani $\mathbf{x} \in \mathbb{R}^{n\times 1}$ uchun yechishimiz kerak. Agar A singulyar emas va uning teskarisi mavjud bo'lsagina, bu tenglamaning yechimini $\mathbf{x} = A^{-1}\mathbf{b}$ ko'rinishda topishimiz mumkin.

Ammo, agar $A \in \mathbb{R}^{m\times n}$, ya'ni, agar A to'rtburchak matritsa bo'lsa va $m > n$ bo'lsa, u holda A^{-1} mavjud emas va demak, yuqoridagi yo'l bilan $\mathbf{x}$ ni topa olmaymiz. Bunday hollarda biz $\mathbf{x}^* = (A^TA)^{-1}A^T\mathbf{b}$ ko'rinishdagi optimal yechimni olamiz. $(A^TA)^{-1}A^T$ matritsa psevdo-teskari (yoki, Moore-Penrose psevdo-teskari matritsa) deb ataladi, chunki u optimal yechimni ta'minlash uchun teskari matritsa bo'lib xizmat qiladi.

```
# Masalaning qo'yilishi: Python-da matritsaning psevdo-
   teskarisini toping
# Yechim: NumPy-dan chiziqli algebra bo'limiga tegishli pinv
   metodidan foydalanamiz

# Kutubxonani yuklash
import numpy as np

# 4 x 5 Matritsani qurish
A = np.array([[1, 2104, 5, 1, 45],
              [1, 1416, 3, 2, 40],
              [1, 1534, 3, 2, 30],
              [1, 852,  2, 1, 36]])

# b vektorni qurish
b = np.array([[460],
```

```
               [232],
               [315],
               [178]])

# A matritsaning Moore-Penrose psevdo-teskarisini topish:
pseudo_inv = np.linalg.pinv(A)
print (pseudo_inv)

# Natijada 5 x 4 matritsa olinadi:
[[-3.54295655e-01 -3.26297205e+00  2.30499970e+00  2.14781421
   e+00]
 [ 9.84762983e-05 -2.99261486e-03  2.79517835e-03  8.71527337
   e-04]
 [ 2.88749831e-01  1.05485650e+00 -9.65470411e-01 -7.27902810
   e-01]
 [-3.48938544e-01  1.37422970e+00 -3.42942596e-01 -8.04774699
   e-01]
 [ 1.16202032e-03  6.46871447e-02 -6.70168955e-02  1.02840226
   e-02]]

# Tenglamaning yechimini topib qo'yamiz: x* = pseudo_inv @ b
x_star = pseudo_inv @ b
print (x_star)

# Natijada 5 x 1 vektor olinadi:
[[188.40031942]
 [  0.3866255 ]
 [-56.13824955]
 [-92.9672536 ]
 [ -3.73781915]]
#------------------------------------------------------------
# Izoh: Agar matritsaning determinanti nol yoki aniqlanmagan
   bo'lsa, bu matritsa teskari matritsaga ega bo'lmaydi va
   linalg.inv funktsiya ishlamaydi. Bu holatlarda linalg.pinv
    funksiya bilan ishlash mumkin.
#------------------------------------------------------------
```

Listing 2.16: Matritsaning psevdo-teskarisini topish

Ma'lum bir vektor yo'nalishi bo'yicha birlik vektor

Muayyan vektor yo'nalishi bo'yicha birlik vektor shu vektorni o'zining moduliga nisbati orqali topiladi. Misol uchun, $\mathbf{x} = \begin{pmatrix} 3 & 4 \end{pmatrix}^T$ vektor yo'nalishidagi birlik vektor Yevklid fazosida quyidagicha bo'ladi:

$$\frac{\mathbf{x}}{\|x\|_2} = \frac{\mathbf{x}}{(\mathbf{x}^T\mathbf{x})^{1/2}} = \frac{\begin{pmatrix} 3 & 4 \end{pmatrix}^T}{5} = \begin{pmatrix} 0.6 & 0.8 \end{pmatrix}^T.$$

```
# Masalaning qo'yilishi: Biror vektor yo'nalishi bo'yicha
   birlik vektor toping
# Yechim: NumPy-da bu uchun maxsus funksiya mavjud emas, va u
    vektorni uning uzunligiga bo'lib topiladi.

# Kutubxonani yuklash
import numpy as np

# Vektorni qurish
x = np.array([[460],
              [232],
              [315],
              [178]])

# Vektorning uzunligi
length = np.linalg.norm(x)

# x vektor yo'nalishidagi birlik vektor, u:
u = x / length
print (u)

# Natija:
[[0.73068083]
 [0.36851729]
 [0.50035753]
 [0.28274171]]
#-------------------------------------------------------------
# Izoh: Birlik vektorning uzunligi 1 ga teng.
#-------------------------------------------------------------
```

Listing 2.17: Biror vektor yo'nalishi bo'yicha birlik vektor topish

Vektorning boshqa Vektor yo'nalishi bo'yicha proeksiyasi

Biror $\mathbf{v}_1$ vektorning boshqa bir $\mathbf{v}_2$ vektor yo'nalishiga proyeksiyasi deganda $\mathbf{v}_1$ bilan $\mathbf{v}_2$ yo'nalishdagi birlik vektor skalyar ko'paytmasi tushuniladi:

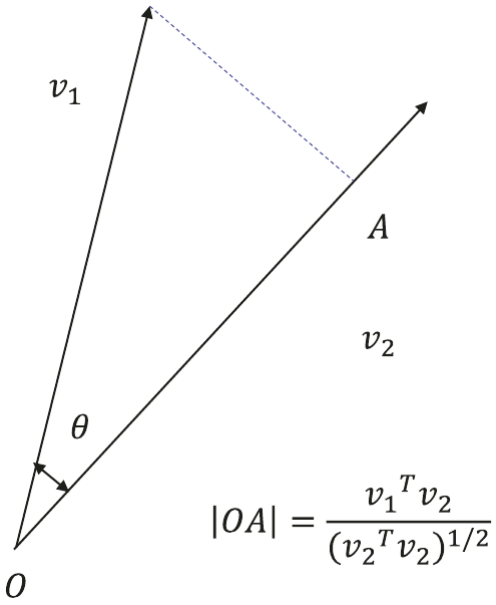
$$\|\mathbf{v}_{12}\| = \mathbf{v}_1^T \mathbf{u}_2.$$

Mazkur formulada $\|\mathbf{v}_{12}\|$ bu $\mathbf{v}_1$ ning $\mathbf{v}_2$ yo'nalishga proyeksiyasi va $\mathbf{u}_2$ bu $\mathbf{v}_2$ yo'nalishdagi birlik vektor.

Birlik vektorning ta'rifi bo'yicha $\mathbf{u}_2 = \frac{\mathbf{v}_2}{\|\mathbf{v}_2\|_2}$ ifodani yozish mumkin va bundan proyeksiya uchun quyidagi formula kelib chiqadi:

$$\|\mathbf{v}_{12}\| = \mathbf{v}_1^T \mathbf{u}_2 = \mathbf{v}_1^T \frac{\mathbf{v}_2}{\|\mathbf{v}_2\|_2} = \mathbf{v}_1^T \frac{\mathbf{v}_2}{(\mathbf{v}_2^T \mathbf{v}_2)^{1/2}}.$$

Masalan, $\mathbf{v}_1 = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$ vektorning $\mathbf{v}_2 = \begin{pmatrix} 3 & 4 \end{pmatrix}^T$ vektor yo'nalishi bo'ylab



Rasm 2.5: $\mathbf{v}_1$ vektorning $\mathbf{v}_2$ ga proeksiyasi uzunligi.

proyeksiyasi bu $\begin{pmatrix}1 & 1\end{pmatrix}^T$ hamda $\begin{pmatrix}3 & 4\end{pmatrix}^T$ ning birlik vektori, ya'ni $\mathbf{u}_2 = \begin{pmatrix}0.6 & 0.8\end{pmatrix}^T$ bilan skalyar ko'paytmasidir:

$$proyeksiya = \begin{pmatrix}1 & 1\end{pmatrix} \begin{pmatrix}0.6\\0.8\end{pmatrix} = 1 \times 0.6 + 1 \times 0.8 = 1.4.$$

Bajarilgan hisoblashlarni yaxshiroq tasavvur qilish uchun 2.5-rasmdan foydalanamiz. Unga ko'ra, $\mathbf{v}_1$ va $\mathbf{v}_2$ vektorlar Yevklid fazosida, o'zaro θ burchak hosil qilib joylashgan. $\mathbf{v}_1$ vektorning $\mathbf{v}_2$ vektor yo'nalishiga proeksiyasi OA segment chizig'iga mos keladi va uning uzunligi

$$l_{OA} = \frac{\mathbf{v}_1^T \mathbf{v}_2}{(\mathbf{v}_2^T \mathbf{v}_2)^{1/2}}$$

formula bilan ifodalanadi.

Xususiy vektorlar

Biz hozir chiziqli algebraning eng muhim tushunchalardan bo'lgan - Xususiy vektor va Xususiy qiymat mavzulari muhokamasiga yetib keldik. Xususiy qiymat va Xususiy vektor masalasiga Sun'iy o'rganishning bir necha sohalarida duch kelinadi. Masalan, prinsipial-komponentlar analizidagi asosiy komponentalar kovariatsion matritsasining xususiy vektorlari, xususiy qiymatlari esa asosiy komponentlar kovariatsiyalardir. Shuningdek, Google qidiruv

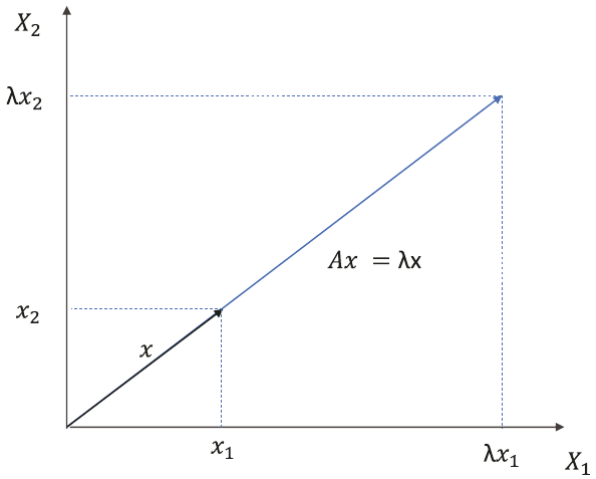
motorining web-sahifalarni reytinglash ("page-rank") algoritmida "page-rank score" vektori web-sahifaga o'tish ehtimoli matritsasining Xususiy vektoridir; bu matritsaning Xususiy qiymati 1 qiymatiga mos keladi.

Matritsa vektorga operator sifatida ta'sir etadi. Matritsaning vektorga bu ta'siri vektorni boshqa vektorga aylantirishdan iborat bo'lib, natijaviy vektorning o'lchamlari dastlabki vektor bilan bir xil bo'lishi yoki bo'lmasligi mumkin. Bu esa matritsaning o'lchamiga bog'liq ravishda yuzaga keladi.

$A \in \mathbb{R}^{n\times n}$ matritsa $\mathbf{x} \in \mathbb{R}^{n\times 1}$ vektorga ta'sir etsa, natijada yana $\mathbf{x} \in \mathbb{R}^{n\times 1}$ vektor hosil bo'ladi. Odatda, yangi vektorning kattaligi va yo'nalishi dastlabki vektornikidan farq qiladi. Agar, yangi hosil bo'lgan vektor dastlabki vektor bilan bir xil yo'nalishga ega yoki teskari yo'nalishga ega bo'lsa, unda bunday yo'nalishdagi har qanday vektor Xususiy vektor bo'ladi. Vektor uzunligining cho'zilish (yoki qisqarish) kattaligiga Xususiy qiymati deyiladi (2.6-rasmga qarang).

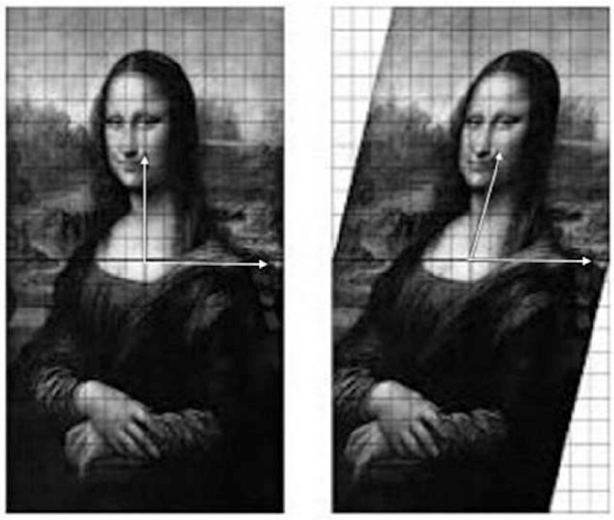
$$A\mathbf{x} = \lambda\mathbf{x},$$

bu yerda A matritsa - $\mathbf{v}$ vektorga ko'paytirish amali orqali ta'sir etuvchi operator, $\mathbf{x}$ xususiy vektor, va λ - xususiy qiymat.



Rasm 2.6: A matritsa transformatsiyasidan ta'sirlanmagan (yo'nalishi o'zgarmagan) $\mathbf{x}$ Xususiy vektor, bu yerda λ Xususiy qiymat.

2.7-rasmdan ko'rinib turibdiki, surat tekisligiga transformatsiya qo'llanilganda vertikal o'q bo'ylab joylashgan piksellar vektori yo'nalishni o'zgartirdi, vaholanki gorizontal yo'nalishda joylashgan piksellar vektori esa yo'nalishni o'zgartirmadi. Demak, gorizontal o'q bo'ylab joylashgan piksellar vektori



Rasm 2.7: Mashhur "Mona Liza" kartinasi piksellangan vektor tekisligiga transformatsiya matritsasi tatbiq etilgan.

"Mona Liza" suratiga ta'sir etgan transformatsiya matritsasining Xususiy vektoridir.

```
# Masalaning qo'yilishi: Python-da matritsaning Xususiy qiymati va Xususiy vektorini toping
# Yechim: NumPy-dan chiziqli algebra bo'limiga tegishli eig metodidan foydalanamiz.
# eig funksiya --> inglizcha "eigenvalue" (xususiy qiymat) va "eigenvector" (xususiy vektor)

# Kutubxonani yuklash
import numpy as np

# Matritsani qurish
matrix = np.array([[1, -1, 3],
                   [1, 1, 6],
                   [3, 8, 9]])

# "eigenvalue" (xususiy qiymat) va "eigenvector" (xususiy vektor) larni hisoblash
eigenvalues, eigenvectors = np.linalg.eig(matrix)

# View eigenvalues
print (eigenvalues)

# Natija:
array([ 13.55075847,   0.74003145,  -3.29078992])

```

```
# View eigenvectors
print (eigenvectors)

# Natija:
array([[-0.17622017, -0.96677403, -0.53373322],
       [-0.435951  ,  0.2053623 , -0.64324848],
       [-0.88254925,  0.15223105,  0.54896288]])
```

Listing 2.18: Matritsaning Xususiy qiymati va Xususiy vektorini topish

Matritsaning xarakteristik tenglamasi

$A \in \mathbb{R}^{n\times n}$ matritsa xarakteristik tenglamasining ildizlari matritsaning Xususiy qiymatlarini beradi. $n-$tartibli kvadrat matritsa uchun n ta Xususiy vektorga mos keladigan n ta Xususiy qiymat olish mumkin.

λ Xususiy qiymatga mos keluvchi $\mathbf{v} \in \mathbb{R}^{n\times 1}$ vektor uchun biz quyidagi tenglamani yozamiz:

$$A\mathbf{v} = \lambda\mathbf{v} \Rightarrow (A - \lambda I)\mathbf{v} = 0.$$

Bu yerda $\mathbf{v}$ Xususiy vektor sifatida nolga teng emas, demak yuqoridagi tenglama to'g'ri bo'lishi uchun $(A - \lambda I)$ singulyar bo'lishi kerak.

O'z navbatida, $(A-\lambda I)$ singulyar bo'lishi uchun $det(A-\lambda I) = 0$ bo'lishi kerak, bu esa A matritsa uchun xarakteristik tenglama hisoblanadi. Xarakteristik tenglamaning ildizlari bizga Xususiy qiymatlarini beradi. Xususiy qiymatlarini $A\mathbf{v} = \lambda\mathbf{v}$ tenglamaga olib borib va uni $\mathbf{v}$ uchun yechib Xususiy qiymatga mos keluvchi Xususiy vektor topiladi.

Misol uchun, quyidagi matritsaning Xususiy qiymat va Xususiy vektorlarini topaylik:

$$A = \begin{pmatrix} 0 & 1 \\ -2 & -3 \end{pmatrix}.$$

A matritsa uchun xarakteristik tenglama $det(A - \lambda I) = 0$ ko'rinishga ega. Tenglamani yechib

$$A = \begin{vmatrix} -\lambda & 1 \\ -2 & -3-\lambda \end{vmatrix} = 0 \Rightarrow \lambda^2 + 3\lambda + 2 = 0,$$

matritsaning Xususiy qiymatlarini topamiz $\lambda_1 = -2$, $\lambda_2 = -1$.

Birinchi Xususiy qiymat $\lambda_1 = -2$ ga mos keluvchi Xususiy vektor $\mathbf{u} = \begin{pmatrix} a & b \end{pmatrix}^T$ bo'lsin. U holda

$$\begin{pmatrix} 0 & 1 \\ -2 & -3 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = -2 \begin{pmatrix} a \\ b \end{pmatrix}.$$

Bundan quyidagi ikkita tenglama kelib chiqadi:

$$0a + 1b = -2a \Rightarrow 2a + b = 0, \tag{2.1}$$

$$-2a - 3b = -2b \Rightarrow 2a + b = 0. \tag{2.2}$$

Bu tenglamalar bir xil ko'rinishga ega, ya'ni $2a+b=0 \Rightarrow \frac{a}{b} = \frac{1}{-2}$. $a = k_1$ va $b = -2k_1$ deb olaylik, bu yerda k_1 biror o'zgarmas. Demak, birinchi Xususiy qiymat $\lambda_1 = -2$ ga mos keluvchi Xususiy vektor $\mathbf{u} = k_1 \begin{pmatrix} 1 \\ -2 \end{pmatrix}$ bo'ldi.

Yuqoridagi kabi ammallarni bajarib, ikkinchi Xususiy qiymat $\lambda_2 = -1$ ga mos keluvchi Xususiy vektorni ham topamiz, $\mathbf{w} = k_2 \begin{pmatrix} 1 \\ -1 \end{pmatrix}$.

Shuni ta'kidlash kerakki, Xususiy vektor va Xususiy qiymat har doim muayyan bir operator bilan bog'liq (yuqoridagi holatda A matritsa ham operator edi) va bu operator vektor fazosida ta'sir etadi. Lekin, Xususiy qiymat va Xususiy vektor faqatgina vektor fazosiga xos emas.

Biror funksiya ham vektor sifatida qaralishi mumkin. Aytaylik, $f(x) = e^{ax}$ funksiya berilgan bo'lsin. x ning cheksiz qiymatlarining har biri o'lcham deb qaralsa, bu qiymatlarda aniqlangan $f(x)$ qiymati shu o'lcham bo'ylab vektor komponentasi bo'ladi. Shunday qilib, biz cheksiz vektor fazosiga ega bo'lamiz.

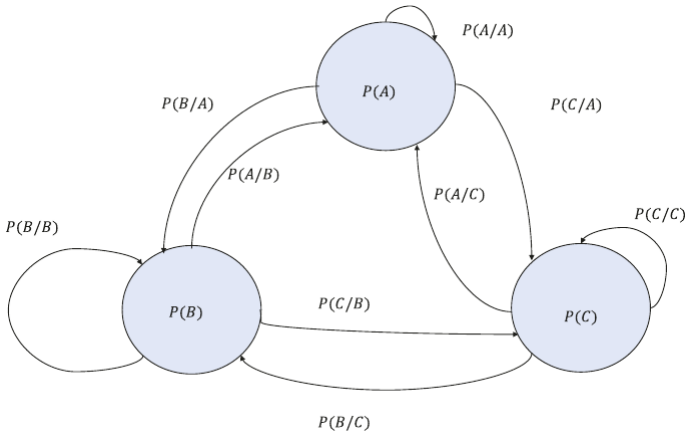
Endi, misol sifatida differensiallash operatorini ko'raylik:

$$\frac{d}{dx} f(x) = \frac{d}{dx}(e^{ax}).$$

Bu erda $\frac{d}{dx}$ - operator va e^{ax} - operatorga nisbatan Xususiy funksiya, a esa - operatorga mos keluvchi Xususiy qiymat.

Avvalroq ta'kidlab o'tilganidek, Xususiy vektor va Xususiy qiymat masalalari ko'plab sohalarda qo'llanilib, Sun'iy o'rganish sohasi bundan mustasno emas. Xususiy vektorlar zamonaviy dasturlarga qanday ta'sir qilgani haqida tasavvurga ega bo'lish uchun Google qidiruv-motorining sahifalarni reytinglash algoritmini sodda tarzda ko'raylik.

2.8-rasmda uchta A, B va C sahifalarga ega bo'lgan sodda web-sayt uchun algoritm ko'rsatilgan. Web-sozlamalarda, bir sahifadan boshqasiga o'tish mumkin, agar bosh sahifadan keyingi sahifaga o'tish havolasi (ya'ni, "link") bo'lsa, albatta. Bundan tashqari, sahifa o'z-o'ziga yo'llanma berishi ham mumkin. Shunday qilib, foydalanuvchi A sahifadan B sahifaga o'tadigan bo'lsa va bu o'tish A dan B ga o'tish linki borligi sababli sodir bo'lsa, hodisa B/A orqali ifodalanadi. $P(B/A)$ ehtimollikni A sahifadan B sahifaga



Rasm 2.8: Uchta A, B va C web-sahifalar uchun o'tish ehtimolligi diagrammasi.

tashriflarning umumiy sonini, A sahifaga tashriflarning umumiy soniga nisbati orqali topish mumkin. Qolgan barcha sahifalararo o'tish ehtimolliklari ham huddi shunday hisoblanishi mumkin. Har bir sahifaning ehtimolligi shu yo'sinda – normallashtirish orqali topilganligi sababli, sahifalar uchun individual ehtimollik sahifalar ahamiyatini aks ettiradi.

Barqaror holatda, har bir sahifaning ehtimolligi o'zgarmas bo'ladi va aynan mana shu o'zgarmas ehtimolikni hisoblashimiz kerak.

Barqaror holatda biror sahifaning ehtimolligi o'zgarmas qolishi uchun, shu sahifadan chiqib ketish ehtimollik zichligi unga kirib kelish ehtimollik zichligiga teng bo'lishi kerak, va yana ularning har biri sahifada qoladigan ehtimollik zichligi bilan to'ldirilganda sahifaning ehtimollik qiymatiga teng bo'lishi kerak. Shu nuqtai nazardan, A sahifaning atrofidagi muvozanat tenglamasini ko'rib chiqamiz: A dan chiqib ketish ehtimollik zichligi

$$P(B/A)P(A) + P(C/A)P(A),$$

A ga kirish ehtimollik zichligi esa

$$P(A/B)P(B) + P(A/C)P(C).$$

A ning o'zida qolish ehtimollik zichligi esa $P(A/A)P(A)$. Demak, muvozanat holatida tashqaridan kelish ehtimollik zichligi, ya'ni $P(A/B)P(B)+P(A/C)P(C)$ va A-da qolish ehtimollik zichligi, ya'ni $P(A/A)P(A)$ yig'indisi $P(A)$ ga teng:

$$P(A/A)P(A) + P(A/B)P(B) + P(A/C)P(C) = P(A). \qquad (2.3)$$

Shunga o'xshab, B va C sahifalar atrofidagi muvozanatni ko'rib chiqsak, quyidagilar o'rinli bo'ladi:

$$P(B/A)P(A) + P(B/B)P(B) + P(B/C)P(C) = P(B), \tag{2.4}$$

$$P(C/A)P(A) + P(C/B)P(B) + P(C/C)P(C) = P(C). \tag{2.5}$$

Endi chiziqli algebra qismiga keladik. Uchta tenglamani vektorga ta'sir qiluvchi bitta matritsaga quyidagicha joylashimiz mumkin:

$$\begin{pmatrix} P(A/A) & P(A/B) & P(A/C) \\ P(B/A) & P(B/B) & P(B/C) \\ P(C/A) & P(C/B) & P(C/C) \end{pmatrix} \begin{pmatrix} P(A) \\ P(B) \\ P(C) \end{pmatrix} = \begin{pmatrix} P(A) \\ P(B) \\ P(C) \end{pmatrix}.$$

O'tish-ehtimollik matritsasi sahifa-ehtimollik vektorini yana hosil qilish uchun sahifa-ehtimollik vektoriga ta'sir etdi. Ko'rib turganimizdek, sahifa-ehtimolligi vektori - bu sahifaga o'tish ehtimoli matritsasidagi Xususiy vektoridan boshqa narsa emas va unga mos keluvchi Xususiy qiymat 1 ga teng.

Shunday qilib, Xususiy qiymatga mos keluvchi Xususiy vektorini hisoblash bizga sahifa-ehtimollik vektorini beradi, bu esa o'z navbatida sahifalarni reytinglash uchun ishlatiladi. Nufuzli qidiruv-motorlarining sahifani reytinglash algoritmlari huddi shu printsip asosida ishlaydi. Albatta, zamonaviy qidiruv-motorlarining algoritmlarida ushbu sodda modelga bir nechta o'zgarishlar kiritilgan, ammo asosiy tushunchalar bir xil. Ehtimollik vektori keyingi bo'limda muhokama qilinganidek, darajani iteratsiyaash kabi usullar yordamida aniqlanishi mumkin.

Xususiy Vektorni hisoblashda darajani iteratsiyalash usuli

Darajani iteratsiyalash usuli bu matritsaning eng katta qiymatli Xususiy qiymatiga mos keluvchi Xususiy vektorini hisoblash uchun ishlatiladigan iteratsiya usuli.

Biror $A \in \mathbb{R}^{n\times n}$ matritsa va uning n ta $\lambda_1 > \lambda_2 > \lambda_3 > \ldots > \lambda_n$ kattalikdagi Xususiy qiymatlariga mos keluvchi $\mathbf{v}_1 > \mathbf{v}_2 > \mathbf{v}_3 > \ldots > \mathbf{v}_n$ Xususiy vektorlari berilgan bo'lsin.

Darajani iteratsiyalash shunday tasodifiy $\mathbf{v}$ vektor bilan boshlanadiki, uning biror komponentasi eng katta Xususiy qiymatga mos keladigan Xususiy vektor yo'nalishi bo'yicha bo'lishi kerak; ya'ni, $\mathbf{v}_1$ yo'nalishi bo'yicha.

Qandaydir iteratsiyadagi taqribiy Xususiy vektor quyidagicha bo'ladi:

$$\mathbf{v}^{(k+1)} = \frac{A\mathbf{v}^{(k)}}{\|A\mathbf{v}^{(k)}\|}.$$

Yetarli miqdordagi iteratsiyalardan so'ng $\mathbf{v}^{(k+1)}$ vektor $\mathbf{v}_1$ ga yaqinlashadi. Har bir iteratsiyada biz A matritsani oldingi iteratsiya qadamida olingan vektorga ko'paytiramiz. Agar vektorning normallashishini iterativ usulda uni birlik vektoriga aylantirish uchun olib tashlasak, $\mathbf{v}^{(k+1)} = A^k\mathbf{v}$.

Tasavvur qilaylik, tasodifiy tanlab olingan boshlang'ich vektor Xususiy vektorlar kombinatsiyasi sifatida ifodalasin: $\mathbf{v} = k_1\mathbf{v}_1 + k_2\mathbf{v}_2 + \ldots + k_n\mathbf{v}_n$, bu yerda $k_i\ \forall i \in \{1, 2, 3, \ldots n\}$ konstantalar.

$$\begin{aligned}\mathbf{v}^{(k+1)} = A^k\mathbf{v} &= A^k(k_1\mathbf{v}_1 + k_2\mathbf{v}_2 + \ldots + k_n\mathbf{v}_n) \\ &= k_1A^k\mathbf{v}_1 + k_2A^k\mathbf{v}_2 + \ldots + k_nA^k\mathbf{v}_n \\ &= k_1\lambda_1^k\mathbf{v}_1 + k_2\lambda_2^k\mathbf{v}_2 + \ldots + k_n\lambda_n^k\mathbf{v}_n \\ = \lambda_1^k\left(k_1\mathbf{v}_1 + k_2\left(\frac{\lambda_2}{\lambda_1}\right)^k\mathbf{v}_2 + \ldots + k_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\mathbf{v}_n\right).\end{aligned}$$

Agar k yetarlicha katta bo'lsa, ya'ni $(k \to \infty)$, birinchisidan tashqari barcha hadlar yo'qoladi, chunki

$$\left(\frac{\lambda_i}{\lambda_1}\right)^k \to 0\ \forall i \in \{1, 2, 3, \ldots n\}.$$

Demak, $\mathbf{v}^{(k+1)} = \lambda_1^k k_1\mathbf{v}_1$, va bu bizga eng katta Xususiy qiymatga mos keluvchi Xususiy vektorini beradi. Yaqinlashish tezligi eng katta Xususiy qiymat va ikkinchi eng katta Xususiy qiymat orasidagi farqqa bog'liq: agar bu ikkita Xususiy qiymatlar kattaligi jihatdan yaqin bo'lsa, metodning yaqinlashishi sekin bo'ladi.

```
# Masalaning qo'yilishi: Python-da "Power iteration" metodi yordamida Xususiy Vektorni toping
# Yechim:
# Ushbu metoddan faqat absolyut qiymati eng katta bo'lgan Xususiy qiymatni topish uchun foydalanish mumkin - biz bu Xususiy qiymatni "dominant Xususiy qiymat" deb ataymiz.

# Kutubxonani yuklash
import numpy as np

# Darajani iteratsiyalash funksiyasi yaratiladi:
def power_iteration(A, num_simulations: int):
    # Ideally choose a random vector to decrease the chance that our vector is orthogonal to the eigenvector

    # matritsaning ustunlari sonicha komponentaga ega bo'lgan vektor -- boshlang'ich vektor sifatida
    b_k = np.random.rand(A.shape[1])
```

```
14
15      for _ in range(num_simulations):
16          # calculate the matrix-by-vector product Ab
17          b_k1 = np.dot(A, b_k)
18
19          # calculate the norm
20          b_k1_norm = np.linalg.norm(b_k1)
21
22          # re normalize the vector
23          b_k = b_k1 / b_k1_norm
24
25      return b_k
26
27  A = np.array([[2, -12],
28                [1, -5]])
29
30  print ( power_iteration(A, 100) )
31
32  # Natija:
33  [0.9486833  0.31622777]
34
35  #-----------------------------------------------------------
36  # Izoh: A matritsaning Xususiy qiymatlari: -1 va -2. Demak,
      Dominant Xususiy qiymati 2. Dominant Xususiy qiymatga mos
      keluvchi Dominant Xususiy vektor [0.9486833  0.31622777]
      ekan.
37  #-----------------------------------------------------------
```

Listing 2.19: Xususiy Vektorni hisoblashda darajani iteratsiyalash usuli

2.1.6 Vektorlar fazosi

Vektorlar fazosi chiziqli algebra uchun qurilgan asosiy muhit – platformadir. V vektorlar fazosi - bu (har bir elementi vektordan iborat) shunday to'plamki, unda ikkita operatsiya aniqlanadi: vektorlar yig'indisi va biror skalyarning vektorga ko'paytmasi. Ushbu V vektorlar fazosi quyidagilarni qanoatlantirishi kerak:

(i) V da shunday xarakteristik qo'shiluvchi mavjudki ($\mathbf{0}$ kabi yoziladi), barcha $\mathbf{x} \in V$ uchun $\mathbf{x} + \mathbf{0} = \mathbf{x}$ bo'ladi

(ii) Har bir $\mathbf{x} \in V$ uchun shunday teskari qo'shiluvchi mavjudki ($-\mathbf{x}$ kabi yoziladi), $\mathbf{x} + (-\mathbf{x}) = 0$ bo'ladi

(iii) $\mathbb{R}$ da shunday birlik ko'paytuvchi mavjudki (1 kabi yoziladi), barcha $\mathbf{x} \in V$ uchun $1\mathbf{x} = \mathbf{x}$ bo'ladi

(iv) Kommutativlik: barcha $\mathbf{x}$, $\mathbf{y} \in V$ uchun $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$

(v) Assosiativlik: barcha $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z} \in V$ va α, $\beta \in \mathbb{R}$ uchun $(\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z})$ va $\alpha(\beta\mathbf{x}) = (\alpha\beta)\mathbf{x}$

(vi) Distributivlik: barcha $\mathbf{x}$, $\mathbf{y} \in V$ va α, $\beta \in \mathbb{R}$ uchun $\alpha(\mathbf{x}+\mathbf{y}) = \alpha\mathbf{x}+\alpha\mathbf{y}$ va $(\alpha + \beta)\mathbf{x} = \alpha\mathbf{x} + \beta\mathbf{x}$

Oldinroq qayd etilganidek, biror vektorlar fazosida olingan vektorlar to'plami $\mathbf{v}_1$, $\mathbf{v}_2$, ..., $\mathbf{v}_n \in V$ **chiziqli mustaqil** deyiladi, agar

$$a_1\mathbf{v}_1 + \ldots + a_n\mathbf{v}_n = 0$$

ayniyat bajarilishi uchun $a_1 = \ldots = a_n = 0$ o'rinli bo'lsa.

$\mathbf{v}_1$, $\mathbf{v}_2$, ..., $\mathbf{v}_n \in V$ vektorlarning bog'lanishi (inglizcha "span") bu barcha vektorlar to'plami bo'lib, ularning chiziqli kombinatsiyasi quyidagicha ifodalash mumkin:

$$span\{\mathbf{v}_1, \ldots, \mathbf{v}_n\} = \{\mathbf{v} \in V : \exists a_1, \ldots, a_n \; bunda \; a_1\mathbf{v}_1 + \ldots + a_n\mathbf{v}_n = \mathbf{v}\}.$$

Agar vektorlar to'plami chiziqli mustaqil bo'lsa va uning bog'lanishi yaxlit holda olingan V bo'lsa, bu vektorlar V uchun **bazis** deyiladi. Umuman olganda, har bir mustaqil vektorlar to'plami o'zining bog'lanishi uchun bazisni tashkil qiladi.

Agar vektorlar fazosi chekli sondagi vektorlar bilan bog'langan bo'lsa, demak bu fazo **chekli o'lchamli** fazodir. Aks holda bu cheksiz o'lchamlidir. Chekli o'lchamdagi vektorlar fazosi V uchun berilgan bazisdagi vektorlar soni V **ning o'lchami** deyiladi va $dimV$ orqali belgilanadi.

2.1.7 Chiziqli akslantirish

Chiziqli akslantirish bu T funksiyadir, $T : V \to W$, bunda V va W vektorlar fazosi hisoblanib, quyidagilarni qanoatlantiradi:

(i) $T(\mathbf{x} + \mathbf{y}) = T\mathbf{x} + T\mathbf{y}$, $\mathbf{x}$, $\mathbf{y} \in V$

(ii) $T(\alpha\mathbf{x}) = \alpha T\mathbf{x}$, $\mathbf{x} \in V$, $\alpha \in \mathbb{R}$

Chiziqli akslantirish uchun standart yozuvda (biz ham shu yo'ldan boramiz) keraksiz qavslar tashlab yuboriladi, agar ikkilantirish xavfi bo'lmasa, $T(\mathbf{x})$ o'rniga $T\mathbf{x}$ deb yoziladi. Shuningdek, chiziqli akslantirish tarkibini odatdagi $S \circ T$ o'rniga ST deb belgilanadi.

V ning o'zini o'ziga chiziqli akslantirish **chiziqli operator** deyiladi.

E'tibor bering, chiziqli akslantirishning ta'rifi vektorlar fazosining strukturasini aks ettirishga moslashgan, chunki u vektorlar fazosida ikkita asosiy amalni, yig'indi va skalyar ko'paytmani saqlaydi. Algebraik nuqtai nazardan, chiziqli akslantirish vektorlar fazosining **gomomorfizmi** deb nomlanadi. Teskarilanuvchi gomomorfizm (bu yerda teskarilash ham gomomorfizm) **izomorfizm** deb ataladi. Agar V dan W ga tomon izomorfizm mavjud bo'lsa, unda V va W **izomorf** deyiladi va biz $V \cong W$ deb yozamiz. Bir nechta izomorfik vektorlar fazosi o'zlarining algebraik strukturasi jihatidan "bir xil". Shunisi qiziqki, o'lchamlari bir xil bo'lgan chekli o'lchamdagi vektorlar fazolari har doim izomorfdir; agar V, W haqiqiy vektor fazolari bo'lsa, hamda $dimV = dimW = n$ bo'lsa, bu holda tabiiy izomorfizm bo'ladi:

$$\varphi : V \to W$$
$$a_1\mathbf{v}_1 + \ldots + a_n\mathbf{v}_n \mapsto a_1\mathbf{w}_1 + \ldots + a_n\mathbf{w}_n,$$

bunda $\mathbf{v}_1, \ldots, \mathbf{v}_n$ va $\mathbf{w}_1, \ldots, \mathbf{w}_n$ vektorlar V va W lar uchun bazislar. Ushbu akslantirish yaxshi-aniqlangan, chunki V dagi har bir vektorni $\mathbf{v}_1$, $\mathbf{v}_2$, ..., $\mathbf{v}_n$ ning chiziqli birikmasi sifatida ifodalash mumkin va bu ifoda yagonadir. φ ning izomorfizm ekanligi ko'rinib turipti, shuning uchun aslida $V \cong W$. Xususan, har bir n-o'lchamli vektorlar fazosi $\mathbb{R}^n$ ga izomorfdir.

Chiziqli akslantirish matritsasi

Vektorlar fazosi juda mavhum tushunchadir. Kompyuterda vektorlar va chiziqli akslantirishni ko'rsatish va boshqarish uchun biz **matritsalar** deb nomlanuvchi to'rtburchak massivlardan foydalanamiz.

Aytaylik, V va W chekli o'lchamdagi vektor fazolari mos ravishda $\mathbf{v}_1, \ldots, \mathbf{v}_n$ va $\mathbf{w}_1, \ldots, \mathbf{w}_n$ bazisalar bilan berilgan bo'lsin, va $T : V \to W$ chiziqli akslantirish bo'lsin. Bu holda T ning matritsasi, ya'ni A_{ij}, bunda $i = 1, \ldots, m$ va $j = 1, \ldots, n$, quyidagicha aniqlanadi:

$$T\mathbf{v}_j = A_{1j}\mathbf{w}_1 + \cdots + A_{mj}\mathbf{w}_m.$$

Ya'ni, A matritsaning j-ustuni W uchun tanlangan bazisda $T\mathbf{v}_j$ koordinatalaridan iborat. Aksincha, har bir matritsa $A \in \mathbb{R}^{m\times n}$ chiziqli akslantirish $T : \mathbb{R}^n \to \mathbb{R}^m$ ni paydo qiladi

$$T\mathbf{x} = A\mathbf{x}$$

va $\mathbb{R}^n$ va $\mathbb{R}^m$ standart bazislarga nisbatan akslantirish matritsasi A ekanligi ma'lumdir. Agar $A \in \mathbb{R}^{m\times n}$ bo'lsa, uning transponiri $A^T \in \mathbb{R}^{n\times m}$ har bir (i, j) uchun $(A^T)_{ij} = A_{ji}$ bilan berilgan. Boshqacha qilib aytganda, A

ustunlari A^T satrlariga, A satrlari esa A^T ustunlariga aylanadi. Transponirning bir nechta ajoyib algebraik xususiyatlari borki, ularni ta'rifdan osongina tekshirish mumkin:

(i) $(A^T)^T = A$

(ii) $(A + B)^T = A^T + B^T$

(iii) $(\alpha A)^T = \alpha A^T$

(iv) $(AB)^T = B^T A^T$

2.2 Differensial va Integral hisob

Hisoblashlar, sodda qilib aytganda, bu funksiyalar differensiali va integrallari bilan shug'ullanadigan matematikaning bir sohasidir. Hisoblashllarni yaxshi tushunish bir necha sabablarga ko'ra Sun'iy o'rganish uchun muhimdir:

- Sun'iy o'rganishning turli xil modellari ko'p o'zgaruvchili funksiyalar orqali ifodalanadi.
- Sun'iy o'rganish modelini yaratish uchun odatda ma'lumotlar va model parametrlari asosida model uchun "cost function" hisoblanadi va "cost function"ni optimallashtirish orqali berilgan ma'lumotlarni eng yaxshi tushuntiruvchi model parametrlari olinadi.

2.2.1 Differensial hisob

Funksiyaning differensiali deganda funksiya bog'liq bo'lgan biror kattalik o'zgarishiga nisbatan funksiyaning o'zgarish tezligi tushuniladi.

Aytaylik, jism bir o'lchovli fazoda, ya'ni to'g'ri chiziq bo'ylab harakatlanmoqda va uning biror t vaqt davomida bosib o'tgan masofasi funksiya orqali ifodalanadi, $f(t) = 5t^2$.

Bu jismning t vaqt momentidagi oniy tezligi $f(t)$ funksiyadan olingan vaqt bo'yicha hosila yordamida topiladi.

Funksiyaning hosilasi $\frac{df(t)}{dt}$ kabi aniqlanadi va odatda quyidagi formula bilan ifodalanadi:

$$\frac{df}{dt} = \lim_{h \to 0} \frac{f(t+h) - f(t)}{h}$$

yoki

$$\frac{df}{dt} = \lim_{h \to 0} \frac{f(t+h) - f(t-h)}{2h}.$$

Ko'p o'zgaruvchili funksiyaga kelsak, boshqa o'zgaruvchilarga tegmay turib faqat bir o'zgaruvchiga nisbatan funksiyaning hosilasi "xususiy hosila" deb ataladi va xususiy hosilalar vektori "funksiyaning gradienti" deyiladi.

Aytaylik uyning narxi z ikki o'zgaruvchiga bog'liq: uyning kvadrat-metr maydoni x va xonalari soni y, $z = f(x, y)$.

Shunda, z ning x bo'yicha xususiy hosilasi

$$\frac{\partial z}{\partial x} = \lim_{h \to h} \frac{f(x+h, y) - f(x, y)}{h},$$

va y bo'yicha xususiy hosilasi esa

$$\frac{\partial z}{\partial y} = \lim_{h \to h} \frac{f(x, y+h) - f(x, y)}{h}$$

orqali ifodalanadi.

2.2.2 Funksiyaning gradienti

Sun'iy o'ranish konsepsiyasi uchun differensial hisob bo'limida yagona muhim tushuncha bu **gradient**dir. Gradient ko'p o'zgaruvchili skalyar funksiyalarga oid differensialni umumlashtiradi. Biror ko'p-o'zgaruvchili funksiya $f(x_1, x_2, \ldots, x_n) : \mathbb{R}^d \to \mathbb{R}$ ni ixchamlashtirib $f(\mathbf{x})$ ko'rinishda yozib olsak bo'ladi, bu yerda

$$\mathbf{x} = \begin{pmatrix} x_1 & x_2 & \ldots & x_n \end{pmatrix}^T \in \mathbb{R}^{n \times 1}.$$

Ushbu funksiyaning $\mathbf{x}$ ga nisbatan gradient-vektori, uni $\vec{\nabla} f$ orqali belgilaymiz, quyidagicha beriladi:

$$\vec{\nabla} f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \ldots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}, \; ya'ni \; (\vec{\nabla} f)_i = \frac{\partial f}{\partial x_i}$$

Misol uchun, ikki o'zgaruvchili $z = f(x, y)$ funksiya uchun, uning xususiy hosilalaridan tuzilgan vektor $\begin{pmatrix} \frac{\partial z}{\partial x} & \frac{\partial z}{\partial y} \end{pmatrix}^T$ bu funksiyaning gradienti deyiladi va $\vec{\nabla} z$ kabi yoziladi.

Konkret ko'rinishdagi boshqa bir misol, uch-o'zgaruvchili $f(x, y, z) = x + y^2 + z^3$ funksiyaning gradienti quyidagi ifodaga teng:

$$\vec{\nabla} f = \begin{pmatrix} 1 \\ 2y \\ 3z^2 \end{pmatrix}.$$

Gradient quyidagi o'ta muhim xususiyatga ega: $\vec{\nabla} f(\mathbf{x})$ gradient-vektor $\mathbf{x}$ vektordan boshlab eng tik ko'tarilish yo'nalishi bo'ylab yo'naladi. Xuddi shunday, $-\vec{\nabla} f(\mathbf{x})$ vektor $\mathbf{x}$ vektordan boshlab eng tez tushish yo'nalishi bo'yicha yo'naladi. Sun'iy o'rganish algoritmlarida, model parametrlari bo'yicha qiymat funksiyasini (inglizcha "cost function") maksimal yoki minimallashtirish amaliyotlarida gradient va xususiy hosilalar muhimdir. Chunki funksiyaning maximum va minimumlarida uning gradient-vektori nolga teng.

2.2.3 Funksiyaning Yakobi matritsasi

$f(x_1, x_2, \ldots, x_n) : \mathbb{R}^n \to \mathbb{R}^m$ ning Yakobiani birinchi tartibli xususiy hosilalarining matritsasi hisoblanadi:

$$\mathbf{J}_f = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}, \; ya'ni \; (\mathbf{J}_f)_{ij} = \frac{\partial f_i}{\partial x_j}.$$

Qayd etish lozimki, $m = 1$ bo'lgan hol uchun $\vec{\nabla} f = \mathbf{J}_f^T$ o'rinlidir.

2.2.4 Funksiyaning aralash xususiy hosilalari

Ko'p-o'zgaruvchili funksiyaning aralash xususiy hosilalariga ega bo'lishimiz mumkin. Masalan, $z = f(x, y)$ funksiyasi uchun

$$\frac{\partial}{\partial y}(\frac{\partial z}{\partial x}) = \frac{\partial^2 z}{\partial y \partial x}.$$

Topilgan ifoda z ning avval x ga nisbatan va so'ngra y ga nisbatan xususiy hosilasidir. Shuning barobarida, quyidagi ifodani ham yozishimiz mumkin:

$$\frac{\partial}{\partial x}(\frac{\partial z}{\partial y}) = \frac{\partial^2 z}{\partial x \partial y}.$$

Agar z funksiyaning ikkinchi tartibli hosilalari uzluksiz bo'lsa, aralash hosilani topishda qaysi o'zgaruvchi bo'yicha xususiy hosila olish tartibining ahamiyati yo'q, ya'ni

$$\frac{\partial^2 z}{\partial x \partial y} = \frac{\partial^2 z}{\partial y \partial x}.$$

2.2.5 Funksiyaning Hesse matritsasi

Ko'p-o'zgaruvchili funksiyaning Hesse matritsasi deganda uning ikkinchi tartibli xususiy hosilalaridan tuzilgan matritsa tushuniladi. $f(x, y, z)$ funksiyasi

uchun Hesse matritsasi $H_{ij}(f) = \frac{\partial^2 f}{\partial x_i \partial x_j}$ quyidagicha aniqlanadi:

$$H(f) = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} \end{pmatrix}.$$

Hesse matritsasi, biz tez-tez Sun'iy o'rganish sohasida duch keladigan optimallashtirish masalalarida ishlatiladi. Masalan, model parametrlari to'plamini topish uchun qiymat funksiyani minimallashtirishda, Hessian parametrlarining keyingi to'plamini yaxshiroq baholash uchun foydalanadi, xususan qiymat funksiya nochiziqli bo'lsa. Nyuton usuli, Broyden-Fletcher-Goldfarb-Shanno (BFGS) kabi nochiziqli optimallashtirish usullarida Hessian-dan qiymat funksiyani minimallashtirish uchun foydalaniladi.

2.2.6 Funksiyaning maksimum va minimumlari

Funksiyalarning maksimum va minimumlarini baholash Sun'iy o'rganishda juda katta amaliy ahamiyatga ega. Sun'iy o'rganish modellarini qurish, nazorat qilinadigan va nazorat qilinmaydigan o'rganishda "cost function"ni minimallashtirish yoki ehtimollik funktsiyalarini, entropiyani va boshqalarni maksimal darajada oshirishga tayanadi.

Bir-o'zgaruvchili funksiyaning maksimum va minimumlik qoidalari

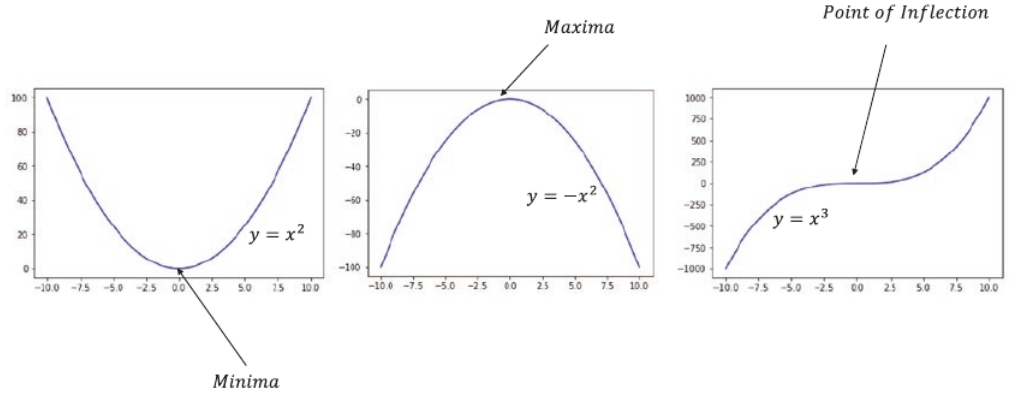
- Maksimum va minimum nuqtalarda $f(x)$ funksiyaning x bo'yicha hosilasi nolga teng
- $f(x)$ ning ikkinchi tartibli hosilasi $\frac{d^2 f(x)}{dx^2}$ birinchi tartibli hosila nol bo'lgan joyda tekshirilishi kerak. Agar ikkinchi tartibli hosila noldan kichik bo'lsa, u $f(x)$ ning maksimum nuqtasidir, noldan katta bo'lsa, bu minimum nuqtasidir. Agar ikkinchi tartibli hosila ham nolga aylansa, u holda bu nuqtaga egilish nuqtasi ("point of inflection", "tochka peregiba") deyiladi.

$y = f(x) = x^2$ ko'rinishdagi funksiyani olaylik. Agar bu funksiyaning x ga nisbatan hosilasini olsak va nolga tenglasak, $\frac{dy}{dx} = 2x = 0$, bu bizga $x = 0$ ni beradi. Shuningdek, ikkinchi tartibli hosilasi $\frac{d^2 y}{dx^2} = 2$. Demak, x ning barcha qiymatlari uchun, shu jumladan $x = 0$ uchun ham, ikkinchi tartibli hosila noldan katta va shuning uchun $x = 0$ nuqta $f(x)$ funksiyaning minimal nuqtasidir.

Endi $y = g(x) = x^3$ uchun xuddi shu mashqni bajarishga harakat qilaylik.

$\frac{dy}{dx} = 3x^2 = 0$ tenglamadan $x = 0$ ni topamiz. Ikkinchi tartibli hosila $\frac{d^2y}{dx^2} = 6x$ ga teng va uning $x = 0$ nuqtadagi qiymati 0. Demak, $x = 0$ nuqta $g(x)$ funksiyasi uchun minimum ham, maksimum ham emas. Ikkinchi tartibli hosila nolga teng bo'lgan nuqtaga egilish nuqtasi deyiladi va ushbu nuqtada funksiyaning egrilik ishorasi o'zgaradi.

Bir-o'zgaruvhili funksiyaning hosilasi nolga teng bo'lgan yoki ko'p-o'zgaruvchili funksiyaning gradient vektori nol vektorga teng bo'lgan nuqtalarga statsionar nuqtalar deyiladi. Ular maksimum/minimum nuqtalari bo'lishi yoki bo'lmasligi ham mumkin. 2.9-rasmda turli xil statsionar nuqtalar -



Rasm 2.9: Turli xil statsionar nuqtalar - maksimum, minimum va egilish nuqtasi.

maksimum, minimum va egilish nuqtalari tasvirlangan.

Ko'p-o'zgaruvchili funksiya uchun maksimum va minimumlar biroz murakkabroq. Keling, misolni ko'rib chiqamiz, shundan keyin biz qoidalarni aniqlaymiz. Soddalik uchun ikki o'zgaruvchili funksiyani ko'raylik:

$$f(x, y) = x^2y^3 + 3y + x + 5.$$

Bu funksiyaning statsionar nuqtalarini aniqlash uchun uning gradient vektorini nolga tenglaymiz:

$$\begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{pmatrix}^T = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$\frac{\partial f}{\partial x}$ va $\frac{\partial f}{\partial y}$ ni nolga tenglab, quyidagilarni olamiz:

$$\frac{\partial f}{\partial x} = 2xy^3 + 1 = 0, \quad \frac{\partial f}{\partial y} = 3x^2y^2 + 3 = 0.$$

Shu bilan birga, Hesse matritsa elementlarini ham hisoblaymiz:

$$\frac{\partial^2 f}{\partial x^2} = f_{xx} = 2y^3,$$
$$\frac{\partial^2 f}{\partial y^2} = f_{yy} = 6x^2 y,$$
$$\frac{\partial^2 f}{\partial x \partial y} = f_{xy} = 6xy^2,$$
$$\frac{\partial^2 f}{\partial y \partial x} = f_{yx} = 6xy^2.$$

Funksiya gradinetini nol deb olib ($x = a, y = b$), quyidagi hollarni tahlil qilamiz:

- Agar ($x = a, y = b$) nuqtada $f_{xx}f_{yy} - (f_{xy})^2 < 0$ bo'lsa, u holda ($x = a, y = b$) *egar nuqta* deb ataladi.

- Agar ($x = a, y = b$) nuqtada $f_{xx}f_{yy} - (f_{xy})^2 > 0$ bo'lsa, u holda ($x = a, y = b$) *ekstremum nuqta* deb ataladi, ya'ni maksimum yoki minimumlar mavjud bo'ladi:

  a) agar ($x = a, y = b$) nuqtada $f_{xx} < 0$ va $f_{yy} < 0$ bo'lsa, u holda $f(x, y)$ funksiya ($x = a, y = b$) nuqtada maksimumga ega bo'ladi;

  b) agar ($x = a, y = b$) nuqtada $f_{xx} > 0$ va $f_{yy} > 0$ bo'lsa, u holda $f(x, y)$ funksiya ($x = a, y = b$) nuqtada minimumga ega bo'ladi.

- Agar $f_{xx}f_{yy} - (f_{xy})^2 = 0$ bo'lsa, u holda statsionar nuqtani to'g'ri baholash uchun yuqoriroq usullarga murojaat qilinadi.

Quyida n o'zgaruvchili funksiya uchun funksiyaning maksimal, minimal va egar nuqtalarini aniqlash bo'yicha ko'rsatmalar keltirilgan:

- Gradientni hisoblash va uni nol vektorga tenglash orqali statsionar nuqtalar o'rni aniqlanadi.

- $x_0 \in \mathbb{R}^{n\times 1}$ statsionar nuqta uchun, agar x_0 da funksiyasining Hesse matritsasi ham musbat, ham manfiy Xususiy qiymatiga ega bo'lsa, x_0 - egar nuqtasidir. Agar Hesse matritsasining barcha Xususiy qiymatlari musbat bo'lsa, statsionarlik nuqtasi *lokal minimum* bo'ladi, va aksincha Xususiy qiymatlarining barchasi manfiy bo'lsa, statsionarlik nuqtasi *lokal maksimum* bo'ladi.

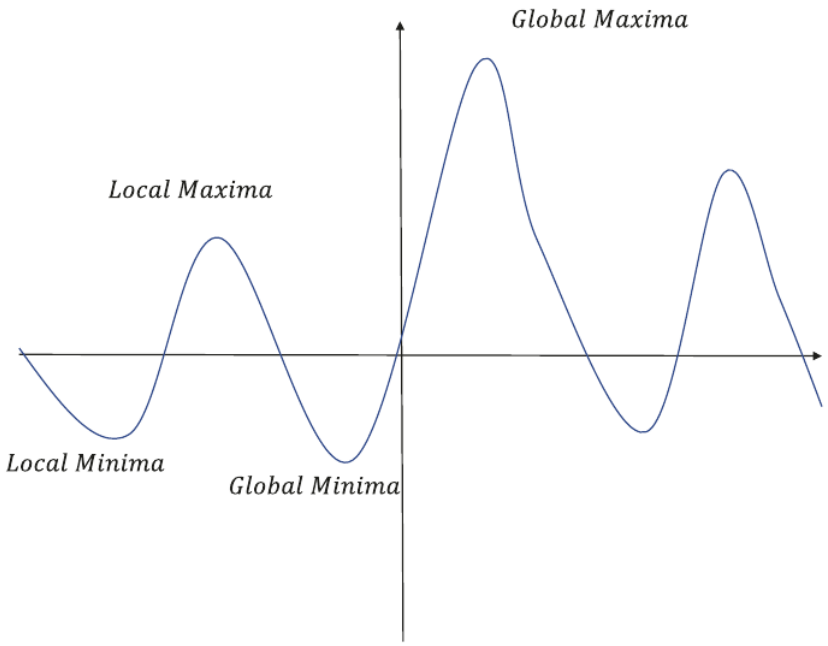
2.2.7 Lokal va Global minimumlar

Gradiyenti nolga teng bo'lgan nuqtalarda funksiya bir nechta minimumlarga ega bo'lishi mumkin, ularning har biri *lokal minimum* nuqtasi deb ataladi. Funksiya o'zining minimal qiymatiga ega bo'lgan lokal minimum *global minimum* deb nomlanadi. Huddi shu narsa maximum nuqtalar uchun ham amal qiladi. Funksiyaning maksimum va minimumlari optimallashtirish usullari bilan olinadi. Oshkor ko'rinishdagi yechimlar har doim ham mavjud emasligi yoki hisoblash qiyin bo'lganligi sababli, minimum va maximumlar ko'pincha iterativ yaqinlashishlar orqali olinadi, masalan, gradient-tushish, gradient-ko'tarilish va h. k. Minimum va maximum nuqtalarni olishning iterativ usulida, optimallashtirish usuli lokal minimum/maximum nuqtalar bilan cheklanishi va global minimum yoki maximumlargacha yetib borolmasligi mumkin. Iterativ usullarda algoritm yanada optimal nuqtaga erishish uchun funksiyaning gradientidan foydalanadi. Ushbu usulda nuqtalar to'plamini kesib o'tayotganda, nol gradientli nuqta topilsa, algoritm kerakli minimum yoki maksimum nuqtani topgandek bo'ladi va qidirishdan to'xtaydi. Funksiyaning global minimum yoki maksimum nuqtalari uchun bu yaxshi ishlaydi. Yana shunday bo'lishi ham mumkinki, unda optimallashtirish egar nuqtasiga yopishib olishi ham mumkin. Bunday barcha holatlarda biror suboptimal modelga murojaat qilinadi.

2.10-rasmda global va lokal minima, shuningdek funksiyaning global va lokal maksimallari ko'rsatilgan.

2.2.8 Yarim-musbat va Musbat aniqlangan matritsalar

Agar nolga teng bo'lmagan biror $\mathbf{x} \in \mathbb{R}^{n\times 1}$ vektor uchun $\mathbf{x}^T A\mathbf{x} \geq 0$ bo'lsa, u holda $A \in \mathbb{R}^{n\times n}$ kvadrat matritsa Yarim-Musbat aniqlangan deyiladi. Agar $\mathbf{x}^T A\mathbf{x} > 0$ bo'lganda esa, A matritsa Musbat aniqlangan deb ataladi. Yarim-Musbat aniqlangan matritsaning barcha Xususiy qiymatlari manfiy bo'lmasligi kerak, Musbat aniqlangan matritsaning Xususiy qiymatlari esa musbat bo'lishi kerak. Masalan, agar biz A matritsani 2×2 birlik matritsasi deb olsak, ya'ni $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ - bu Musbat aniqlangandir, chunki uning ikkala Xususiy qiymati ham, ya'ni $1, 1$ - musbat kattaliklardir. Bundan tashqari, agar biz $\mathbf{x}^T A\mathbf{x}$ ni hisoblasak, unda $\mathbf{x} = (x_1 x_2)^T$, biz $\mathbf{x}^T A\mathbf{x} = x_1^2 + x_2^2$ ni olamiz, bu har doim nol bo'lmagan vektor $\mathbf{x}$ uchun noldan katta bo'ladi, bu A Musbat aniqlangan matritsa ekanligini tasdiqlaydi.



Rasm 2.10: Lokal va Global minimum/maximumlar.

2.2.9 Integral hisob

Hosilaga "qarama-qarshi" amal integral deyiladi. Uning eng keng tarqalgan qo'llanilishi, har holda biz ishlatadigan, egri chiziq ostidagi yuzani hisoblashdan iborat. Hech qachon biz integrallarni qo'l bilan hisoblashimiz shart emas, garchi ularning xossalari bilan yaxshi tanish bo'lsak ham.

Odatda, integral yordamida "yuzani hisoblash" amalida ikkita chegara mavjud, a (pastki chegara) va b (yuqori chegara). Berilgan a va b oraliqdagi f funksiya orqali chizilgan egri chiziq ostidagi yuzani ifodalash $\int_a^b dx f(x)$ integral bilan ifodalanadi. Mazkur ko'rinishdagi integrallarni oddiygina yig'indining uzluksiz analogi deb hisoblash kerak. Ya'ni, ushbu integralni $\sum_a^b f(x)$ summa kabi tushunish mumkin.

Integralni yig'indi sifatida talqin qilish quyidagi fikriy tajribadan kelib chiqadi. Faraz qilaylik, biz $[a, b]$ oraliqni bo'laklar soni R va kengligi $(b-a)/R$ bo'lgan qismlarga ajratamiz. So'ngra, egri chiziq ostidagi yuzani har bir nuqtada $f(x)$ ning qiymatini topib (to'rtburchaklar balandligini olish uchun), uni kenglik $(b-a)/R$ ga ko'paytirib, ushbu birlik yuzalar yig'indisiga yaqinlashtiramiz. Ushbu elementar yuzachalar yig'indisi qaralayotgan yuzaga yaqinlashadi. $R \to \infty$ bo'lganda, yana va yana yaqinlashadi. Biroq, $R \to \infty$ da har bir to'rtburchakning kengligi 0 ga intiladi. Biz bu kenglikni dx deb nomlaymiz va shu sababli integral ifodasi deyarli "to'rtburchaklar yig'indisi" ifo-

dasiga o'xshaydi (dx kenglikning $f(x)$ balandliklarga ko'paytmalari yig'indisi).

Shunday xosmas integrallar ham borki, unda quyi va yuqori chegaralar cheksizlikda yotadi, masalan $\int_{-\infty}^{\infty} dx f(x)$. Yig'indini cheksiz sohada amalga oshirish mantiqsizdek tuyulishi mumkin bo'lsa-da, aslida ushbu integralning qiymati chekli bo'lgan juda ko'p $f(x)$ funksiyalar mavjud. Masalan, $1/x^2$ ning yarim chegaralangan integrali chekli qiymatdir:

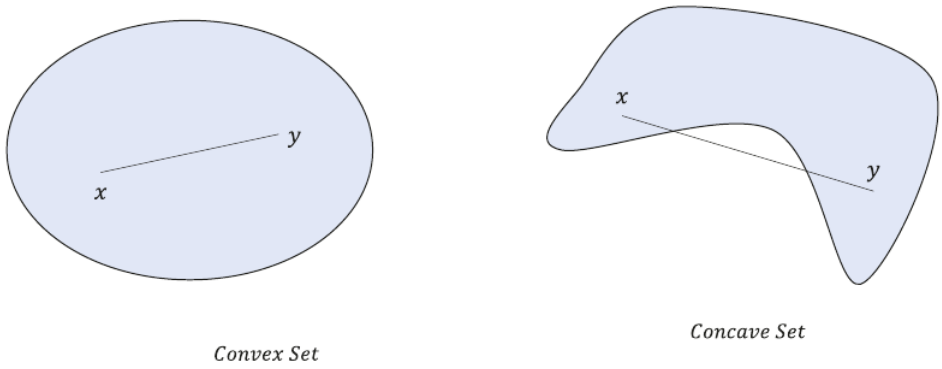
$$\int_1^{\infty} dx \frac{1}{x^2} = \lim_{b\to\infty} \left[-\frac{1}{b} - (-\frac{1}{1})\right] = 0 + 1 = 1.$$

Shunga o'xshash hisoblash quyidagilarni ko'rsatishi mumkin (Gauss integrali deb nomlanadi):

$$\int_{-\infty}^{\infty} dx e^{-x^2} = \sqrt{\pi}.$$

2.2.10 Qavariq to'plam

Biror to'plamga tegishli bo'lgan ikkita x va y nuqta berilgan bo'lsin. Mana shu x va y nuqtalarni tutashtiruvchi to'g'ri chiziqning barcha nuqtalari ham shu to'plamga tegishli bo'lsa, bu nuqtalar to'plamiga qavariq deyiladi. 2.11-rasmda qavariq va qavariq-bo'lmagan, ya'ni botiq to'plam ko'rsatilgan.



Rasm 2.11: Qavariq va botiq to'plamlar.

2.2.11 Qavariq funksiya

D qavariq to'plamda aniqlangan $f(x)$ funktsiya qavariq deyiladi, bu yerda $x \in \mathbb{R}^{n\times 1}$ va D soha, agar funksiyaning har qanday ikkita nuqtasini birlashtiruvchi to'g'ri chiziq uning grafigida yoki grafikdan yuqorida joylashgan bo'lsa. Matematik jihatdan buni quyidagicha ifodalash mumkin:

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y) \quad \forall x, y \in D, \ \forall t \in [0,1].$$

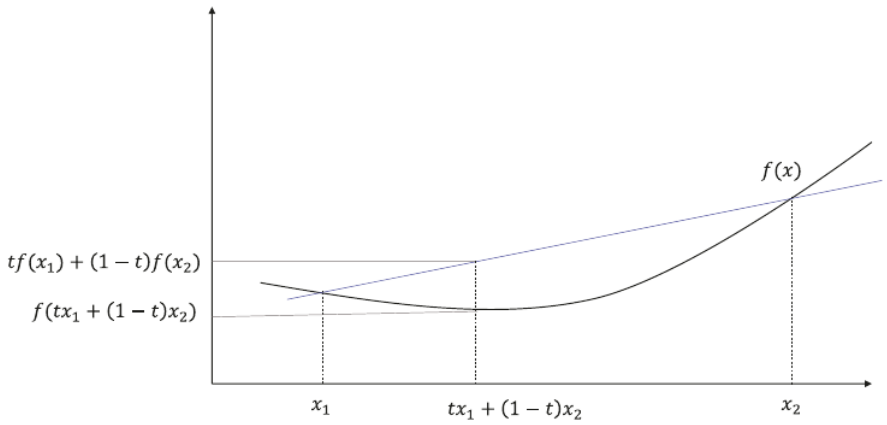
Ikki marta uzluksiz differensiallanuvchi qavariq funksiya uchun, funksiyaning D sohadagi har bir nuqtasida topilgan Hesse matritsasi Yarim-Musbat aniqlangan bo'lishi kerak; ya'ni har qanday $\mathbf{x} \in \mathbb{R}^{n\times 1}$ vektor uchun

$$\mathbf{x}^T H\mathbf{x} \geq 0$$

o'rinli bo'lishi kerak.

Qavariq funksiyaning lokal minimumi uning global minimumidir. Shuni unutmaslik krakki, bir nechta global minimumlar bo'lishi mumkin, ammo qavariq funksiya uchun global minimumlardagi qiymatlar bir xil bo'ladi.

2.12-rasmda $f(x)$ qavariq funksiya ko'rsatilgan. Ko'rib turganimizdek, $f(x)$ yuqorida bayon qilingan qavariq funksiyalarning xususiyatlariga aniq mos keladi.



Rasm 2.12: Qavariq funksiyaning ko'rinishi.

2.2.12 Qavariqmas funksiya

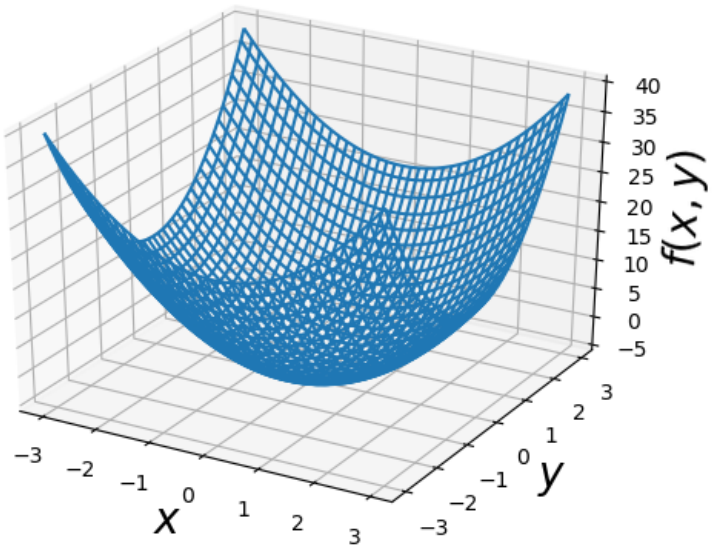
Qavariq bo'lmagan funksiyada ko'plab lokal minimumlar bo'lishi mumkin, ammo ularning hammasi ham global minimum bo'lavermaydi.

Har qanday Sun'iy o'rganish modelini yaratish jarayonida, biz qiymat funksiyani minimallashtirish orqali model parametrlarini o'rganishga harakat qilsak, biz qiymat funksiyaning qavariq bo'lishini afzal ko'ramiz, chunki to'g'ri optimallashtirish texnikasi bilan biz global minimumga erishamiz. Optimallashtirish texnikasi qavariq bo'lmagan qiymat funksiyaning lokal minimumi yoki egar nuqtasida qotib qolish ehtimoli katta va shuning uchun u global minimumga erisha olmaydi.

2.2.13 Ko'p-o'zgaruvchili qavariq va qavariqmas funksiyalarga misollar

Modomiki, Sun'iy o'rganish sohasida ko'p o'zgaruvchili funksiyalar bilan ishlar ekanmiz, ikki o'zgaruvchili qavariq va qavariq bo'lmagan funksiyalarni o'rganish mantiqan to'g'ri keladi.

$f(x, y) = 2x^2 + 3y^2 - 5$ ko'rinishdagi funksiya qavariq bo'lib, uning minimumi $x = 0$, $y = 0$ nuqtada va qiymati $f(x = 0, y = 0) = -5$. Ushbu funksiya haqida yaqqol tasavvur olish uchun 2.13-rasmda uning grafigini keltirdik.



Rasm 2.13: Ikki o'zgaruvchili qavariq funksiya $f(x, y) = 2x^2 + 3y^2 - 5$ ning grafigi.

```
# Masalaning qo'yilishi: Python-da ikki o'zgaruvchili qavariq
    funksiya grafigini chizing
# Yechim:
#

# Kutubxonani yuklash
import numpy as np
import matplotlib.pyplot as plt
```

```
from pylab import *
import math
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.mlab import griddata

fig = plt.figure()
ax = fig.gca(projection='3d')

x = np.arange(-3, 3, 0.05)
y = np.arange(-3, 3, 0.05)

X, Y = np.meshgrid(x, y)

# Ikki o'zgaruvchili funksiya
def convex_function(x, y):
  return 2. * x**2 + 3. * y**2 - 5.

ax.plot_wireframe(X, Y, convex_function(X, Y))
ax.set_xlabel('$x$',size=20)
ax.set_ylabel('$y$',size=20)
ax.set_zlabel('$f(x, y)$',size=20)
plt.show()

#------------------------------------------------------------
# Izoh:
#------------------------------------------------------------
```

Listing 2.20: Python-da ikki o'zgaruvchili funksiya grafigi

Endi $f(x,y) = \log(x/y),\ \forall x > 0, y > 0$ funksiyasini ko'rib chiqaylik.

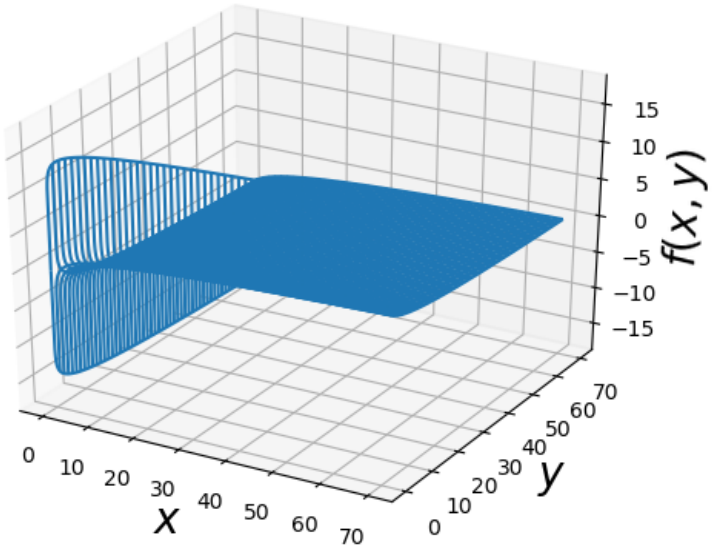
Bu funksiya qavariq bo'lmagan funksiyasdir va buni tekshirishning eng oson usuli funksiyaning Hesse matritsasini tekshirishdir:

$$Hesse\ matritsa\ H = \begin{pmatrix} -\frac{1}{x^2} & 0 \\ 0 & \frac{1}{y^2} \end{pmatrix}.$$

Hesse matritsasining Xususiy qiymatlari $-\frac{1}{x^2}$ va $\frac{1}{y^2}$; x ning haqiqiy qiymatlari uchun $-\frac{1}{x^2}$ har doim manfiy bo'ladi. Demak, Hesse matritsasi Yarim-musbat aniqlangan emas va $f(x,y)$ funksiyaning o'zi ham qavariq emas.

2.14-rasmda ko'rishimiz mumkinki $\log(x/y)$ funksiya qavariq bo'lmagan funksiyadir.

```
# Masalaning qo'yilishi: Python-da ikki o'zgaruvchili qavariq
    bo'lmagan funksiya grafigini chizing
# Yechim:
#

# Kutubxonani yuklash
```



Rasm 2.14: Qavariq bo'lmagan funksiya $f(x, y) = \log(x/y)$ ning chizmasi.

```
import numpy as np
import matplotlib.pyplot as plt
from pylab import *
import math
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.mlab import griddata

fig = plt.figure()
ax = fig.gca(projection='3d')

x = np.arange(1.e-6, 70, 0.05)
y = np.arange(1.e-6, 70, 0.05)

X, Y = np.meshgrid(x, y)

# Ikki o'zgaruvchili funksiya
def concave_function(x, y):
  return log( x / y )

ax.plot_wireframe(X, Y, concave_function(X, Y))
ax.set_xlabel('$x$',size=20)
ax.set_ylabel('$y$',size=20)
```

```
ax.set_zlabel('$f(x, y)$',size=20)
plt.show()
```

Listing 2.21: Python-da ikki o'zgaruvchili funksiya grafigini chizish dasturi

Eng kichik kvadratlar orqali chiziqli regressiya yoki log-loss qiymatlar funksiyasi (ikkilik cross entropiya) orqali logistik regressiya bularning hammasi qavariq optimallashtirish muammolari hisoblanadi va shuning uchun optimallashtirish orqali o'rganilgan model parametrlari global minimal yechimi hisoblanadi. Xuddi shunday, SVM-da biz optimallashtiradigan qiymat funksiyasi ham qavariq hisoblanadi.

Har qanday modelda yashirin qatlamlar yoki yashirin omillar mavjud bo'lganda, qiymat funksiyasi qavariq bo'lmagan xarakterga ega bo'ladi. Yashirin qatlamlari mavjud bo'lgan neyron to'ri, regressiya yoki tasniflash masalasini yechishimizdan qat'i nazar, qavariq bo'lmagan qiymat yoki xatolik sirtini beradi.

Qavariq bo'lmagan masalani hal etishda parametrni boshlash juda muhimdir. Boshlang'ich parametrlar global minimaga yoki ba'zi maqbul lokal minimumlarga qanchalik yaqin bo'lsa, shuncha yaxshi bo'ladi.

2.2.14 Teylor qatori

Har qanday funksiya ma'lum bir nuqta atrofida funksiyaning qiymatini va uning hosilalarini hisobga olgan holda cheksiz qator shaklida ifodalanishi mumkin. Funksiyaning bunday yoyilishi Teylor qatori deb nomlanadi. Bir o'zgaruvchili funksiyaning x nuqta atrofida Teylor qatoriga yoyishni quyidagicha ifodalash mumkin:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \ldots + \frac{h^n}{n!}f^{(n)}(x) + \ldots,$$

bu yerda $f^{(n)}(x)$ – $f(x)$ funksiyadan olingan n-tartibli hosila va $n!$ ("n faktorial") n sonining faktorialini bildiradi. h kattalik x o'lchovi bilan bir xil o'lchovga ega va har ikkala kattaliklar, ya'ni h va x, skalyar hisoblanadi.

- Agar $f(x)$ doimiy funksiya bo'lsa, u holda barcha hosilalar nolga teng, $f(x+h)$ va $f(x)$ teng bo'ladi, ya'ni $f(x+h) = f(x)$.
- Agar funksiya x nuqtaning atrofida chiziqli bo'lsa, u holda chiziqlilik sohasidagi har qanday nuqta $(x+h)$ uchun $f(x+h) = f(x) + hf'(x)$ o'rinli bo'ladi.
- Agar funktsiya x ning atrofida kvadratik bo'lsa, u holda kvadratiklik sohasidagi har qanday nuqta $(x+h)$ uchun $f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x)$ o'rinli bo'ladi.

- Teylor qatori gradient-tushish va Nyuton optimallashtirish usullari, shuningdek integrallash va differentsiatsiyaning sonli usullari kabi iterativ usullarda juda muhim ahamiyat kasb etadi.

Ko'p o'zgaruvchili funksiyalar uchun $\mathbf{x} \in \mathbb{R}^{n\times 1}$ nuqta atrofida Teylor qatori quyidagicha ifodalanishi mumkin:

$$f(\mathbf{x}+\Delta\mathbf{x}) = f(\mathbf{x}) + \Delta\mathbf{x}^T\nabla f(\mathbf{x})$$
$$+\frac{1}{2}\Delta\mathbf{x}^T\nabla^2 f(\mathbf{x})\Delta\mathbf{x} + \textit{yuqori tartibli hadlar},$$

bunda $\nabla f(\mathbf{x})$ gradient-vektor va $\nabla^2 f(\mathbf{x})$ esa $f(\mathbf{x})$ funksiya uchun Hesse matritsasi.

Umuman olganda, amaliy maqsadlar uchun biz Sun'iy o'rganish dasturlarida ikkinchi darajali Teylor qatorlaridan chetga chiqmaymiz, chunki ularni sonli usulda hisoblash qiyin. Hesse matritsasining ikkinchi darajali kengayishi uchun hisoblash ham qimmatga tushadi va shuning uchun ikkinchi darajali optimallashtirishning bir necha usuli to'g'ridan-to'g'ri baho berishning o'rniga taxminiy Hesse matritsa gradientni hisoblashga tayanadi. Qayd etish kerakki, uchinchi darajali hosila $\nabla^3 f(\mathbf{x})$ uchinchi rangli tenzor bo'ladi.

2.3 Ehtimollar Nazariyasi

Ehtimollar Nazariyasiga kirishishdan avval, stoxastik tajriba va hodisalar fazosi tushunchalarini eslatib o'tamiz.

Tadqiqot laboratoriyasida yoki boshqa usulda, deyarli har xil sharoitlarda takroriy tajriba o'tkazish odatiy amaliyotdir. Masalan, tibbiy tadqiqotchini yangi turdagi dorining ta'siri qiziqtirishi mumkin yoki agronom ma'lum bir o'simlik hosiliga kimyoviy o'g'itning ta'sirini o'rganishni xohlashi mumkin. Ushbu qiziqishlar haqida ma'lumot olishning yagona usuli - tajriba o'tkazish. Ba'zida eksperimentlar o'tkazishga ehtiyoj qolmasligi mumkin, chunki tajribalar tabiat tomonidan o'tkaziladi va biz shunchaki ma'lumotlarni yig'ishimiz kerak.

Har bir tajriba natijaga olib keladi. Aytaylik, eksperimentlar natijasini mutlaq aniqlik bilan oldindan aytib bo'lmaydi. Ammo, tajriba o'tkazishdan oldin, barcha mumkin bo'lgan natijalar to'plamini bilamiz deylik. Agar bunday tajribalarni deyarli bir xil sharoitlarda takrorlash mumkin bo'lsa, unda tajriba **stoxastik eksperiment** deb ataladi va barcha mumkin bo'lgan natijalar to'plamiga **elementar hodisalar fazosi** (inglizcha "sample space") deyiladi. Elementar hodisalar fazosi Ω orqali belgilanadi.

Shuni yodda tutish kerakki, elementar hodisalar fazosi - bu bizni qiziqtirgan natijalar to'plami. O'yin soqqasi tashlanganda bir nechta natijalarga olib kelishi mumkin. Ulardan biri o'yin soqqasi yerga tushganda uning ustida joylashishi kutiladigan natijalar to'plamidir. Boshqasi esa o'yin soqqasining yerga urilgandagi tezligi bo'lishi mumkin. Agar bizni o'yin soqqasining yuqori qismidagi raqam qiziqtiradigan bo'lsa, unda elementar hodisalar fazosi $\Omega = \{1, 2, 3, 4, 5, 6\}$, ya'ni o'yin soqqasi ustida turgan raqamlar to'plami tushuniladi.

Keling, o'yin soqqasini tashlash tajribasini davom ettiraylik va natijada u yerga tushib qolgan yuzani belgilab qo'yamiz. Aytaylik, biz tajribani n marta takrorladik, va 1 yuzali qism m marta takrorlandi. Keyin, eksperimentdan shuni aytishimiz mumkinki, tajribalar soni orttirib borilganda, o'yin soqqasida 1 tushish hodisasining ehtimolligi shu hodisa sodir bo'lish umumiy sonining o'tkazilgan tajribalar umumiy soniga nisbatiga intiladi, ya'ni $P(x = 1) = m/n$, bu yerda x kubikning yuzidagi raqamni bildiradi.

Keling, masalani aniqroq qo'yaylik: o'yin soqqasi tashlanish tajribasida 1 sonining tushish ehtimoli qanday?

Agar o'yin soqqasining simmetrik (ya'ni, uning 6 ta yog'idan har birining tushish imkoniyati teng) ekanligini hisobga olsak, 600 marta tashlangan kubikdan 100 tasida 1 raqami (katta ehtimollik bilan) tushadi, deya olamiz. Bu bizga 1/6 kabi ehtimollikni beradi.

Endi, 1000 marta tashlangan o'yin soqqasining quyidagi zaylda berilgan statistika ma'lumotlariga nazar tashlaylik:

$$1 \to 200\ marta$$

$$2 \to 100\ marta$$

$$3 \to 100\ marta$$

$$4 \to 100\ marta$$

$$5 \to 300\ marta$$

$$6 \to 200\ marta$$

Bunday holda, soqqaning 1 tushish ehtimolligini $P(x = 1) = 200/1000 = 0.2$ kabi hisoblaymiz. Bu tajribadan ushbu taxmin kelib chiqadi: o'yin soqqasi nosimmetrik bo'lgan yoinki tajriba sharoiti bir xil bo'lmagan.

2.3.1 Hodisalar yig'indisi va ko'paytmasi. Shartli ehtimollik

- $P(A \cup B)$ – A hodisa yoki B hodisa yoki ikkisining ham ro'y berish ehtimolligi

- $P(A \cap B)$ – A va B hodisalarning birgalikda ro'y berish ehtimolligi
- $P(A/B)$ – B hodisa ro'y bergandan so'ng A hodisaning ro'y berish ehtimolligi.

Yuqoridagi belgilashlardan foydalanib, ikkita A va B hodisalarning birgalikda ro'y berish ehtimolligi uchun quyidagi tenglikni yozishimiz mumkin:

$$P(A \cap B) = P(A/B)P(B) = P(B/A)P(A).$$

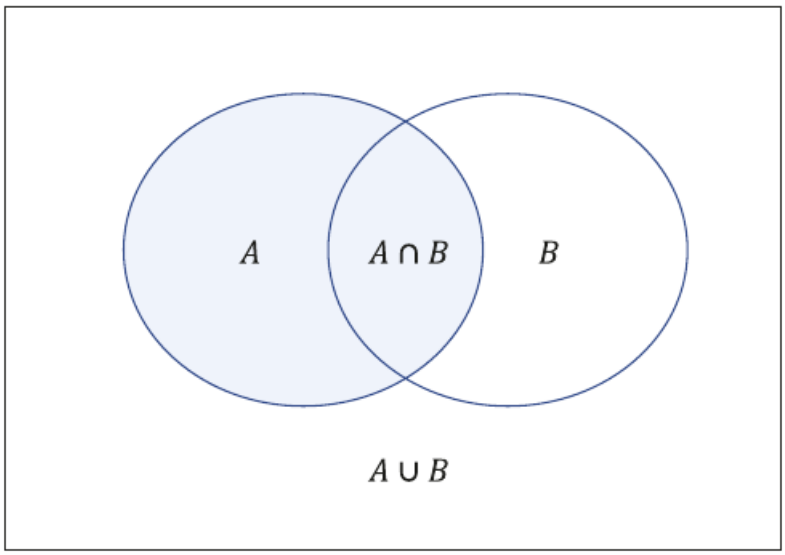
Ushbu tenglik ehtimolliklarni ko'paytirish qoidasidir.

Shu yerdan boshlab biz $A \cap B$ (A va B hodisalarning bir paytda sodir bo'lishi) yozuvi orqali ifodalangan "A kesishgan B" belgisini ishlatmaymiz va yozish qulay bo'lishi uchun uni AB deb belgilaymiz.

Osongina tekshirish mumkinki, A va B hodisalar ayirmasi ehtimolligi uchun quyidagi tenglik o'rinli:

$$P(A - B) = P(A) - P(AB).$$

Yuqorida A va B hodisalar uchun keltirilgan barcha tasdiqlarni oson tushunishda 2.15-rasmda keltirilgan *Venn Diagrammasi*dan foydalanamiz.



Rasm 2.15: Ikkita A va B hodisalarning birlashmasi va kesishmasini ko'rsatuvchi Venn diagrammasi.

Aytaylik, biror tajribada n ta natija qayd etildi: bulardan n_1 tasida A hodisa va n_2 tasida B hodisa ro'y berdi, A va B lar birgalikda esa m marta sodir bo'ldi.

Buni Venn Diagrammasida tasvirlaylik.

$P(A \cup B)$ ehtimollik uchta bir-biriga bog'liq bo'lmagan hodisalarning ehtimolliklar yig'indisi bilan ifodalanishi mumkin: $(A - B)$, $(B - A)$ va AB.

$$\begin{aligned} P(A \cup B) = P(A - B) + P(B - A) + P(AB) = \\ P(A) - P(AB) + P(B) - P(AB) + P(AB) = \\ P(A) + P(B) - P(AB). \end{aligned}$$

$P(A/B)$ ehtimollik bu B hodisa ro'y bergandan so'ng A ning ro'y berish ehtimolligi. B hodisa n_2 marta ro'y berganligini hisobga olinsa, A hodisa m marta sodir bo'lgan AB hodisa bilan cheklangan bo'ladi. Demak, B allaqachon ro'y bergan holatda A ning ehtimolligi quyidagicha ifodalanishi mumkin:

$$P(A/B) = \frac{m}{n_2}.$$

O'z navbatida $\frac{m}{n_2}$ ni

$$\frac{\frac{m}{n}}{\frac{n_2}{n}} = \frac{P(AB)}{P(B)}$$

kabi yozish mumkin.

Demak, $P(A/B) = P(AB)/P(B) \Rightarrow P(AB) = P(B)P(A/B)$.

Shunga o'xshab, agar $P(B/A)$ uchun ham yuqoridagi kabi amallarni bajarsak, $P(AB) = P(A)P(B/A)$ munosabatga kelamiz.

2.3.2 Hodisaning kesishishi uchun ehtimollikning zanjir qoidasi

Ikki hodisa bo'lgan hol uchun muhokama qilingan kesishishning ko'paytirish qoidasini n ta hodisa bo'lgan hol uchun ham qo'llash mumkin.

Agar n ta A_1, A_2, A_3, $\ldots A_n$ hodisalar to'plami bo'lsa, unda bu hodisalarning qo'shma ehtimolligi quyidagicha ifodalanishi mumkin:

$$\begin{aligned} P(A_1, A_2, A_3, \ldots A_n) \\ = P(A_1)P(A_2/A_1)P(A_3/A_1A_2)\ldots P(A_n/A_1A_2A_3\ldots A_{n-1}) \\ = P(A_1)\prod_{i=2}^{n} P(A_i/A_1A_2A_3\ldots A_{n-1}). \end{aligned}$$

2.3.3 O'zaro taqiqlanuvchi hodisalar

Ikkala A va B hodisalar o'zaro taqiqlanuvchi hodisalar deyiladi, agar ular bir paytda sodir bo'lmasa. Boshqacha qilib aytganda, A va B bir-birini o'zaro

taqiqlovchi hodisalardir, qachonki $P(AB) = 0$ tenglik o'rinli bo'lsa. O'zaro taqiqlanuvchi hodisalar uchun $P(A \cup B) = P(A) + P(B)$.

Umuman olganda, n ta o'zaro taqiqlanuvchi hodisalar birlashmasining ehtimolligi ularning ehtimolliklar yig'indisi sifatida yozilishi mumkin:

$$P(A_1 \cup A_2 \ldots \cup A_n) = P(A_1) + P(A_2) + \ldots + P(A_n) = \sum_{i=1}^{n} P(A_i).$$

2.3.4 Hodisalarning bog'liqsizligi

Ikkita A va B hodisalar bog'liqsiz (yoki, mustaqil) deyiladi, agar ularning kesishish (ya'ni, birgalikda ro'y berish) ehtimolligi ularning alohida ehtimolliklari ko'paytmasiga teng bo'lsa, ya'ni

$$P(AB) = P(A)P(B).$$

Bunday yozish mumkin, chunki berilgan B hodisa ehtimolligi sharti bilan A ning ehtimolligi bir xil; ya'ni, $P(A/B) = P(A)$.

Bu shuni anglatadiki, A barcha hodisalar to'plamida, B zonasida bo'lgani kabi sodir bo'ladi. Xuddi shunday, $P(B/A) = P(B)$, A va B hodisalar mustaqil bo'lishi uchun. Ikkala voqea mustaqil bo'lganda, boshqa voqealar sodir bo'lganligi sababli, ikkala voqea ham ta'sir qilmaydi.

2.3.5 Hodisalarning shartli bog'liqsizligi

Ikkita A va B hodisa, uchinchi bir C hodisa sodir bo'lgan holatda, shartli mustaqil deyiladi qachonki C berilgan holatda A va B larning birgalikda sodir bo'lish ehtimolini quyidagicha yozish mumkin bo'lsa:

$$P(AB/C) = P(A/C)P(B/C),$$

bunda faktorizatsiya xossasi $P(AB/C) = P(A/C)P(B/AC)$ orqali berilgan.

Oldingi tenglamalarni birlashtirib, quyidagi ifodani olamiz:

$$P(B/AC) = P(B/C).$$

Yodda tutish kerakki, A va B hodisalarning shartli mustaqilligi A va B ning mustaqil hodisalar ekanligini bildirmaydi. Hodisalarning shartli mustaqilligi Sun'iy o'rganish sohalarida juda ko'p qo'llaniladi. Shuningdek, Bayes tarmoqlari deb nomlanuvchi tarmoq modellari sinfi tarmoqni soddalashtirish uchun bir necha omillardan biri sifatida shartli mustaqillikdan foydalanadi.

2.3.6 Bayes qoidasi

Elementar ehtimollik haqida asosiy tushunchaga ega bo'lganimiz uchun, Bayes qoidasi deb nomlangan juda muhim teoremani muhokama qilaylik. Biz teoremani tushuntirish uchun ikkita A va B hodisalarni olamiz, ammo buni har qanday sondagi hodisalar uchun ham umumlashtirish mumkin.

Ehtimolliklarni ko'paytirish qoidasidan quyidagi ifodani olamiz:

$$P(AB) = P(A)P(B/A). \tag{2.6}$$

Xuddi shunday, ikkinchi ifodani yozish mumkin:

$$P(AB) = P(B)P(A/B). \tag{2.7}$$

(2.6) va (2.7) ni birlashtirib, quyidagini olamiz:

$$\begin{aligned} P(A)P(B/A) &= P(B)P(A/B) \\ \Rightarrow P(A/B) &= P(A)P(B/A)/P(B) \end{aligned}$$

Yuqorida olingan qoida *Bayes qoidasi* deb ataladi va u Sun'iy o'rganishning ko'plab sohalarida, masalan, "posterior" taqsimotni hisoblashda, Markov zanjir modellaridan foydalanishda, "posterior" algoritmni maksimalga ko'tarish va hokazolarda foydalidir.

Wikipedia masalasi: Ikkita quti berilgan va ularga sharchalar solingan. Birinchisida o'nta (10) qora va o'ttizta (30) oq sharcha mavjud; ikkinchisida har ikki rangdagi sharchadan yigirmatadan (20) solingan. Biz tasodifiy ravishda qutilaridan biridan bitta sharcha olamiz. Agar olingan sharcha oq rangda bo'lsa, uning birinchi qutidan olingan bo'lishlik ehtimolini toping.

Intuitiv ravishda biz shuni tushunamizki, bu sharcha ikkinchi qutidan emas, balki birinchisidan keladi. Shunday qilib, bu ehtimollik 50 % dan katta bo'lishi kerak. Aniq javobni (60 %) Bayes teoremasidan hisoblash mumkin.

Aytaylik, sharchaning birinchi qutidan olinishini H_1 hodisa deb belgilaylik, o'z navbatida ikkinchi qutidan olinishini H_2 hodisa sifatida belgilaymiz. Sharchani olishda biror qutida ustunlik bo'lmagani uchun, H_1 va H_2 hodisalarning sodir bo'lish ehtimolligi teng, $P(H_1) = P(H_2)$. Bundan tashqari, sharcha ikkita qutidan biridan albatta olinadi. Demak, ikkita ehtimollikning yig'indisi 1 ga teng: ularning har biri 50 % ga teng.

Endi, "Oq sharcha olinish" hodisasini D orqali belgilaymiz. Qutilaridan biridan tasodifiy ravishda sharchani tortayotganimizda, H_1 shart ostida D hodisaning sodir bo'lish ehtimoli quyidagiga teng:

$$P(D|H_1) = 30/40 = 75\%.$$

Xuddi shunday, H_2 shart ostida D hodisaning ro'y berish ehtimolligi:

$$P(D|H_2) = 20/40 = 50\%.$$

Bayes formulasidan so'ralgan ehtimollikni osongina hisoblash mumkin:

$$P(H_1|D) = \frac{P(H_1) \cdot P(D|H_1)}{P(H_1) \cdot P(D|H_1) + P(H_2) \cdot P(D|H_2)} = \frac{50\% \cdot 75\%}{50\% \cdot 75\% + 50\% \cdot 50\%} = 60\%.$$

Shunday qilib, birinchi sharcha oq ekanligi ma'lum bo'lsa, bu sharchaning birinchi qutidan olinganlik ehtimoli 60 % ekan.

2.3.7 Ehtimollik massasi funksiyasi

Tasodifiy o'zgaruvchining ehtimollik massasi funksiyasi (pmf) bu tasodifiy o'zgaruvchi qabul qilishi mumkin bo'lgan har bir diskret qiymatning ehtimolligini beruvchi funksiyadir. Ehtimolliklar yig'indisi 1 ga teng bo'lishi kerak.

Masalan, o'yin soqqasi tashlanganda, soqqaning yuzidagi raqam biror tasodifiy o'zgaruvchi X ga teng bo'lsin. Shunda, pmf quyidagicha aniqlanishi mumkin:

$$P(X = i) = \frac{1}{6},\ i \in \{1, 2, 3, 4, 5, 6\}$$

2.3.8 Ehtimollik zichligi funksiyasi

Ehtimollik zichligi funksiyasi (pdf) bu uzluksiz tasodifiy o'zgaruvchining qiymatlar sohasidagi har bir qiymatiga mos keluvchi ehtimollik zichligidir. Bu yerda uzluksiz o'zgaruvchi haqida gap ketmoqda, va uning qiymatlar sohasida ehtimollik zichligi funksiyasidan olingan integral 1 ga teng bo'lishi kerak.

Aytaylik, X tasodifiy miqdorning qiymatlar sohasi D bo'lsin. $P(x)$ bu ehtimolliklar taqsimotining zichligi funksiyasini bildiradi, shuning uchun

$$\int_D P(x)dx = 1.$$

Masalan, 0 dan 1 gacha qiymatlarni qabul qiluvchi uzluksiz tasodifiy miqdorning ehtimollik zichligi funksiyasi $P(x) = 2x$ orqali berilgan bo'lsin, bunda $x \in [0, 1]$. Buning ehtimolliklar taqsimotining zichlik funksiyasi ekanligini tekshirib ko'raylik.

$P(x)$ ehtimollik zichligi funksiyasi bo'lishi uchun $\int_{x=0}^{1} P(x)dx = 1$ bo'lishi kerak:

$$\int_{x=0}^{1} P(x)dx = \int_{x=0}^{1} 2xdx = x^2\Big|_0^1 = 1.$$

Demak, $P(x)$ ehtimollik zichligi funksiyasi ekan.

Shuni ta'kidlash kerakki, integral egri chiziq ostidagi yuzani hisoblaydi va $P(x)$ ehtimollik zichligi funksiyasi (pdf) bo'lganligi sababli, ehtimollik egri chizig'i ostidagi yuza 1 ga teng bo'lishi kerak.

2.3.9 Tasodifiy miqdorning matematik kutilmasi

Tasodifiy miqdorning matematik kutilmasi bu tasodifiy miqdorning aynan o'rtacha qiymatidir. Aytaylik, X tasodifiy miqdor n ta diskret $x_1, x_2, \ldots, x_n$ qiymatlarni qabul qiladi, va ehtimolliklari $p_1, p_2, \ldots, p_n$. Boshqacha qilib aytganda, X bu diskret tasodifiy miqdor va uning ehtimollik massa funksiyasi $P(X = x_i) = p_i$. Keyin, X tasodifiy miqdorning matematik kutilmasi quyidagicha beriladi:

$$E[X] = x_1p_1 + x_2p_2 + \ldots + x_np_n = \sum_{i=1}^{n} x_ip_i.$$

Agar X uzluksiz tasodifiy miqdor bo'lsa va $P(x)$ uning ehtimollik zichligi funksiyasi bo'lsa, X ning matematik kutilmasi quyidagicha topiladi:

$$E[X] = \int_D xP(x)dx,$$

bunda D soha $P(x)$ ning aniqlanish sohasi.

2.3.10 Tasodifiy miqdorning tarqoqligi

Tasodifiy miqdorning tarqoqligi ushbu miqdorning o'zgaruvchanligini bildiradi. Bu degani, tasodifiy o'zgaruvchi o'rtachasi (matematik kutilmasi)dan uning kvadratik og'ishi o'rtachasidir (yoki matematik kutilmasi).

X tasodifiy miqdorning o'rtacha qiymati $\mu = E[X]$ bo'lsin. U holda

$$Var[X] = E[(X - \mu)^2],$$

bu yerda $\mu = E[X]$.

Agar X diskret tasodifiy miqdor bo'lib, n ta diskret qiymatlarni qabul qilsa va ehtimollik massa funksiyasi $P(X = x_i) = p_i$ berilgan bo'lsa, X ning tarqoqligini quyidagicha aniqlash mumkin:

$$Var[X] = E[(X - \mu)^2] = \sum_{i=1}^{n}(x_i - \mu)^2p_i.$$

Agar X uzluksiz tasodifiy miqdor bo'lsa va u $P(x)$ ehtimollik zichligi funksiyasiga ega bo'lsa, u holda $Var[X]$ quyidagicha aniqlanadi:

$$Var[X] = \int_D (x-\mu)^2 P(x)dx,$$

bunda D soha $P(x)$ ning aniqlanish sohasi.

```

# Masalaning qo'yilishi: Python-da tasodifiy o'zgaruvchining tarqoqligini hisoblang
# Yechim: Dastlab Numpy paketlar majmuasidan foydalanib, masalani yechamiz

# Kutubxonani yuklash
import numpy as np

# Ma'lumotlar namunasi list ko'rinishida berilgan
results = [-14.82381293, -0.29423447, -13.56067979,
           -1.6288903, -0.31632439, 0.53459687,
           -1.34069996, -1.61042692, -4.03220519,
           -0.24332097]

print('Variance of results using numpy:', np.var(results))
# Javob: Variance of results using numpy: 28.822364260579157

# Tarqoqlikni formulaga ko'ra hisoblasa ham bo'ladi
# O'rtacha qiymat, mean
m = sum(results) / len(results)

# calculate variance using a list comprehension
var_res = sum((xi - m) ** 2 for xi in results) / len(results)

print ('Variance of results by hand:',var_res)
# Javob: Variance of results by hand: 28.822364260579157
```

Listing 2.22: Python-da tasodifiy miqdorning tarqoqligi

2.3.11 Asimmetriya va Kurtosis

Asimmetriya va Kurtosis tasodifiy miqdor uchun yuqori darajali momentlar orqali aniqlanuvchi sonli xarakteristikalardir. Asimmetriya ehtimollik taqsimoti simmetriyasini o'lchasa, Kurtosis ehtimollik taqsimotining chetlari og'irmi yoki yo'qmi – shuni o'lchaydi. Asimmetriya uchinchi tartibli moment orqali quyidagicha aniqlanadi:

$$Skew(X) = \frac{E[(X-\mu)^3]}{(Var[X])^{3/2}}.$$

Mukammal simmetriyaga ega bo'lgan ehtimollik taqsimoti, 2.16-rasmda ko'rsatilgandek, 0 asimmetriyaga ega. Musbat qiymatli asimmetriya shuni anglatadiki, taqsimotning asosiy qismi chap tomonda bo'ladi, 2.17-rasmga qarang. Asimmetriyaning manfiy qiymatida esa ma'lumotlarning asosiy qismi o'ng tomonga yo'naltirilgan bo'ladi, 2.18-rasmda ko'rsatilgan.

```

# Masalaning qo'yilishi: Python-da tasodifiy o'zgaruvchining asimmetriyasini hisoblang
# Yechim: Quyidagi kutubxonadan foydalanib, Python-da har qanday ma'lumotning asimmetriyasini osongina topishimiz mumkin.

# Kutubxonani yuklash
from Scipy.stats import skew

# Ma'lumotlar namunasi list ko'rinishida berilgan
x = np.random.normal(0, 5, 10)

print('X:', x)
print('Skewness for data:', skew(x))

# Javoblardan biri:
# X: [-1.40370937  6.28101961 -0.96663508  2.41419594  5.51670985 -1.56326869 -2.8109994   3.332224    8.17142599  7.08181734]

# Skewness for data: -0.007739223874939867
```

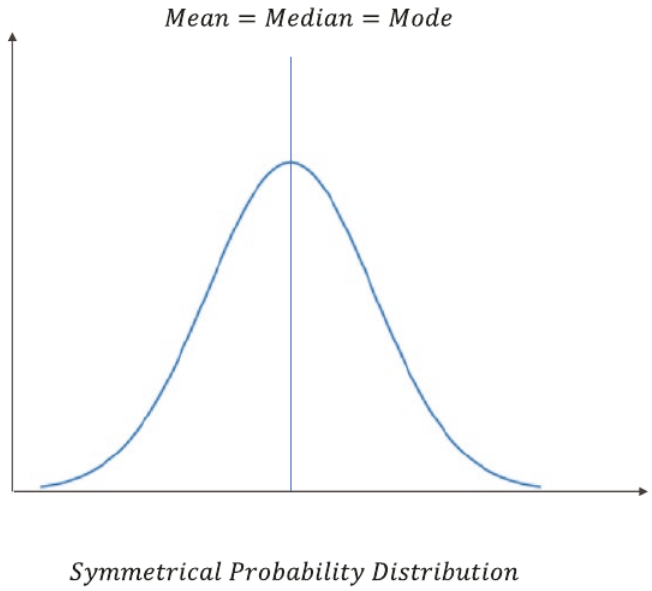
Listing 2.23: Python-da tasodifiy miqdorning asimmetriyasi

Kurtosis to'rtinchi tartibli statistika hisoblanib, o'rtacha qiymati μ bo'lgan tasodifiy miqdor X uchun u quyidagicha aniqlanadi:

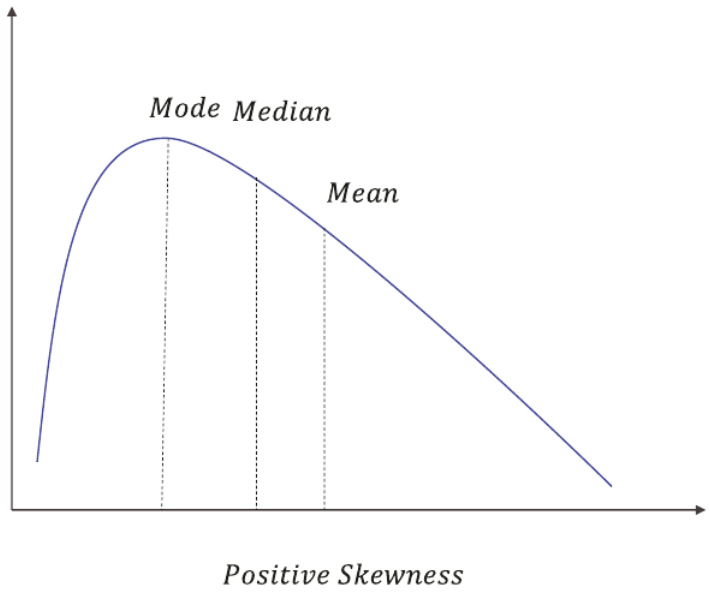
$$Kurt(X) = \frac{E[(X-\mu)^4]}{(Var[X])^2}.$$

Kurtosisning yuqori qiymatlarida ehtimollik taqsimoti uchun og'irroq dumlarni keltirib chiqaradi, buni biz 2.20-rasmda ko'rib turibmiz. Normal taqsimot uchun Kurtosis (2.19-rasmga qarang) 3 ga teng. Ammo, boshqa taqsimotlarning Kurtosisini o'lchash uchun, odatda, haqiqiy Kurtosisdan normal taqsimot uchun berilgan Kurtosisni, ya'ni 3 ni ayirib topiladi.

```

# Masalaning qo'yilishi: Python-da tasodifiy o'zgaruvchining kurtosisini toping
# Yechim: Quyidagi kutubxonadan foydalanib, Python-da har qanday ma'lumotning kurtosisini osongina topishimiz mumkin.
```

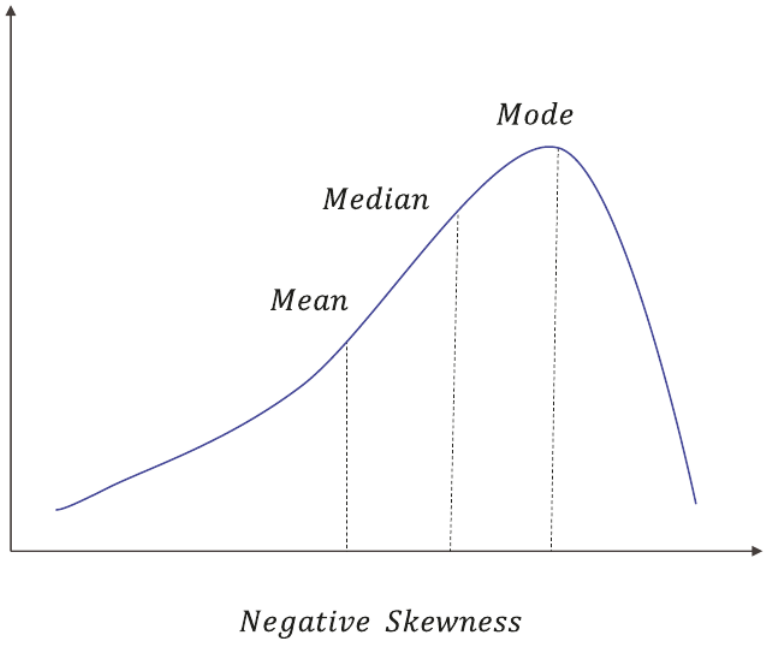


Rasm 2.16: Simmetrik ehtimollik taqsimoti.

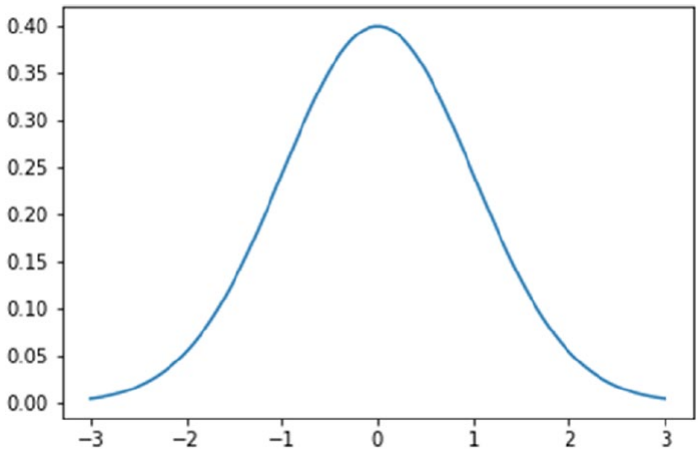


Rasm 2.17: Asimmetriyasi musbat bo'lgan ehtimollik taqsimoti.

```
4
5 # Kutubxonani yuklash
6 import numpy as np
7 import pandas as pd
```

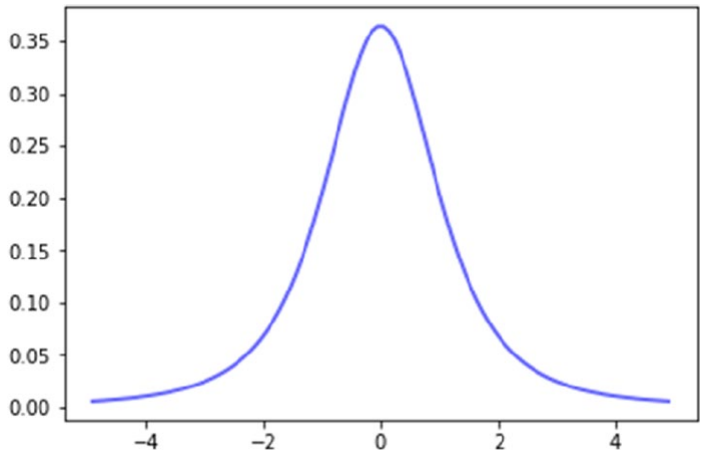


Rasm 2.18: Asimmetriyasi manfiy bo'lgan ehtimollik taqsimoti.



Rasm 2.19: Kurtosis 3 ga teng bo'lgan standard normal taqsimot.

```
8  from scipy.stats import kurtosis
9  from scipy.stats import skew
10
11 import matplotlib.pyplot as plt
12 plt.style.use('ggplot')
13
```



Rasm 2.20: Kurtosis ∞ bo'lgan talabalar T taqsimoti.

```
# Normal taqsimot
data = np.random.normal(0, 1, 10000000)
np.var(data)

plt.hist(data, bins=60)
print("mean : ", np.mean(data))
print("var  : ", np.var(data))
print("skew : ",skew(data))
print("kurt : ",kurtosis(data))

# Output:
mean :  0.000410213500847
var  :  0.999827716979
skew :  0.00012294118186476907
kurt :  0.0033554829466604374
```

Listing 2.24: Python-da tasodifiy o'zgaruvchining kurtosisi

2.3.12 Kovariatsiya

Ikkita tasodifiy miqdor X va Y orasidagi kovariatsiya ularning qo'shma o'zgaruvchanligining o'lchovidir. Agar X ning katta qiymatlari Y ning katta qiymatlariga va yoki X ning kichik qiymatlari Y ning kichik qiymatlariga mos kelsa, kovariatsiya musbat bo'ladi. Boshqa tomondan, agar X ning yuqoriroq qiymatlari Y ning kichik qiymatlariga va yoki X ning kichik qiymatlari Y ning kattaroq qiymatlariga mos kelsa, u holda kovariatsiya manfiy bo'ladi.

X va Y ning kovariatsiyasi uchun formula quyidagicha:

$$cov(X, Y) = E[X - u_x][Y - u_y],$$

bu yerda $u_x = E[X]$, $u_y = E[Y]$.

Avvalgi formulaning o'ng tomonini soddalashtirib, quyidagicha yozish mumkin:

$$cov(X, Y) = E[XY] - u_x u_y.$$

Agar ikkita tasodifiy miqdor o'zaro mustaqil bo'lsa, ularning kovariatsiyasi nolga teng, chunki $E[XY] = E[X]E[Y] = u_x u_y$.

```

# Masalaning qo'yilishi: Python-da ikki tasodifiy o'zgaruvchining Kovariatsiyasini toping
# Yechim: Quyidagi kutubxonadan foydalanib, Python-da ma'lumotning Kovariatsiyasini topishimiz mumkin.

# Kutubxonani yuklash
import numpy as np

x = np.array([[0, 2], [1, 1], [2, 0]]).T

x
array([[0, 1, 2],
       [2, 1, 0]])

np.cov(x)
array([[ 1., -1.],
       [-1.,  1.]])
```

Listing 2.25: Python-da ikki tasodifiy miqdorning kovariatsiyasi

2.3.13 Korrelatsiya koeffitsienti

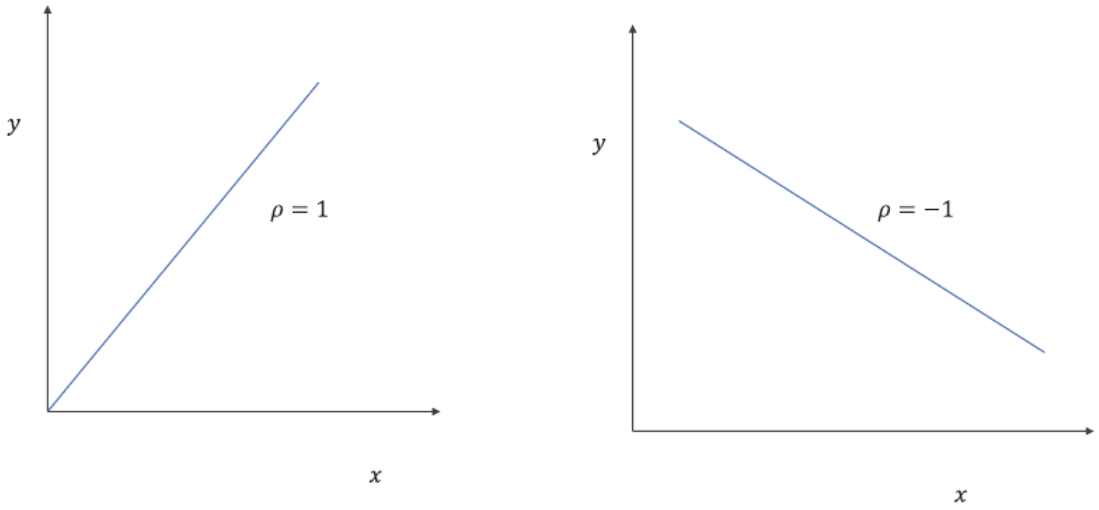
Umuman olganda, kovariatsiya ikki tasodifiy miqdorning o'zaro bog'liqlik darajasi to'g'risida har doim ham yetarli ma'lumot bermaydi, chunki ikkita miqdor juda farqli masshtabda bo'lishi mumkin. Shu ma'noda, miqdorlarning korrelatsiya koeffitsientlari orasidagi chiziqli bog'liqlikni aniqlash ancha qulaydir.

X va Y tasodifiy miqdorlar orasidagi korrelatsiya koeffitsienti quyidagicha aniqlanadi:

$$\rho = \frac{cov(X, Y)}{\sigma_x \sigma_y},$$

bu yerda σ_x va σ_y mos ravishda X va Y larning standart og'ishlari. ρ ning qiymati -1 va $+1$ oralig'ida bo'ladi.

2.21-rasmda X va Y ikkita tasodifiy miqdorlar orasidagi musbat va manfiy korrelatsiyalar ko'rsatilgan.



Rasm 2.21: Korrelyatsiya koeffitsientlari $+1$ va -1 ga teng bo'lgan o'zgaruvchilar grafigi.

2.3.14 Muhim ehtimollik taqsimotining ba'zi turlari

Ushbu bo'limda biz Sun'iy o'rganish va chuqur o'rganish sohalarida tez-tez ishlatiladigan ba'zi ehtimollik taqsimotlarini ko'rib chiqamiz.

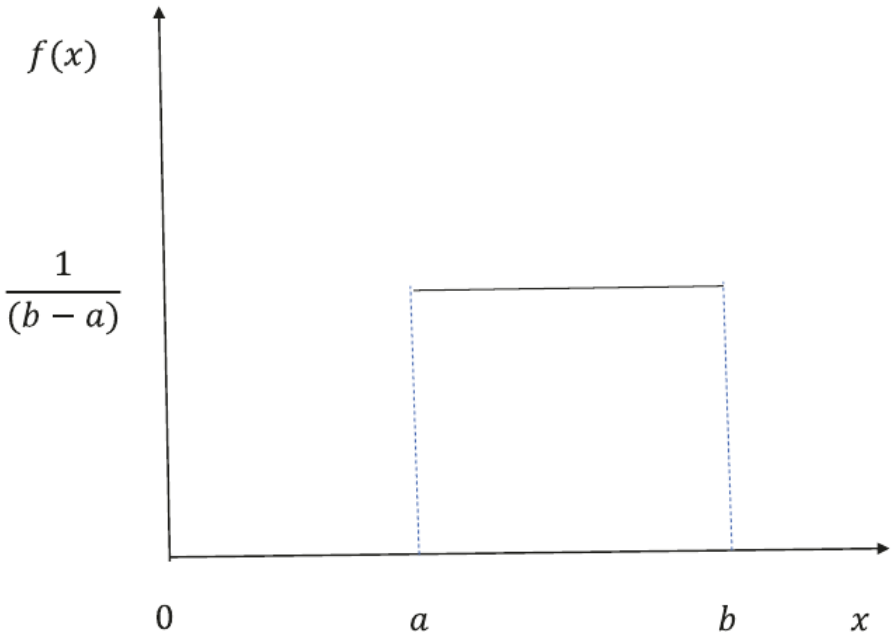
Bir jinsli taqsimot

Bir jinsli taqsimot (yoki, tekis taqsimot) uchun ehtimollik zichligi funksiyasi doimiydir. Qiymatlari a va b $(b > a)$ oraliqda yotuvchi uzluksiz tasodifiy miqdor uchun ehtimollik zichligi funksiyasi quyidagicha ifodalanadi

$$P(X = x) = f(x) = \begin{cases} 1/(b-a), & x \in [a, b] \\ 0, & boshqa\ sohada \end{cases}$$

2.22-rasmda bir jinsli taqsimot uchun ehtimollik zichligi grafigi ko'rsatilgan. Bir jinsli taqsimot uchun turli xil statistikalar bu yerda keltirilgan:

$$E[X] = \frac{(b+a)}{2}$$
$$Median[X] = \frac{(b+a)}{2}$$
$$Mode[X] = [a, b]\ intervaldagi\ barcha\ nuqtalar$$
$$Var[X] = (b-a)^2/12$$
$$Skew[X] = 0$$
$$Excessive\ Kurt[X] = -6/5$$



Rasm 2.22: Bir jinsli ehtimollik taqsimoti.

Shuni yodda tutish kerakki, ortiqcha Kurtosis - bu haqiqiy Kurtosisdan 3 ning ayirmasi, 3 - normal taqsimot uchun haqiqiy Kurtosis. Demak, ortiqcha Kurtosis normal taqsimotga nisbatan nisbiy Kurtosisdir.

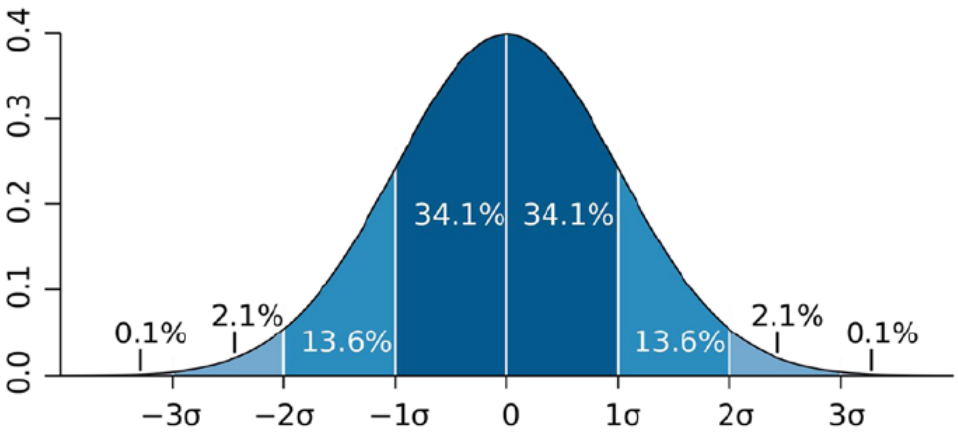
Normal Taqsimot

Bu, ehtimol, reallikda ehtimollik taqsimotining eng muhim senariyidir. Normal taqsimotda maksimal ehtimollik zichligi taqsimotning o'rtacha qiymatiga to'g'ri keladi va zichlik nosimmetrik va ekspansional ravishda o'rtacha masofadan kvadratga tushadi. Normal taqsimotning ehtimollik zichligi funksiyasi quyidagicha beriladi:

$$P(X = x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad -\infty < x < \infty,$$

bu yerda μ o'rtacha qiymat va σ^2 berilgan tasodifiy miqdor X ning tarqoqligi. Bir o'zgaruvchili normal taqsimotning zichlik funksiyasi 2.23-rasmda keltirilgan.

2.23-rasmda ko'rsatilgandek, normal taqsimotdagi ma'lumotlarning 68.2 % o'rtacha qiymatdan bitta $(+1/-1\sigma)$ standart og'ish oralig'iga to'g'ri keladi va ma'lumotlarning qariyb 95.4 % o'rtacha qiymatdan $+2/-2\sigma$ atrofida bo'lishi kutilmoqda. Normal taqsimot uchun muhim statistika bu yerda



Rasm 2.23: Normal ehtimollik taqsimoti.

keltirilgan:

$$\begin{aligned} E[X] &= \mu \\ Median[X] &= \mu \\ Mode[X] &= \mu \\ Var[X] &= \sigma^2 \\ Skew[X] &= 0 \\ Excessive\ Kurt[X] &= 0 \end{aligned}$$

Har qanday normal taqsimot quyidagicha almashtirishlardan so'ng standart normal taqsimotga aylantirilishi mumkin:

$$z = \frac{(x - \mu)}{\sigma}.$$

Standart normal tasodifiy o'zgaruvchi z uchun o'rtacha qiymat va standart og'ish mos ravishda 0 va 1 ga teng. Standart normal taqsimot statistik ta'sir o'tkazish testlarida juda ko'p qo'llaniladi. Xuddi shunday, chiziqli regressiyada xatolar odatda normal taqsimlangan deb hisoblanadi.

Ko'p o'zgaruvchili normal taqsimot

Ko'p o'zgaruvchili normal taqsimot, yoki n o'zgaruvchili Gauss taqsimoti $x \in \mathbb{R}^{n\times 1}$ vektor bilan ifodalanadi. Ushbu taqsimot qo'shma o'zgaruvchilarning birgalikdagi ehtimollik taqsimoti bo'lib, $\mu \in \mathbb{R}^{n\times 1}$ o'rtacha vektor va $\Sigma \in \mathbb{R}^{n\times n}$ kovariant matritsa bilan belgilanadi.

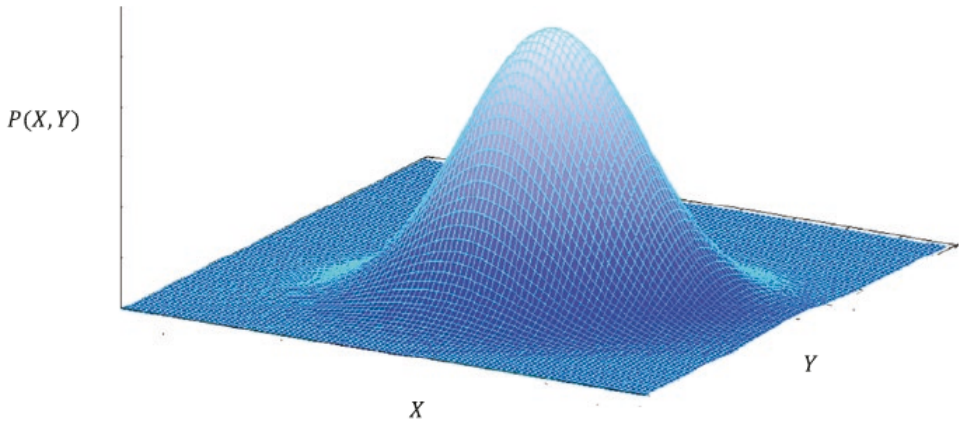
Ko'p o'zgaruvchili normal taqsimotning ehtimollik zichligi funksiyasi (inglizcha "probability density function", yoki qisqacha "pdf") quyidagicha:

$$P(x/\mu;\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{-1/2}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)},$$

bu yerda

$$x = [x_1 x_2 \dots x_n]^T,\ -\infty < x_i < +\infty,\ \forall i \in \{1, 2, 3, \dots n\}.$$

2.24-rasmda tasvirlangan chizma ko'p o'zgaruvchili normal taqsimotning ehtimollik zichligi funksiyasidir. Ko'p o'zgaruvchili normal taqsimot, yoki Gauss taqsimoti, Sun'iy o'rganishning turli masalalarida ishlatiladi. Masalan, korrelyatsiyaga ega bo'lgan ko'p o'zgaruvchili kiritma ma'lumotlar uchun kiritma belgilar ko'pincha ko'p o'zgaruvchili normal taqsimotni kuzatish uchun qabul qilinadi va ehtimollik zichligi funksiyasiga asoslanib, kichik ehtimollik zichligi bo'lgan nuqtalar anomaliyalar deb belgilanadi. Shuningdek, ko'p o'zgaruvchili normal taqsimot Gauss modellari aralashmasida keng qo'llaniladi, unda bir nechta xususiyatlarga ega ma'lumotlar nuqtasi turli ehtimolliklarga ega bo'lgan bir nechta ko'p o'zgaruvchili normal taqsimotlarga tegishli deb taxmin qilinadi. Gaussiyanlarning aralashmalari bir nechta sohalarda qo'llaniladi, masalan, klasterlash, anomaliyani aniqlash, Markov yashirin modellari va boshqalar.



Rasm 2.24: Ikki o'zgaruvchili normal taqsimot.

Bernoulli Taqsimoti

Biror eksperimentda ikkita hodisa o'zaro bir-birini ta'qiqlovchi va to'liq (ikki hodisa ehtimolligi yig'indisi 1 ga teng) bo'lsa, bunday eksperiment Bernoulli tajribasi deb ataladi.

Bernoulli tajribasi Bernoulli taqsimotiga bo'ysinadi. Aytaylik, Bernoulli tajribasidagi ikkita hodisa "g'alaba" va "mag'lubiyat" bo'lsin. G'alaba ehtimolligi p bo'lsa, mag'lubiyat ehtimolligi $1-p$ bo'ladi. G'alaba $x=1$ orqali berilsin. U holda, g'alaba yoki mag'lubiyat ehtimolligi quyidagicha belgilanishi mumkin:

$$P(X=x) = f(x) = p^x(1-p)^{(1-x)},\ x \in \{0,1\}.$$

$P(X=x)$ uchun oldingi ifoda Bernoulli taqsimotining ehtimollik massasi funksiyasini bildiradi. Ehtimollik massasi funksiyasining matematik kutilmasi va tarqoqligi quyidagicha:

$$E[X] = p$$
$$Var[X] = p(1-p)$$

Bernoulli taqsimoti o'zaro bir-birini ta'qiqlovchi (eksklyuziv) va to'ldiruvchi bo'lgan ko'p tasnifli hodisalarga qo'llanilishi mumkin. Har qanday ikki sinfli tasniflash masalasi Bernoulli tajribasi sifatida modellashtirilishi mumkin. Masalan, logistik regressiya ehtimolligi funksiyasi har bir o'qitish ma'lumotlari bo'yicha Bernoulli taqsimotiga asoslanadi, va bunda p ehtimollik sigmoid funksiya orqali beriladi.

Binomial Taqsimot

Bernoulli tajribalar ketma-ketligida biz ko'pincha muvaffaqiyat va muvaffaqiyatsizliklar umumiy soni ehtimolligi bilan qiziqamiz, ushbu hodisalar qaysi bir tajribada sodir bo'lganligi bilan emas. Agar, n ta Bernoulli tajribalar ketma-ketligida x muvaffaqiyatlar sonini bildirsa, mana shu n ta Bernoulli tajribasida sodir bo'lgan x ta muvaffaqiyat ehtimolligi ehtimollik massa funksiyasi orqali quyidagicha ifodalanadi:

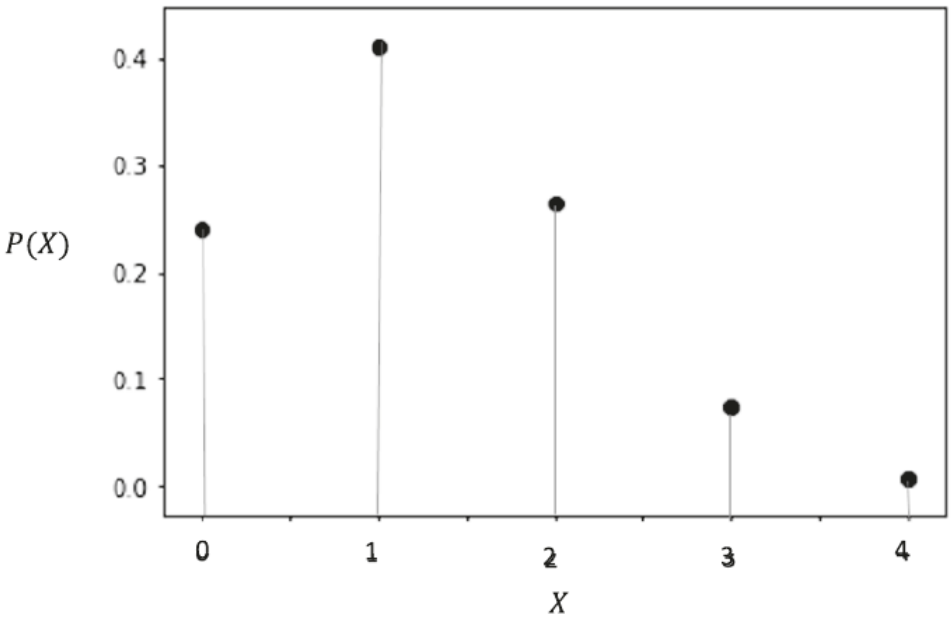
$$P(X=x) = \binom{n}{x} p^x(1-p)^{(n-x)},\ x \in \{0,1,2\ldots,n\}$$

bu yerda p - muvaffaqiyat ehtimolligi.

Taqsimotning matematik kutilmasi va tarqoqligi quyidagicha:

$$E[X] = np$$
$$Var[X] = np(1-p)$$

2.25-rasmda $n=4$ va $p=0.3$ bo'lgan binomial taqsimotning ehtimollik massa funksiyasi ko'rsatilgan.



Rasm 2.25: Binomial taqsimotning ehtimollik massa funksiyasi, $n = 4$ va $p = 0.3$ bo'lgan holat.

Poisson Taqsimoti

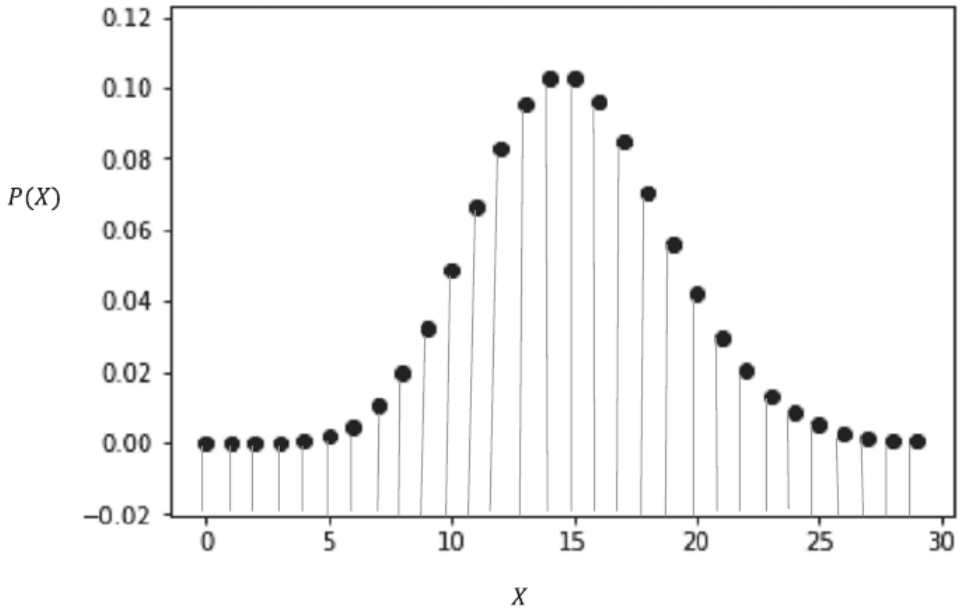
Har safar biron miqdorning takrorlanish tezligi bilan ishlaganimizda, masalan, 1000 ta mahsulot partiyasidagi nuqsonlar soni, to'rt soat davomida radioaktiv modda chiqargan alfa zarrachalar soni va boshqa bunday hodisalarni tasvirlash usuli sifatida Poisson taqsimoti odatda eng yaxshisi hisoblanadi. Poisson taqsimoti uchun ehtimollik massa funksiyasi quyidagicha:

$$P(X = x) = \frac{e^{-\lambda}\lambda^x}{x!}, \; x \in \{0, 1, 2, \ldots \infty\}$$
$$E[X] = \lambda,$$
$$Var[X] = \lambda$$

2.26-rasmda Poisson taqsimotining massa funksiyasi ko'rsatilgan, bunda o'rtacha qiymat $\lambda = 15$.

O'xshashlik funksiyasi

O'xshashlik bu asosiy ma'lumotlarni paydo qiluvchi parametrlar berilgan holda kuzatilgan ma'lumotlarning ehtimolligi. Tushuntiramiz: faraz qilaylik, biz n ta $x_1, x_2, \ldots x_n$ kuzatuvlarni olib boramiz. Yana faraz qilaylikki,



Rasm 2.26: Poisson taqsimotning ehtimollik massa funksiyasi, $\mu = 15$ bo'lgan holat.

kuzatuvlar mustaqil va bir xil ravishda o'rtacha qiymati μ va tarqoqligi σ^2 bo'lgan normal taqsimot bilan berilgan.

Bu holda o'xshashlik funksiyasi quyidagicha bo'ladi:

$$P(Ma'lumotlar/Model\ parametrlar) = P(x_1, x_2, \ldots x_n/\mu, \sigma^2)$$

Kuzatuvlar mustaqil bo'lganligi sababli, ehtimollikni quyidagicha aniqlanadi:

$$P(Ma'lumotlar/Model\ parametrlar) = \prod_{i=1}^{n} P\left(x_i/\mu, \sigma^2\right)$$

Har bir $x_i \sim Normal(\mu, \sigma^2)$, shuning uchun o'xshashlik funksiyasi quyidagi tarzda kengaytirilishi mumkin:

$$P(Ma'lumotlar/Model\ parametrlar) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}.$$

Noma'lum parametrni maksimal o'xshashlik usuli orqali baholash

Maksimal o'xshashlik orqali baholash (inglizcha "Maximum Likelihood Estimate", MLE) bu taqsimot yoki model parametrlarini baholash usulidir. Bunga, o'xshashlik funksiyasini maksimallashtiruvchi parametrlarni olish orqali

erishiladi, ya'ni model parametrlari berilgan holda ma'lumotlarni kuzatish ehtimolligini maksimallashtiradi. Quyida bir misol yordamida maksimal o'xshashlik usulini tushuntiramiz.

Aytaylik, Holmat tangani 10 marta otdi: 7 marta gerb va 3 marta raqam tomonlarini ko'rdi. Shuningdek, hodisalar mustaqil va bir xil deb faraz qilinadi. Berilgan tanganing gerb tushish ehtimolligi uchun maksimal o'xshashlik bahosi qanday bo'ladi?

Aytish kerakki, ushbu ekperiment Bernoulli tajribasiga misol bo'ladi, gerb tushish ehtimolligini p deb olamiz va bu biz baholamoqchi bo'lgan noma'lum parametrdir. Shuningdek, gerb tushish hodisasini 1 bilan belgilanadi va raqamli tomon tushishi 0 bilan belgilanadi.

O'xshashlik funksiyasini quyidagicha ko'rsatish mumkin:

$$P(Ma'lumotlar/parametr) = L(p) = P(x_1, x_2, \dots x_{10}/p) = \\ = \prod_{i=1}^{10} P(x_i/p) = \\ p^7(1-p)^3$$

Shunchaki tushuntirish uchun L o'xshashlikning $p^7(1-p)^3$ ga teng bo'lganini bilib olaylik.

Har bir gerb tomon uchun Bernoulli taqsimotidan kelib chiqadigan ehtimollik $P(x_i = 1/p) = p^1(1-p)^0 = p$. Xuddi shunday, har bir raqamli tomon uchun ehtimollik $P(x_i = 0/p) = p^0(1-p)^1 = 1-p$. Bizda 7 marta gerb va 3 marta raqam tushgan ekan, o'xshashlik uchun $L(p) = p^7(1-p)^3$ ni yozish mumkin.

L o'xshashlikni maksimallashtirish uchun p ga nisbatan L hosilasini olib, uni 0 ga tenglashimiz kerak.

Endi $L(p)$ o'xshashlikni maksimallashtirish o'rniga, biz uning logarifmni, ya'ni $\log L(p)$ maksimal darajaga ko'tarishimiz mumkin. Logarifmik funksiya monotonik o'suvchi funksiya bo'lgani uchun, $L(p)$ ni maksimallashtiruvhi parametr $\log L(p)$ ni ham maksimallashtiradi.

$$\log L(p) = 7 \log p + 3 \log(1-p).$$

Tenglikning har ikkala tomondan olingan hosilasini olib nolga tenglaymiz:

$$\frac{d \log(L(p))}{dp} = \frac{7}{p} - \frac{3}{1-p} \\ \Rightarrow p = 7/10$$

Qiziqqan talabalarimiz ikkinchi tartibli hosila $\frac{d^2 \log(L(p))}{dp^2}$ ni $p = \frac{7}{10}$ nuqtada hisoblashlari mumkin; natija manfiy qiymatni beradi va $p = \frac{7}{10}$ haqiqatan ham maximal nuqta ekanligini tasdiqlaydi.

Optimallashtirish masalalari bilan ishlaganda yordam berishi mumkin bo'lgan yana bir nozik usulni ko'rib chiqaylik. $f(x)$ funksiyasining maksimalini hisoblash, $-f(x)$ funksiya uchun minimalni hisoblash bilan bir xil. $f(x)$ uchun maximal va $-f(x)$ uchun minimal qiymat x ning bir xil qiymatida bo'ladi. Xuddi shunday, $f(x)$ uchun maximal va $1/f(x)$ uchun minimal x ning bir xil qiymatida bo'ladi.

Sun'iy o'rganish va chuqur o'rganish amaliyotlarida ko'pincha optimallashtirish paketlaridan foydalanamiz, ular faqat model parametrlarini hisoblash uchun qiymat funksiyasini minimallashtirishni biladi xolos. Bunday hollarda biz maksimallashtirish masalasini ishorani o'zgartirish yoki funksiyaning teskarisini olish orqali minimallashtirish masalasiga qulay tarzda aylantiramiz (albatta qaysi biri ko'proq ma'noga ega bo'lsa shuni tanlagan holatda). Masalan, oldingi masalada biz o'xshashlik funksiyasi logarifmini manfiy ishora bilan olishimiz mumkin edi, ya'ni, $-\log L(p)$; va uni minimallashtirib 0.7 ga teng ehtimollik bahosini olgan bo'lar edik.

Gipotezani tekshirish va p qiymat

Ko'pincha biz aholidan to'plangan namunalar asosida ba'zi gipotezalarni sinab ko'rishimiz kerak. Biz nolinchi gipotezadan boshlaymiz va o'tkazilgan statistik tekshiruv asosida nolinchi gipotezani qabul qilamiz yoki rad qilamiz.

Gipotezalarni tekshirishdan oldin, avval statistikaning asosiy asoslaridan birini, ya'ni **Markaziy Limit** teoremasini ko'rib chiqaylik.

Aytaylik, aholidan olingan n ta mustaqil va bir xil taqsimlangan

$$x_1, x_2, x_3, \ldots, x_n$$

kuzatuv namunalari bo'lsin; bunda o'rtacha qiymat μ va chekli tarqoqlik σ^2.

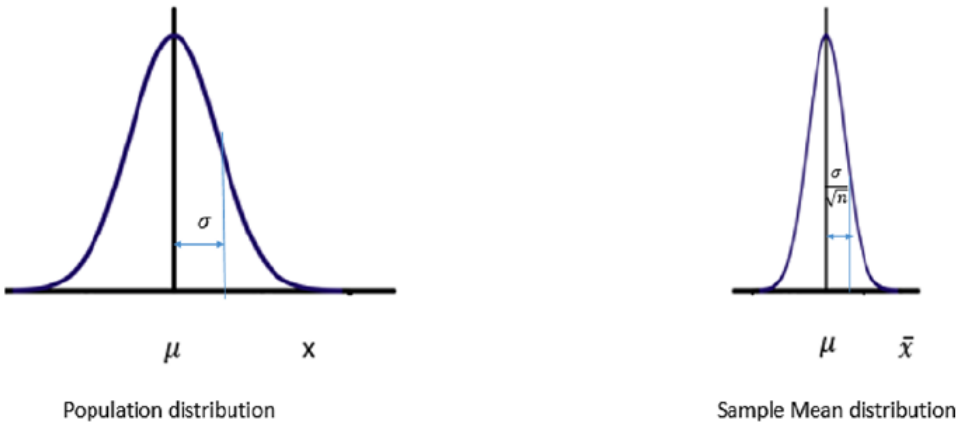
$\overline{x} = \frac{1}{n}(x_1 + x_2 + x_3 + \ldots + x_n)$ o'rtacha qiymatni qaraylik. U holda $\overline{x}$ ning taqsimoti Normal taqsimotga bo'ysinib, bu Normal taqsimotning o'rtacha qiymati μ va tarqoqligi $\frac{\sigma^2}{n}$ bo'ladi; ya'ni,

$$\overline{x} \sim Normal\left(\mu, \frac{\sigma^2}{n}\right).$$

Bu Markaziy Limit teorema deyiladi. Namuna o'lchami n kattalashgan sari, $\overline{x}$ ning tarqoqligi kamayib, $n \to \infty$ bo'lganda nolga intiladi.

2.27-rasmda aholi taqsimoti va namunaning o'rtacha qiymat taqsimoti ko'rsatilgan bo'lib, namunaning o'lchami n ta etib belgilangan.

Shuni yodda tutish kerakki, namuna o'rtacha qiymati, aholiga mos keluvchi o'zgaruvchilar normal taqsimlangan yoki yo'q – bundan qat'iy nazar,



Rasm 2.27: Aholi taqsimoti va namunaning o'rtacha qiymati bo'yicha taqsimot.

normal taqsimotga bo'ysinadi. Endi oddiy gipotezani tekshirish masalasini ko'rib chiqaylik.

10 yoshli o'g'il bolalarning o'rtacha vazni 85 funtga teng, standart og'ish 11.6 ga teng. Biror hududdagi o'g'il bolalarning semirib ketganligini o'rganaylik. Buni o'rganish uchun hududdagi o'g'il bolalar guruhidan 25 ta tasodifiy o'rtacha og'irlik yig'iladi. O'rtacha vazn 89.16 funtga teng bo'lsin.

Biz nolinchi gipotezani shakllantirishimiz kerak va agar, nolinchi gipotezaga qarshi dalillar yetarlicha kuchli bo'lsa, uni tekshiruv orqali rad etamiz.

Nolinchi gipotezani ko'rib chiqaylik, ya'ni H_0 : { hududdagi bolalar semiz emas: o'rtacha vazn qiymati $\mu = 85$ funt bo'lgan aholidan kelib chiqqan }.

H_0 nolinchi gipoteza ostidagi namunaviy o'rtacha qiymat quyidagicha bo'ladi:

$$\overline{x} \sim Normal\left(85, \frac{11.6^2}{25}\right).$$

Kuzatiladigan namuna o'rtacha vazn qiymati aholining o'rtacha vazn qiymatiga qancha yaqin bo'lsa, nolinchi gipotezaning haqiqati shunchalik yaxshi bo'ladi. Boshqa tomondan, tanlangan o'rtacha ko'rsatkich aholi o'rtacha darajasidan uzoqroq bo'lsa, nolinchi gipotezaga qarshi dalillar shunchalik kuchli bo'ladi. Normal o'zgaruvchining qiymati quyidagiga teng:

$$z = (\overline{x} - \mu)/(\sigma/\sqrt{n}) = (89.16 - 85)/(11.6/\sqrt{25}) = +1.75$$

Har bir gipoteza tekshiruvi uchun biz p ning qiymatini aniqlaymiz. Ushbu gipoteza tekshiruvining p qiymati bu tanlangan o'rtacha qiymatni kuzatish ehtimoli kuzatiladigan narsadan uzoqroq; ya'ni $P(\overline{x} \geq 89.16)$ yoki $P(z \geq$

1.75). Shunday qilib, p qiymat qanchalik kichik bo'lsa, dalil nolinchi gipotezaga qarshi shuncha kuchli bo'ladi.

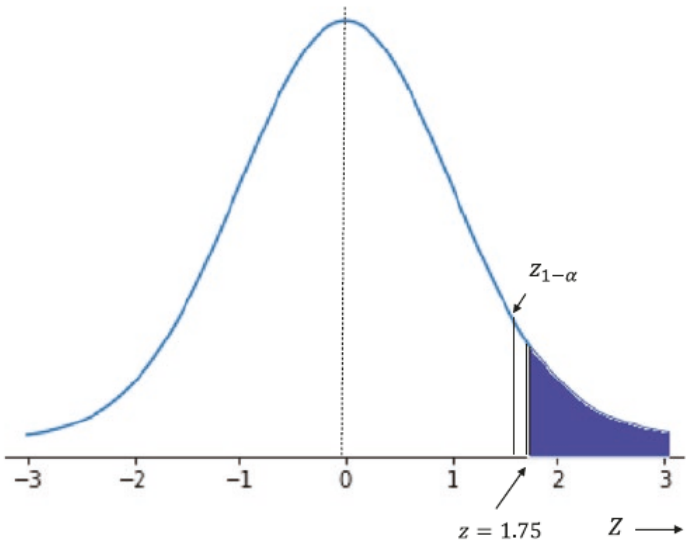
Agar p qiymat "1-tip xatolik" deb ataluvchi belgilangan α foizdan kam bo'lsa, nolinchi gipoteza rad etiladi.

Shuni yodda tutish kerakki, namuna o'rtacha qiymatining aholidan chetga chiqishi faqat tasodifiylik natijasi bo'lishi mumkin, chunki tanlab olingan o'rtacha qiymat chegarasi chekli tarqoqlik σ^2/n ga teng. α bizga nolinchi gipoteza to'g'ri bo'lsa ham nolinchi gipotezani rad etishimiz kerak bo'lgan chegarani beradi. Biz xato qilishimiz mumkin va tasodifiylik tufayli katta og'ish bo'lishi mumkin. Ammo bunday bo'lish ehtimolligi juda oz, ayniqsa namuna hajmi katta bo'lsa, namuna o'rtacha og'ish sezilarli darajada pasayadi.

Agar nolinchi gipoteza to'g'ri bo'lsa ham, nolinchi gipotezani rad qilsak, biz 1-tipdagi xatolikka yo'l qo'yamiz va shu sababli α bizga 1-tipdagi xatolik qilish ehtimolini beradi.

Ushbu sinov uchun p qiymat $P(Z \geq 1.75) = 0.04$.

Tanlash kerak bo'lgan 1-tipdagi xatolik, test o'tkaziladigan muayyan domen to'g'risidagi ma'lumotga bog'liq. Odatda, $\alpha = 0.05$ qiymat 1-tipdagi xatolik sozlamalari uchun yaxshi qiymat. Hisoblangan p qiymat sinov uchun belgilangan 1-tipdagi xatolik darajasidan past bo'lganligi sababli biz nolinchi gepotezani qabul qila olmaymiz. Sinov statistik ahamiyatga ega deb aytamiz. p qiymat 2.28-rasmda ko'rsatilgan.



Rasm 2.28: p qiymatni ko'rsatuvchi Z tekshiruv.

To'q rang bilan ajratilgan qism p qiymatga mos keladi; ya'ni, $P(z \geq 1.75)$. $z_{1-\alpha}$ nuqta z qiymatga to'g'ri keladi va bu nuqtadan o'ng tomonda,

agar nolinchi gipoteza to'g'ri bo'lsa, 1-tip xatolik bo'lishi mumkin. $z_{1-\alpha}$ dan yuqori bo'lgan qism, ya'ni, $P(z \geq z_{1-\alpha})$ bu 1-tip xatolik ehtimolligi uchun mos keladi. p qiymat tekshiruv uchun 1-tip xatolik ehtimolligidan kichik bo'lgani uchun, nolinchi gepotezani to'g'ri deb qabul qilib bo'lmaydi. Bunday Z tekshiruv, odatda yana bir yaxshi amaliyot - Ishonch oralig'i tekshiruvi bilan yakunlanadi.

Shuningdek, Z tekshiruv sifatida mashhur bo'lgan oldingi tekshiruv, berilgan aholi tarqoqligi bo'lmasa, har doim ham mumkin emas. Muayyan masalalar uchun aholi tarqoqligi bo'lmasligi mumkin. Bunday holatlarda Student-T tekshiruvi qulayroq, chunki u aholi tarqoqligi o'rniga namunaviy tarqoqliklarni ishlatadi.

O'quvchiga ushbu statistik tekshiruvlar haqida ko'proq ma'lumot olish tavsiya etiladi.

2.4 Sun'iy o'rganish algoritmi va Optimallashtirish

Modellashtirishning maqsadi turli xil optimallashtirish usullaridan foydalangan holda berilgan ma'lumotlar bazasi asosida model parametrlari uchun qiymat funksiyani minimallashtirishdir. Shunday ekan, o'z oldimizga ushbu savolni qo'yamiz: agar qiymat funksiyaning hosilasini yoki gradientini nolga tenglashtirsak, model parametrlariga ega bo'lamizmi? Bu har doim ham mumkin emas, chunki barcha yechimlar ham tugal formada bo'lavermaydi, tugal formada bo'lsa ham hisoblash uchun qimmat bo'lishi mumkin, yoinki yechim sifatida foydalanib bo'lmasligi ham mumkin. Bundan tashqari, ma'lumotlar hajmi katta bo'lganda, tugal formadagi yechimni topishda xotira cheklovlari paydo bo'ladi. Shunday qilib, iterativ usullar odatda optimallashtirishning murakkab masalalari uchun ishlatiladi.

Keng ma'noda, Sun'iy o'rganish ikki turga bo'linishi mumkin:

- Boshqaruvli Sun'iy o'rganish ("Supervised machine learning")
- Boshqaruvsiz Sun'iy o'rganish ("Unsupervised machine learning")

2.4.1 Boshqaruvli o'rganish

Boshqaruvli o'rganish jarayonida o'qitish ma'lumotlari ("training data") kiritma belgiga ega bo'ladi – odatda kiritma belgili vektor va unga tegishli yorliq, ya'ni "label". Model bir nechta parametrlarga ega bo'lib, ular kirish ("input") vektorini hisobga olgan holda chiqish ("output") yorlig'ini topishga

harakat qiladi. Model parametrlari prognozlash xatosiga asoslangan qiymat funksiyaning ba'zi formalarini optimallashtirish orqali olinadi; ya'ni, amaldagi yorliqlar va o'qitish ma'lumotlari asosida topilgan yorliqlar o'rtasidagi farqni optimallashtirish. Shu bilan bir qatorda, o'qitish ma'lumotlari ehtimolini maksimal darajaga ko'tarish ham model parametrlarini berishi mumkin.

Chiziqli regressiya Boshqaruvli o'rganish usuli sifatida

Bizda chiqish yorlig'i sifatida uylarning narxlari bo'yicha ma'lumotlar to'plami bo'lishi mumkin, shu bilan birga uyning maydoni, yotoq xonalari soni, vannaxonalar soni va boshqalar uning kiritma belgili vektori bo'lishi mumkin. U holda, kiritma belgili vektorga asoslangan holda uyning narxini taxmin qiluvchi funksiyani aniqlash mumkin.

Kiritma belgili vektor x' bilan ifodalansin va taxmin qilinayotgan qiymat esa y_p bo'lsin. Uy-joy narxining haqiqiy qiymati, ya'ni, chiqish yorlig'i y bilan belgilansin. Quyidagi tenglamada ko'rsatilganidek, chiqish yorlig'i kiritma belgili vektorining funksiyasi sifatida ifodalangan modelni aniqlay olamiz. Model, biz o'qitish jarayoni orqali o'rganishni istagan bir nechta doimiylar orqali parametrlangan.

$$y/x' = \theta'^T x' + b + \epsilon,$$

bu uerda ϵ – taxmin qilishdagi tafodifiy o'zgaruvchi va $\epsilon \sim Normal(0, \sigma^2)$.

Shunday qilib, berilgan kiritma uchun uy-joy narxi (y/x'), kiritma belgili vektor x', "bias" had b hamda tasodifiy komponent ϵ larning chiziqli kombinatsiyasidan iborat bo'ladi. O'z navbatida, ϵ o'rtacha qiymati nol va chekli tarqoqlik σ^2 bilan normal taqsimotga bo'ysinadi.

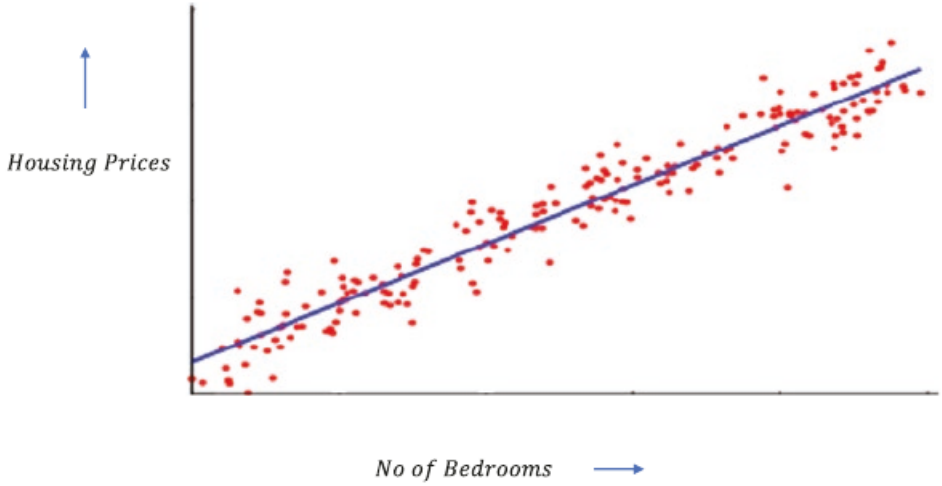
ϵ tasodifiy komponenta bo'lganligi sababli, uni oldindan aytib bo'lmaydi, va biz bashorat qilishimiz mumkin bo'lgan eng yaxshi yo'l - belgilari berilgan turar-joy narxlarining o'rtacha qiymati, ya'ni taxmin qilingan qiymati $y_p/x' = E[y/x'] = \theta'^T x' + b$. Bu yerda, θ' - chiziqli koeffitsient va b - og'ish yoki kesishish. Ushbu ikki kattaliklar θ' va b - biz o'rganmoqchi bo'lgan model parametrlari. Yuqoridagi ifodani $y_p = \theta^T x$ ko'rinishda yozish ham mumkin, bu yerda doimiy belgi 1 ga mos keladigan model parametriga "bias" qo'shilgan. Ushbu amal masala yechimini topishni soddalashtiradi.

Aytaylik, bizda m ta $(x(1), y(1)), (x(2), y(2)), \ldots (x(m), y(m))$ namunalar bor. Uy-joy narxlarining taxmin qilingan va haqiqiy qiymatlari o'rtasidagi farqning kvadratlari yig'indisini oladigan qiymat funksiyani hisoblashimiz mumkin va model parametrlarini olish uchun uni minimallashtirishga harakat qilamiz.

Qiymat funksiyasini quyidagicha aniqlash mumkin

$$C(\theta) = \sum_{i=1}^{m} \left(\theta^T x_i - y^{(i)}\right)^2.$$

Model parametrini aniqlash uchun θ ga nisbatan qiymatlar funksiyasini minimallashtirishimiz mumkin. Bu chiqish yorlig'i yoki maqsad uzluksiz bo'lgan chiziqli regressiya masalasidir. Qayd etish kerakki, Regressiya Boshqaruvli Sun'iy o'rganish sinfiga to'g'ri keladi. 2.29-rasmda uy-joy narxlari va yotoq xonalari soni o'rtasidagi bog'liqlik ko'rsatilgan.



Rasm 2.29: Regressiya uy-joy narxlariga va yotoq xonalari soniga mos keladi. Qizil nuqtalar ma'lumotlar nuqtalarini, ko'k chiziq esa o'rnatilgan regressiya chizig'ini bildiradi.

Aytaylik, kirish vektori $\mathbf{x}' = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}^T$ berilgan bo'lsin, bu yerda

$$\begin{aligned} x_1 &\rightarrow uyning\ maydoni \\ x_2 &\rightarrow xonalar\ soni \\ x_3 &\rightarrow vanna - xonalar\ soni \end{aligned}$$

Kiritma belgili vektoriga mos keladigan parametr vektori $\theta' = \begin{pmatrix} \theta_1 & \theta_2 & \theta_3 \end{pmatrix}^T$ bo'lsin, bunda

$$\begin{aligned} \theta_1 &\rightarrow uyning\ birlik\ maydoni\ uchun\ narx \\ \theta_2 &\rightarrow xonalar\ soni\ uchun\ narx \\ \theta_3 &\rightarrow vanna - xonalar\ soni\ uchun\ narx \end{aligned}$$

"Bias" hadni hisobga olgandan so'ng, kiritma belgili vektor

$$\mathbf{x} = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \end{pmatrix}^T$$

ko'rinishda bo'ladi, bu yerda

$$\begin{aligned} x_0 \to \text{"}bias\text{"}\ had ga\ mos\ keluvchi\ o'zgarmas\ 1\ qiymatli\ belgi \\ x_1 \to uyning\ maydoni \\ x_2 \to xonalar\ soni \\ x_3 \to vanna - xonalar\ soni \end{aligned}$$

va $\theta = \begin{pmatrix} \theta_0 & \theta_1 & \theta_2 & \theta_3 \end{pmatrix}$, bunda

$$\begin{aligned} \theta_0 \to \text{"}bias\text{"}\ had \\ \theta_1 \to uyning\ birlik\ maydoni\ uchun\ narx \\ \theta_2 \to xonalar\ soni\ uchun\ narx \\ \theta_3 \to vanna - xonalar\ soni\ uchun\ narx \end{aligned}$$

Endi biz regressiya muammosini va u bilan bog'liq qiymatlar funktsiyasini qanday qurish haqida bir oz tushunchaga ega bo'lganimiz sababli, muammoni soddalashtiramiz va model parametrlarini olishga kirishamiz.

Model parametr $\theta^* = \underbrace{ArgMin}_{\theta} C(\theta) = \underbrace{ArgMin}_{\theta} \sum_{i=1}^{m} \left(\theta^T x_i - y^{(i)}\right)^2$.

Barcha namunalar uchun kirish vektorlari matritsa X bilan birlashtirilishi va mos keluvchi maqsad chiqishi vektori $\mathbf{Y}$ sifatida ifodalanishi mumkin.

$$X = \begin{pmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & x_3^{(2)} \\ x_0^{(3)} & x_1^{(3)} & x_2^{(3)} & x_3^{(3)} \\ \vdots & \vdots & \vdots & \vdots \\ x_0^{(m)} & x_1^{(m)} & x_2^{(m)} & x_3^{(m)} \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

Agar bashorat qilish vektorini $\mathbf{Y}_p$ sifatida ifodalasak, u holda $\mathbf{Y}_p = X\theta$. Demak, bashorat qilish vektoridagi xatolik $\mathbf{e}$ quyidagicha ifodalanishi mumkin:

$$\mathbf{e} = X\theta - \mathbf{Y}.$$

Demak, $C(\theta)$ xatolik vektori $\mathbf{e}$ ning l^2 normasining kvadrati sifatida ifodalanishi mumkin, ya'ni

$$C(\theta) = \| \mathbf{e} \|_2^2 = \| X\theta - \mathbf{Y} \|_2^2 = (X\theta - \mathbf{Y})^T (X\theta - \mathbf{Y}).$$

Endi bizda matritsa shaklida soddalashtirilgan qiymat funksiyasi mavjud bo'lib, unda turli xil optimallashtirish usullarini qo'llash oson bo'ladi. Ushbu usullar ko'plab qiymat funksiyalari uchun, ular hoh konveks bo'lsin, hoh bo'lmasin, ishlatilishi mumkin. Konveks bo'lmagan qiymat funksiyalar uchun neyron tarmoqlarni ko'rib chiqishda batafsil muhokama qiladigan ba'zi qo'shimcha narsalar mavjud.

Biz gradientni hisoblash va uni nol vektorga tenglash orqali to'g'ridan-to'g'ri model parametrlarini olishimiz mumkin. Minimallik shartlari qoniqtirilishini tekshirish uchun siz oldin o'rgangan qoidalarni qo'llashingiz mumkin.

Parametr vektori θ ga nisbatan qiymatlar funksiyasining gradienti bu yerda ko'rinib turibdi:

$$\nabla C(\theta) = 2X^T(X\theta - \mathbf{Y}).$$

$\nabla C(\theta) = 0$ tenglikdan $X^T X\theta = X^T\mathbf{Y} \Rightarrow \widehat{\theta} = (X^T X)^{-1}X^T\mathbf{Y}$ ni olamiz.

Agar ushbu yechimni sinchkovlik bilan ko'rib chiqsak, X matritsaning psevdo-teskarisi, ya'ni $(X^T X)^{-1}X^T$, chiziqli regressiya masalasining yechimiga kelishini ko'rish mumkin. Buning sababi, chiziqli regressiya parametrlari vektoriga $X\theta = \mathbf{Y}$ tenglamaning yechimi sifatida qarash mumkin, bu yerda X – $m \times n$ bo'lgan to'rtburchak matritsa va $m > n$.

$\widehat{\theta}$ uchun oldingi ifoda model uchun yopiq shaklli yechimdir. Ushbu olingan $\widehat{\theta}$ ni yangi x_{new} nuqta uchun ishlatib, biz uyning narxini $\widehat{\theta}^T x_{new}$ deb taxmin qilishimiz mumkin.

$(X^T X)$ ning teskarisini hisoblash katta ma'lumotlar to'plamlari uchun ham xarajat, ham xotirani talab qiladi. Bundan tashqari, $X^T X$ matritsa singulyar bo'lgan va natijada uning teskari qiymati aniqlanmagan holatlar bo'lishi ham mumkin. Shuning uchun, minimal nuqtasiga erishish uchun muqobil usullarni ko'rib chiqishimiz kerak.

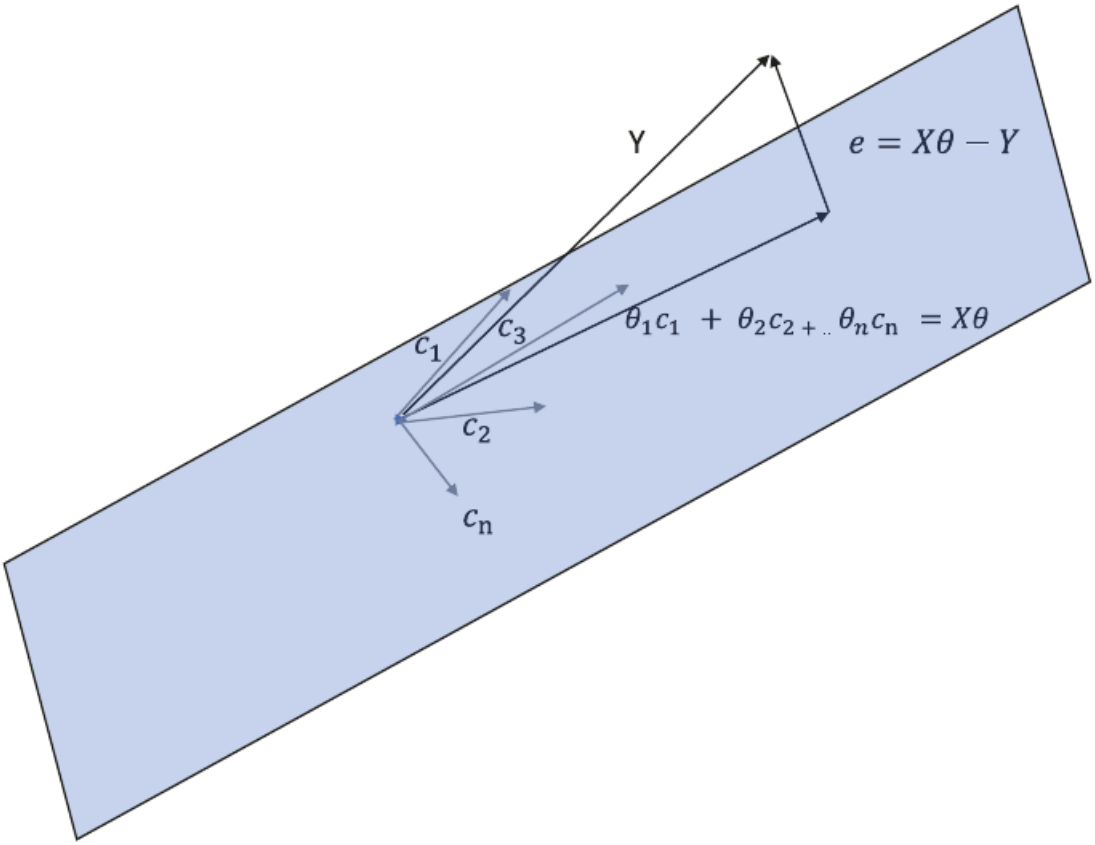
Chiziqli regressiya modelini yaratgandan keyin tekshirish kerak bo'lgan narsa - bu o'quv ma'lumotlari uchun chegirmaviy xatolik taqsimotidir. Xatoliklar, taqriban olganda, o'rtacha qiymat 0 bilan normal taqsimlanishi va ba'zi chekli tarqoqlik bilan bo'lishi kerak. Xatoliklar taqsimoti uchun nazariy kvantil qiymatlari bilan xatoliklar taqsimoti uchun haqiqiy kvantil qiymatlarini ko'rsatuvchi QQ grafik qoldiq uchun Gaussian bo'lishligini tekshirishi mumkin.

Vektor fazo yaqinlashuvi orqali chiziqli regressiya

Chiziqli regressiya masalasi yordamida vektor parametr θ ni shunday aniqlanadiki, $X\theta$ chiqish vektori $\mathbf{Y}$ ga imkon qadar yaqinlashtiriladi. $X \in \mathbb{R}^{m\times n}$ bu ma'lumotlar matritsasi va uni n ta ustunli vektorlar deb hisoblash mumkin

$c_i \in \mathbb{R}^{m\times 1}$, bu yerda ko'rsatilganidek, ketma-ket ketma-ket joylashtirilgan:

$$X = [c_1 c_2 \dots c_n].$$



Rasm 2.30: Chiziqli regressiya vektor fazo masalasi sifatida.

Ustun fazosining o'lchamlari m ta, ustun vektorlari soni esa n ta. Demak, ustun vektorlari m o'lchovli vektor fazosida n o'lchamli qism-fazoni egallashi mumkin. Ushbu qism-fazo 2.30-rasmda tasvirlangan. Bu 2 o'lchovli sirtga o'xshasa-da, biz uni n o'lchovli fazo sifatida tasavvur qilishimiz kerak. X ustunlarining parametrlar bilan chiziqli kombinatsiyasi bu yerda ko'rsatilgan prognozlash vektorini beradi:

$$X\theta = \theta_1 c_1 + \theta_2 c_2 + \dots \theta_n c_n.$$

$X\theta$ ko'paytma X ning ustun vektorlari chiziqli kombinatsiyasidan boshqa narsa emas, va $X\theta$ ustun vektorlar c_i $\forall i = \{1, 2, 3 \dots n\}$ bilan bir xil bo'lgan qism-fazoda joylashgan.

Endi haqiqiy qiymat vektori $\mathbf{Y}$ X ning ustun vektorlari bilan chegaralangan qism-fazo tashqarisida yotadi, shuning uchun biz X ni θ bilan qanday birlashtirmaylik, $X\theta$ hech qachon $\mathbf{Y}$ bilan bir yo'nalishda bo'lmaydi yoki unga teng bo'lmaydi. Bu degani, $\mathbf{e} = \mathbf{Y} - X\theta$ orqali berilgan nolmas xatolik vektori paydo bo'ladi.

Xato borligini bilganimizdan so'ng, xatoning l^2 normasini qanday kamaytirishni o'rganishimiz kerak. Xatolik vektorining l^2 normasi minimal bo'lishi uchun u $X\theta$ vektoriga perpendikulyar bo'lishi kerak. $\mathbf{e} = \mathbf{Y} - X\theta$ xatolik vektori $X\theta$ ga perpendikulyar bo'lganligi sababli, u ushbu qism-fazodagi barcha vektorlarga perpendikulyar bo'lishi kerak.

Shunday qilib, X ning barcha ustun vektorlari bilan $\mathbf{Y} - X\theta$ xatolik vektori "dot" ko'paytmasi nolga teng bo'lishi kerak, bu bizga quyidagilarni beradi:

$$c_1^T[\mathbf{Y} - X\theta] = 0,\ c_2^T[\mathbf{Y} - X\theta] = 0,\ \ldots c_n^T[\mathbf{Y} - X\theta] = 0$$

Buni quyidagi tarzda matritsa shaklida o'zgartirish mumkin:

$$[c_1 c_2 \ldots c_n]^T[\mathbf{Y} - X\theta] \Rightarrow X^T[\mathbf{Y} - X\theta] = 0 \Rightarrow \widehat{\theta} = (X^T X)^{-1} X^T \mathbf{Y}$$

Shuni ham yodda tutingki, xatolik vektori bashorat qilish vektoriga perpendikulyar bo'ladi, agar $X\theta$ X ustun vektorlaridan tashkil topgan qism-fazoda $\mathbf{Y}$ proektsiyasi bo'lsa. Ushbu illustratsiyaning yagona maqsadi Sun'iy o'rganish bilan bog'liq muammolarni yechishda vektor bo'shliqlarining ahamiyatini ta'kidlashdir.

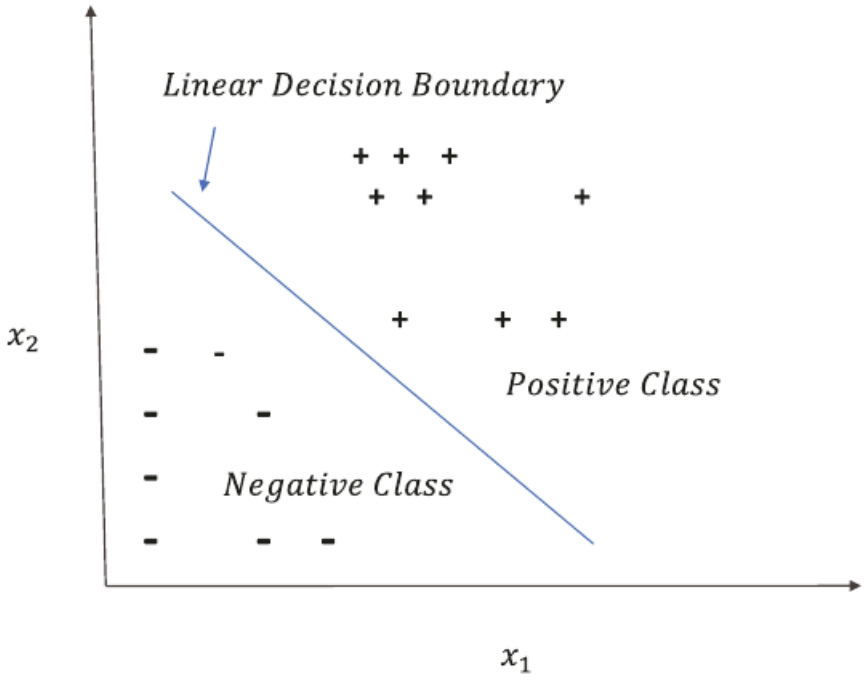
Klassifikatsiya

Huddi shunday, biz Klassifikatsiya, ya'ni tasniflash masalalariga qarashimiz mumkin, bunda uzluksiz o'zgaruvchining qiymatini bashorat qilish o'rniga kiritma belgili vektor bilan bog'liq klass yorlig'ini bashorat qilamiz. Masalan, to'lov tarixi va tranzaksiyalar tafsilotlari, shuningdek demografiya va bandlik to'g'risidagi ma'lumotlarga asoslanib mijozning to'lov qobiliyatini taxmin qilishga harakat qilishimiz mumkin. Bunday masalaga duch kelsak, biz kirish sifatida ko'rsatilgan funktsiyalar va mijoz sinf yorlig'i sifatida standart bo'lmaganligini ko'rsatadigan maqsadga ega bo'lgan ma'lumotlarga ega bo'lamiz. Ushbu yorliqli ma'lumotlarga asoslanib, biz mijozning to'lov bo'yicha majburiyatni bajarishini ko'rsatadigan sinf yorlig'ini bashorat qiladigan klassifikatorni qura olamiz, yoki mijozning to'lov qobiliyati ehtimolligi ballini taqdim eta olamiz. Ushbu senariyda klassifikator ikkita holni ajratadi: to'lay olishlik yoki to'lay olmaslik. Bunday klassifikatorni qurishda eng kichik kvadratlar metodi yaxshi qiymat funksiyasini bermasligi mumkin, chunki

biz yorliqni topishga harakat qilamiz va uzluksiz o'zgaruvchini bashorat qilmaymiz. Tasniflash muammolari uchun keng foydalaniladigan qiymat funksiyalari, odatda, Gini Entropiyasi va Shannon Entropiyasi kabi maksimal ehtimollik va entropiyaga asoslangan qiymat funksiyalariga asoslangan log-loss qiymat funksiyalari.

Chiziqli qaror chegaralariga ega bo'lgan klassifikatorlarga chiziqli klassifikator deyiladi. Qaror chegaralari - bu turli sinflarni ajratib turadigan gipertekislik yoki sirtlardir. Agar qaror chegaralari chiziqli bo'lsa, ajratuvchi tekislik gipertekislikdir.

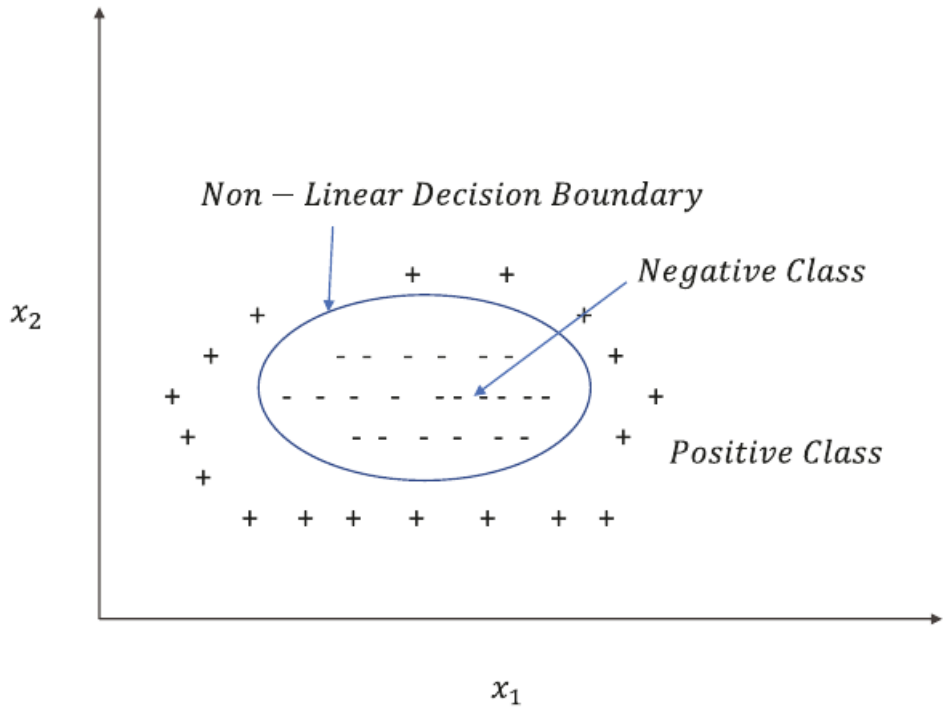
2.31 va 2.32-rasmlar mos ravishda ikki sinfni ajratish uchun chiziqli va chiziqli bo'lmagan qarorlar chegaralarini ko'rsatadi.



Rasm 2.31: Chiziqli qaror chegarasi bo'yicha tasniflash.

keling, eng ommabop va sodda klassifikatorlardan biri, logistik regressiya haqida qisqacha to'xtalib o'taylik.

Aytaylik $(x^{(i)}, y^{(i)})$ belgilangan ma'lumot nuqtalari, bunda $x^{(i)} \in \mathbb{R}^{n\times 1}$ $\forall i \in \{1, 2, \ldots, m\}$ - doimiy vektorni o'z ichiga olgan va o'zgarmas belgi qiymati 1 bo'lgan kirish vektori, va $y^{(i)}$ sinfni belgilaydi. Agar ma'lumotlar punkti qarzni uzgan mijozga mos kelsa, $y^{(i)}$ qiymati 1 ga tenglanadi, agar qarzni to'lamagan bo'lsa 0 ga. Kiritma vektor $x^{(i)}$ komponentalari bo'yicha



Rasm 2.32: No-chiziqli qaror chegarasi bo'yicha tasniflash.

quyidagicha ifodalanishi mumkin:

$$x^{(i)} = \begin{pmatrix} 1x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \end{pmatrix}.$$

Umuman olganda, logistik regressiya chiziqli klassifikator bo'lib, chiziqli baholashni ehtimollikka aylantirish uchun siqish funksiyasidan foydalanadi. Keling, $\theta = \begin{pmatrix} \theta_0 & \theta_1 & \theta_2 & \dots & \theta_n \end{pmatrix}$ model parametr bo'lsin, va har bir θ_j $\forall j \in \{0, 1, 2, \dots, n\}$ mos ravishda $\mathbf{x}$ kiritma vektorning j-chi belgisi x_j uchun model parametri komponentasi bo'lsin.

θ_0 had "bias"dir. Chiziqli regressiyada, "bias" faqat chiqish y o'qidagi kesishishdan boshqa narsa emas. Ushbu "bias" hadi logistik regressiya (va boshqa chiziqli klassifikatorlar) uchun nimani anglatishini qisqacha ko'rib chiqamiz.

"Dot" ko'paytma $\theta^T x$, berilgan data-nuqta musbat sinf yoki manfiy sinf bo'lishini aniqlaydi. Biz qarayotgan masalada, musbat klass mijozning kreditni to'lash bo'yicha majburiyatlarini bajarganligi va manfiy klass - bu mijoz majburiyatni bajarmaganligini bildiradi. Kiritma ma'lumotlar va modelni hisobga olgan holda, mijozning qarzni to'lashlik ehtimoli quyidagicha

aniqlanadi:

$$P(y = 1/x, \theta) = \frac{1}{1 + \exp(-\theta^T x)} = p$$

$$P(y = 0/x, \theta) = 1 - \frac{1}{1 + \exp(-\theta^T x)} = \frac{\exp(-\theta^T x)}{1 + \exp(-\theta^T x)} = q$$

Endi $\theta^T x$ ning turli qiymatlari uchun ehtimollik qiymatlarini ko'rib chiqaylik:

- $\theta^T x = 0$ bo'lsa, musbat klass ehtimoli 1/2 ga teng.
- Agar $\theta^T x > 0$ bo'lsa, musbat klassning ehtimolligi 1/2 dan katta va 1 dan kam bo'ladi.
- Agar $\theta^T x < 0$ bo'lsa, musbat klassning ehtimolligi 1/2 dan kam va 0 dan katta.
- Agar $\theta^T x$ yetarlicha katta va musbat bo'lsa, ya'ni $\theta^T x \to \infty$, ehtimollik$\to$ 1.
- Agar $\theta^T x > 0$ yetarlicha katta va manfiy bo'lsa, ya'ni $\theta^T x \to -\infty$, ehtimollik$\to$ 0.

Ushbu ehtimollik formulasining yaxshi tomoni shundaki, u 0 va 1 o'rtasidagi qiymatlarni ushlab turadi, bu esa chiziqli regressiya bilan mumkin emas edi. Bundan tashqari, u mavjud klass o'rniga doimiy ehtimollikni beradi. Shunday qilib, mavjud masalaga qarab, klassni aniqlash uchun kesish ehtimoli chegarasi aniqlanishi mumkin.

Ushbu ehtimollik modeli funksiyasi logistik yoki sigmoid funksiya deb ataladi. U modelni o'qishni matematik jihatdan qulaylashtiradigan silliq, uzluksiz gradientga ega.

Agar sinchkovlik bilan qarasak, har bir o'quv namunasi bo'yicha mijozlar sinfi biz ilgari muhokama qilgan Bernoulli taqsimotiga mos kelishini ko'ramiz. Har bir ma'lumotlar nuqtasi uchun $y^{(i)}/x^{(i)} \sim Bernoulli(1, p_i)$. Bernulli taqsimotining ehtimollik massasi funksiyasiga asoslanib, quyidagilarni aytishimiz mumkin:

$$P(y^{(i)}/x^{(i)}, \theta) = (1 - p_i)^{1-y^{(i)}},$$

bu erda $p_i = \frac{1}{1+\exp(-\theta^T x)}$.

Endi qiymat funksiyasini qanday aniqlaymiz? Berilgan ma'lumotlar parametrlari model parametrlarini hisoblab chiqamiz va keyin hisoblangan ehtimollikni maksimal darajada oshiradigan model parametrlarini aniqlaymiz. Biz belgilaymiz L ning ehtimolligi va u quyidagicha ifodalanishi mumkin

$$L = P(Ma'lumot/model) = P(D^{(1)}D^{(2)}\dots D^{(m)}/\theta),$$

bu erda $D^{(i)}$ i-chi o'quv namunasi $(x^{(i)}, y^{(i)})$ ni ifodalaydi.

O'quv namunalari mustaqil, ya'ni bir-biriga bog'liq emas deb hisoblasak, ushbu modelni hisobga olgan holda L ni quyidagicha yozish mumkin:

$$\begin{aligned} L &= P(D^{(1)}D^{(2)}\dots D^{(m)}/\theta) \\ &= P(D^{(1)}/\theta)P(D^{(2)}/\theta)\dots P(D^{(m)}/\theta) \\ &= \prod_{i=1}^{m} P(D^{(i)}/\theta). \end{aligned}$$

Tenglikning har ikki tomonini logarifmlab, ehtimolliklar ko'paytmasidan ehtimolliklar logarifmining yig'indisiga o'tamiz. Shuningdek, logarifmlash natijasida optimallashtirish masalasi o'zgarmas saqlanadi: L va $\log L$ uchun maximum nuqta bir xil bo'ladi, chunki log monoton ravishda oshib boruvchi funksiyadir.

Shunday ekan, tenglikning ikkala tomonini logarifmlab, biz quyidagilarni olamiz:

$$\log L = \sum_{i=1}^{m} \log P(D^{(i)}/\theta).$$

Endi esa, quyidagi ifodani yozish mumkin:

$$P(D^{(i)}/\theta) = P((x^{(i)}, y^{(i)})/\theta) = P(x^{(i)}/\theta)P(y^{(i)}/x^{(i)}, \theta).$$

Biz bu yerda "data point" ehtimolligi bilan shug'ullanmaymiz – ya'ni, $P(x^{(i)}/\theta)$ va shunday deb hisoblaymizki, o'qitishdagi barcha "data point"lar ushbu model uchun teng ehtimollikka ega. Shunday qilib, $P(D^{(i)}/\theta) = kP(y^{(i)}/x^{(i)}, \theta)$, bu yerda k doimiy.

Ikkala tomonni yana bir bor logarifmlab, biz quyidagilarni olamiz:

$$\log P(D^{(i)}/\theta) = \log k + y^{(i)} \log p_i + (1 - y^{(i)}) \log(1 - p_i).$$

Barcha ma'lumotlar to'plamlarini yig'ib, biz quyidagilarga ega bo'lamiz:

$$\log L = \sum_{i=1}^{m} \log k + y^{(i)} \log p_i + (1 - y^{(i)}) \log(1 - p_i).$$

Model parametri θ ni olish uchun $\log L$ ni maksimallashtirishimiz kerak. $\log L$ ni maksimallashtirish $-\log L$ ni minimallashtirish bilan bir xil, shuning uchun biz $-\log L$ ni logistik regressiya uchun qiymat funksiyasi sifatida olishimiz va uni minimallashtirishimiz mumkin. Bundan tashqari, $\log k$ summani tashlab yuborish mumkin, chunki bu doimiy va minimaldagi model parametr $\log k$ summa bor yoki yo'qligidan qat'iy nazar bir xil bo'ladi. Agar biz qiymat funksiyasini $C(\theta)$ sifatida tasvirlasak, uni quyidagi ko'rinishda ifodalash mumkin:

$$C(\theta) = \sum_{i=1}^{m} -y^{(i)} \log p_i - (1 - y^{(i)}) \log(1 - p_i),$$

bu erda $p_i = 1/\left(1 + \exp(-\theta^T x)\right)$.

$C(\theta)$ - bu θ bo'yicha konveks funksiya, va o'quvchiga buni "Hisoblash" bo'limida ilgari o'rganilgan qoidalar bilan tekshirish tavsiya etiladi. $C(\theta)$ ni optimallashtirishning umumiy usullari yordamida minimallashtirish mumkin.

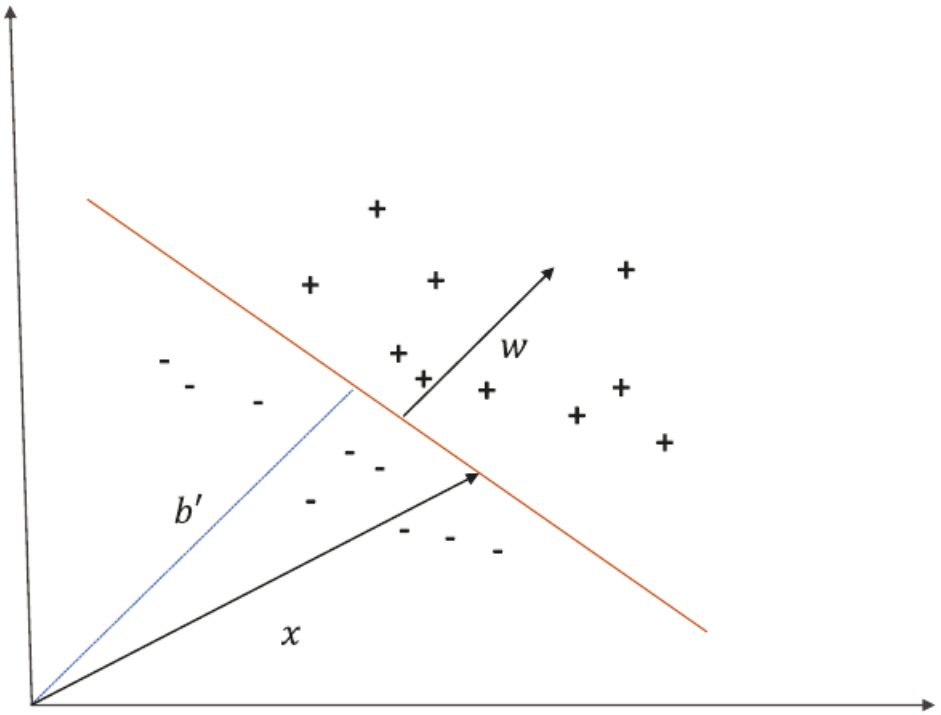
Gipertekisliklar va Chiziqli Klassifikatorlar

Chiziqli klassifikatorlar u yoki bu yo'l bilan gipertekislikka bog'liq, shuning uchun bu munosabatlarni o'rganish mantiqan to'g'ri hisoblanadi. Shu ma'noda, chiziqli klassifikatorni o'rganish bu musbat klassni manfiy klassdan ajratib turadigan gipertekislik haqida bilishdir.

n o'lchovli vektor fazosidagi gipertekislik bu $(n-1)$ o'lchovli tekislik bo'lib, bu tekislik n o'lchovli vektor fazosini ikki sohaga ajratadi. Bu sohalardan birinchisi gipertekislik ustida yotgan vektorlardan tashkil topgan bo'lsa, ikkinchisi esa gipertekislikning pastki qismida joylashgan vektorlardan iborat bo'ladi. Ikki o'lchovli vektor fazosi uchun to'g'ri chiziqlar gipertekislik vazifasini bajaradi. Huddi shunday, uch o'lchovli vektor fazosi uchun ikki o'lchovli tekislik gipertekislik sifatida qaralishi mumkin.

Har qanday gipertekislik ikkita asosiy parametr bilan aniqlanadi: (i) koordinata boshidan gipertekislikkacha bo'lgan perpendikulyar masofa, bu masofa b' "bias" had orqali tasvirlanadi; (ii) gipertekislikning orintatsiyasi (yo'nalishi), bu parametr esa gipertekislik yuzasiga o'tkazilgan perpendikulyar birlik vektor $\mathbf{w}$ bilan aniqlanadi, 2.33-rasmga qarang.

Biror $\mathbf{x} \in \mathbb{R}^{n\times 1}$ vektor gipertekislikda yotishi uchun, ushbu vektorning $\mathbf{w}$ tomonga yo'naltirilgan proeksiyasi koordinata boshidan gipertekislikkacha bo'lgan perpendikulyar masofaga teng bo'lishi kerak, ya'ni $\mathbf{w}^T\ \mathbf{x} = b'$. Shunday qilib, giperplet ustida yotgan har qanday nuqta $\mathbf{w}^T\ \mathbf{x} - b' = 0$ tenglikni qanoatlantirishi kerak. Shunga o'xshab, $\mathbf{w}^T\ \mathbf{x} - b' > 0$ shartni qanoatlantiruvchi nuqtalar gipertekislikdan yuqorida va $\mathbf{w}^T\ \mathbf{x} - b' < 0$ shartni qanoatlantiruvchi nuqtalar gipertekislikdan pastda yotadi.



Rasm 2.33: Ikki klassni ajratib turuvchi gipertekisliklar.

Chiziqli klassifikatorlarda biz gipertekislikni qanday modellashtirish kerakligini o'rganamiz. Boshqacha aytganda, model parametrlar $\mathbf{w}$ va b ni o'rganamiz. $\mathbf{w}$ vektor odatda musbat klassga to'g'ri keladi. Perceptron va chiziqli "Support Vector Machine" (SVM) chiziqli klassifikatorlar sirasiga kiradi. Albatta, SVM va Perceptronning gipertekislikni o'rganishdagi usullari mutlaqo boshqacha va shuning uchun ular turli xil gipertekisliklarni taqdim etadi, garchi bir xil o'quv ma'lumotlarini ishlatishsa ham.

Biz logistik regressiyani qarayotgan bo'lsak ham, uning chiziqli qaror chegarasiga asoslanganligini ko'ramiz. Chiziqli qaror chegarasi - bu gipertekislikdan boshqa narsa emas va gipertekislikda joylashgan nuqtalar (ya'ni, $\mathbf{w}^T\ \mathbf{x} - b' = 0$) har ikkala klass uchun ham ehtimolligi 0.5 ga tengdir. Shuningdek, logistika regressiyasining qarorlar chegarasini o'rganishi SVM va Perceptron modellarinikidan mutlaqo farq qiladi.

2.4.2 Boshqaruvsiz o'rganish

Boshqaruvsiz sun'iy o'rganish algoritmlari ma'lumot to'plamlarida yorliqsiz yoki mo'ljalsiz kiritma ma'lumot nuqtalarini o'z ichiga olgan ma'lumotlarning

tashqi yoki ichki tuzilishini aniqlashga qaratilgan. K-ma'noli klasterlash, aralash Gaussianlar va boshqalar bu boshqarilmaydigan o'rganish usullaridir. Prinsipial Komponent Tahlili ("Principal Component Analysis", yoki PCA), Yakka Qiymat Dekompozitsiyasi ("Singular Value Decomposition", yoki SVD), avto-kodlash va shu kabi ma'lumotlarni qisqartirish texnikalari ham boshqarilmaydigan o'rganish usullari tarkibiga kiradi.

2.4.3 Sun'iy o'rganishda optimallash usullari

Gradient tushish

Gradient tushish, o'zining bir nechta variantlari bilan birga, ehtimol Sun'iy o'rganish va chuqur o'rganishda eng ko'p ishlatiladigan optimallashtirish texnikasi bo'lsa kerak. Bu tasodifiy model parametri bilan boshlanadigan va model parametrini yangilash yo'nalishini aniqlash uchun model parametriga nisbatan qiymat funksiyasining gradientidan foydalanadigan iterativ usuldir.

Aytaylik, bizda $C(\theta)$ qiymat funksiyasi mavjud, bu yerda θ model parametrlarini ifodalaydi. Bilamizki, qiymat funksiyasining θ ga nisbatan gradienti $C(\theta)$ ning maksimumga tomon o'sish yo'nalishini beradi. Shunday qilib, $C(\theta)$ ning maksimal kamayishga tomon yo'nalishini chiziqli ma'noda olish uchun gradientning manfiy qiymatini ishlatish kerak.

Model parametr θ ni $(t+1)$ iteratsiya vaqtida yangilash qoidasi quyidagicha berilgan:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla C(\theta^{(t)}).$$

Bu yerda η o'rganish tezligini va $\theta^{(t+1)}$ hamda $\theta^{(t)}$ mos ravishda $(t+1)$ va t iteratsiya vaqtlaridagi parametr vektorlarini bildiradi.

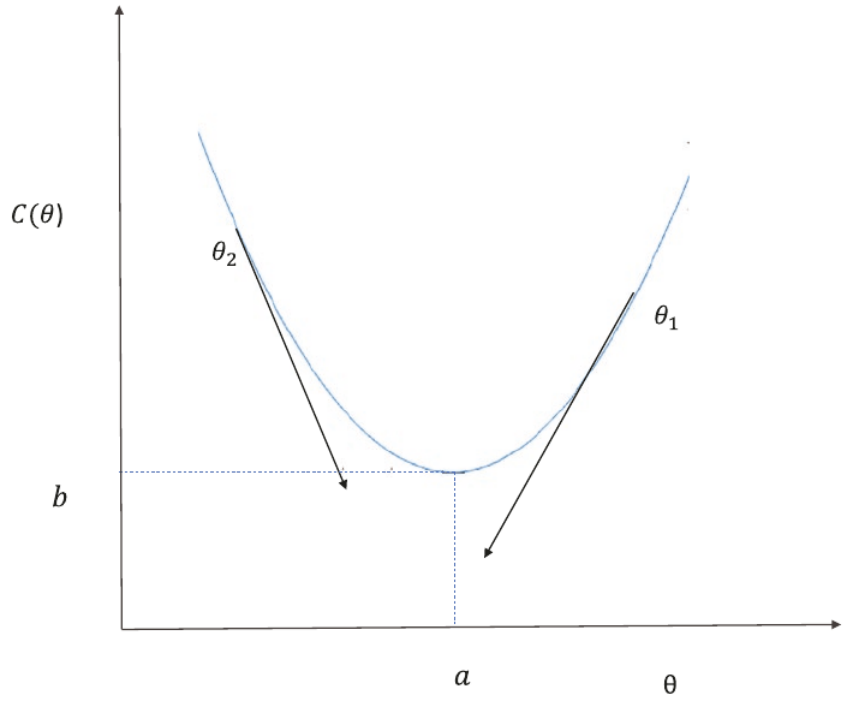
Ushbu takroriy jarayon orqali minimumga erishilsa, minimumdagi qiymat funksiyasining gradienti texnik jihatdan nolga teng bo'ladi va shuning uchun θ yangilanmasa ham bo'ladi. Biroq, hisoblashlardagi yaxlitlash xatoliklari va boshqa kompyuter cheklovlari sababli haqiqiy minimumga erishish mumkin bo'lmasligi mumkin. Shunday ekan, iterativ jarayonni to'xtatish uchun minimumni topganimizga ishonganimizda yoki hech bo'lmaganda minimumga yetarlicha yaqin bo'lganida, iteratsiya jarayonini to'xtatish uchun boshqa mantiqiy amalni o'ylab topishga to'g'ri keladi. Odatda foydalaniladigan usullardan biri bu gradient vektorning kattaligini tekshirish va agar u oldindan belgilangan juda kichik ε qiymatdan kichik bo'lsa, iterativ jarayonni to'xtatishdir. Qo'llanilishi mumkin bo'lgan yana bir usul - bu 1000 ga o'xshash qat'iy belgilangan iteratsiyalardan so'ng parametrlarni yangilashning iterativ jarayonini to'xtatish.

O'rganish tezligi gradientning minimum nuqtaga yaqinlashishida juda muhim rol o'ynaydi. Bir tomondan, agar o'rganish tezligi katta bo'lsa, yaqin-

lashuv tezroq bo'lar, ammo minimum nuqtasi atrofida kuchli tebranishlarga olib kelishi mumkin. Ikkinchi tomondan, agar o'rganish tezligi juda kichik bo'lsa, minimal nuqtaga erishish ko'proq vaqtni talab qiladi, biroq yaqinlashish odatda tebranishlarsiz kechadi.

Gradient tushish qanday ishlashini bilish uchun, keling bitta parametr va qiymat funksiyasi $C(\theta) = (\theta - a)^2 + b$ bo'lgan modelni olaylik va bu usul nima uchun ishlashini bilib olaylik.

2.34-rasmdan ko'rinib turibdiki, θ_1 nuqtada gradient (ya'ni, θ bo'yicha hosila) musbat bo'ladi va shuning uchun gradient yo'nalishi bo'yicha harakat qilsak qiymat funksiyasi o'sadi. Buning o'rniga, agar θ_1 dan boshlab gradientning manfiy tomoni bo'ylab harakat qilsak, qiymat funksiya kamayadi. Endi, gradient manfiy bo'lgan boshqa bir θ_2 nuqtani olaylik. Agar bu yerda θ ni yangilash uchun gradient yo'nalishini olsak, qiymat funktsiyasi oshadi. Gradientning manfiy yo'nalishini olish bizning minimum tomon harakatlanishimizni ta'minlaydi. Minimumga $\theta = a$ nuqtada yetganimizda, gradient 0 ga teng va shuning uchun θ ustida yangilanish bo'lmaydi.



Rasm 2.34: Sodda, bir o'zgaruvchili qiymat funksiyasi uchun gradient tushish tasviri.

Ko'p o'zgaruvchili qiymat funksiyasi uchun gradient tushish

Biz bitta parametrga bog'liq bo'lgan qiymat funksiyasining gradient tushishi haqida bir oz tasavvurga ega bo'ldik. Endi esa qiymat funksiyasi bir nechta parametrlarga bog'liq bo'lgan hol uchun gradient tushishni qarab chiqamiz.

Keling ko'p o'zgaruvchili funksiyaning Teylor qatoriga yoyilmasini ko'rib chiqaylik. Buning uchun ko'p o'zgaruvchilarni θ vektor orqali ifodalaymiz va $C(\theta)$ qiymat funksiyasini ko'rib chiqamiz, bu yerda $\theta \in \mathbb{R}^{n\times 1}$.

Avvalroq muhokama qilinganidek, θ nuqta atrofidagi Taylor qatori yoyilmasi quyida ko'rsatilganidek, matritsali yozuv bilan ifodalanishi mumkin:

$$C(\theta + \Delta\theta) = C(\theta) + \Delta\theta^T \nabla C(\theta) + \frac{1}{2}\Delta\theta^T H(\theta)\Delta\theta + \ldots,$$

bu yerda

$$\Delta\theta \to \theta \; vektor \; o'zgarish$$

$$\nabla C(\theta) \to C(\theta) ning \; gradiyent \; vektori$$

$$H(\theta) \to C(\theta) ning \; Hesse \; matritsasi$$

Faraz qilaylikki, gradient tushishning t iteratsion vaqtida model parametr θ bo'lsin, va biz θ ni $\theta + \Delta\theta$ ga yangilamoqchimiz. Bunda $C(\theta + \Delta\theta)$ ning qiymati $C(\theta)$ dan kam bo'lishi kerak.

Agar funksiyani θ ning atrofida chiziqli deb hisoblasak, Teylor qatori yoyilmasi quyidagicha olinadi:

$$C(\theta + \Delta\theta) = C(\theta) + \Delta\theta^T \nabla C(\theta)$$

Biz $\Delta\theta$ ni shunday tanlaylikki, $C(\theta + \Delta\theta)$ ning qiymati $C(\theta)$ dan kichik bo'lsin.

Bir xil qiymatdagi barcha $\Delta\theta$ lardan biri $\Delta\theta^T \nabla C(\theta)$ "dot" ko'paytmani maksimal qilishi uchun, u gradient vektor $\nabla C(\theta)$ yo'nalishga ega bo'lishi kerak. Ammo, bu bizga $\Delta\theta^T \nabla C(\theta)$ ning maksimumini olishga imkon beradi. Demak, $\Delta\theta^T \nabla C(\theta)$ "dot" ko'paytmaning minimal qiymatini olish uchun, $\Delta\theta$ ning yo'nalishi $\nabla C(\theta)$ ga aynan qarama-qarshi kelishi kerak. Boshqacha aytganda, $\Delta\theta$ vektor manfiy ishorali gradient vektor $\nabla C(\theta)$ ga proporsional bo'lishi kerak:

$$\Delta\theta \propto -\nabla C(\theta)$$

$\Rightarrow \Delta\theta = -\eta \nabla C(\theta)$, bu yerda η - o'rganish tezligi

$$\Rightarrow \theta + \Delta\theta = \theta - \eta \nabla C(\theta)$$

$\Rightarrow \theta^{(t+1)} = \theta^{(t)} - \eta \nabla C(\theta^{(t)})$ bu esa gradient tushish uchun mashhur tenglama. Gradient tushish minimalgacha qanday borishini tasavvur qilish uchun biz kontur grafiklari va kontur chiziqlari haqida bazaviy tushunchalarga ega bo'lishimiz kerak.

Kontur grafigi va kontur chiziqlari

Keling, $C(\theta)$ funksiyani ko'rib chiqamiz, bunda $\theta \in \mathbb{R}^{n\times 1}$. Kontur chizig'i bu $C(\theta)$ funksiyasining bir xil qiymatga ega bo'lgan nuqtalarini tutashtirib turuvchi θ vektor fazosidagi *chiziq/egri chiziq*. $C(\theta)$ ning har bir yagona qiymati uchun biz alohida kontur chiziqlariga ega bo'lamiz.

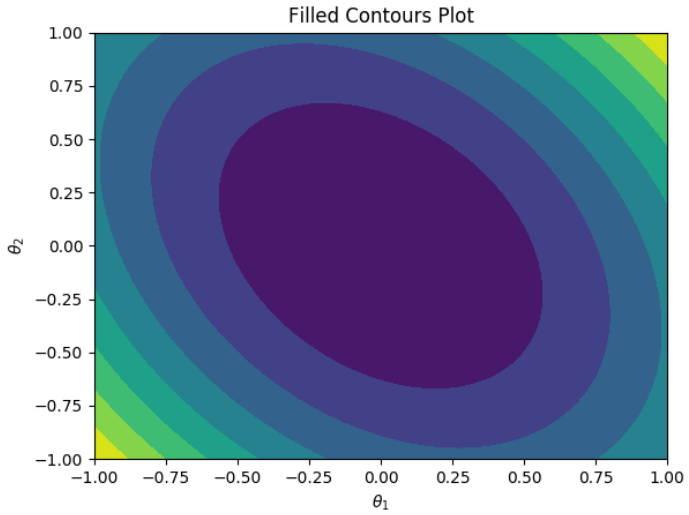
Misol sifatida $C(\theta) = \theta^T A\theta$ funksiya uchun kontur chiziqlarini chizamiz, bu yerda $\theta = \begin{pmatrix}\theta_1 & \theta_2\end{pmatrix}^T \in \mathbb{R}^{2\times 1}$ va

$$A = \begin{pmatrix}7 & 2\\ 2 & 5\end{pmatrix}$$

$C(\theta)$ uchun yozilgan ifodani yoyib, quyidagi ko'rinishga keltiramiz:

$$C(\theta_1, \theta_2) = 7\theta_1^2 + 5\theta_2^2 + 4\theta_1\theta_2.$$

Qiymat funksiyasi uchun kontur grafigi 2.35-rasmda tasvirlangan. Ushbu grafikni olish uchun Python-da yozilgan dastur esa 2.26-listda berilgan.



Rasm 2.35: Kontur grafigi.

```
# Masalaning qo'yilishi: Python-da ikki o'zgaruvchili qiymat funksiya kontur grafigini chizing
# Yechim: Matplotlib-ning contourf() funksiyasidan foydalib, to'ldirilgan kontur chiziqlarini ko'rsatamiz.
#

# Kutubxonani yuklash
import numpy as np
import matplotlib.pyplot as plt

#Ikki o'zgaruvchili qiymat funksiya C(x, y) ning berilishi:
def C(x, y):
return 7.* x**2 + 5.* y**2 + 4.* x * y

# O'zgaruvchilarni o'rnatish:
theta_1 = np.linspace(-1.0, 1.0, 100)
theta_2 = np.linspace(-1.0, 1.0, 100)

#theta_1 va theta_2 qiymatlar grafikdagi vaziyatni va C ning qiymatlari esa kontur darajalarini ifodalaydi. Bunday ma'lumotlarni tayyorlashning eng oddiy usuli bu bir o'lchovli massivlardan ikki o'lchovli panjara quradigan "np.meshgrid" funktsiyasidir:
theta_1, theta_2 = np.meshgrid(theta_1, theta_2)

C = C(theta_1, theta_2)

fig,ax=plt.subplots(1,1)
cp = ax.contourf(theta_1, theta_2, C)


ax.set_title('Filled Contours Plot')
ax.set_xlabel('$\\theta_1$')
ax.set_ylabel('$\\theta_2$')
plt.show()

#-----------------------------------------------------------
# Izoh: Matplotlib API contains contour() and contourf() functions that draw contour lines and filled contours, respectively. Both functions need three parameters theta_1, theta_2 and C.
#-----------------------------------------------------------
```

Listing 2.26: Python-da ikki o'zgaruvchili funksiya kontur grafigi

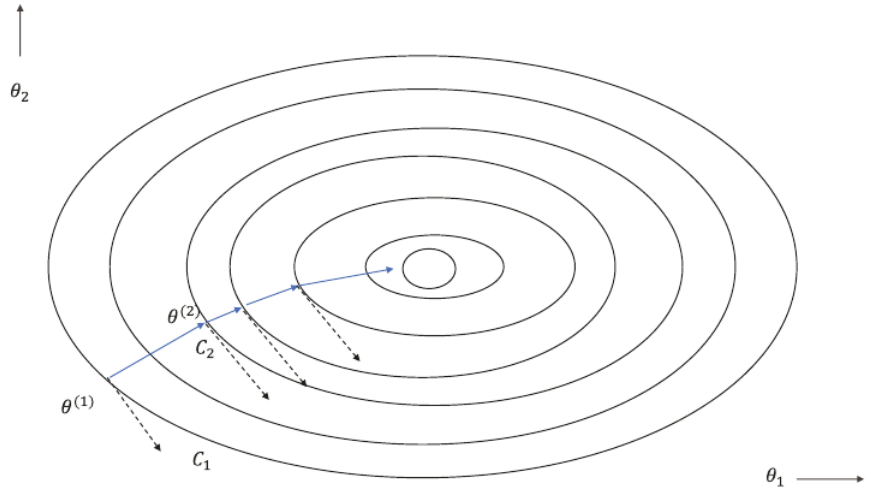
Qayd etish lozimki, 2.35-rasmdagi har bir ellips chiziqlar bu $C(\theta_1, \theta_2)$ funksiyaning aynan biror qiymatiga mos keluvchi kontur chiziqlaridir. Haqiqatan ham, agar $C(\theta_1, \theta_2) = a$ deb olinsa, bu yerda a doimiy, quyidagi ifoda ellips

tenglamasini anglatadi:

$$a = 7\theta_1^2 + 5\theta_2^2 + 4\theta_1\theta_2.$$

a ning turli xil doimiy qiymatlari uchun biz 2.35-rasmda ko'rsatilganidek, turli xil ellipslarni olamiz. Muayyan kontur chizig'idagi barcha nuqtalar majmuasi funksiyaning biror o'zgarmas qiymatiga mos keladi.

Endi, kontur chizig'i nima ekanligini bilgan holda, $C(\theta)$ qiymat funksiyasi uchun kontur chizig'idadagi gradient tushish progressiyasini ko'rib chiqaylik, bu yerda $\theta \in \mathbb{R}^{2\times 1}$. Gradient tushish qadamlari 2.36-rasmda ko'rsatilgan.



Rasm 2.36: Ikki o'zgaruvchili qiymat funksiyasi uchun gradient tushish.

Keling, C_1 qiymatiga mos keladigan eng katta ellipsni tanlab olaylik va θ ning komponentalari C_1 bo'ylab $\theta^{(1)}$ nuqtada joylashgan.

$C(\theta)$ ni θ atrofida chiziqli ekanligini hisobga olsak, $C(\theta)$ qiymat funksiyasining o'zgarishini quyidagicha ko'rsatish mumkin:

$$\Delta C(\theta) = \Delta\theta^T \nabla C(\theta).$$

Agar bitta kontur chizig'ida bir-biriga juda yaqin bo'lgan ikkita nuqta o'rtasida qiymatning kichik o'zgarishini olsak, u holda $\Delta C(\theta) = 0$, chunki bir xil kontur chizig'idagi barcha nuqtalar aynan bir xil qiymatga ega. Yana bir muhim jihat: bitta kontur chizig'ida bir-biriga juda yaqin bo'lgan ikkita nuqtani olsak, $\delta\theta$ kontur chizig'iga tangensial strelkalar bilan ko'rsatilgan urinma ekanligini bildiradi.

$\Delta C(\theta) = 0 \rightarrow \Delta\theta^T \nabla C(\theta) = 0$, bu gradient C_1 kontur chizig'i ustidagi $\theta^{(1)}$ nuqtaga o'tkazilgan urinmaga perpendikulyar ekanligini anglatadi. Gradient tashqariga yo'naltirilgan bo'lishi ham mumkin, qachonki, urinmaga

perpendikulyar strelka orqali ko'rsatilganidek, manfiy ishora bilan olingan gradient ichkariga yo'nalgan bo'lsa. O'rganish tezligidan kelib chiqib, u C_2 bilan ifodalangan boshqa kontur chizig'ida $\theta^{(2)}$ nuqtaga yetadi, va uning qiymati C_1 qiymatidan past bo'ladi. Yana gradient $\theta^{(2)}$ da baholanadi va xuddi shunday jarayon minimallashgan nuqtaga yetguncha bir nechta iteratsiyalar uchun takrorlanadi, bu yerda gradient 0 ga tushadi, so'ngra θ uchun yangilanishlar bo'lmaydi.

Eng tez tushish

Eng tez tushish - bu gradient tushishning bir shakli bo'lib, unda o'rganish tezligi doimiy bo'lmay, aksincha har bir iteratsiyada hisoblab chiqiladi. Natijada, gradient tushishi orqali parametrlarni yangilash qiymat funksiyasini o'rganish tezligiga nisbatan minimallashtirishga imkon beradi. Boshqacha qilib aytganda, har bir iteratsiyada o'rganish tezligi gradient yo'nalishi bo'yicha harakat maksimal darajada ishlatilishini ta'minlash uchun optimallashtiriladi.

Odatdagiday $C(\theta)$ qiymat funksiyasini olamiz hamda t va $(t+1)$ ketmaket iteratsiyalarni ko'rib chiqamiz. Gradient tushishida bo'lgani kabi, parametrlarni yangilash qoidasi quyidagicha yoziladi:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla C(\theta^{(t)}).$$

Shunday qilib, $(t+1)$ iteratsiyada qiymat funksiyasi quyidagicha ifodalanishi mumkin

$$C\left(\theta^{(t+1)}\right) = C\left(\theta^{(t)} - \eta \nabla C(\theta^{(t)})\right).$$

O'rganish tezligiga nisbatan $(t+1)$ iteratsiyada qiymat funksiyasini minimallashtirish uchun quyidagilarni qaraymiz:

$$\begin{aligned} \frac{\partial C\left(\theta^{(t+1)}\right)}{\partial \eta} &= 0 \\ \Rightarrow \nabla C\left(\theta^{(t+1)}\right) \frac{\partial \left[C\left(\theta^{(t)} - \eta \nabla C(\theta^{(t)})\right)\right]}{\partial \eta} &= 0 \\ \Rightarrow -\nabla C\left(\theta^{(t+1)}\right)^T \nabla C\left(\theta^{(t)}\right) &= 0 \\ \Rightarrow \nabla C\left(\theta^{(t+1)}\right)^T \nabla C\left(\theta^{(t)}\right) &= 0 \end{aligned}$$

Shunday qilib, eng tez tushish bo'lishi uchun gradiyentlarning dot ko'paytmasi $(t+1)$ va t momentlarda 0 ga teng bo'lishi kerak ekan, bu har iteratsiyada gradient vektori oldingi iteratsiyada gradient vektoriga perpendikulyar bo'lishi kerakligini anglatadi.

Stoxastik gradient tushish

Har ikkala, eng tez tushish va gradient tushish, to'liq partiyali ("full-batch") modellardir. Ya'ni, gradientlar butun o'quv ma'lumotlar bazasi asosida hisoblanadi. Shunday qilib, agar ma'lumotlar to'plami juda katta bo'lsa, gradient hisoblash qimmatga tushadi va xotira talablari oshadi. Bundan tashqari, agar ma'lumotlar to'plamida juda katta zaxira mavjud bo'lsa, unda to'liq ma'lumot bazasida gradientni hisoblash foydasiz, chunki shunga o'xshash gradientlarni *kichik partiyalar* deb nomlangan mini partiyalardan foydalanib hisoblash mumkin. Ushbu sanab o'tilgan muammolarni bartaraf etishning eng mashhur usuli - stoxastik gradient tushishi deb nomlangan optimallashtirish texnikasidan foydalanish. *Stoxastik gradient tushish* bu gradient tushish usuli asosida qiymat funksiyasini minimallashtirish jarayoni bo'lib, bunda har bir bosqichdagi gradient butun boshli ma'lumotlar to'plamiga emas, balki yagona ma'lumot nuqtalariga asoslanadi.

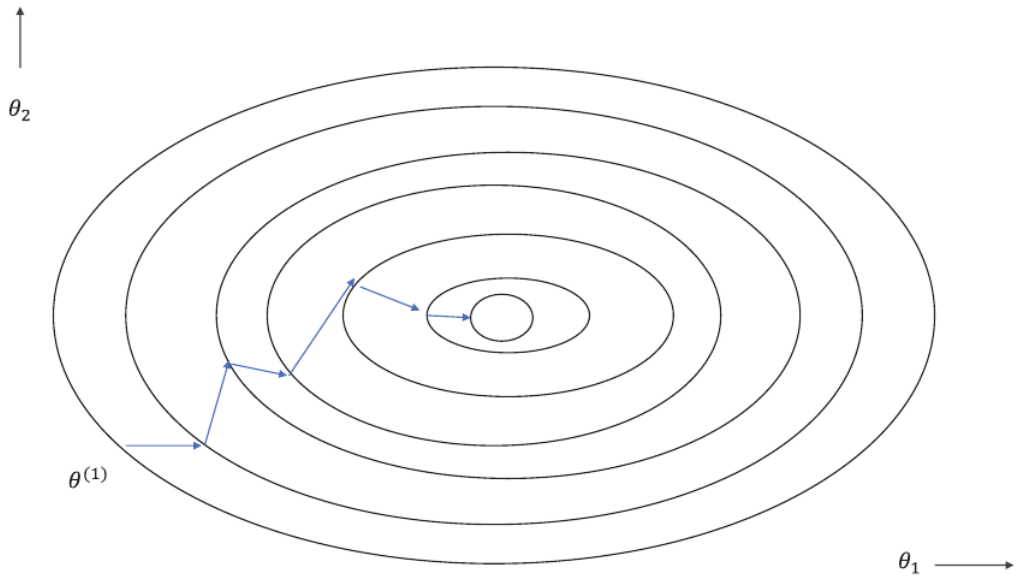
Aytaylik, $C(\theta)$ qiymat funksiyasi m ta o'quv ma'lumot namunalariga asoslangan bo'lsin. Har bir bosqichda gradient tushishi $C(\theta)$ ga emas, balki $C^{(i)}(\theta)$ ga – i-o'quv ma'lumot namunasi bilan berilgan qiymat funksiyasiga asoslangan bo'ladi. Shunday qilib, agar har bir iteratsiyada gradient vektorlarini umumiy qiymat funksiyasiga to'g'ri keltirishimiz kerak bo'lsa, kontur chiziqlari urinmalarga perpendikulyar bo'lmasligi mumkin, chunki ular $C^{(i)}(\theta)$ gradientga asoslangan va $C(\theta)$ ga emas.

$C^{(i)}(\theta)$ qiymat funksiyalari har bir iteratsiyada gradientni hisoblash uchun ishlatiladi va model parametr vektori θ har bir iteratsiyada standart gradient tushishi bo'yicha yangilanadi. Bu yangilanish to butun o'quv ma'lumotlari to'plamini bosib o'tguncha davom etadi. Ma'lumotlar bazasi bo'yicha bunday o'tishlar esa muvofiqlashtirilgan yaqinlashish bo'lmaguncha bir necha marta amalga oshirilishi mumkin.

Har bir iteratsiyadagi gradientlar butun o'quv ma'lumotlar to'plamiga emas, balki bitta o'quv namunalariga asoslanganligi sababli, ular odatda juda shovqinli va yo'nalishni tezda o'zgartiruvchi bo'lishi mumkin. Bu olib kelishi mumkin nimaga: qiymat funksiyasining minimali yaqinidagi tebranishlarga va shuning uchun parametr vektori yangilanishi imkon qadar kichik bo'lishi uchun minimalga o'tishda o'rganish tezligi kamroq bo'lishi kerak. Gradientni hisoblash oson va tez, shuning uchun gradient tushish tezroq yaqinlashadi.

Stoxastik gradient tushishida muhim bo'lgan narsa shundaki, o'quv namunalari iloji boricha tasodifiy bo'lishi kerak. Bu bir nechta o'quv namunalari davridagi stoxastik gradientning tushishi model parametriga o'xshash yangilanishni ta'minlaydi, chunki tasodifiy namunalar umumiy o'quv ma'lumotlarini aks ettirishi mumkin. Agar stoxastik gradient tushishining har bir iteratsiyasidagi namunalar noaniq bo'lsa, ular haqiqiy ma'lumotlar to'plamini aks

ettirmaydi va shuning uchun model parametrlari yangilanishi stoxastik gradient tushishining uzoq vaqt davom etishiga olib keladigan yo'nalishda bo'lishi mumkin.



Rasm 2.37: Stokastik gradient tushish parametrlarini yangilash.

2.37-rasmda ko'rsatilgandek, stoxastik gradient tushish har bir qadamdagi gradient kontur chiziqlaridagi urinmalarga perpendikulyar emas. Ammo, ular individual o'quv namunalari uchun kontur chiziqlaridagi urinmalarga perpendikulyar bo'lishi mumkin. Shuningdek, kuchli tebranuvchi gradient tufayli iteratsiyaga bog'liq qiymat shovqinli bo'ladi.

Agar yagona o'quv ma'lumot nuqtalaridan foydalansak, gradient hisobi ancha arzonlashadi. Shuningdek, yaqinlashish juda tez, ammo uning quyidagi kamchiliklari ham bor:

- Har bir iteratsiyada hisoblangan gradientlar umumiy qiymat funksiyasiga emas, balki yagona ma'lumot nuqtalari bilan bog'liq bo'lgan qiymat funksiyasiga asoslanganligi sababli, gradientlar juda shovqinli. Bu minimal nuqtadagi yaqinlashish muammolariga olib keladi va kuchli tebranishlarni paydo qiladi.

- O'rganish tezligini sozlash muhim ahamiyat kasb etadi, chunki katta qiymatli o'rganish tezligi minimalga yaqinlashish paytida tebranishlarga olib kelishi mumkin. Buning sababi shundaki, gradientlar juda shovqinli va agar gradientning yaqinlashishi bahosi nolga yaqin bo'lmasa,

yuqori o'rganish tzligi minimal darajadan pastga va yangilanishga olib keladi, bu jarayon minimalning har ikkala tomonida takrorlanishi mumkin.

- Gradientlar shovqinli bo'lganligi sababli, har bir iteratsiyadan keyin model parametrlarining qiymatlari juda shovqinli va shuning uchun model parametrlarining qaysi qiymatini olish kerakligini aniqlash uchun evristik stokastik gradient nasliga qo'shilishi kerak. Bu yana bir savolni tug'diradi: mashg'ulotni qachon to'xtatish kerak.

To'liq partiyali model va stoxastik gradientning tushishi o'rtasidagi kelishuv mini-yondashuv bo'lib, unda gradient to'liq o'quv ma'lumotlar bazasiga va bitta ma'lumotlar to'plamiga asoslanmagan. Aksincha, u qiymat funksiyasini hisoblash uchun o'quv ma'lumotlarining mini to'plamidan foydalanadi. Ko'pincha chuqur o'rganish algoritmlarida stoxastik gradientning tushishi uchun mini-tizimli yondashuv qo'llaniladi. Gradient kamroq shovqinli va shu bilan birga ko'pgina xotira cheklovlarini keltirib chiqarmaydi, chunki mini-partiyalarning o'lchamlari o'rtacha hisoblanadi.

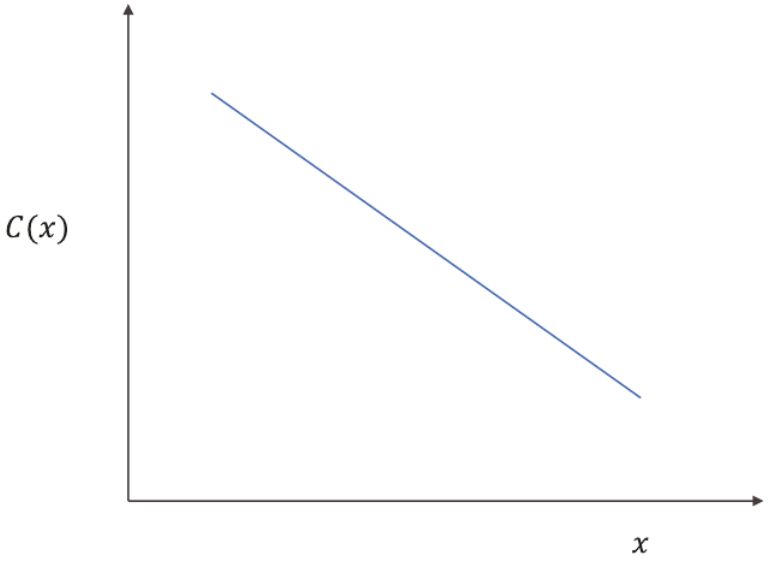
Newton metodi

Minimallashtirish uchun qiymat funksiyasini optimallashtirishda Newton metodini boshlashdan oldin, gradient-tushish usullarining kamchiliklarini ko'rib chiqaylik.

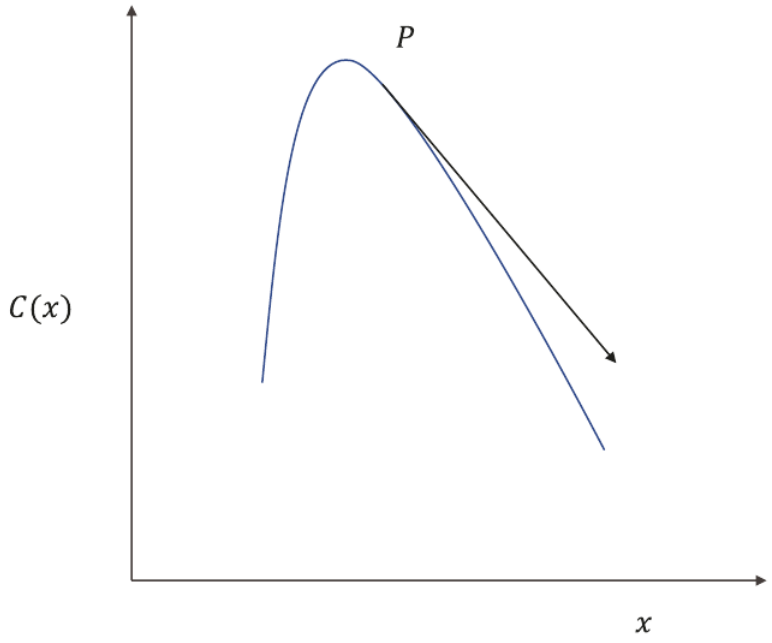
Gradient-tushish usullari ketma-ket iteratsiyalar o'rtasidagi qiymat funksiyasining chiziqliligiga asoslanadi; ya'ni $(t + 1)$ iteratsiyada parametr qiymatiga t vaqtidagi parametr qiymatidan gradientga qarab erishish mumkin, chunki $C(\theta)$ qiymat funktsiyasining t-dan $(t + 1)$ gacha bo'lgan yo'li chiziqli yoki to'g'ri chiziq bilan bog'langan bo'lishi mumkin. Bu juda soddalashtirilgan taxmin va agar qiymat funksiyasi nochiziqli bo'lsa yoki egri chiziqqa ega bo'lsa, gradient tushishi uchun yaxshi yo'nalish bermaydi. Bu haqda yaxshiroq tasavvurga ega bo'lish uchun uchta har xil holatlar uchun bir o'zgaruvchili bo'lgan qiymat funktsiyasining sxemasini ko'rib chiqaylik.

To'g'ri chiziq

2.38-rasmda ko'rsatilgandek, chiziqli qiymat funksiyasi uchun gradientning manfiyligi minimalga erishish uchun eng yaxshi yo'nalish beradi, chunki funksiya chiziqli va hech qanday egrilikka ega emas.



Rasm 2.38: Chiziqli qiymat funksiyasi.

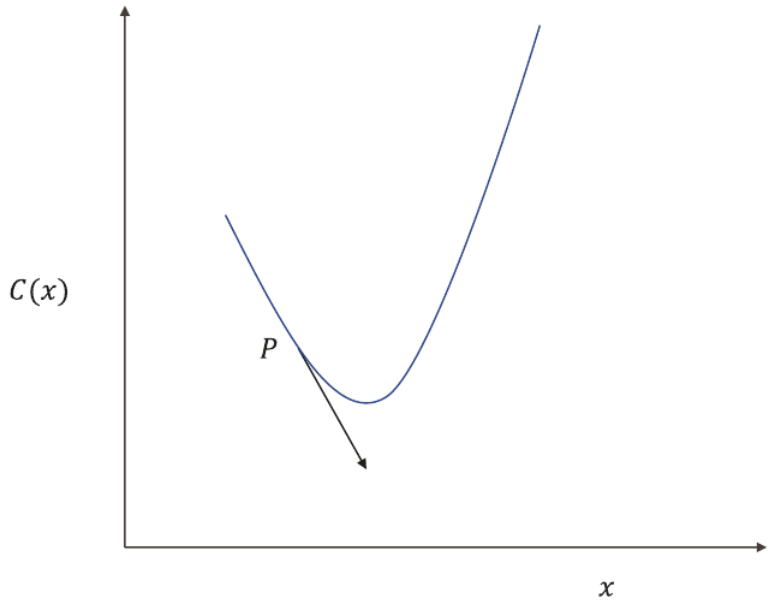


Rasm 2.39: P nuqtada manfiy egrilangan qiymat funksiyasi.

Manfiy egrilangan chiziq

Musbat egrilangan chiziq

2.39 va 2.40-rasmlarda ko'rsatilgani kabi musbat va manfiy egrilangan qiymat funksiyalarining hosilalari minimal uchun yaxshi yo'nalish bermaydi. Shun-



Rasm 2.40: P nuqtada musbat egrilangan qiymat funksiyasi.

ing uchun egrilangan funksiya bilan ishlaganda Hesse matritsasi ham kerak bo'ladi. Hesse matritsasi, biz ko'rganimizdek, elementlari ikkinchi tartibli xususiy hosilalardan iborat matritsasidan boshqa narsa emas. Ular egrilik haqida ma'lumotni o'z ichiga oladi va shuning uchun, oddiy gradient tushishidan farqli o'laroq, parametrlarni yangilash uchun yaxshiroq yo'nalish beradi.

Gradient tushish usullari optimallashtirishning birinchi tartibli yaqinlashish usullari bo'lsa, Nyuton metodlari optimallashtirishning ikkinchi tartibli usullari hisoblanadi, chunki bunda Hesse matritsasini gradient bilan birga qiymat funksiyasidagi egrilanishlar uchun ishlatiladi.

Odatdagidek $C(\theta)$ qiymat funksiyasini qaraylik, bu yerda $\theta \in \mathbb{R}^{n\times 1}$ qandaydir n-o'lchovli model parametr vektori. $C(\theta)$ qiymat funksiyasini θ atrofida Taylor qatoriga yoyamiz va ikkinchi darajali had bilan cheklanamiz:

$$C(\theta + \Delta\theta) = C(\theta) + \Delta\theta^T \nabla C(\theta) + \frac{1}{2}\Delta\theta^T H(\theta)\Delta\theta,$$

bunda $\nabla C(\theta)$ bu gradient va $H(\theta)$ esa $C(\theta)$ qiymat funksiyadan tuzilgan Hesse matritsasi.

Endi, agar θ model parametr vektorining t iteratsiya vaqtidagi qiymati va $(\theta + \Delta\theta)$ esa $(t+1)$ iteratsiya paytidagi model parametr qiymati bo'lsa, u holda

$$C\left(\theta^{(t+1)}\right) = C\left(\theta^{(t)}\right) + \Delta\theta^T \nabla C\left(\theta^{(t)}\right) + \frac{1}{2}\Delta\theta^T H\left(\theta^{(t)}\right)\Delta\theta,$$

bu yerda $\Delta\theta = \theta^{(t+1)} - \theta^{(t)}$.

Gradientni $\theta^{(t+1)}$ ga nisbatan olib, quyidagi ifodaga ega bo'lamiz:

$$\nabla C\left(\theta^{(t+1)}\right) = \nabla C\left(\theta^{(t)}\right) + H\left(\theta^{(t)}\right)\Delta\theta.$$

$\nabla C\left(\theta^{(t+1)}\right)$ gradientni 0 ga tenglab, navbatdagi ifodani olamiz:

$$\nabla C\left(\theta^{(t)}\right) + H\left(\theta^{(t)}\right)\Delta\theta = 0$$
$$\Rightarrow \Delta\theta = -H\left(\theta^{(t)}\right)^{-1}\nabla C\left(\theta^{(t)}\right).$$

Shunday qilib, Nyuton metodi uchun parametr yangilanishi quyidagicha:

$$\Rightarrow \theta^{(t+1)} = \theta^{(t)} - H\left(\theta^{(t)}\right)^{-1}\nabla C\left(\theta^{(t)}\right).$$

Qayd etish kerakki, Nyuton metodida o'rganish tezligi mavjud emas, ammo gradient tushishda bo'lgani kabi o'ranish tezligini yanlab olish mumkin. Nochiziqli qiymat funksiyalari uchun yo'nalishlar Nyuton metodi bilan yaxshiroq bo'lsa, minimalga yaqinlashish iteratsiyalari gradient tushishga nisbatan kamroq bo'ladi. Bir narsani ta'kidlash kerakki, agar biz optimallashtirishga harakat qilayotgan qiymat funksiyasi kvadratik qiymat funktsiyasi bo'lsa, masalan, chiziqli regressiya, Nyuton metodi minimalga bitta qadamda yaqinlashadi.

Biroq, Hesse matritsasini va uning teskarisini hisoblash ba'zan hisoblash qimmat yoki qiyin bo'ladi, ayniqsa kiritma vektor kattaliklar soni katta bo'lsa. Shuningdek, ba'zida hatto shunday qiymat funksiyalari mavjud bo'lishi mumkinki, ular uchun Hesse matritsasi to'g'ri aniqlanmagan bo'ladi. Shunday qilib, ulkan hajmdagi Sun'iy o'rganish va chuqur o'rganish dasturlari uchun gradient tushish, xususan, stoxastik gradient tushish - mini to'plamli texnikadan foydalaniladi, chunki ular nisbatan kamroq intensivroq va ma'lumotlar hajmi katta bo'lganda juda yaxshi ishlaydi.

Cheklovli optimallashtirish masalasi

Cheklovli optimallashtirish masalasida biz optimallashtirishimiz kerak bo'lgan qiymat funksiyasi bilan bir qatorda, biz rioya qilishimiz kerak bo'lgan bir qator cheklovlar mavjud. Bu cheklovlar tenglamalar yoki tengsizliklar bo'lishi mumkin.

Qachonki tenglik cheklovi o'rnatilgan biror funksiyani minimallashtirish masalasiga duch kelsak, biz Lagranj formulirovkasidan foydalanamiz. Aytaylik, $f(\theta)$ ni minimallashtirish kerak va bunda $g(\theta) = 0$ cheklov qo'yilgan, bu yerda $\theta \in \mathbb{R}^{n\times 1}$. Bunday cheklovli optimallashtirish muammosi uchun

$L(\theta, \lambda) = f(\theta) + \lambda g(\theta)$ funksiyasini minimallashtirishimiz kerak. Ushbu kiritilgan L funksiya Lagranjian deb ataladi. Lagranjiandan θ, λ qo'shma vektor bo'yicha gradient olib, uni 0 ga tenglash izlangan θ ni beradi; topilgan ushbu θ vektor $f(\theta)$ ni minimallashtiradi va o'rnatilgan cheklovga rioya qiladi. λ Lagranj multiplikatori deb nomlanadi. Bir nechta cheklovlar mavjud bo'lganda, biz har bir cheklov uchun alohida Lagranj multiplikatoridan foydalanib, barcha bunday cheklovlarni qo'shishimiz kerak. Aytaylik, m ta cheklov qo'yilgan $g_i(\theta) = 0 \ \forall i \in \{1, 2, 3, \ldots m\}$ hol uchun $f(\theta)$ funksiya minimalini topmoqchimiz; bu holda Lagranjian quyidagicha ifodalanishi mumkin:

$$L(\theta, \lambda) = f(\theta) + \sum_{i=1}^{m} \lambda_i g_i(\theta),$$

bu yerda $\lambda = \begin{pmatrix} \lambda_1 & \lambda_2 & \ldots & \lambda_m \end{pmatrix}^T$.

Funksiyani minimallashtirish uchun $L(\theta, \lambda)$ gradienti ikkala θ va λ vektorlarga nisbatan nol vektor bo'lishi kerak; ya'ni,

$$\nabla_\theta L(\theta, \lambda) = 0, \nabla_\lambda L(\theta, \lambda) = 0.$$

Yuqorida keltirilgan usul tengsizlik bilan berilgan cheklovlar uchun to'g'ridan-to'g'ri ishlatilishi mumkin emas. Bunday hollarda, Karush Kahn Tucker usuli deb nomlangan yanada umumiy yondashuvdan foydalanish mumkin.

Aytaylik, $C(\theta)$ qiymat funksiyasini minimallashtirish kerak bo'lsin, bu yerda $\theta \in \mathbb{R}^{n \times 1}$. Shuningdek, θ ga qo'yilgan cheklovlar soni k ga teng bo'lsin

$$\begin{aligned} f_1(\theta) &= a_1 \\ f_2(\theta) &= a_2 \\ f_3(\theta) &\leq a_3 \\ f_4(\theta) &\geq a_4 \\ &\ldots \\ &\ldots \\ f_k(\theta) &= a_k \end{aligned}$$

Bu cheklovli optimallashtirish muammosiga aylanadi, chunki θ bo'ysunishi kerak bo'lgan cheklovlar mavjud. Har bir tengsizlikni standart shaklga aylantirish mumkin, bunda ma'lum bir funksiya nolga teng yoki undan kichikdir. Masalan:

$$f_4(\theta) \geq a_4 \Rightarrow -f_4(\theta) \leq -a_4 \Rightarrow -f_4(\theta) + a_4 \leq 0.$$

Har bir bunday cheklov qat'iy noldan kichik, noldan kichik yoki teng shartlar bilan $g_i(\theta)$ orqali ifodalansin. Shuningdek, qat'iy tenglik bilan qo'yilga

cheklovlar $e_j(\theta)$ tenglamalar orqali qo'yilgan bo'lsin. Bunday minimallashtirish muammolari Lagranj formulirovkasining Karush Kuhn Tucker versiyasi orqali hal qilinadi.

$C(\theta)$ minimallashtirish o'rniga, $L(\theta, \alpha, \beta)$ qiymat funksiyasini quyidagicha minimallashtirishimiz kerak:

$$L(\theta, \alpha, \beta) = C(\theta) + \sum_{i=1}^{k_1} \alpha_i g_i(\theta) + \sum_{j=1}^{k_2} \beta_j e_j(\theta).$$

$\alpha_i \ \forall i \in \{1, 2, 3, \ldots k_1\}$ va $\beta_j \ \forall j \in \{1, 2, 3, \ldots k_2\}$ kattaliklar Lagranj multiplikatorlari deb nomlanadi. Shunday qilib, biz cheklovli minimallashtirish masalasini cheklovsiz minimallashtirish masalasiga keltirdik.

Masalani yechish uchun Karush Kuhn Tucker shartlarini minimal nuqtada quyidagicha bajarish kerak:

- $L(\theta, \alpha, \beta)$ ning θ ga nisbatan gradienti nol vektor bo'lishi kerak; ya'ni,

$$\nabla_\theta L(\theta, \alpha, \beta) = 0$$
$$\Rightarrow \nabla_\theta C(\theta) + \sum_{i=1}^{k_1} \alpha_i \nabla_\theta g_i(\theta) + \sum_{j=1}^{k_2} \beta_j \nabla_\theta e_j(\theta) = 0$$

- $L(\theta, \alpha, \beta)$ ning β ga nisbatan (bu tenglik shartlariga mos keluvchi Lagranj multiplikatori vektori hisoblanadi) gradienti nolga teng bo'lishi kerak:

$$\nabla_\beta L(\theta, \alpha, \beta) = 0$$
$$\Rightarrow \nabla_\beta C(\theta) + \sum_{i=1}^{k_1} \alpha_i \nabla_\beta g_i(\theta) + \sum_{j=1}^{k_2} \beta_j \nabla_\beta e_j(\theta) = 0$$

- Tengsizlik shartlari minimal nuqtada tenglik shartlariga aylanishi kerak. Shuningdek, tengsizlik orqali berilgan Lagranj multiplikatorlari manfiy bo'lmasligi ham kerak:

$$\alpha_i g_i(\theta) = 0, \ va \ \alpha_i \geq 0 \ \forall i \in \{1, 2, \ldots, k_1\}$$

Yuqoridagi shartlarni yechish cheklovli optimallashtirish muammosini minimallashtirishga imkon beradi.

2.4.4 Sun'iy o'rganishning ba'zi muhim nuqtalari

Ushbu bo'limda biz Sun'iy o'rganish uchun juda muhim bo'lgan bir nechta muhim mavzularni muhokama qilamiz. Ularning matematikasi juda boy.

O'lchamni qisqartirish usullari

Asosiy komponent tahlili va yagona qiymat bo'yicha ajratish ("Singular Value Decomposition") Sun'iy o'rganish sohasida eng ko'p ishlatiladigan o'lchamni qisqartirish usullaridir. Biz ushbu usullarni ma'lum darajada bu yerda muhokama qilamiz. Shuni esda tutingki, ma'lumotlarni qisqartirish usullari chiziqli korrelyatsiyaga asoslangan va nochiziqli korrelyatsiyani dastaklamaydi. Kitobning ikkinchi qismida biz sun'iy neyron tarmoqlarga, masalan, "auto-encoder"ga asoslangan o'lchamni qisqartirish usullari haqida gaplashamiz.

Asosiy komponent tahlili Asosiy komponent tahlili bu o'lchamni qisqartirish texnikasi bo'lib, odatda "Chiziqli algebra" bo'limida muhokama qilinadi. Ammo, uning matematikasini tushunishni osonlashtirish uchun biz cheklovli optimallashtirish masalasidan so'ng berishni lozim ko'rdik. 2.41-rasmdagi ikki o'lchovli ma'lumotlar chizmasini ko'rib chiqaylik. Unda ko'rsatilganidek, ma'lumotlar maksimal tarqoqligi ("variance") na x yo'nalishda va na y yo'nalishda bo'lmay, balki ular orasidagi oraliq bir yo'nalishda sodir bo'lgan. Demak, biz ma'lumotlarni maksimal tarqoqlik tomonga yo'naltiradigan bo'lsak, u ma'lumotlarning o'zgaruvchanligini aks ettirgan bo'ladi. Bundan tashqari, qolgan sust tarqoqliklarni shovqin sifatida e'tiborsiz qoldirish ham mumkin.

x va y yo'nalishi bo'yicha ma'lumotlar juda bog'liqdir (2.41-rasmga qarang). Kovariatsion matritsa (inglizcha "Covariance matrix") ortiqcha miqdorni kamaytirish uchun ma'lumotlar to'g'risida kerakli ma'lumotlarni taqdim etadi. x va y yo'nalishlar o'rniga biz maksimal tarqoqlikka ega bo'lgan a_1 yo'nalishni ko'rib chiqamiz.

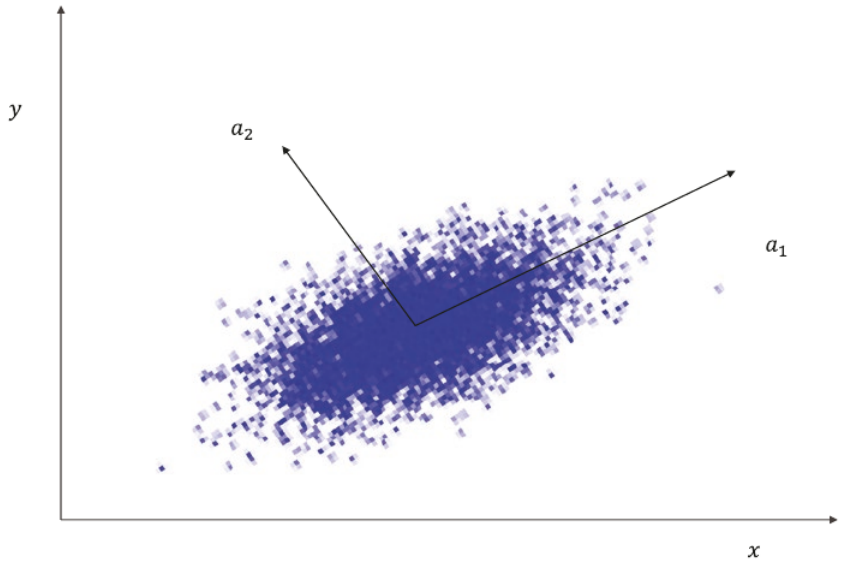
Endi bir oz matematikaga murojaat qilaylik. Hozircha heh qanday chizma bilan qiziqmaymiz va bizda soni m bo'lgan ma'lumotlar mavjud $x^{(i)} \in \mathbb{R}^{n\times 1}$.

Biz n-o'lchamli tekislikda tarqoqlikning kamayish tartibida mustaqil yo'nalishlarni topishimiz kerak. Mustaqil yo'nalishlar deganda shuni nazarda tutyapmizki, ushbu yo'nalishlar orasidagi kovariantlik ("covariance between those directions") 0 bo'lishi kerak. Aytaylik, a_1 birlik vektor bo'lsin va bu vektor yo'nalishi bo'ylab ma'lumotlar tarqoqligi maksimal bo'lsin. Dastlab biz ma'lumotlar vektori o'rtachasini ma'lumotlarni koordinata boshiga markazlashtirish uchun ajratamiz. Aytaylik, x ma'lumotlar vektorining o'rtachasi μ bo'lsin; ya'ni, $E[x] = \mu$.

$(x - \mu)$ vektorning a_1 yo'nalishdagi komponentasi bu $(x - \mu)$ ning a_1 ga proektsiyasi; uni z_1 bilan belgilaymiz:

$$z_1 = a_1^T(x - \mu)$$

$var(z_1) = var[a_1^T(x - \mu)] = a_1^T cov(x) a_1$, bu yerda var variantlik-tarqoqlik va $cov(x)$ kovariatsion matritsani bildiradi.



Rasm 2.41: Korrelyatsiya qilingan $2D$ ma'lumotlar. a_1 va a_2 yo'nalishlar bo'ylab ma'lumotlar korrelatsiya qilinmagan va ular asosiy komponentlardir.

Berilgan ma'lumot nuqtalari uchun tarqoqlik bu a_1 ning funktsiyasidir. Shunday ekan, a_1 birlik vektor ekanligini hisobga olib, tarqoqlikni a_1 ga nisbatan maksimallashtirishimiz kerak:

$$a_1^T a_1 = 1 \Rightarrow a_1^T a_1 - 1 = 0$$

Shunday qilib, maksimallashtirish kerak bo'lgan funksiyani ifodasi:

$$L(a_1, \lambda) = a_1^T cov(x) a_1 - \lambda(a_1^T a_1 - 1),$$

bu yerda λ Lagranj multiplikatori.

Maksimalni olish uchun a_1 ga nisbatan L ning gradiyentini 0 ga tenglashtirib, quyidagilarga ega bo'lamiz:

$$\nabla L = 2cov(x)a_1 - 2\lambda a_1 = 0 \Rightarrow cov(x)a_1 = \lambda a_1.$$

Biz ilgari o'rgangan narsalarimiz paydo bo'lishini ko'rishimiz mumkin. a_1 vektor bu kovariatsion matritsaning Xususiy vektoridan boshqa narsa emas, λ esa unga mos keluvchi Xususiy qiymatidir.

Endi buni a_1 yo'nalish bo'ylab tarqoqlik ifodasiga qo'yib, quyidagilarga ega bo'lamiz:

$$var(z_1) = a_1^T cov(x) a_1 = a_1^T \lambda a_1 = \lambda a_1^T a_1 = \lambda.$$

a_1 yo'nalish bo'ylab tarqoqlik ifodasi Xususiy qiymatning o'zginasi, eng katta Xususiy qiymatiga mos keluvchi Xususiy vektor birinchi asosiy komponentni beradi. Boshqacha aytganda, ma'lumotlar to'plamidagi tarqoqlik maksimal bo'lgan yo'nalishni beradi.

Keling, ikkinchi asosiy komponentni olaylik. Aniqrog'i, tarqoqlik maksimal bo'lgan yo'nalishlar ichida a_1 dan keyin keluvchi yo'nalish.

Ikkinchi asosiy komponentning yo'nalishi a_2 birlik vektor bilan berilgan bo'lsin. Biz ortogonal komponentlarni qidirayotganimiz uchun, a_2 yo'nalishi a_1 ga perpendikulyar bo'lishi kerak.

Ma'lumotlarning a_2 bo'ylab proeksiyasi $z_2 = a_2^T(x-\mu)$ o'zgaruvchi orqali ifodalanishi mumkin. Demak, a_2 bo'ylab ma'lumotlar tarqoqligi quyidagicha beriladi:

$$Var(z_2) = Var[a_2^T(x-\mu)] = a_2^T cov(x) a_2.$$

$a_2^T a_2 = 1$ (chunki a_2 birlik vektor) va $a_2^T a_1 = 0$ (chunki a_2 va a_1 ortogonal) cheklovlarni hisobga olgan holda $Var(z_2)$ ni maksimallashtirish kerak.

a_2, α, β parametrlarga nisbatan $L(a_2, \alpha, \beta)$ funksiyani maksimallashtirish kerak:

$$L(a_2, \alpha, \beta) = a_2^T cov(x) a_2 - \alpha(a_2^T a_2 - 1) - \beta(a_2^T a_1).$$

Dastlab, a_2 ga nisbatan gradientni olamiz va uni nol-vektorga tenglaymiz:

$$\nabla L = 2cov(x)a_2 - 2\alpha a_2 - \beta a_1 = 0.$$

∇L gradient va a_1 vektorlarning skalyar ko'paytmasidan quyidagi ifodani yozamiz:

$$2a_1^T cov(x) a_2 - 2\alpha a_1^T a_2 - \beta a_1^T a_1 = 0,$$

bu yerda $a_1^T cov(x) a_2$ skalyar kattalik bo'lgani uchun, uni $a_2^T cov(x) a_1$ ko'rinishda yozsa ham bo'ladi.

Soddalashtiramiz: $a_2^T cov(x) a_1 = a_2^T \lambda a_1 = \lambda a_2^T a_1 = 0$. Shuningdek, $2\alpha a_1^T a_2$ had ham 0 ga teng bo'lib, $\beta a_1^T a_1 = 0$ ni beradi. $a_1^T a_1 = 1$ bo'lganligi sababli, $\beta = 0$ bo'lishi kerak.

$\beta = 0$ ni $\nabla L = 2cov(x)a_2 - 2\alpha a_2 - \beta a_1 = 0$ ifodaga olib borib qo'ysak, quyidagilarga ega bo'lamiz:

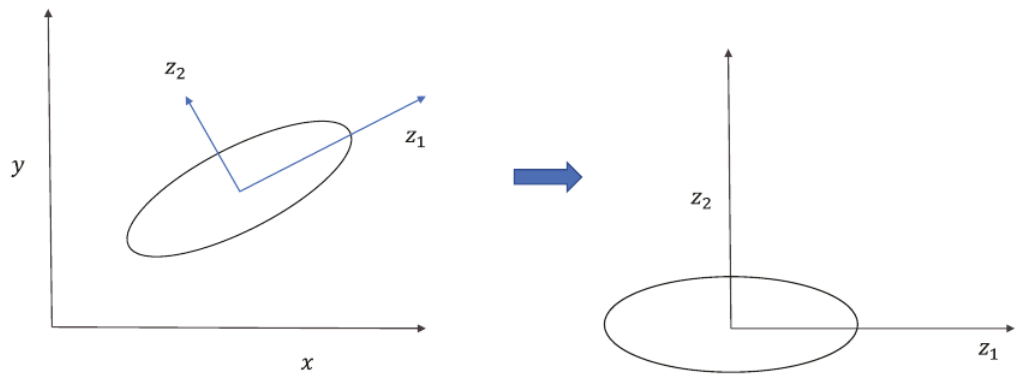
$$2cov(x)a_2 - 2\alpha a_2 = 0,\ ya'ni\ cov(x)a_2 = \alpha a_2.$$

Demak, ikkinchi asosiy komponent ham kovariatsion matritsasining Xususiy vektori ekan va unga mos keluvchi Xususiy qiymat α kattaligi bo'yicha λ dan keyin turuvchi eng katta Xususiy qiymat bo'lishi kerak. Huddi shu yo'l

bilan, $cov(x) \in \mathbb{R}^{n\times 1}$ kovariatsion matritsadan n ta Xususiy vektorlarni olishimiz mumkin va ularning har bir Xususiy qiymat yo'nalishi (yoki, asosiy komponentlari) bo'yicha ma'lumotlar tarqoqligi Xususiy qiymatlar orqali ifodalanishi kerak bo'ladi. Ta'kidlash kerakki, kovariatsion matritsa har doim simmetrikdir. Shuning uchun uning Xususiy vektorlari har doim bir-biriga ortogonal va yo'nalishlari o'zaro bog'liq emas (mustaqil).

Kovariatsion matritsa har doim musbat yarim-aniqlangan bo'ladi. Bu haqiqatdir, chunki kovariatsion matritsaning Xususiy qiymatlari tarqoqliklarni anglatadi va tarqoqlik (ya'ni, dispersiya) manfiy bo'la olmaydi. Agar $cov(x)$ musbat aniqlangan bo'lsa, ya'ni $a^T cov(x)a > 0$ bo'lsa, u holda kovariatsion matritsalarning Xususiy qiymatlari musbat bo'ladi.

2.42-rasmda ma'lumotlarning asosiy komponent tahlili transformatsiyasi ko'rsatilgan. Ko'rinib turibdiki, PCA (inglicha "Principal Component Analysis") ma'lumotlarni markazlashtiradi va PCA orqali transformatsiyalangan o'zgaruvchilar orasidagi korrelyatsiyani yo'qotadi.



Rasm 2.42: Asosiy komponentlar tahlili ma'lumotlarni markazlashtiradi va so'ngra ularni maksimal tarqoqlikka ega bo'lgan o'qlarga proyeksiyalaydi. Agar tarqoqlik z_2 bo'yla juda kichik bo'lsa, z_2 yo'nalishdagi ma'lumotlarni e'tiborsiz qoldirish mumkin.

Ma'lumotlarni qisqartirishda qachon PCA foydali bo'ladi? Kiritma ma'lumotlarining turli o'lchamlari o'rtasida yuqori bog'liqlik, ya'ni korrelatsiya mavjud bo'lganda, ma'lumotlarning tarqoqligi yuqori bo'lgan kam sonli mustaqil yo'nalishlar bo'lishi mumkin va boshqa yo'nalishlarda ham tarqoqlik juda kam bo'ladi. PCA yordamida ma'lumotlar komponentlarini tarqoqlik katta bo'lgan kam sonli yo'nalishlarda ushlab turish mumkin va qolgan ma'lumotlarni e'tiborsiz qoldirib, umumiy o'zgarishga katta hissa qo'shish mumkin.

Tanlangan asosiy komponentlar tomonidan qancha tarqoqlik saqlanganini qayerdan bilamiz? Agar z vektor x Kiritma vektor uchun transformatsiyalangan (o'zgartirilgan) komponentlarni bildirsa, $cov(z)$ diagonal matritsa bo'ladi va uning diagonal elementlari $cov(x)$ matritsaning Xususiy qiymatlari bo'ladi.

$$cov(x) = \begin{pmatrix} \lambda_1 & \dots & 0 \\ \vdots & \lambda_2 & \vdots \\ 0 & \cdots & \lambda_n \end{pmatrix}$$

Tasavvur qilaylik, Xususiy qiymatlar $\lambda_1 > \lambda_2 > \lambda_3 \dots > \lambda_n$ kabi tartiblangan bo'lsin.

Aytaylik, biz faqat birinchi k asosiy komponentlarini saqlashni tanlaymiz; u holda ushlab qolingan ma'lumotlarda tarqoqlik darajasi quyidagicha bo'ladi:

$$\frac{\lambda_1 + \lambda_2 + \lambda_3 + \ldots + \lambda_k}{\lambda_1 + \lambda_2 + \lambda_3 + \ldots + \lambda_k + \ldots + \lambda_n}.$$

Yagona qiymatlar bo'yicha ajratish Yagona qiymatlar bo'yicha ajratish o'lchamni qisqartirish texnikasi bo'lib, u $A \in \mathbb{R}^{m \times n}$ matritsani uchta matritsaning ko'paytmasi ko'rinishida yoyadi, $A = USV^T$. Bunda

1) $U \in \mathbb{R}^{m \times m}$ va u AA^T matritsaning barcha Xususiy vektorlaridan iborat;
2) $V \in \mathbb{R}^{n \times n}$ va u A^TA matritsaning barcha Xususiy vektorlaridan iborat;
3) $S \in \mathbb{R}^{m \times n}$ va u ikkala A^TA va AA^T matritsaning Xususiy vektorlarining k kvadrat ildizidan tashkil topgan, bu yerda k – A matritsaning rangi.

U ning ustun vektorlari bir-biriga nisbatan ortogonaldir va shuning uchun ortogonal bazisni tashkil qiladi. Xuddi shunday, V ning ustun vektorlari ham ortogonal bazisni hosil qiladi:

$$U = \begin{pmatrix} u_1 & u_2 & \dots & u_m \end{pmatrix},$$

bu yerda $u_i \in \mathbb{R}^{m \times 1}$ – U ning ustun vektorlari.

$$V = \begin{pmatrix} v_1 & v_2 & \dots & v_m \end{pmatrix},$$

bu yerda $v_i \in \mathbb{R}^{n \times 1}$ – V ning ustun vektorlari.

$$S = \begin{pmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \sigma_2 & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$$

A ning rangiga bog'liq ravishda, A matritsaning k rangiga mos keluvchi diagonal elementlar $\sigma_1, \sigma_2, \ldots \sigma_k$ bo'ladi:

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_k u_k v_k^T.$$

Yagona qiymatlar (inglizcha "singular values") deb yuritiluvchi σ_i $\forall i \in \{1, 2, 3, \ldots k\}$ ikkala $A^T A$ va AA^T matritsaning Xususiy qiymatlari kvadrat ildizi hisoblanadi va shuning uchun ular ma'lumotlar tarqoqlik o'lchovidir. $\sigma_i u_i v_i^T$ $\forall iin \in \{1, 2, 3, \ldots k\}$ ning har biri bu birinchi rangli matritsa. Biz faqat birinchi rangli matritsalarni olib qolamiz, chunki ular uchun yagona qiymatlar ahamiyatlidir va ulargina ma'lumotlardagi tarqoqlikning katta qismini tushuntirishi mumkin.

Agar, faqatgina dastlabki eng katta amplitudali p yagona qiymatlarga mos keluvchi birinchi rangli p matritsalarni olsak, ma'lumotlarda olib qolingan tarqoqlik-dispersiya quyidagicha beriladi:

$$\frac{\sigma_1^2 + \sigma_2^2 + \sigma_3^2 + \ldots + \sigma_p^2}{\sigma_1^2 + \sigma_2^2 + \sigma_3^2 + \ldots + \sigma_p^2 + \ldots + \sigma_k^2}.$$

Yagona qiymatlar bo'yicha ajratish metodi orqali tasvirlarni siqish mumkin. Huddi shunday, foydalanuvchi rangli matritsani foydalanuvchi vektor (inglizcha "user-vector matrix") va element vektorlarini (inglizcha "item vector") o'z ichiga olgan ikkita matritsaga ajratish uchun qo'shma filtrlashda ham bu metod qo'llaniladi. Matritsaning yagona qiymatlar bo'yicha ajratilishi USV^T orqali beriladi. Foydalanuvchi rangli matritsa R ni quyidagicha ajratish mumkin:

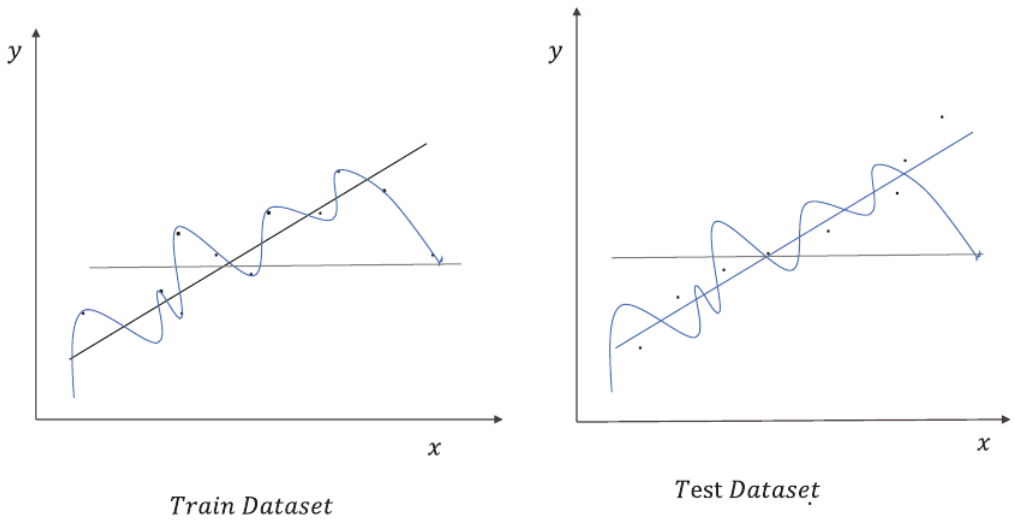
$$R = USV^T = US^{1/2}S^{1/2}V^T = U'V'^T,$$

bu yerda U' ishchi-vektor matritsa (inglizcha "user-vector matrix") va u $US^{1/2}$ ga teng; V' esa elementlar-vektor matritsa, $V' = S^{1/2}V^T$.

Tartibga solish – Regulyarizatsiya Sun'iy o'rganish modelini yaratish jarayoni o'quv ma'lumotlariga mos keluvchi parametrlarni olishni o'z ichiga oladi. Agar model sodda bo'lsa, unda qurilgan model ma'lumotlardagi o'zgarishni sezmaslik va katta og'ishlardan kamchilik sezadi. Ammo, agar model juda murakkab bo'lsa, u iloji boricha ko'proq o'zgarishga va jarayon ma'lumotlarida tasodifiy shovqin uchun o'quv ma'lumotlaridagi modellashtirishga harakat qiladi. Bu oddiy modellar tomonidan ishlab chiqarilgan noto'g'ri og'ishlarni yo'q qiladi, ammo yuqori tarqoqlikni keltirib chiqaradi; ya'ni, model kiritma ma'lumotdagi juda kichik o'zgarishlarga sezgir bo'lib qoladi. Model uchun yuqori tarqoqlik-dispersiya yaxshi narsa emas, ayniqsa ma'lumotlardagi shovqin

katta bo'lsa. Bunday hollarda, o'quv ma'lumotlarini juda yaxshi bajarishga intilayotgan model test ma'lumotlar bazasida sust ishlaydi, chunki model yangi ma'lumotlarni umumlashtirish imkoniyatini yo'qotadi. Modellarning yuqori tarqoqlikka sezuvchan bo'lib qolishi *o'tamoslashish* (inglizcha "overfitting") deyiladi.

2.43-rasmda ko'rib turganimizdek, ma'lumotlarga moslashuvchi uchta model mavjud. Gorizontal o'qqa parallel bo'lgan birinchi model yuqori og'ishdan qiynalsa, egri chiziq ko'rinishidagi ikkinchi model esa yuqori tarqoqlik-dispersiyadan qiynaladi. Shu ikki model orasidagi model – taxminan 45° qiyalikka ega bo'lgan to'g'ri chiziq esa na katta tarqoqlikka, na yuqori og'ishga ega.



Rasm 2.43: Yuqori tarqoqlik va yuqori og'ishga ega modellar ko'rinishi.

Katta tarqoqlikka ega bo'lgan model o'quv ma'lumotlariga juda yaxshi moslashadi, ammo unchalik katta bo'lmagan sinov ma'lumotlar bazasida yaxshi natijalarga erisha olmaydi. Ko'k rangda berilgan model o'quv ma'lumotiga aynan mos kelmasligi mumkin, ammo u sinov ma'lumotlari bo'yicha yaxshiroq ishlaydi, chunki model yuqori tarqoqlikka duch kelmaydi. Mahorat bilan ishlab chiqilgan model bu yuqori og'ishdan aziyat chekmaydigan va o'ta murakkab bo'lmagan model bo'lishi kerakki, u tasodifiy shovqin uchun ham ishlasin.

Yuqori tarqoqlikka ega modellar odatda katta qiymatga ega model parametrlariga ega, chunki bu modellarning kichik o'zgarishlarga sezgirligi yuqori. Yuqori model tafovutlari natijasida yuzaga keladigan o'tamoslashish muammosini bartaraf etish uchun *regulyarizatsiya* deb ataladigan mashhur usul keng qo'llaniladi.

Tushunchalarimizni oydinlashtirish uchun, oldin ko'rib chiqilgan chiziqli regressiya qiymat funksiyasini ko'rib chiqaylik:

$$C(\theta) = \|X\theta - Y\|_2^2 = (X\theta - Y)^T(X\theta - Y).$$

Avval muhokama qilinganidek, katta tarqoqlikka ega modellar katta o'lchamdagi model parametrlariga ega. Model parametr vektorining qiymati katta bo'lsa, $C(\theta)$ qiymat funksiyasiga qo'shimcha had qo'shishimiz mumkin va bu global qiymat funksiyasiga ta'sir etadi.

Shunday qilib, biz yangi qiymat funksiyasiga ega bo'lishimiz mumkin, $L(\theta) = \|X\theta - Y\|_2^2 + \lambda\|\theta\|_2^2$, bu yerda $\|\theta\|_2^2$ model parametr vektorining l^2 normasi kvadrati. Shundan so'ng, optimallashtirish masalasi quyidagicha ko'rinish oladi:

$$\theta^* = \underbrace{Arg\text{Min}}_{\theta} L(\theta) = \|X\theta - Y\|_2^2 + \lambda\|\theta\|_2^2.$$

θ ga nisbatan olingan ∇L gradientni 0 ga tenglab, $\theta^* = \left(X^TX + \lambda I\right)^{-1} X^TY$ ni olamiz.

Ko'rinib turibdiki, qiymat funksiyasidagi $\|\theta\|_2^2$ had borligi sababli, model parametrlarining kattaligi juda katta bo'lmasligi mumkin, chunki bu global qiymat funksiyasiga ta'sir etadi. Bunda λ regulyarizatsiya hadining vaznini aniqlaydi. λ ning kattaroq qiymati $\|\theta\|_2^2$ ning kichikroq qiymatlarini keltirib chiqaradi, shu bilan model soddalashadi va yuqori og'ishga yoki kam-moslashishga moyil bo'ladi. Umuman olganda, λ ning yanada kichik qiymatlari modelning murakkabligi va tarqoqligini kamaytirishga hissa qo'shadi. λ odatda "cross validation" orqali optimallashtiriladi.

Agar l^2 normaning kvadrati regulyarizatsiya hadi sifatida ishlatilsa, optimallashtirish usuli l^2 regulyarizatsiyasi deb ataladi. Ba'zida model parametr vektorlarining l^1 normasi normallashtirish hadii sifatida ishlatiladi va optimallashtirish usuli l^1 regulyarizatsiyasi deb ataladi. Regressiya masalalarida qo'llaniladigan l^2 regulyarizatsiya *tizma regressiya* (inglizcha "ridge regression") deb nomlanadi, shu kabi regressiya masalalarida qo'llaniladigan l^1 regulyarizatsiya esa *lasso regressiya* deb ataladi.

l^1 regulyarizatsiya uchun yuqorida berilgan regressiya masalasi quyidagicha bo'ladi:

$$\theta^* = \underbrace{Arg\text{Min}}_{\theta} L(\theta) = \|X\theta - Y\|_2^2 + \lambda\|\theta\|_1.$$

Tizma regressiya matematik jihatdan qulayroqdir, chunki uning tugal ko'rinishdagi yechimi mavjud, lasso regressiyasida esa tugal ko'rinishdagi

yechim yo'q. Biroq, nomutanosib ma'lumotlar bilan ishlaganda lasso regressiyasi tizma regressiyasiga qaraganda ancha kuchliroqdir. Lasso masalalari siyrak yechimlarni beradi, shuning uchun bu belgilarni tanlash uchun juda yaxshi, ayniqsa kiritma belgilarda o'rtacha va yuqori korrelyatsiya mavjud bo'lganda.

Regulyarizatsiya – cheklovli optimallashtirish masalasi sifatida Biz model parametr vektorining qiymati biror o'zgarmas qiymatdan kichik yoki unga teng bo'lishi uchun ta'sir etuvchi had qo'shgan edik. Buning o'rniga cheklovni qo'shishimiz ham mumkin. Bu holda optimallashtirish masalasi quyidagicha bo'lishi mumkin:

$$\theta^* = argmin_\theta C(\theta) = \|X\theta - Y\|_2^2,$$

shunday holdaki $\|\theta\|_2^2 \leq b$, bunda b doimiy.

Ushbu cheklovli minimallashtirish masalasini, yangi Lagranj formulirovkasi orqali, cheklovsiz minimallashtirish masalasiga o'zgartirish mumkin:

$$L(\theta, \lambda) = \|X\theta - Y\|_2^2 + \lambda \left(\|\theta\|_2^2 - b\right)$$

Karush Kuhn Tucker shartlari bilan Lagranj qiymat funksiyasini minimallashtirish uchun quyidagilar muhimdir:

- θ ga nisbatan olingan gradient $\nabla_\theta L(\theta, \lambda)$ nol vektor bo'lishi kerak va soddalashtirish natijasida:

$$\theta = (X^T X + \lambda I)^{-1} X^T Y \tag{2.8}$$

- Shuningdek, optimal nuqtada $\lambda\left(\|\theta\|_2^2 - b\right) = 0$ va $\lambda \geq 0$. Agar regulyarizatsiyani qarasak, ya'ni $\lambda > 0$ bo'lsa, unda

$$\|\theta\|_2^2 = 0 \tag{2.9}$$

(2.8)-dan ko'rinib turibdiki, olingan θ bu λ ning funksiyasi. λ shunday sozlanishi kerakki, (2.9) orqali berilgan cheklov qanoatlantirilishi kerak.

(2.8)-dan olingan $\theta = (X^T X + \lambda I)^{-1} X^T Y$ yechim l^2 regulyarizatsiyadan olinadigani bilan bir xil bo'ladi.

Regulyarizatsiyaga qaytadigan bo'lsak, modeldagi murakkablikni kamaytiruvchi qiymat funksiyasining har qanday komponenti regulyarizatsiyani ta'minlaydi.

Hatto modellashtirish jarayonini erta to'xtatish ham regulyarizatsiyani ta'minlaydi. Masalan, gradient tushish usulida qanchalik ko'p iteratsiyalar mavjud bo'lsa, biz modellar shunchalik murakkablashadi, chunki har bir iteratsiyada gradient tushishi qiymat funksiyalar qiymatini yanada pasaytirishga

harakat qiladi. Modelni o'rganish jarayonini ba'zi mezonlarga, masalan, iterativ jarayonda testlar to'plamining qiymat funksiyasi kattaligi oshishiga qarab to'xtatishimiz mumkin. Qayta o'qitishni takrorlash jarayonida qiymatning funktsional qiymati sinov qiymatining oshishi bilan pasayadi, bu ortiqcha ishlov berishning boshlanishining belgisi bo'lishi mumkin va shu sababli iterativ o'qishni to'xtatish mantiqiy bo'ladi. Har doim o'qitish ma'lumotlari model o'rganishi kerak bo'lgan parametrlar soni bilan solishtirganda kamroq bo'lsa, ortiqcha ishlov berish ehtimoli katta, chunki model kichik ma'lumotlar to'plami uchun juda ko'p qoidalarni bilib oladi va ko'rinmaydigan ma'lumotlarga yaxshi umumlashtira olmasligi mumkin. Agar ma'lumotlar to'plami parametrlar soni bilan taqqoslanadigan bo'lsa, u holda o'rganilgan qoidalar populyatsion ma'lumotlarning salmoqli qismini tashkil etadi va shuning uchun modelni haddan tashqari moslashtirish ehtimoli kamayadi.

2.5 Xulosa

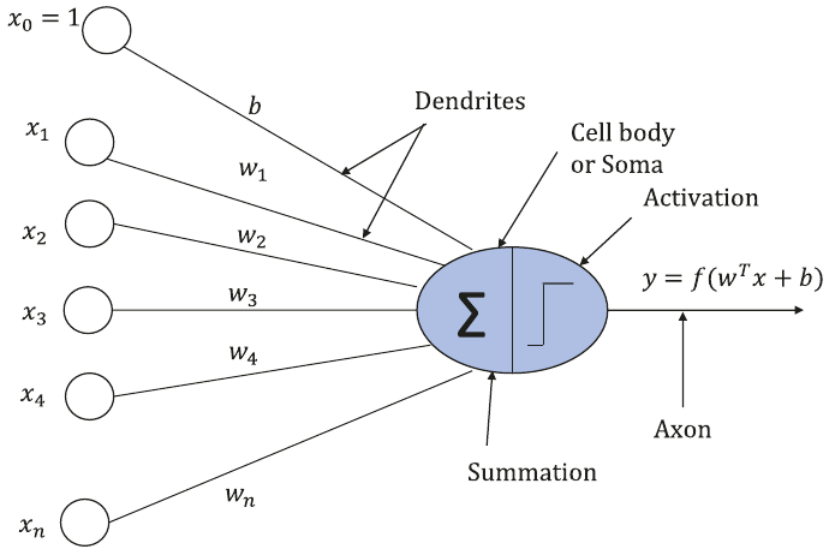
Ushbu bobda biz Sun'iy o'rganish va chuqur o'rganish tushunchalarini davom ettirish uchun barcha kerakli matematik tushunchalarni ko'rib chiqdik. Kitobxonga, ko'proq ma'lumot olish uchun ushbu mavzularga oid tegishli darsliklarni o'qishni tavsiya etiladi. Biroq, ushbu bob yaxshi boshlang'ich nuqta. Keyingi bobda biz Sun'iy Neyron To'rlari mavzusidan boshlaymiz.

3

Sun’iy Neyron To’rlari

3.1 Sun'iy o'rganish

XX-asrning 40-yillaridan beri mavjud bo'lgan Sun'iy Neyron To'rlaridan sun'iy o'rganish rivojlandi. Neyron to'rlari bu biologik miyada joylashgan aksonlarni taqlid qiladigan sun'iy neyronlar deb ataladigan qayta ishlash birliklarining o'zaro bog'liq tarmoqlari. Biologik neyronda dendritlar turli xil qo'shni neyronlardan, odatda ularning soni mingdan ortiq, kirish signallarini qabul qiladi. Ushbu o'zgartirilgan signallar hujayralar tanasiga yoki neyronning bir qismiga yuboriladi, u erda bu signallar birlashtirilib, neyronning aksoniga beriladi. Agar qabul qilingan kirish signali belgilangan chegaradan yuqori bo'lsa, akson signalni chiqaradi va bu signal qo'shni dendritlarga yuboriladi.



Rasm 3.1: *Sun'iy neyron strukturasi. Har bir sun'iy neyron to'rida uchta qatlam mavjud - kirish qatlami,yashirin qatlam va chiqish qatlami. Bir nechta tugunlar yoki neyronlar to'plamidan hosil bo'lgan kirish qatlami kiritma ma'lumotlarni qabul qiladi. To'rdagi har bir neyron o'z funksiyasiga ega va har bir ulanish u bilan bog'liq bo'lgan vazn qiymatiga ega. Kiritma ma'lumotlar kirish qatlamidan so'ng alohida neyronlar to'plamiga - yashirin qatlamga o'tadi. Chiqish qatlami yakuniy natijalarni beradi.*

Sun'iy neyron to'rlari biologik neyronlardan ilhomlanib qurilgan va qulaylik uchun ba'zi moslashtirishlar qilingan. Har bir sun'iy neyron to'rida uchta qatlam mavjud - *kirish qatlami,yashirin qatlam* va *chiqish qatlami.* Dendritlar singari, neyronga kirish ulanishlari boshqa qo'shni neyronlarning siqilgan yoki kuchaytirilgan kirish signallarini o'tkazadi. Signallar neyronga uzati-

ladi, u yerda kirish signallari yig'ilib, so'ng qabul qilingan umumiy ma'lumot asosida qanday chiqish kerakligi to'g'risida qaror qabul qilinadi. Masalan, neyronlarning ikkilik chegarasi uchun umumiy kirish miqdori oldindan belgilangan qiymatdan oshib ketganda 1 ga teng qiymat beriladi; aks holda chiqish 0 ga yetadi. Sun'iy neyron to'rlarida boshqa bir nechta neyron turlari qo'llaniladi va ularni amalga oshirish faqat neyron chiqishini ishlab chiqarish uchun umumiy kirishda faollashtirish funktsiyasi bilan farq qiladi. 3.1-rasmda sun'iy neyron strukturasi ko'rsatilgan va turli xil biologik ekvivalentlar qiyoslash uchun berilgan.

Sun'iy neyron to'rlari haqidagigi tushunchalarni yanada rivojlantirishdan avval, neyron to'rlarning eng sodda turi bo'lgan *Perseptron* bilan tanishamiz.

3.2 Perseptron va Perseptronda o'rganish algoritmi

Perseptronda o'rganish algoritmlari bajarishi mumkin bo'lgan cheklovlar mavjud bo'lsa-da, ular hozirda mavjud bo'lgan sun'iy o'rganish ilg'or uslublarining asosidir. Shunday qilib, Perseptron va Perseptronda o'rganish algoritmini batafsil o'rganish maqsadga muvofiqdir. Perseptron - bu ikkita sinfni ajratish uchun gipertekislikdan foydalanuvchi chiziqli ikkilik tasniflagichdi. Perseptronda o'rganish algoritmi, agar bunda vaznlar va biaslar to'plami o'rnatilishi mumkin bo'lsa, barcha kiritma ma'lumotlarni to'g'ri tasniflovchi vazn va biasni kiritish kafolatlanadi.

Perseptron - bu chiziqli tasniflovchi hisoblanadi va 2-bobda ko'rganimizdek, chiziqli tasniflagichlar, odatda, musbat sinfni manfiy sinfdan ajratib turuvchi gipertekislik qurish orqali ikkilik tasnifini amalga oshiradi.

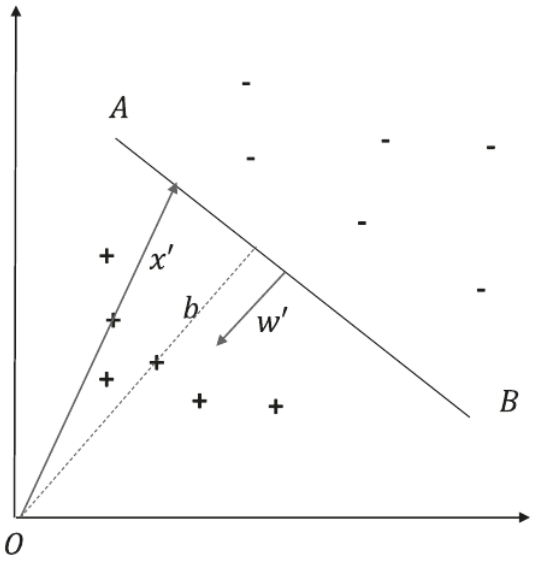
Gipertekislik unga perpendikulyar bo'lgan birlik vazn vektori $\omega' \in \mathbb{R}^{n\times 1}$ va gipertekislikdan asosgacha bo'lgan masofani aniqlovchi bias had b bilan ifodalanadi. Bunda birlik vektor $\omega' \in \mathbb{R}^{n\times 1}$ yo'nalishi musbat sinf tomonga qarab yo'naltirilgan.

3.2-rasmda ko'rsatilgandek, har qanday kiritma vektor $x' \in \mathbb{R}^{n\times 1}$ va manfiy ishora bilan olingan birlik vektor $\omega' \in \mathbb{R}^{n\times 1}$ skalyar ko'paytmasi asosdan gipertekislikkacha bo'lgan masofa b ni beradi.

Formal ravishda qaraganda, gipertekislikda yotgan nuqtalar uchun quyidagi ifodani yozish mumkin:

$$-\omega'^T x' = b \Rightarrow \omega'^T x' + b = 0.$$

Xuddi shunday, gipertekislik ostida joylashgan nuqtalar uchun, ya'ni, musbat sinfga tegishli bo'lgan kiritma vektor $x'_+ \in \mathbb{R}^{n\times 1}$ uchun, manfiy ishora



Rasm 3.2: *Ikki sinfni ajratib turuvchi gipertekislik.*

bilan olingan x'_+ ning ω' ga proeksiyasi b dan kam bo'lishi kerak. Shunday qilib, musbat sinfga tegishli bo'lgan nuqtalar uchun,

$$-\omega'^T x' < b \Rightarrow \omega'^T x' + b > 0.$$

Xuddi shunday, gipertekislik ustida joylashgan nuqtalar uchun, ya'ni, manfiy sinfga tegishli bo'lgan kiritma vektor $x'_- \in \mathbb{R}^{n\times 1}$ uchun, manfiy ishora bilan olingan x'_- ning ω' ga proeksiyasi b dan katta bo'lishi kerak. Shunday qilib, manfiy sinfga tegishli bo'lgan nuqtalar uchun

$$-\omega'^T x' > b \Rightarrow \omega'^T x' + b < 0.$$

Yuqorida yozganlarimizni jamlab, biz quyidagicha xulosa chiqarishimiz mumkin:

- $\omega'^T x' + b = 0$ gipertekislikka to'g'ri keladi va gipertekislik ustida yotgan barcha $x' \in \mathbb{R}^{n\times 1}$ bu shartni qondiradi. Odatda, gipertekislik ustidagi nuqtalar manfiy sinfga tegishli bo'ladi.
- $\omega'^T x' + b > 0$ musbat sinfning barcha nuqtalariga to'g'ri keladi.
- $\omega'^T x' + b \leq 0$ manfiy sinfning barcha nuqtalariga to'g'ri keladi.

Biroq, Perseptron uchun vazn vektori ω' ni birlik vektori sifatida ushlab turish shart emas, aksincha uni har qanday umumiy vektor sifatida olsa

ham bo'ladi. Bunday holatda b gipertekislikdan asosgacha bo'lgan masofaga to'g'ri kelmaydi. Aksincha, bu masshtablangan masofa bo'lib, ω' vektorning l^2 normasi (ya'ni, $\| \omega' \|_2$) masshtab koeffitsienti bo'lib xizmat qiladi. Oxirgi jumlalarni umumlashtiramiz: agar ω' gipertekislikka perpendikulyar va musbat sinfga tomon yo'nalgan biror umumiy vektor bo'lsa, u holda $\omega'^T x' + b = 0$ gipertekislikni ifodalaydi, bu yerda b – asosdan gipertekislikkacha bo'lgan masofa ko'paytirilgan ω' ning uzunligi.

Sun'iy o'rganish sohasidagi qilinishi kerak bo'lgan vazifa - gipertekislik parametrlarini, ya'ni, ω' va b o'rnatishdir. Umuman olganda, biz bias haddan xalos bo'lish uchun masalani soddalashtirishga harakat qilamiz va uni, 2-bobda muhokama qilganimizdek, ω ichidagi kiritma xususiyatga mos keluvchi o'zgarmas parametr 1 sifatida qabul qilamiz.

Aytaylik, bias hadni qo'shgandan keyingi yangi parametr vektori $\omega \in \mathbb{R}^{(n+1)\times 1}$ bo'lsin va doimiy 1 qo'shilgandan so'ng yangi kiritma xuxusiyatli vektor $x \in \mathbb{R}^{(n+1)\times 1}$ bo'lsin. Aniqrog'i, quyidagicha:

$$\begin{aligned} x' &= \begin{pmatrix} x_1 & x_2 & x_3 & \dots & x_n \end{pmatrix}^T \\ x &= \begin{pmatrix} 1 & x_1 & x_2 & x_3 & \dots & x_n \end{pmatrix}^T \\ \omega' &= \begin{pmatrix} \omega_1 & \omega_2 & \omega_3 & \dots & \omega_n \end{pmatrix}^T \\ \omega &= \begin{pmatrix} b & \omega_1 & \omega_2 & \omega_3 & \dots & \omega_n \end{pmatrix}^T \end{aligned}$$

Avvalgi amallarni bajarib, $\mathbb{R}^n$ vektor fazoda, asosdan muayyan masofada joylashgan gipertekislikdan $\mathbb{R}^{(n+1)}$ vektor fazosiga o'tdik. Gipertekislik endi faqat uning vazn parametrlar vektori $\omega \in \mathbb{R}^{(n+1)\times 1}$ bilan aniqlanadi, va tasniflash qoidalari quyidagicha soddalashtiriladi:

- $\omega^T x = 0$ gipertekislikka to'g'ri keladi va gipertekislikda joylashgan barcha $x \in \mathbb{R}^{(n+1)\times 1}$ bu shartni qondiradi.
- $\omega^T x > 0$ musbat sinfning barcha nuqtalariga to'g'ri keladi. Bu degani, endi tasniflash faqat ω va x vektorlari orasidagi burchak bilan aniqlanadi. Agar kiritma vektor x hamda vazn parametrlar vektori ω orasiqdagi burchak $-90°$ va $+90°$ oraliqda bo'lsa, u holda chiqish sinfi musbat bo'ladi.
- $\omega^T x \leq 0$ manfiy sinf nuqtalariga to'g'ri keladi. Tenglik holati turli tasniflash algoritmlarida turlicha talqin qilinadi. Perseptronlar uchun gipertekislikdagi nuqtalar manfiy sinfga tegishli deb hisoblanadi.

Endi bizda Perseptronda o'rganish algoritmini davom ettirish uchun hamma narsa mavjud.

Aytaylik, m ta $x^{(i)} \in \mathbb{R}^{(n+1)\times 1}$ $\forall i = \{1, 2, \ldots, m\}$ kiritma xususiyatli vektorlar va $y^{(i)} \in \{0, 1\}$ $\forall i = \{1, 2, \ldots, m\}$ tegishli sinf yorlig'i bo'lsin.

U holda, Perseptronda o'rganish masalasi quyidagicha bo'ladi:

1-qadam: Vaznning tasodifiy to'plamini yaratishdan boshlanadi, $\omega \in \mathbb{R}^{(n+1)\times 1}$.

2-qadam: Berilgan nuqtaviy ma'lumotlar uchun taxmin qilingan sinf aniqlanadi. Kiritma ma'lumotlar $x^{(i)}$ uchun, agar $\omega^T x^{(i)} > 0$ bo'lsa, u holda taxmin qilingan sinf $y_p^{(i)} = 1$, yo'qsa $y_p^{(i)} = 0$. Perseptron tasniflagichi uchun gipertekislikdagi nuqtalar odatda manfiy sinfga tegishli, deb hisoblanadi.

3-qadam: Vazn vektori ω quyidagicha yangilanadi:

- Agar $y_p^{(i)} = 0$ va shu paytdagi sinf $y^{(i)} = 1$ bo'lsa, vazn vektorini $\omega = \omega + x^{(i)}$ sifatida yangilanadi.
- Agar $y_p^{(i)} = 1$ va shu paytdagi sinf $y^{(i)} = 0$ bo'lsa, vazn vektorini $\omega = \omega - x^{(i)}$ sifatida yangilanadi.
- Agar $y_p^{(i)} = y^{(i)}$ bo'lsa, ω uchun yangilanishlar talab qilinmaydi.

4-qadam: 2-bosqichga o'tiladi va keyingi ma'lumotlar qayta ishlanadi.

5-qadam: Barcha ma'lumotlar to'g'ri tasniflangach, hisoblash to'xtatiladi.

Perseptron faqat ikkita sinfni to'g'ri tasniflashi mumkin, agar vazn vektori ω mavjud bo'lsa va bu ikkita sinfni to'g'ri ajratib tursa. Bunday holatlarda *Perseptron Yaqinlashish* teoremasi yaqinlashishni kafolatlaydi.

Yuqorida o'rganilgan Perseptronda o'rganish algoritmi bo'yicha nazariy bilimlarimizni Python dasturlash tilida amalga oshiramiz:

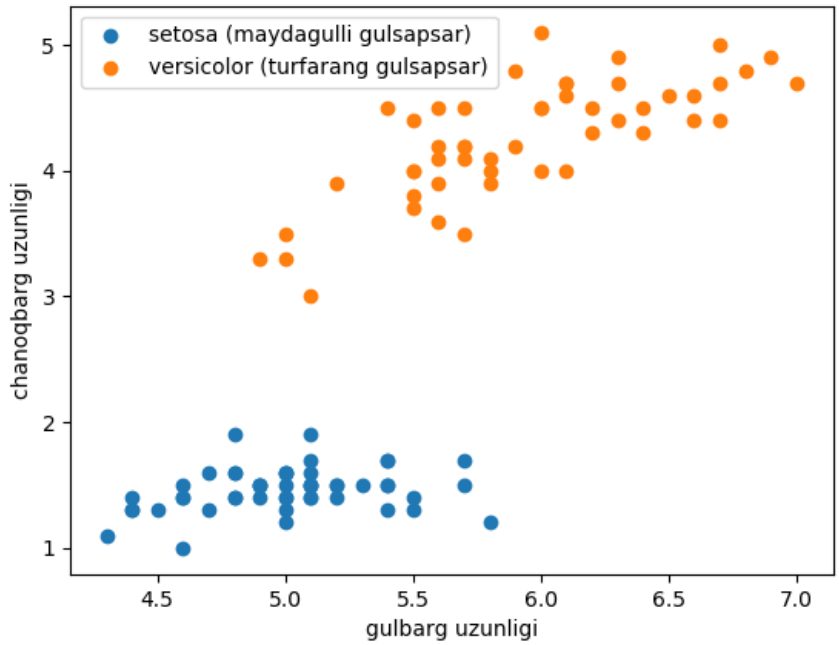
```
# Masalaning qo'yilishi: Perseptronda o'rganish algoritmi
    asosida Python-da namunaviy dastur tuzing
# Yechim: Perceptron-ni amalga oshirish uchun ko'rib
    chiqadigan ma'lumotlar to'plami -- "Iris gullari to'plami
    ". Ushbu ma'lumotlar to'plamida gulni tavsiflaydigan va
    ularni 3 sinfdan biriga tegishli deb tasniflaydigan 4 ta
    kiritma xususiyatli vektorlar mavjud. Biz "Iris-virginica"
     sinfiga tegishli ma'lumotlar to'plamining oxirgi 50
    qatorini olib tashlaymiz va faqat 2-sinf "Iris-setosa" va
    "Iris-versicolor" dan foydalanamiz, chunki bu sinflar bir-
    biridan chiziqli ajraladi va algoritm lokal minimum tomon
    yaqinlashadi.
#
# Kutubxonani yuklash
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
8
9  # Ma'lumotlarni yuklovchi funksiya
10 def load_data():
11   URL_='https://archive.ics.uci.edu/ml/machine-learning-
       databases/iris/iris.data'
12   data = pd.read_csv(URL_, header = None)
13   print(data)
14
15   # bir-biridan chiziqli ajraluvchi ma'lumotlarnigina
       qaraymiz
16   data = data[:100]
17   data[4] = np.where(data.iloc[:, -1]=='Iris-setosa', 0, 1)
18   data = np.asmatrix(data, dtype = 'float64')
19   return data
20
21 # Ma'luotlarni yuklash
22 data = load_data()
23
24 # Ma'lumotlar bazasini ikkita xususiyat bilan vizual ravishda
       ko'rib chiqsak, biz ma'lumotlar bazasini ular orasidan to
     'g'ri chiziq chizish orqali aniq ajratish mumkinligini ko'
     rishimiz mumkin.
25
26 # Bizning maqsadimiz bu chiziqni aniqlovchi va ushbu ma'
     lumotlarning barchasini to'g'ri tasniflovchi algoritm
     yozish.
27
28 plt.scatter(np.array(data[:50,0]), np.array(data[:50,2]),
     marker='o', label='setosa')
29 plt.scatter(np.array(data[50:,0]), np.array(data[50:,2]),
     marker='o', label='versicolor')
30 plt.xlabel('gulbarg uzunligi')
31 plt.ylabel('chanoqbarg uzunligi')
32 plt.legend()
33 plt.show()
34
35 #------------------------------------------------------------
36 # Izoh: Dasturning ushbu qismini ishga tushirib, bitta rasmda
       ikki xil gul uchun grafiklarga ega bo'lamiz.
37 #------------------------------------------------------------
```

Listing 3.1: Perseptronda o'rganish algoritmi va uni Python-da dasturlash

Endi biz yuqorida aytib o'tilgan algoritmni qanday bo'lsa, shunday bajaramiz va qanday ishlashini ko'rib chiqamiz. Bizda 4 ta xususiyat mavjud va shuning uchun har bir xususiyat bilan bog'liq 4 ta vazn mavjud. Esda tutingki, biz $x_0 = 1$ ga teng bo'lgan bias had kiritdik va uni 5 vazn qilamiz.

Biz iteratsiyalar sonini 10 ta etib belgiladik. Bu algoritm tomonidan o'rganilgan ω kabi tizim parametrlaridan farqli o'laroq, giperparametrlardan



Rasm 3.3: *3.1-listda berilgan dastur bo'yicha olingan rasm.*

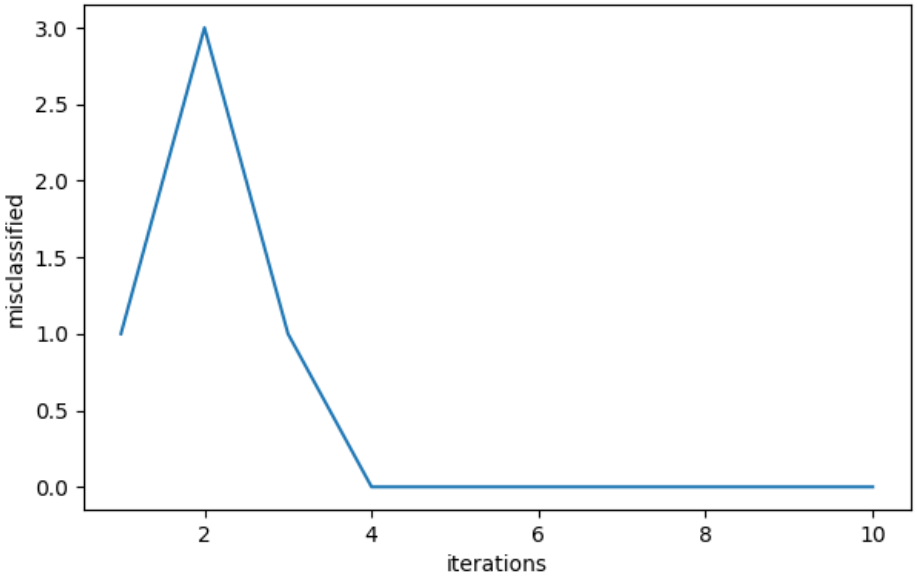
biri hisoblanadi. Har bir iteratsiyada algoritm barcha ma'lumotlar nuqtalari uchun sinfni (0 yoki 1) hisoblaydi va har bir noto'g'ri tasniflash bilan vaznlarni yangilaydi.

Agar namuna noto'g'ri tasniflangan bo'lsa, unda vaznlar qarama-qarshi yo'nalishda siljigan delta tomonidan yangilanadi. Shunday qilib, agar namunani yana tasniflash kerak bo'lsa, natija "kamroq noto'g'ri" bo'ladi. Biz har qanday yorliqni '0' (Iris-setosa) ga '1' (Iris-versicolor) bo'lish uchun tasniflaymiz.

```
# Masalaning qo'yilishi: Perseptronda o'rganish algoritmi
   asosida Python-da namunaviy dastur tuzing
# Yechim:

def perceptron(data, num_iter):
  features = data[:, :-1] # data-dagi oxirgi ustun kirmaydi
  labels = data[:, -1] # data-dagi oxirgi ustun

  # elementlari nol bo'lgan vazn vektori, w ning 1 ta qator
   va 4+1=5 ta ustuni bor
  w = np.zeros(shape=(1, features.shape[1]+1))

```

```
  misclassified_ = []

  for epoch in range(num_iter):
    misclassified = 0
    for x, label in zip(features, labels):
      x = np.insert(x,0,1)
      y = np.dot(w, x.transpose())
      target = 1.0 if (y > 0) else 0.0

      delta = (label.item(0,0) - target)

      if(delta): # misclassified
        misclassified += 1
        w += (delta * x)

    misclassified_.append(misclassified)
  return (w, misclassified_)

num_iter = 10
w, misclassified_ = perceptron(data, num_iter)

# Keling, har bir iteratsiyadagi tasniflanmagan namunalar
   sonini aniqlaylik. Algoritm 4-chi iteratsiyada
   birlashishini ko'ramiz. ya'ni, barcha namunalar ma'
   lumotlarning 4-o'tishida to'g'ri tasniflanadi.

epochs = np.arange(1, num_iter+1)
plt.plot(epochs, misclassified_)
plt.xlabel('iterations')
plt.ylabel('misclassified')
plt.show()

#-------------------------------------------------------------
# Izoh: Qayd etish kerakki, bir qatlamli perseptron faqat ma'
   lumotlar to'plami chiziqli ajraladigan bo'lsagina ishlaydi
   . Algoritm faqat Ikkilik tasniflash muammolari uchun
   ishlatiladi. Shu bilan birga, bitta sinfga bitta
   perseptronni kiritish orqali ko'p sinfli tasniflash
   muammosini yechish uchun algoritmni kengaytirishimiz
   mumkin. Ya'ni, har bir perseptron ushbu sinfga tegishli
   yoki yo'qligini bildiruvchi 0 yoki 1 natijani beradi.
#-------------------------------------------------------------
```

Listing 3.2: Perseptronda o'rganish algoritmi va uni Python-da dasturlash (davomi)



Rasm 3.4: *3.2-listda berilgan dastur bo'yicha olingan rasm.*

3.2.1 Peseptronda o'rganish cheklovlari

Perseptronda o'rganish qoidasi faqatgina kiritma ma'lumotlar fazosida chiziqli ajratilgan sinflarnigina ajratishga mo'ljallangan xolos. Hattoki, bazaviy narsalardan bo'lgan **XOR** kirish mantig'ini ham perseptronda o'rganish qoidasi orqali bajarib bo'lmaydi. Keling, shuni yaqqolroq ko'rib chiqaylik.

XOR mantig'i uchun kirish va unga mos keluvchi yorliq yoki sinflar quyida sanab o'tilgan:

- $x_1 = 1,\ x_2 = 0 \quad y = 1$
- $x_1 = 0,\ x_2 = 1 \quad y = 1$
- $x_1 = 1,\ x_2 = 1 \quad y = 0$
- $x_1 = 0,\ x_2 = 0 \quad y = 0$

Aytaylik, vazn vektori $\omega \to \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$ ko'rinishda berilsin, bu yerda vazn vektorining birinchi komponenti biasga to'g'ri keladi. Xuddi shunday, barcha kiritma vektorlarning birinchi komponenti 1 bo'lsin.

- $x_1 = 1,\ x_2 = 0,\ y = 1$ bo'lganda prognozlash $\omega^T x = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} =$

0 bo'ladi. $\omega^T x = 0$ ekanligi sababidan, ma'lumotlar nuqtasi 0 deb tasniflanishi kerak, bu esa voqelikdagi 1 sinfiga mos kelmaydi. Demak, perseptron qoidasi bo'yicha yangilangan vazn vektori quyidagicha bo'lishi kerak:

$$\omega \to \omega + x = \begin{pmatrix}0\\0\\0\end{pmatrix} + \begin{pmatrix}1\\1\\0\end{pmatrix} = \begin{pmatrix}1\\1\\0\end{pmatrix}.$$

- $x_1 = 0$, $x_2 = 1$, $y = 1$ bo'lganda prognozlash $\omega^T x = \begin{pmatrix}1 & 1 & 0\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} =$ 1 bo'ladi. $\omega^T x > 0$ ekanligi sababidan, ma'lumotlar nuqtasi 1 deb tasniflansa to'g'ri bo'ladi. Demak, vazn vektorini yangilashga hojat yo'q va $\begin{pmatrix}1\\1\\0\end{pmatrix}$ ko'rinishda qoladi.

- $x_1 = 1$, $x_2 = 1$, $y = 0$ bo'lganda prognozlash $\omega^T x = \begin{pmatrix}1 & 1 & 0\end{pmatrix}\begin{pmatrix}1\\1\\1\end{pmatrix} =$ 2 bo'ladi. $\omega^T x = 2$ ekanligi sababidan, ma'lumotlar nuqtasi 1 deb tasniflanishi kerak, bu esa voqelikdagi 0 sinfiga mos kelmaydi. Demak, yangilangan vazn vektori quyidagicha bo'lishi kerak:

$$\omega \to \omega - x = \begin{pmatrix}1\\1\\0\end{pmatrix} - \begin{pmatrix}1\\1\\1\end{pmatrix} = \begin{pmatrix}0\\0\\-1\end{pmatrix}.$$

- $x_1 = 0$, $x_2 = 0$, $y = 0$ bo'lganda prognozlash $\omega^T x = \begin{pmatrix}0 & 0 & -1\end{pmatrix}\begin{pmatrix}1\\0\\0\end{pmatrix} =$ 0 bo'ladi. $\omega^T x = 0$ ekanligi sababidan, ma'lumotlar nuqtasi 0 deb tasniflansa to'g'ri bo'ladi. Demak, vazn vektori ω uchun yangilashga hojat yo'q.

Shunday qilib, ma'lumot nuqtalari orqali birinchi o'tishdan keyin vazn vektori $\omega = \begin{pmatrix}0 & 0 & -1\end{pmatrix}^T$ bo'ladi. Yangilangan ushbu vazn vektori ω ga asoslanib, nuqtalar qanchalik tasniflanganligini ko'rib chiqamiz.

- Ma'lumot nuqtasi 1 uchun, $\omega^T x = \begin{pmatrix}0 & 0 & -1\end{pmatrix}\begin{pmatrix}1\\1\\0\end{pmatrix} = 0$, va bu nuqta yanglish tarzda 0 sinfiga kiritiladi.

- Ma'lumot nuqtasi 2 uchun, $\omega^T x = \begin{pmatrix} 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = -1$, va bu nuqta yanglish tarzda 0 sinfiga kiritiladi.

- Ma'lumot nuqtasi 3 uchun, $\omega^T x = \begin{pmatrix} 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = -1$, va bu nuqta to'g'ri tasniflanadi va 0 sinfiga kiritiladi.

- Ma'lumot nuqtasi 4 uchun, $\omega^T x = \begin{pmatrix} 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = 0$, va bu nuqta to'g'ri tasniflanadi va 0 sinfiga kiritiladi.

Oldingi tasniflarga asoslanib, birinchi iteratsiyadan keyin Perseptron algoritmi faqat manfiy sinfni to'g'ri tasniflashga muvaffaq bo'lganligini ko'ramiz. Agar Perseptronda o'rganish qoidasini yana bir bor ma'lumot nuqtalariga qo'llasak, ikkinchi o'tishdagi vazn vektorining yangilanishi quyidagicha bo'ladi:

- Ma'lumot nuqtasi 1 uchun, $\omega^T x = \begin{pmatrix} 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = 0$, va bu nuqta yanglish tarzda 0 sinfiga kiritiladi. Shunday ekan, perseptron qoidasi bo'yicha yangilangan vazn vektori quyidagicha bo'lishi kerak:

$$\omega \to \omega + x = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}.$$

- Ma'lumot nuqtasi 2 uchun, $\omega^T x = \begin{pmatrix} 1 & 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = 0$, va bu nuqta yanglish tarzda 0 sinfiga kiritiladi. Shunday ekan, perseptron qoidasi bo'yicha yangilangan vazn vektori quyidagicha bo'lishi kerak:

$$\omega \to \omega + x = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}.$$

- Ma'lumot nuqtasi 3 uchun, $\omega^T x = \begin{pmatrix} 2 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = 3$, va bu nuqta yanglish tarzda 1 sinfiga kiritiladi. Shunday ekan, perseptron qoidasi

bo'yicha yangilangan vazn vektori quyidagicha bo'lishi kerak:

$$\omega \to \omega - x = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}.$$

- Ma'lumot nuqtasi 4 uchun, $\omega^T x = \begin{pmatrix} 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = 1$, va bu nuqta yanglish tarzda 1 sinfiga kiritiladi. Shunday ekan, perseptron qoidasi bo'yicha yangilangan vazn vektori quyidagicha bo'lishi kerak:

$$\omega \to \omega - x = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}.$$

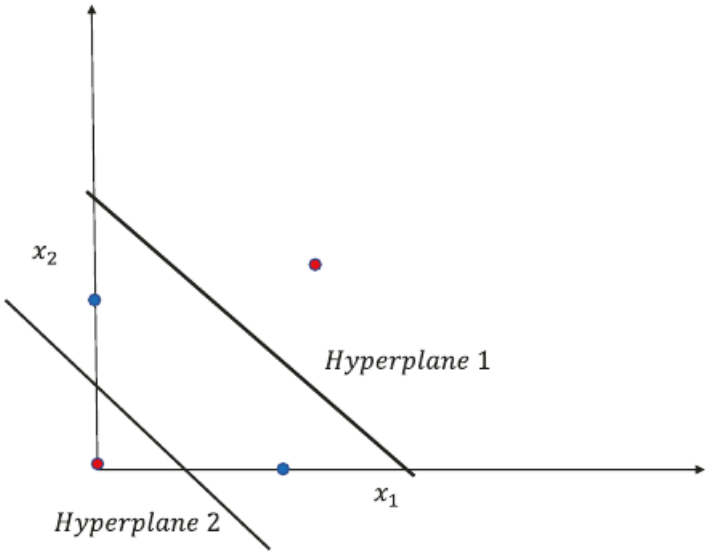
Ikkinchi o'tishdan keyingi vazn vektori $\begin{pmatrix} 0 & 0 & -1 \end{pmatrix}^T$, bu birinchi o'tishdan keyingi vazn vektori bilan bir xil. Perseptronda o'rganish birinchi va ikkinchi bosqichlarida o'tkazilgan kuzatuvlardan ko'rinib turibdiki, ma'lumot nuqtalaridan qancha o'tishimizdan qat'iy nazar, har doim bir xil vazn vektori $\begin{pmatrix} 0 & 0 & -1 \end{pmatrix}^T$ ga duch kelamiz. Avval aytib o'tganimizdek, bu vazn vektori faqat manfiy sinfni to'g'ri tasniflashi mumkin va shuning uchun biz Perseptron algoritmi har doim XOR mantig'ini modellashtira olmasligi haqida ishonch bilan xulosa qila olamiz.

3.2.2 Nochiziqlilik zarurati

Ko'rib turganimizdek, Perseptron algoritmi tasniflash uchun faqat chiziqli qarorlar chegarasini o'rganishi mumkin va shu sababli qaror chegarasida no-chiziqlik zarur bo'lganda masalani hal qila olmaydi. XOR masalasi yordamida biz Perseptron ikkita sinfni to'g'ri ajratishga qodir emasligini ko'rdik.

Ikki sinfni ajratish uchun ikkita gipertekislikka ega bo'lishimiz kerak (3.5-rasmda ko'rsatilgandek) va Perseptron algoritmi orqali o'rganilgan bitta gipertekislik zarur tasnifni ta'minlash uchun yetarli emas. 3.5-rasmda ikkita gipertekislik chiziqlari orasidagi ma'lumot nuqtalari musbat sinfga, qolgan ikkita ma'lumot nuqtalari manfiy sinfga tegishli. Ikki sinfni ajratish uchun ikkita gipertekislik talab qilish bitta no-chiziqli tasniflagichga ega bo'lish bilan ekvivalent hisoblanadi.

Ko'p qatlamli Perseptron ("multi-layer Perceptron", MLP) yashirin qatlamlarga nochiziqlilikni kiritish orqali sinflar o'rtasida nochiziqli ajralishni



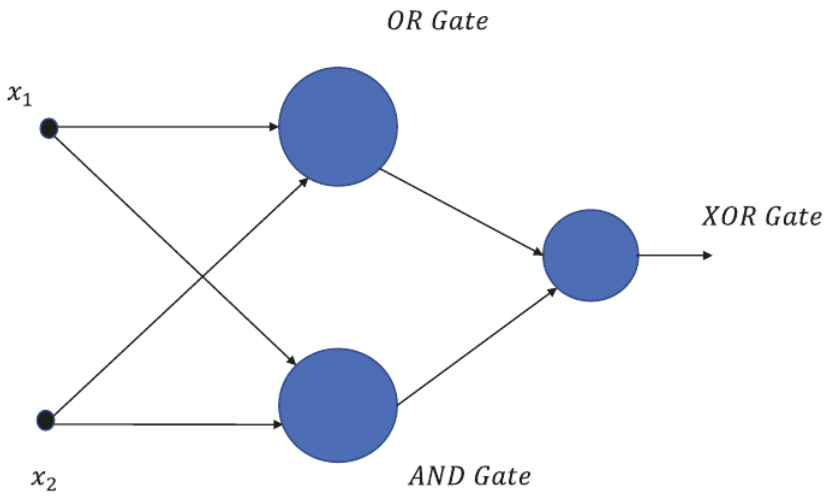
Rasm 3.5: *Ikkita gipertekislik orqali ikki sinfni ajratuvchi XOR masalasi.*

ta'minlaydi. Esda tutingki, Perseptron 0 yoki 1 natijalarini qabul qilingan umumiy ma'lumotlarga asoslanib chiqarganda, chiqish uning kiritilishining chiziqli bo'lmagan funksiyasi hisoblanadi. Ko'p qatlamli Perseptronning vaznlarini o'rganish paytida aytilgan va qilinadigan hamma narsaga Perseptronda o'rganish qoidasi orqali erishish mumkin emas.

3.6-rasmda XOR mantig'i ko'p qatlamli Perceptron to'ri orqali amalga oshirilgan. Agar bizda ikkita Perseptrondan iborat yashirin qatlam bo'lsa, bittasi *OR* mantig'ini bajarishi mumkin boshqasi *AND* mantig'ini bajarishga qodir bo'lsa, unda butun tarmoq XOR mantig'ini amalga oshirishi mumkin. OR va AND mantiqiy tushunchalari Perseptronda o'rganish qoidasidan foydalanib o'qitilishi mumkin. Biroq, Perseptronda o'rganish qoidasi bo'yicha tarmoqni umuman o'qib bo'lmaydi. Agar biz XOR kirishining oxirgi qismiga qarasak, u kirishning chiziqli bo'lmagan funksiyasi bo'lib, no-chiziqli qaror chegarasini hosil qiladi.

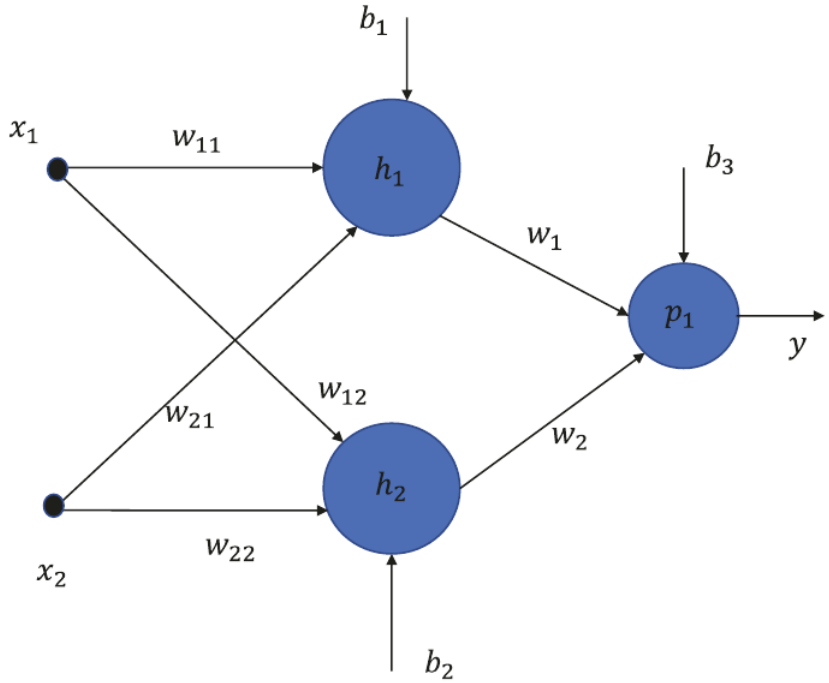
3.2.3 Nochiziqlilik uchun Yashirin qatlamli Perseptron faollashtirish funksiyasi

Agar yashirin qatlamlarning faollashtirish funksiyalarini chiziqli qilsak, u holda oxirgi neyrondan chiqish ham chiziqli bo'ladi va shuning uchun biror bir nochiziqli qaror chegaralarini o'rganib bo'lmaydi. Buni tasvirlash uchun XOR funksiyasini chiziqli faollashtirish funksiyalariga ega bo'lgan yashirin



Rasm 3.6: *XOR mantiqiy amalini ko'p qatlamli Perseptron to'ri yordamida amalga oshirish.*

qatlam birliklari orqali bajarishga harakat qilaylik.



Rasm 3.7: *Ikki qatlamli Perseptron to'rida chiziqli chiqishning yashirin qatlamlari.*

3.7-rasmda bitta yashirin qatlamga ega bo'lgan ikki-qatlamli Perseptron to'ri ko'rsatilgan. Yashirin qatlam ikkita neyron bo'linmasidan iborat. Yashirin bo'linmada faollashtirish chiziqli bo'lganda biz to'rning umumiy natijalariga qaraymiz:

- Yashirin bo'linma h_1 ning chiqishi $h_1 = \omega_{11}x_1 + \omega_{21}x_2 + b_1$
- Yashirin bo'linma h_2 ning chiqishi $h_2 = \omega_{12}x_1 + \omega_{22}x_2 + b_2$
- Chiqish bo'linma p_1 dan chiqish $p_1 = \omega_1(\omega_{11}x_1+\omega_{21}x_2+b_1)+\omega_2(\omega_{12}x_1+\omega_{22}x_2+b_2)+b_3 = (\omega_1\omega_{11}+\omega_2\omega_{12})x_1+(\omega_1\omega_{21}+\omega_2\omega_{22})x_2+\omega_1b_1+\omega_2b_2+b_3$

Yuqorida aytilganidek, to'rning yakuniy chiqishi, ya'ni p_1 bo'linmaning chiqishi - bu kiritmarning chiziqli funksiyasi va shuning uchun to'r sinflar o'rtasida nochiziqli ajralishni yarata olmaydi.

Agar yashirin qatlam tomonidan hosil qilingan chiziqli chiqish o'rniga

$$f(x) = 1/(1 + e^{-x})$$

ko'rinishidagi faollashtirish funksiyasini kiritsak, u holda yashirin bo'linma h_1 dan chiqish $h_1 = 1/\left(1 + e^{-(\omega_{11}x_1+\omega_{21}x_2+b_1)}\right)$. Xuddi shunday, yashirin bo'linma h_2 dan chiqish $h_2 = 1/\left(1 + e^{-(\omega_{12}x_1+\omega_{22}x_2+b_2)}\right)$. Chiqish bo'linmasi p_1 dan chiqish esa

$$p_1 = \omega_1/\left(1 + e^{-(\omega_{11}x_1+\omega_{21}x_2+b_1)}\right) + \omega_2/\left(1 + e^{-(\omega_{12}x_1+\omega_{22}x_2+b_2)}\right) + b_3.$$

Ayon bo'ldiki, yuqorida keltirilgan chiqish natijasi kirish qismida chiziqli emas va shuning uchun tasniflash masalasi uchun chiziqli gipertekislikdan foydalanish o'rniga murakkabroq nochiziqli qarorlar chegaralarini o'rganishi mumkin. Yashirin qatlamlar uchun yozilgan faollashtirish funksiyasi *sigmoid funksiya* deb ataladi va biz bu haqda keyingi boblarda batafsil muhokama qilamiz.

3.2.4 Neyron/Perseptron uchun turli faollashtirish funksiyalari

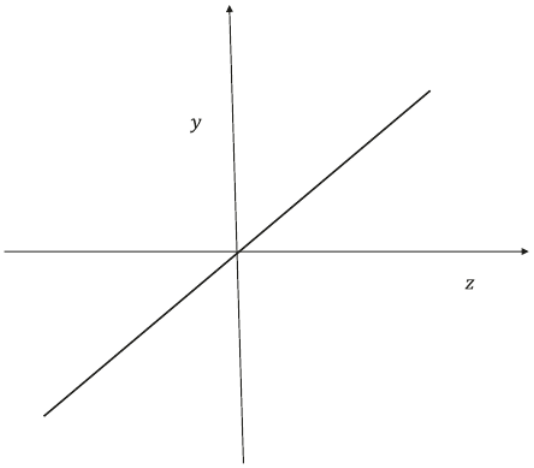
Neyron to'rlari uchun bir nechta faollashtirish funksiyalari mavjud bo'lib, ulardan foydalanish masalaning qo'yilishiga va neyron to'rining topologiyasiga qarab farq qiladi. Ushbu bo'limda biz bugungi kunda sun'iy neyron to'rlarida ishlatiladigan barcha faollashtirish funksiyalarini muhokama qilamiz.

Chiziqli faollashtirish funksiyasi

Chiziqli neyrondan chiqish chiziqli ravishda uning kirishiga bog'liq. Agar neyron uchta kiritma x_1, x_2 va x_3 ni qabul qilsa, u holda chiziqli neyronning chiqishi $y = \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + b$, bu yerda ω_1, ω_2 va ω_3 mos ravishda x_1, x_2 va x_3 kiritmalar uchun *sinaptik vaznlar* va b - neyron bo'linmasidagi bias.

Chiqishni vektorli ko'rinishda $y = \omega^T x + b$ kabi ifodalashimiz mumkin.

Agar $\omega^T x + b = z$ deb olsak, mutloq kiritma z uchun chiqish 3.8-rasmda ko'rsatilgandek bo'ladi.



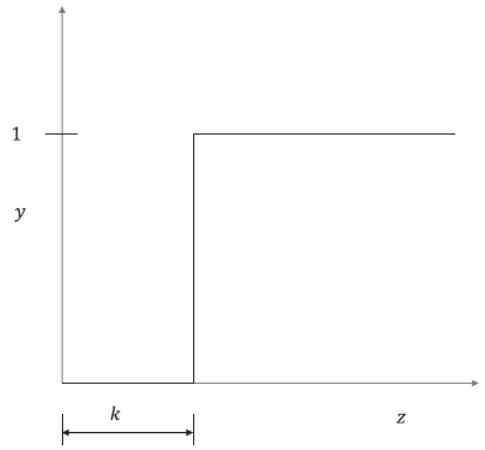
Rasm 3.8: *$\omega^T x + b = z$ kiritma uchun chiziqli faollashtirish funksiyasi.*

Ikkilik chegarali faollashtirish funksiyasi

Ikkilik chegarali neyronda (3.9-rasmga qarang), agar neyronga berilgan mutloq kiritma belgilangan chegaradan oshsa, u holda neyron faollashadi; ya'ni chiqish 1 bo'ladi. Agar neyronga berilgan mutloq chiziqli kiritma $z = \omega^T x + b$ bo'lsa va k neyron faollashadigan chegara bo'lsa, u holda neyron faollashadi:

$$y = 1, \ agar \ z > k$$
$$y = 0, \ agar \ z \leq k$$

Odatda, ikkilik chegarali neyronlar 0 chegarada faollashishga sozlangan bo'ladi va bunda bias moslashtiriladi. Neyron faollashishi uchun $\omega^T x + b > k \Rightarrow \omega^T x + (b - k) > 0$ shart bajarilishi kerak.



Rasm 3.9: *Ikkilik chegarali neyron.*

Sigmoid faollashtirish funksiyasi

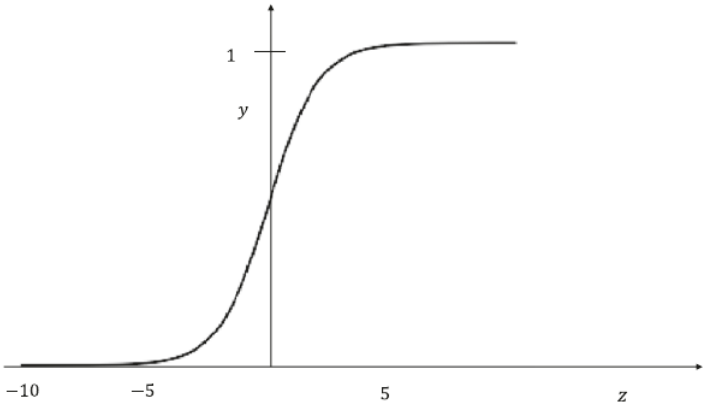
Sigmoid neyron uchun kirish-chiqish munosabatlari quyidagicha ifodalanadi:

$$y = \frac{1}{1 + e^{-z}},$$

bu yerda $z = \omega^T x + b$ - sigmoid faollashtirish funksiyasiga berilgan mutloq kiritma.

- Agar sigmoid funksiyaga mutloq kiritma z juda katta musbat son bo'lsa, $e^{-z} \to 0$, u holda $y \to 1$.
- Agar sigmoid funksiyaga mutloq kiritma z juda katta manfiy son bo'lsa, $e^{-z} \to \infty$, u holda $y \to 0$.
- Agar sigmoid funksiyaga mutloq kiritma z ning qiymati 0 bo'lsa, u holda $e^{-z} = 1$ va natijada $y = 1/2$.

3.10-rasmda sigmoid faollashtirish funksiyasining kirish-chiqish munosabatlari ko'rsatilgan. Sigmoid faollashtirish funksiyasiga ega bo'lgan neyronning chiqishi juda silliq va uzluksiz hosilalarni beradi, ular neyron to'rini o'qitishda yaxshi ishlaydi. Sigmoid faollashtirish funksiyasining chiqishi 0 dan 1 gacha oraliqda o'zgaradi. Bunda 0 dan 1 gacha oraliqda uzluksiz qiymatlarni berish imkoniyati mavjudligi sababli, sigmoid funksiya odatda ikkilik tasniflash uchun berilgan sinfga nisbatan ehtimollikni chiqarish uchun ishlatiladi. Yashirin qatlamlarda sigmoid faollashtirish funksiyalari nochiziqlilikni joriy etadi, shunda model yanada murakkab xususiyatlarni o'rganishi mumkin.



Rasm 3.10: *Sigmoid faollashtirish funksiyasi.*

SoftMax faollashtirish funksiyasi

SoftMax faollashtirish funksiyasi sigmoid funksiyaning umumlashgan ko'rinishi bo'lib, ko'p sinfli tasniflash masalalari uchun ko'proq qo'l keladi. Agar chiqish sinflari soni k bo'lsa va i-sinf uchun vazn vektori $\omega^{(i)}$ bo'lsa, kiritma vektor $x \in \mathbb{R}^{n\times 1}$ bo'lgan holda berilgan i-th sinf uchun taxmin qilingan ehtimollik quyidagicha beriladi:

$$P(y_i = 1/x) = \frac{e^{\omega^{(i)T}x+b^{(i)}}}{\sum_{j=1}^{k} e^{\omega^{(j)T}x+b^{(j)}}},$$

bu yerda $b(i)$ - SoftMax-ning har bir chiqish birligi uchun bias hadi.

Sigmoid funksiya va ikki sinfli SoftMax funksiyasi o'rtasidagi bog'liqlikni topishga harakat qilaylik. Aytaylik, ikkita sinf y_1 va y_2 va ular uchun mos keladigan vazn vektorlari $\omega^{(1)}$ va $\omega^{(2)}$. Shuningdek, ular uchun mos keluvchi bias hadlar $b^{(1)}$ va $b^{(2)}$ bo'lsin. Deylik, $y_1 = 1$ ga mos keluvchi sinf musbat hisoblanadi. U holda SoftMax funksiyasining ko'rinishi quyidagicha bo'ladi:

$$\begin{aligned} P(y_1 = 1/x) &= \frac{e^{\omega^{(1)T}x+b^{(1)}}}{e^{\omega^{(1)T}x+b^{(1)}} + e^{\omega^{(2)T}x+b^{(2)}}} \\ &= \frac{1}{1 + e^{-\left(\omega^{(1)}-\omega^{(2)}\right)^T x-\left(b^{(1)}-b^{(2)}\right)}} \end{aligned}$$

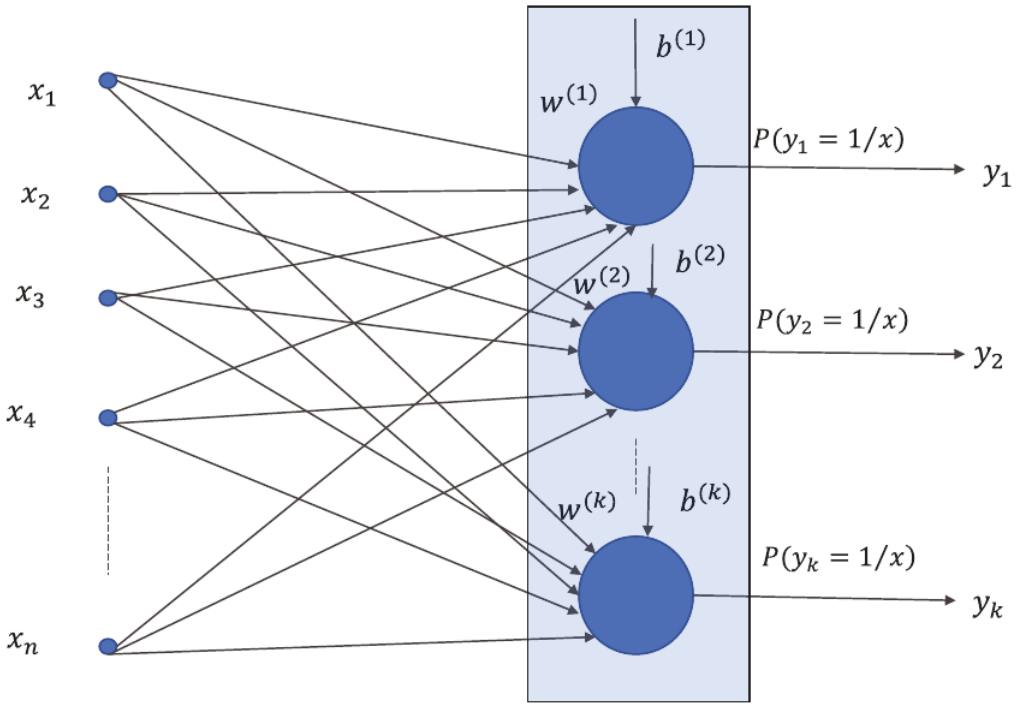
Yuqoridagi ifodadan ko'rinib turibdiki, ikki sinfli SoftMax uchun musbat sinfning ehtimolligi sigmoid faollashtirish funksiyasi bilan bir xil. Ammo, farq shundan iboratki, sigmoidda biz faqat bitta vazn to'plamini ishlatamiz, ikki sinfli SoftMaxda esa ikkita vazn to'plamidan foydalanamiz. Sigmoid faollashtirish funksiyalarida biz ikkita turli sinflar uchun turli xil vazn to'plamlarini

ishlatmaymiz va olingan vazn to'plami odatda musbat sinfning manfiy sinfga nisbatan vaznidir. SoftMax faollashtirish funksiyalarida biz har xil sinflar uchun turli xil vaznlarni olamiz.

SoftMax qatlami uchun yo'qotish funksiyasi kategoriyali *ko'ndalang entropiya* ("cross-entropy") deb ataladi va quyidagicha beriladi:

$$C = -\sum_{i=1}^{k} y_i \log P\left(y_i = 1/x\right)$$

3.11-rasmda SoftMax faollashtirish funksiyasi ko'rsatilgan.



Rasm 3.11: *SoftMax faollashtirish funksiyasi.*

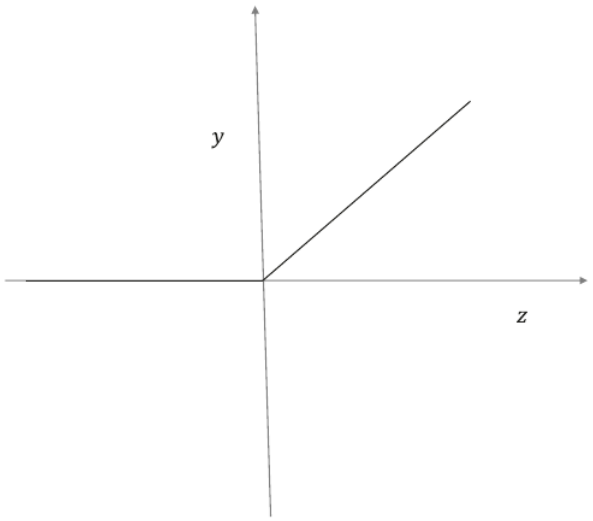
ReLU faollashtirish funksiyasi

3.12-rasmda ko'rsatilgandek to'g'rilangan chiziqli birlik ("Rectified Linear Unit" yoki ReLU) faollashtirish funksiyasida, agar umumiy kirish 0 dan katta bo'lsa, chiqish ma'lumoti neyronga mutloq kirish bilan teng bo'ladi; ammo, agar umumiy kirish miqdori 0 dan kichik yoki unga teng bo'lsa, neyronning chiqishi ham 0 ga teng bo'ladi.

ReLU bo'linmaning chiqishi quyidagicha ifodalanishi mumkin:

$$y = \max\left(0, \omega^T x + b\right).$$

ReLU chuqur o'rganishda inqilob qilgan eng muhim elementlardan biridir.



Rasm 3.12: *ReLU faollashtirish funksiyasi.*

Bu bilan hisoblashlar ancha osonlashadi. ReLU ikki tomonning eng yaxshisini jamlaydi, ya'ni mutloq kirish musbat bo'lganda gradient o'zgarmas bo'lib, boshqa joylarda esa nol gradient o'rnatiladi. Masalan, sigmoid faollashtirish funksiyasini olsak, juda katta musbat va manfiy qiymatlar uchun gradient deyarli nolga teng va shuning uchun neyron to'ri *gradient-yo'qolish* muammosiga duch keladi. Musbat mutloq kiritma uchun doimiy gradient shuni ta'minlaydiki, gradient-tushish algoritmi gradient-yo'qolish tufayli o'rganishdan to'xtab qolmaydi. Shu bilan birga, musbat bo'lmagan mutloq kiritma uchun nol chiqish nochiziqli bo'lib qoladi.

To'grilangangan chiziqli birlik faollashtirish funktsiyalarining bir nechta turlari mavjud bo'lib, Parametrli to'grilangan chiziqli birlik (PReLU) va oquvchi to'g'rilangan chiziqli birlik ("Leaky Rectified Linear Unit") kabilar shular jumlasidandir.

Normal ReLU faollashtirish funksiyasida musbat bo'lmagan kiritma qiymatlari uchun chiqish hamda gradient nolga teng, shuning uchun nol gradient tufayli o'qitish to'xtatilishi mumkin. Modelga berilgan kiritma manfiy bo'lsada, noldan farqli bo'lgan gradient olishda PReLU funksiyasi foydali bo'lishi mumkin. PReLU faollashtirish funksiyasi uchun kirish-chiqish

munosabatlari quyidagicha beriladi:

$$y = \max\left(0, \omega^T x + b\right),$$

bu yerda $z = \omega^T x + b$ – PReLU faollashtirish funksiyasiga berilgan mutloq kiritma va β o'qitish orqali o'rganilgan parametr.

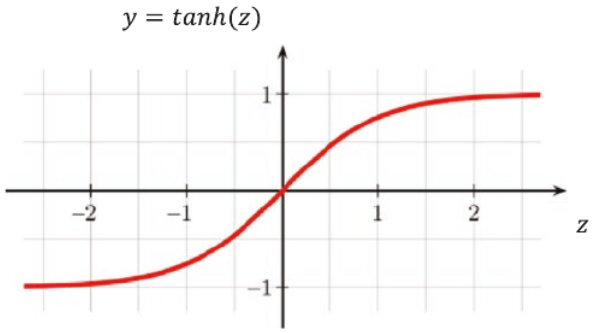
Agar $\beta = -1$ bo'lsa, u holda $y = \mid z \mid$ va faollashtirish funksiyasi ReLU mutlaq qiymati deb ataladi. Qachonki β kichik bir musbat qiymatga teng bo'lsa, odatda 0.01 atrofida bo'ladi, faollashtirish funksiyasi oquvchi ReLU deb nomlanadi.

Tanh faollashtirish funksiyasi

Tanh faollashtirish funksiyasi uchun kirish-chiqish munosabati (3.13-rasmga qarang) quyidagicha ifodalanadi:

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

bu yerda $z = \omega^T x + b$ giperbolik tangens faollashtirish funksiyasiga berilgan mutloq kiritma.



Rasm 3.13: *Tanh faollashtirish funksiyasi.*

- Agar mutloq kiritma z musbat bo'lsa, $e^{-z} \to 0$ va $y \to 1$.
- Agar mutloq kiritma z manfiy katta son bo'lsa, $e^z \to 0$ va $y \to -1$.
- Agar mutloq kiritma z nol bo'lsa, u holda $e^{-z} = 1$ va $y = 0$.

Ko'rib turganimizdek, tanh faollashtirish funksiyalari -1 va $+1$ oralig'idagi qiymatlarni qabul qilishi mumkin.

Sigmoid faollashtirish funksiyasi 0 chiqish atrofida to'yinadi. Neyron to'rni o'qitish davomida, agar qatlamdagi chiqish nolga yaqin bo'lsa, gradient yo'qoladi va o'qitishlar to'xtaydi. Tanh faollashtirish funksiyasi chiqishning -1 va $+1$ qiymatlarida to'yingan va chiqishning 0 qiymatida chekli gradientlarga ega. Shunday qilib, tanh faollashtirish funksiyalari bilan 0 chiqish atrofida bu kabi gradient-yo'qolish muammolarining oldini olish mumkin.

3.2.5 Ko'p qatlamli perseptron to'rlarda o'rganish qoidasi

Oldingi bo'limlardan birida biz Perseptron o'rganish qoidasi faqat chiziqli qaror chegaralarini o'rganishi mumkinligini ko'rdik. Nochiziqsiz murakkab qaror chegaralarini ko'p qatlamli Perseptron orqali modellashtirish mumkin; ammo, bunday modelni Perseptronda o'rganish qoidasi orqali amalga oshirish mumkin emas. Demak, boshqacha o'rganish algoritmiga ehtiyoj seziladi.

Perseptronda o'rganish qoidasida, maqsad barcha o'qitish ma'lumotlari to'g'ri tasniflangunga qadar modelning vaznlarini yangilab turishdir. Agar barcha nuqtalarni to'g'ri tasniflaydigan bunday vaznli vektor mavjud bo'lmasa, algoritm yaqinlashmaydi. Bunday hollarda, algoritmni mashq qilish uchun ketadigan o'tish (iteratsiya) sonini oldindan aniqlash yoki to'g'ri tasniflangan o'quv ma'lumotlari sonining chegarasini belgilash orqali to'xtatish mumkin.

Ko'p qatlamli Perseptron va chuqur o'rganish to'rlari uchun modelni o'qitishning eng yaxshi usuli - bu noto'g'ri tasniflash xatosi asosida qiymat funksiyasini hisoblash va keyin model parametrlariga nisbatan qiymat funksiyasini minimallashtirish. Qiymatga asoslangan o'rganish algoritmlari qiymat funksiyasini minimallashtirganligi sababli, ikkilik tasniflash uchun - odatda log-yo'qotish qiymat funksiyasi - log ehtimollik funksiyasining manfiy qiymati qo'llaniladi. Ma'lumot uchun, log-yo'qotish qiymat funksiyasi maksimal ehtimollik usulidan qanday olinishi 2-bobda "Logistik regressiya" mavzusi ostida ko'rsatilgan.

Ko'p qatlamli Perseptron to'rida yashirin qatlamlar bo'ladi va nochiziqli qaror chegaralarini o'rganish uchun faollashtirish funksiyalari ham nochiziqli bo'lishi kerak; masalan, sigmoid, ReLU, tanh va boshqalar. Ikkilik tasniflash uchun chiqish neyroni log-loss qiymat funksiyasiga va sinflar uchun ehtimollik qiymatlarini qondirish uchun sigmoid faollashtirish funksiyasiga ega bo'lishi kerak. Endi, oldingi mulohazalar bilan, XOR funksiyasini log-yo'qotish qiymat funksiyasini tuzishga va keyin uni modelning vazni va bias parametrlariga nisbatan minimallashtirishga harakat qilaylik. Bunda neyron to'rdagi barcha neyronlar sigmoid faollashtirish funksiyasiga ega deb hisoblanadi.

3.6-rasmga murojaat qilgan holda aytaylikki, h_1 yashirin bo'linmada kirish

va chiqish mos ravishda i_1 va z_1 bo'lsin. Xuddi shunday, h_2 yashirin bo'linmada kirish va chiqish mos ravishda i_2 va z_2 bo'lsin. Va nihoyat, p_1 chiqish qatlamidagi kirish va chiqish mos ravishda i_3 va z_3 bo'lsin.

$$\begin{aligned} i_1 &= \omega_{11}x_1 + \omega_{21}x_2 + b_1, \\ i_2 &= \omega_{12}x_1 + \omega_{22}x_2 + b_2, \\ z_1 &= 1/(1+e^{-i_1}), \\ z_2 &= 1/(1+e^{-i_2}), \\ i_3 &= \omega_1 z_1 + \omega_2 z_2 + b_3, \\ z_3 &= 1/(1+e^{-i_3}). \end{aligned}$$

Log-yo'qotish qiymat funksiyasini hisobga olgan holda XOR masalasi uchun umumiy qiymat funksiyasini quyidagicha aniqlash mumkin:

$$C = \sum_{i=1}^{4} -y^{(i)} \log z_3^{(i)} - (1-y^{(i)}) \log(1-z_3^{(i)}).$$

Agar barcha vazn va biaslarni bitta qilib parametr vektor θ deb hisoblash mumkin bo'lsa, $C(\theta)$ qiymat funksiyasini minimallashtirish orqali modelni o'rganishimiz mumkin:

$$\theta^{\star} = Arg_{\theta} MinC(\theta).$$

Minimal bo'lishi uchun $C(\theta)$ qiymat funksiyaning θ ga nisbatan gradienti nolga teng bo'lishi kerak (ya'ni, $\nabla C(\theta)$).

Minimalga erishish uchun gradient-tushish usullaridan foydalanish mumkin. Gradient-tushishga asoslangan yangilash qoidasi esa $\theta^{(t+1)} = \theta^{(t)} - \eta \nabla C(\theta^{(t)})$ ko'rinishda beriladi, bu yerda η o'rganish tezligi hamda $\theta^{(t+1)}$ va $\theta^{(t)}$ mos ravishda $t+1$ va t iteratsiyadagi parametr vektorlar.

Agar parametr vektoridagi individual vaznni hisobga olsak, gradient-tushish yangilash qoidasi quyidagicha bo'ladi:

$$\omega_k^{(t+1)} = \omega_k^{(t)} - \eta \frac{\partial C\left(\omega_k^{(t)}\right)}{\partial \omega_k}, \ \forall \omega_k \in \theta.$$

Gradient vektorni chiziqli yoki logistika regressiyasidagi kabi hisoblash oson emas, chunki neyron to'rlarda vaznlar ierarxik tartibda bo'ladi. Shu bilan birga, hosila olishdagi zanjir qoidasi vaznlar bo'yicha (shu jumladan biaslar bo'yicha ham) xususiy xosilalarni hisoblashda ba'zi bir uslubiy soddalashtirishlarga olib keladi.

Ushbu usul *ortga tarqalish* ("backpropagation") deb nomlanadi va u gradient hisoblashda soddalashtirishni ta'minlaydi.

3.2.6 Gradientni hisoblashda ortga tarqalish metodi

Ortga tarqalish - bu chiqish qatlamidagi xatolarning orqaga qaytishi demakdir, shuning uchun oldingi qatlamlardagi gradientlarni xosila olishdagi zanjir qoidasi yordamida osonlikcha hisoblash mumkin.

Keling, bitta o'qitish misolini ko'rib chiqamiz va XOR tuzilishini (3.7-rasmga qarang) hisobga olgan holda ortqa tarqalish metodi bilan ishlaymiz. Kiritma ma'lumot $x = \begin{pmatrix} x_1 & x_2 \end{pmatrix}^T$ va unga mos keluvchi sinf y bo'lsin. Shunday qilib, bitta qayd etish uchun qiymat funksiyasi quyidagicha bo'ladi:

$$C = -y \log z_3 - (1-y)\log(1-z_3),$$
$$\frac{\partial C}{\partial \omega_1} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial \omega_1},$$
$$\frac{dC}{dz_3} = \frac{(z_3 - y)}{z_3(1-z_3)}.$$

Endi esa quyidagi ifodalarni yozamiz:

$$z_3 = \frac{1}{1+e^{-i_3}},$$
$$\frac{dz_3}{di_3} = z_3(1-z_3),$$
$$\frac{dC}{di_3} = \frac{dC}{dz_3}\frac{dz_3}{di_3} = \frac{(z_3-y)}{z_3(1-z_3)} z_3(1-z_3) = (z_3 - y).$$

Ko'rib turganimizdek, oxirgi qatlamdagi mutloq kiritmaga nisbatan qiymat funksiyasining hosilasi chiqish $(z_3 - y)$ ni baholashdagi xatolikka aynan mos keladi:

$$\frac{\partial i_3}{\partial \omega_1} = z_1,$$
$$\frac{\partial C}{\partial \omega_1} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial \omega_1} = (z_3 - y)z_1.$$

Xuddi shunday,

$$\frac{\partial C}{\partial \omega_2} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial \omega_2} = (z_3 - y)z_2,$$
$$\frac{\partial C}{\partial b_3} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial b_3} = (z_3 - y).$$

Endi esa oldingi qatlamdagi vaznlar bo'yicha qiymat funksiyasining xususiy hosilalarini hisoblab chiqamiz:

$$\frac{\partial C}{\partial z_1} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial z_1} = (z_3 - y)\omega_1.$$

$\frac{\partial C}{\partial z_1}$ ni yashirin qatlam birligi h_1 dan chiqishga nisbatan xatolik deb hisoblash mumkin. Bu xatolik chiqish birligini yashirin qatlam birligiga bog'lovchi vaznga mutanosib ravishda tarqaladi. Agar bir nechta chiqish birliklari mavjud bo'lsa, unda $\frac{\partial C}{\partial z_1}$ har bir chiqish birliklarining hissasi bo'ladi. Buni keyingi bo'limda batafsil ko'rib chiqamiz.

Xuddi shunday,

$$\frac{\partial C}{\partial i_1} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial z_1}\frac{dz_1}{di_1} = (z_3 - y)\omega_1 z_1(1 - z_1).$$

$\frac{\partial C}{\partial i_1}$ ni yashirin qatlam birligi h_1 ning mutloq kiritmasiga nisbatan xatolik deb hisoblash mumkin. Buni $z_1(1 - z_1)$ hadni $\frac{\partial C}{\partial z_1}$ ga ko'paytirish orqali hisoblash mumkin:

$$\begin{aligned}
\frac{\partial C}{\partial \omega_{11}} &= \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial z_1}\frac{dz_1}{di_1}\frac{\partial i_1}{\partial \omega_{11}} = (z_3 - y)\omega_1 z_1(1 - z_1)x_1, \\
\frac{\partial C}{\partial \omega_{21}} &= \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial z_1}\frac{dz_1}{di_1}\frac{\partial i_1}{\partial \omega_{21}} = (z_3 - y)\omega_1 z_1(1 - z_1)x_2, \\
\frac{\partial C}{\partial b_1} &= \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial z_1}\frac{dz_1}{di_1}\frac{\partial i_1}{\partial \omega_{21}} = (z_3 - y)\omega_1 z_1(1 - z_1).
\end{aligned}$$

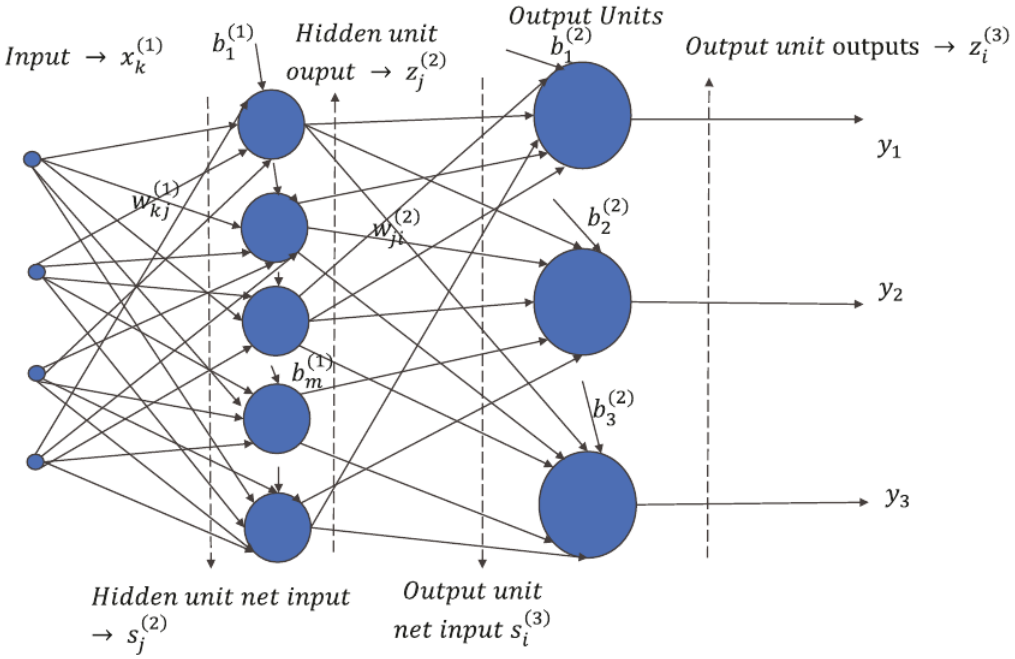
Har bir neyron birlikning kiritmasiga nisbatan qiymat funksiyasining xususiy hosilasiga ega bo'lsak, qiymat funksiyasining xususiy hosilasini kiritmaga hissa qo'shuvchi vazn bo'yicha hisoblashimiz mumkin - shunchaki shu vazn orqali keluvchi kiritmani ko'paytirishimiz kerak.

3.2.7 Gradientni hisoblashda ortqa tarqalish usulini umumlashtirish

Ushbu bo'limda biz yanada murakkab bo'lgan to'r bo'ylab ortqa tarqalish usulini umumlashtirishga harakat qilamiz. Yakuniy chiqish qatlami 3.14 rasmda tasvirlangan uchta mustaqil sigmoid chiqish birliklaridan iborat deb taxmin qilamiz. Shuningdek, matematik ifodalash va o'rganishni soddalashtirish uchun to'rni yagona yozuvga ega deb taxmin qilamiz.

Bitta kiritma yozuvning qiymat funkiyasi quyidagicha beriladi:

$$\begin{aligned}
C &= \sum_{i=1}^{3} -y_i \log P(y_i = 1) - (1 - y_i)\log(1 - P(y_i = 1)) \\
&= \sum_{i=1}^{3} -y_i \log z_i^{(3)} - (1 - y_i)\log(1 - z_i^{(3)}).
\end{aligned}$$



Rasm 3.14: *Sigmoid funksiyali mustaqil chiqish qatlamlari uchun ortqa tarqalish usulini ifodalovchi neyron to'ri.*

Oldingi ifodada y_i uchun xos bo'lgan hodisa faol yoki faolmasligiga qarab $y \in \{0, 1\}$ bo'ladi. $P(y_i = 1) = z_i^{(3)}$ bu i-sinfning taxmin qilingan ehtimolligini bildiradi.

Keling, qiymat funksiyasining vazn $\omega_{ji}^{(2)}$ ga nisbatan xususiy hosilasini hisoblaylik. Ushbu vazn neyron to'rning faqat i-chiqish bo'linmasiga ta'sir qilishi mumkin xolos:

$$\frac{\partial C}{\partial \omega_{ji}^{(2)}} = \frac{\partial C}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial s_i^{(3)}} \frac{\partial s_i^{(3)}}{\partial \omega_{ji}^{(2)}},$$

$$\frac{\partial C}{\partial z_i^{(3)}} = \frac{\left(z_i^{(3)} - y_i\right)}{z_i^{(3)}\left(1 - z_i^{(3)}\right)},$$

$$P\left(y_i = 1\right) = z_i^{(3)} = 1/\left(1 + e^{-s_i^{(3)}}\right),$$

$$\frac{\partial z_i^{(3)}}{\partial s_i^{(3)}} = z_i^{(3)}\left(1 - z_i^{(3)}\right),$$

$$\frac{\partial C}{\partial s_i^{(3)}} = \frac{\partial C}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial s_i^{(3)}} = \frac{\left(z_i^{(3)} - y_i\right)}{z_i^{(3)}\left(1 - z_i^{(3)}\right)} z_i^{(3)}\left(1 - z_i^{(3)}\right) = \left(z_i^{(3)} - y_i\right).$$

Shunday qilib, avvalgidek, i-chiqish birligi uchun mutloq kiritmaga nisbatan qiymat funksiyasining xususiy hosilasi $\left(z_i^{(3)} - y_i\right)$ ga teng, bu i-chiqish birligidagi bashorat qilishdagi xatolikdan boshqa narsa emas.

$$\frac{\partial s_i^{(3)}}{\partial \omega_{ji}^{(2)}} = z_j^{(2)}.$$

$\frac{\partial C}{\partial s_i^{(3)}}$ va $\frac{\partial s_i^{(3)}}{\partial \omega_{ji}^{(2)}}$ ni birlashtirib, quyidagi ifodalarni olamiz:

$$\frac{\partial C}{\partial \omega_{ji}^{(2)}} = \left(z_i^{(3)} - y_i\right) z_j^{(2)},$$

$$\frac{\partial C}{\partial b_i^{(2)}} = \left(z_i^{(3)} - y_i\right).$$

Yuqorida berilgan ifoda neyron to'rining so'ngi qatlamidagi vaznlar va biaslar bo'yicha qiymat funksiyasining xususiy hosilalari uchun umumlashtirilgan ifodani beradi. Endi esa quyi qatlamlardagi vaznlar va biaslarning xususiy hosilalarini hisoblab chiqamiz. Hisoblashlar bir oz ko'proqdek tuyuladi, lekin hali ham umumiy tendentsiyani davom ettiradi. $\omega_{kj}^{(1)}$ vaznga nisbatan qiymat funksiyasining xususiy hosilasini hisoblaylik. Vaznga uchta chiqish birligidagi xatoliklar ta'sir qiladi. Asosan, yashirin qatlamda j-blokning chiqishidagi xatolik, barcha chiqish birliklarining xatolik hissasi bo'lishi mumkin, chiquvchi qatlamlarni j-yashirin birligiga ulanadigan vaznlar bilan o'lchanadi. Xususiy hosilani faqat zanjir qoidasi bo'yicha hisoblab chiqamiz va u biz e'lon qilgan narsaga mos kelishini ko'raylik:

$$\frac{\partial C}{\partial \omega_{kj}^{(1)}} = \frac{\partial C}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial s_j^{(2)}} \frac{\partial s_j^{(2)}}{\partial \omega_{kj}^{(1)}},$$

$$\frac{\partial s_j^{(2)}}{\partial \omega_{kj}^{(1)}} = z_k^{(1)},$$

$$\frac{\partial z_j^{(2)}}{\partial s_j^{(2)}} = z_j^{(2)} \left(1 - z_j^{(2)}\right).$$

$\frac{\partial C}{\partial z_j^{(2)}}$ ni hisoblash juda nozik, chunki $z_j^{(2)}$ uchta chiqish birligining barchasiga ta'sir etadi:

$$\begin{aligned}\frac{\partial C}{\partial z_j^{(2)}} &= \sum_{i=1}^{3} \frac{\partial C}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial s_i^{(3)}} \frac{\partial s_i^{(3)}}{\partial z_j^{(2)}} \\ &= \sum_{i=1}^{3} \left(z_i^{(3)} - y_i\right) \omega_{ji}^{(2)}.\end{aligned}$$

$\frac{\partial s_j^{(2)}}{\partial \omega_{kj}^{(1)}}$, $\frac{\partial z_j^{(2)}}{\partial s_j^{(2)}}$ va $\frac{\partial C}{\partial z_j^{(2)}}$ larni birlashtirib, quyidagi ifodaga kelamiz:

$$\frac{\partial C}{\partial \omega_{kj}^{(1)}} = \sum_{i=1}^{3} \left(z_i^{(3)} - y_i\right) \omega_{ji}^{(2)} z_j^{(2)} \left(1 - z_j^{(2)}\right) x_k^{(1)}.$$

Umuman olganda, ko'p qatlamli neyron to'ri uchun C qiymat funksiyasining neyron birligi mutloq kiritmasi s ga hissa qo'shuvchi ma'lum bir vazn ω ga nisbatan xususiy hosilasini hisoblash uchun biz qiymat funksiyasining xususiy hosilasini mutloq kiritmaga nisbatan (ya'ni, $\frac{\partial C}{\partial s}$) hisoblashimiz kerak va keyin ω vazn bilan bog'liq bo'lgan kiritma x ni quyidagicha ko'paytiramiz:

$$\frac{\partial C}{\partial \omega} = \frac{\partial C}{\partial s}\frac{\partial s}{\partial \omega} = \frac{\partial C}{\partial s} x.$$

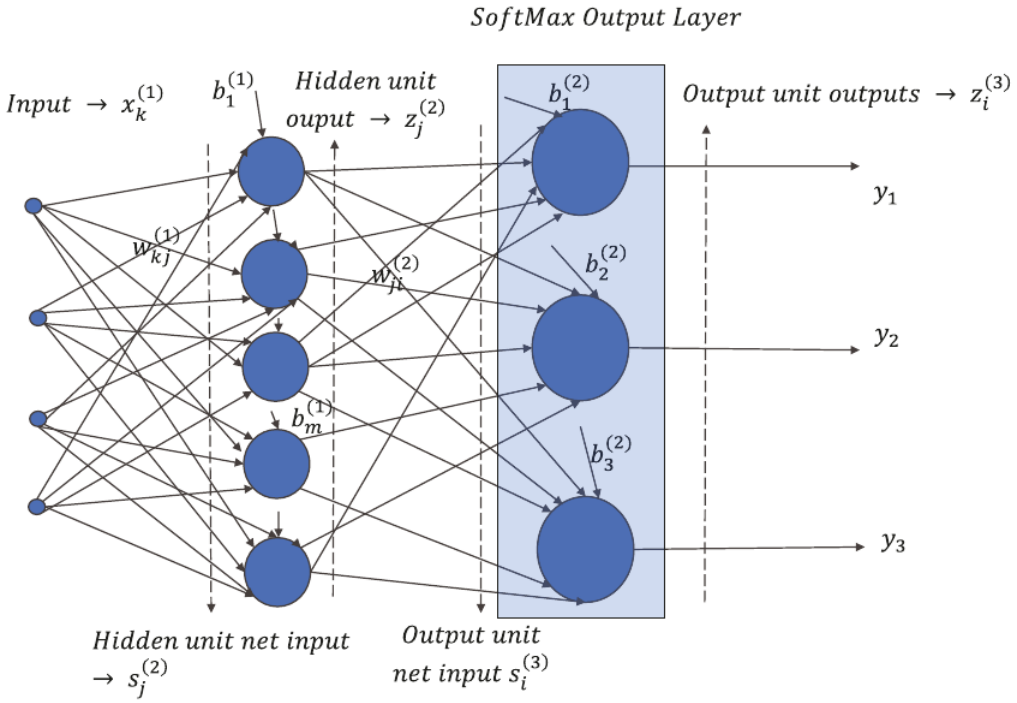
$\frac{\partial C}{\partial s}$ ni neyron birligidagi xatolik deb hisoblash mumkin va chiqish qatlamidagi xatolikni pastki qatlamlardagi neyron birliklarga o'tkazish orqali iterativ ravishda hisoblash mumkin. Ta'kidlash kerak bo'lgan yana bir narsa, yuqori qatlamli neyron birlikdagi xatolik oldingi qatlamning neyron birliklarining chiqishlari orasidagi vazn ulanishlariga mutanosib ravishda taqsimlangan. Shuningdek, qiymat funksiyaning sigmoid faollashuvchi neyron mutloq kiritmasiga nisbatan xususiy hosilasi $\frac{\partial C}{\partial s}$ ni olish uchun qiymat funksiyaning neyron chiqish z ga nisbatan xususiy hosilasi (ya'ni, $\frac{\partial C}{\partial z}$) ni $z(1-z)$ ga ko'paytirish kerak. Chiziqli neyronlar uchun bu ko'paytuvchi had 1 ga teng.

Neyron to'rlarining bu xususiyatlari gradient hisoblashlarni osonlashtiradi. Demak, neyron to'rining har bir iteratsiyadagi ortga tarqalish orqali o'rganishi mana shu zaylda kechadi.

Har bir iteratsiya oldinga va orqaga o'tish ("forward pass and backward pass") yoki ortga tarqalish orqali amalga oshiriladi. Oldinga o'tish paytida, har bir qatlamdagi har bir neyron birlikda mutloq kirish va chiqish hisoblanadi. Oldindan belgilangan chiqish va joriy mo'ljal qiymatlariga asoslanib, xatolik chiqish qatlamlarida hisoblanadi. Xatolik uni oldinga o'tish joyida hisoblangan neyron chiqishi va mavjud vaznlar bilan birlashtirish orqali ortga tarqatiladi. Ortga tarqalish orqali gradientlar iterativ ravishda hisoblab chiqiladi. Gradientlar hisoblangandan so'ng, vaznlar gradient tushish usullari bilan yangilanadi.

E'tibor bering, yuqoridagi xulosalar sigmoid faollashtirish funksiyalari uchun to'g'ri keladi. Boshqa faollashtirish funksiyalari uchun yondashuv bir xil bo'lib qoladi, amalga oshirishda faollashtirish funksiyalariga xos o'zgarishlar talab qilinadi.

SoftMax funksiyasi uchun qiymat funksiyasi mustaqil ko'p sinfli tasniflashdan farq qiladi.



Rasm 3.15: *Softmax chiqish qatlami uchun ortqa tarqalishni ko'rsatuvchi neyron to'r.*

3.15-rasmda ko'rsatilgan to'rdagi SoftMax faollashtirish qatlami uchun cross-entropy qiymat quyidagicha beriladi:

$$C = \sum_{i=1}^{3} -y_i \log P(y_i = 1) = \sum_{i=1}^{3} -y_i \log z_i^{(3)}.$$

$\omega_{ji}^{(2)}$ vazniga nisbatan qiymat funksiyasining xususiy hosilasini hisoblaylik. Endi vazn i-SoftMax birlikka mutloq kirish $s_i^{(3)}$ ga ta'sir qilishi mumkin. Biroq, oldingi to'rdagi mustaqil ikkilik faollashuvlardan farqli o'laroq, bu yerda uchchala SoftMax chiqish birliklari $z_k^{(3)}$ $\forall k \in \{1, 2, 3\}$ mutloq kirish $s_i^{(3)}$ dan ta'sirlanishi mumkin, chunki

$$z_k^{(3)} = \frac{e^{s_k^{(3)}}}{\sum_{l=1}^{3} e^{s_l^{(3)}}} = \frac{e^{s_k^{(3)}}}{\sum_{l \neq i} e^{s_l^{(3)}} + e^{s_i^{(3)}}}.$$

Demak, xususiy hosila $\frac{\partial C}{\partial \omega_{ji}^{(2)}}$ quyidagicha yozilishi mumkin:

$$\frac{\partial C}{\partial \omega_{ji}^{(2)}} = \frac{\partial C}{\partial s_i^{(3)}} \frac{\partial s_i^{(3)}}{\partial \omega_{ji}^{(2)}}.$$

Hozirgina qayd etilganidek, $s_i^{(3)}$ barcha SoftMax chiqish birliklari $z_k^{(3)}$ ga ta'sir etganligi uchun,

$$\frac{\partial C}{\partial s_i^{(3)}} = \sum_{k=1}^{3} \frac{\partial C}{\partial z_k^{(3)}} \frac{\partial z_k^{(3)}}{\partial s_i^{(3)}}.$$

$\frac{\partial C}{\partial z_k^{(3)}}$ xususiy hosilaning individual komponentalari quiyidagicha aniqlanadi:

$$\frac{\partial C}{\partial z_k^{(3)}} = \frac{-y_k}{z_k^{(3)}}.$$

- $k = i$ bo'lganda, $\frac{\partial z_k^{(3)}}{\partial s_i^{(3)}} = z_i^{(3)} \left(1 - z_i^{(3)}\right)$
- $k \neq i$ bo'lganda, $\frac{\partial z_k^{(3)}}{\partial s_i^{(3)}} = -z_i^{(3)} z_k^{(3)}$

$$\frac{\partial s_i^{(3)}}{\partial \omega_{ji}^{(2)}} = z_j^{(2)},$$

$$\begin{aligned}
\frac{\partial C}{\partial s_i^{(3)}} = \sum_{k=1}^{3} \frac{\partial C}{\partial z_k^{(3)}} \frac{\partial z_k^{(3)}}{\partial s_i^{(3)}} &= \sum_{k=i} \frac{\partial C}{\partial z_k^{(3)}} \frac{\partial z_k^{(3)}}{\partial s_i^{(3)}} + \sum_{k \neq i} \frac{\partial C}{\partial z_k^{(3)}} \frac{\partial z_k^{(3)}}{\partial s_i^{(3)}} \\
&= \frac{-y_i}{z_i^{(3)}} z_i^{(3)} \left(1 - z_i^{(3)}\right) + \sum_{k \neq i} \frac{-y_k}{z_k^{(3)}} \left(-z_i^{(3)} z_k^{(3)}\right) \\
&= -y_i \left(1 - z_i^{(3)}\right) + z_i^{(3)} \sum_{k \neq i} y_k \\
&= -y_i + y_i z_i^{(3)} + z_i^{(3)} \sum_{k \neq i} y_k \\
&= -y_i + z_i^{(3)} \sum_{k} y_k \\
&= -y_i + z_i^{(3)}
\end{aligned}$$

Ko'rinib turibdiki, qiymat funksiyasidan i-SoftMax birligiga mutloq kirish bo'yicha olingan xususiy hosila bu i-SoftMax birligidan chiqishni bashorat qilishdagi xatolik ekan. $\frac{\partial C}{\partial s_i^{(3)}}$ hamda $\frac{\partial s_i^{(3)}}{\partial \omega_{ji}^{(2)}}$ ifodalarini birlashtirib, quyidagilarga ega bo'lamiz:

$$\frac{\partial C}{\partial \omega_{ji}^{(2)}} = \frac{\partial C}{\partial s_i^{(3)}} \frac{\partial s_i^{(3)}}{\partial \omega_{ji}^{(2)}} = \left(z_i^{(3)} - y_i\right) z_j^{(2)}.$$

Xuddi shunday, i-SoftMax chiqish birligidagi bias had uchun quyidagi ifodani yozish mumkin:

$$\frac{\partial C}{\partial b_i^{(2)}} = \left(z_i^{(3)} - y_i\right).$$

Oldingi qatlamdagi $\omega_{kj}^{(1)}$ vaznga nisbatan qiymat funksiyasining xususiy hosilasi $\frac{\partial C}{\partial \omega_{kj}^{(1)}}$ ni hisoblash huddi ikkilik sinfli mustaqil neyron to'rdagi kabi ko'rinishda bo'ladi. Bu aniq, chunki neyron to'rlar faqat chiqish birliklaridagi faollashtirish funksiyalari jihatidan farq qiladi xolos va $\frac{\partial C}{\partial s_i^{(3)}}$ hamda $\frac{\partial s_i^{(3)}}{\partial \omega_{ji}^{(2)}}$ uchun olingan ifodalar bir xil bo'lib qoladi. Mashq sifatida, qiziqqan o'quvchilar $\frac{\partial C}{\partial \omega_{kj}^{(1)}} = \sum_{i=1}^{3} \left(z_i^{(3)} - y_i\right) \omega_{ji}^{(2)} z_j^{(2)} \left(1 - z_j^{(2)}\right) x_k^{(1)}$ har doim to'g'ri yoki yo'qligini tekshirishlari mumkin.

3.3 Python-da neyron to'rini boshqarish

3.3.1 Neyron to'rlari, tuzilishi, vazn va matritsalar

Biz neyron to'rlar haqida asosiy g'oyalarni Sun'iy o'rganish bo'yicha ushbu o'quv qo'llanmaning oldingi bo'limlarida tanishtirdik.

Biz biologiyada neyronlar va neyron to'rlari o'rtasidagi o'xshashlikni ta'kidladik. Shuningdek, biz juda kichik artikulyar neyron to'rlarni taqdim etdik va qaror chegaralari va XOR muammolari bilan tanishtirdik.

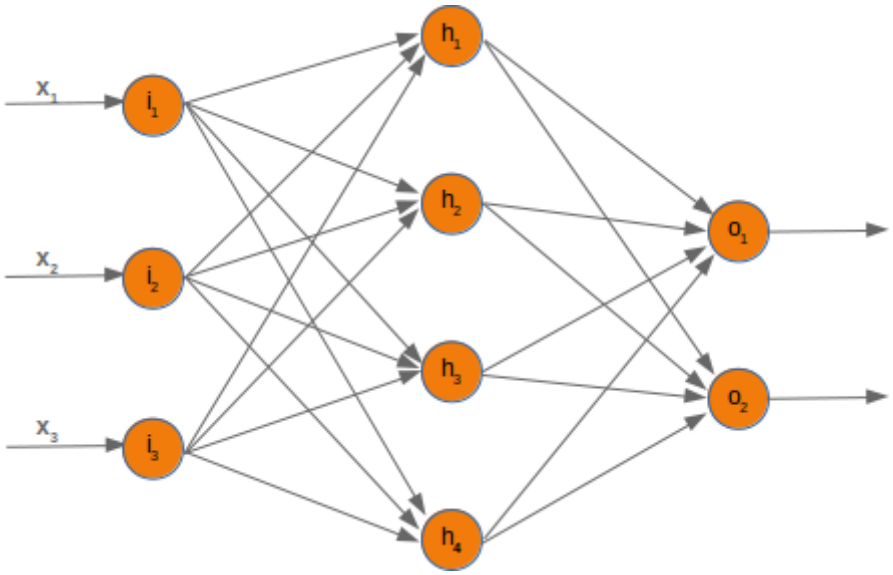
Biz hozirgacha taqdim etgan oddiy misollarda biz vaznlar neyron to'rining ajralmas qismi ekanligini ko'rdik. Bir nechta qatlamli neyron to'rini yozishni boshlashdan oldin, biz vaznlarni diqqat bilan ko'rib chiqishimiz kerak.

Vaznni qanday boshlash kerakligini va vaznni kiritma qiymatlari bilan qanday ko'paytirishni ko'rishimiz kerak.

Keyingi bo'limlarda biz uchta qatlamdan, ya'ni kirish qatlamidan, yashirin qatlamdan va chiqish qatlamidan iborat Python-da neyron to'rini yaratamiz. Ushbu neyron to'ri tuzilishini 3.16-rasmada ko'rishingiz mumkin.

Ushbu neyron to'rida uchta tugunli kirish qatlami bor i_1, i_2, i_3. Ushbu tugunlar mos ravishda x_1, x_2, x_3 kiritma qiymatlarni qabul qiladi. O'rta yoki yashirin qatlamda h_1, h_2, h_3, h_4 to'rtta tugun mavjud. Ushbu qatlamning kiritilishi kirish qatlamidan kelib chiqadi. Bir oz keyinroq bu mexanizmni muhokama qilamiz. Va nihoyat, bizning chiqish qatlamimiz o_1, o_2 ikkita tugundan iborat

Kirish qatlami boshqa qatlamlardan farq qiladi. Kirish qatlamining tugunlari passivdir. Bu shuni anglatadiki, kirish neyronlari ma'lumotlarni



Rasm 3.16: *Uch qatlamli neyron to'r tuzilishi.*

o'zgartirmaydi, ya'ni bu holatda ishlatiladigan vaznlar yo'q. Ular bitta qiymatni oladi va bu qiymatni ko'p chiqishlar uchun ko'paytiradi.

Kirish qatlami i_1, i_2 va i_3 tugunlaridan iborat. Aslida, kirish bu $\begin{pmatrix} 2 & 4 & 11 \end{pmatrix}$ kabi bir o'lchovli vektor. Bir o'lchovli vektor numpy-da quyidagicha ifodalanadi:

```
import numpy as np

input_vector = np.array([2, 4, 11])
print(input_vector)

[ 2  4 11]
```

Keyinchalik yozadigan algoritmda biz uni ustun vektoriga, ya'ni bitta ustunli ikki o'lchovli qatorga o'tkazamiz:

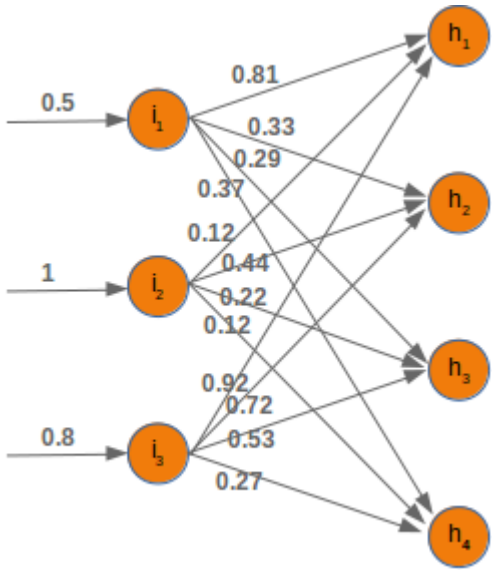
```
import numpy as np

input_vector = np.array([2, 4, 11])
input_vector = np.array(input_vector, ndmin=2).T
print("Kiritma vektor:\n", input_vector)
print("Ushbu vektorning shakli: ", input_vector.shape)
# Natijada quyidagilarni olamiz:
Kiritma vektor:
[[ 2]
[ 4]
[11]]
Ushbu vektorning shakli:  (3, 1)
```

Vazn va matritsalar

Bizga berilgan neyron to'r diagrammasidagi har bir o'qning o'ziga berilgan vazn qiymati bor. Endi biz faqat kirish va chiqish qatlami orasidagi o'qlarni ko'rib chiqamiz.

i_1 tugunga kiruvchi x_1 qiymati vaznlarning qiymatlari bo'yicha taqsimlanadi. 3.17-rasmda biz ba'zi bir namunaviy qiymatlarni qo'shdik. Ushbu qiymatlardan foydalanib, yashirin qatlamning $\begin{pmatrix} h_1 & h_2 & h_3 & h_4 \end{pmatrix}$ tugunlarga keluvchi kiritma qiymatlar $\begin{pmatrix} Ih_1 & Ih_2 & Ih_3 & Ih_4 \end{pmatrix}$ quyidagicha hisoblanishi mumkin:

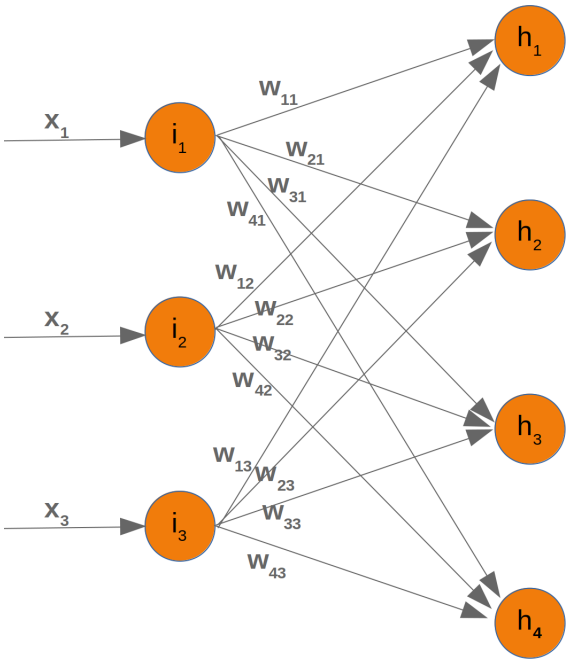
$$Ih_1 = 0.81 * 0.5 + 0.12 * 1 + 0.92 * 0.8$$
$$Ih_2 = 0.33 * 0.5 + 0.44 * 1 + 0.72 * 0.8$$
$$Ih_3 = 0.29 * 0.5 + 0.22 * 1 + 0.53 * 0.8$$
$$Ih_4 = 0.37 * 0.5 + 0.12 * 1 + 0.27 * 0.8$$



Rasm 3.17: *Uch qatlamli neyron to'r tuzilishi.*

Matritsalar va matritsalarni ko'paytirish bilan tanish bo'lgan talabalar uning qayerdan kelganini darhol ko'rishgan bo'lsa kerak. Biz berilgan neyron to'rni o'zgartiramiz va vaznlarni ω_{ij} bilan belgilaymiz:

barcha kerakli hisob-kitoblarni samarali bajarish uchun biz vaznlarni vazn matritsasiga joylashtiramiz. 3.18-rasmdagi vaznlar massivni yaratadi, uni



Rasm 3.18: *Uch qatlamli neyron to'r tuzilishi.*

"Neural Network" sinfida "weights_in_hidden" deb nomlaymiz. Nom vaznlar kirish va yashirin tugunlarni bog'lab turganligini ko'rsatishi kerak, ya'ni ular kirish va yashirin qatlam o'rtasida. Biz shuningdek, ismni ωih deb qisqartiramiz. Yashirin va chiqish qatlamlari orasidagi vazn matritsasi ωho deb belgilanadi:

$$\omega ih = \begin{pmatrix} \omega_{11} & \omega_{12} & \omega_{13} \\ \omega_{21} & \omega_{22} & \omega_{23} \\ \omega_{31} & \omega_{32} & \omega_{33} \\ \omega_{41} & \omega_{42} & \omega_{43} \end{pmatrix}$$

$$\omega ho = \begin{pmatrix} \omega h_{11} & \omega h_{12} & \omega h_{13} & \omega h_{14} \\ \omega h_{21} & \omega h_{22} & \omega h_{23} & \omega h_{24} \end{pmatrix}$$

Endi biz o'z vazn o'lchovlarimizni aniqladik va keyingi bosqichga o'tishimiz kerak. Biz kiritma vektor bilan matritsani ko'paytirishimiz kerak. Darvoqe, bu avvalgi misolimizda qo'lda qilingan ishlarimizdir:

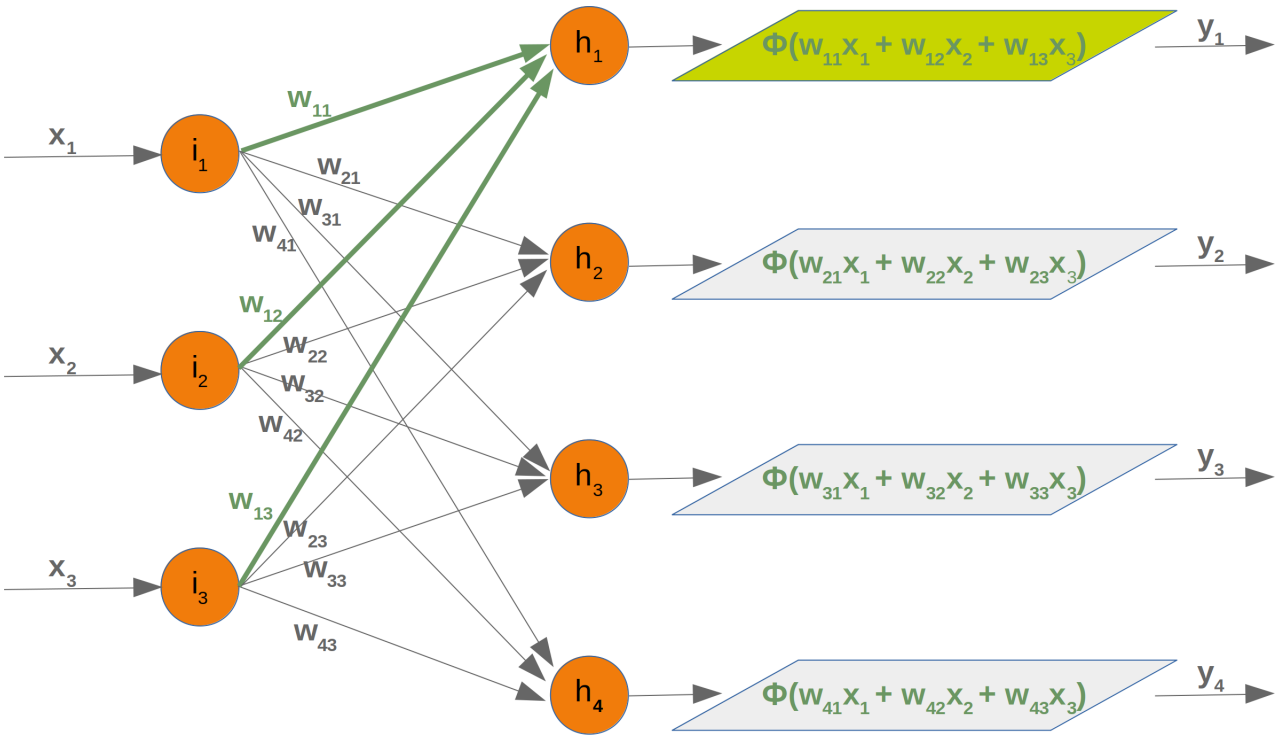
$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} \omega_{11} & \omega_{12} & \omega_{13} \\ \omega_{21} & \omega_{22} & \omega_{23} \\ \omega_{31} & \omega_{32} & \omega_{33} \\ \omega_{41} & \omega_{42} & \omega_{43} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \omega_{11} \cdot x_1 + \omega_{12} \cdot x_2 + \omega_{13} \cdot x_3 \\ \omega_{21} \cdot x_1 + \omega_{22} \cdot x_2 + \omega_{23} \cdot x_3 \\ \omega_{31} \cdot x_1 + \omega_{32} \cdot x_2 + \omega_{33} \cdot x_3 \\ \omega_{41} \cdot x_1 + \omega_{42} \cdot x_2 + \omega_{43} \cdot x_3 \end{pmatrix}$$

Yashirin va chiqish qatlamlari o'rtasidagi ωho matritsasida o'xshash holat mavjud. Shunday qilib, z_1 va z_2 chiqishlarni o_1 va o_2 tugunlaridan matritsali ko'paytirish bilan ham hisoblash mumkin:

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \omega h_{11} & \omega h_{12} & \omega h_{13} & \omega h_{14} \\ \omega h_{21} & \omega h_{22} & \omega h_{23} & \omega h_{24} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

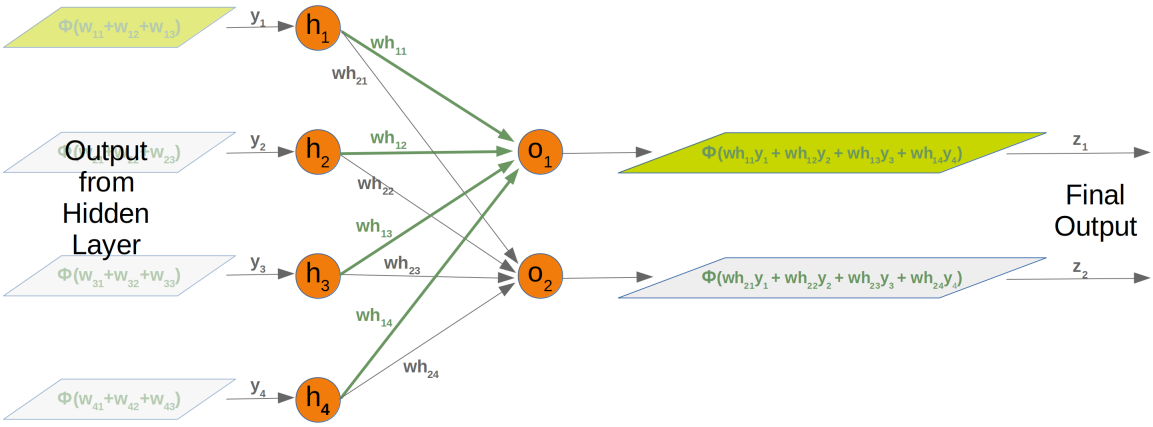
$$= \begin{pmatrix} \omega h_{11} \cdot y_1 + \omega h_{12} \cdot y_2 + \omega h_{13} \cdot y_3 + \omega h_{14} \cdot y_4 \\ \omega h_{21} \cdot y_1 + \omega h_{22} \cdot y_2 + \omega h_{23} \cdot y_3 + \omega h_{24} \cdot y_4 \end{pmatrix}$$

3.19-rasmda hisoblashning butun oqimi, ya'ni matritsa ko'payishi va faollashtirish funksiyasining qanday qo'llanilishi tasvirlangan. ωih matritsasi va x_1, x_2, x_3 kirish tugunlari qiymatlarining matritsasi o'rtasidagi matritsani ko'paytirish faollashtirish funksiyasiga o'tkaziladigan natijani hisoblab chiqadi.



Rasm 3.19: *Uch qatlamli neyron to'r tuzilishi.*

y_1, y_2, y_3, y_4 yakuniy chiqish ωho vazn matritsasining kirishidir: jarayon mutlaqo o'xshash bo'lsa ham, biz yashirin qatlam va chiqish qatlami o'rtasida nima sodir bo'lganligini batafsil ko'rib chiqamiz, 3.20-rasmga qarang:



Rasm 3.20: *Uch qatlamli neyron to'r tuzilishi.*

Vazn matritsasini boshlash

Neyron to'rini o'rgatishdan oldin qilinadigan muhim tanlovlardan biri bu vazn matritsalarini boshlashdir. Biz boshlashimiz mumkin bo'lgan vaznlar haqida hech narsa bilmaymiz. Shunday qilib, biz o'zboshimchalik bilan boshlashimiz mumkinmi?

Ko'rib turganimizdek, kirish tugunlaridan tashqari barcha tugunlarga kirish faollashtirish funksiyasini quyidagi summaga qo'llash orqali hisoblanadi:

$$y_j = \sum_{i=1}^{n} \omega_{ji} \cdot x_i,$$

bu yerda n avvalgi qatlamdagi tugunlar soni va y_j keyingi qatlam tuguniga kiritma qiymat.

Barcha vazn qiymatlarini 0 ga tenglash yaxshi fikr bo'lmasligini biz osongina ko'rishimiz mumkin, chunki bu holda bu yig'indining natijasi har doim nolga teng bo'ladi. Bu bizning to'rimiz o'rganishga qodir emasligini anglatadi. Bu eng yomon tanlov, ammo vazn matritsasini 1 bilan boshlash ham yomon tanlovdir.

Vazn matritsalari qiymatlari tasodifiy va o'zboshimchalik bilan tanlanmasligi kerak. Tasodifiy normal taqsimotni tanlab, biz nosimmetrik vaziyatlarni sindirdik, ular o'quv jarayoni uchun tez-tez yomonlashishi mumkin.

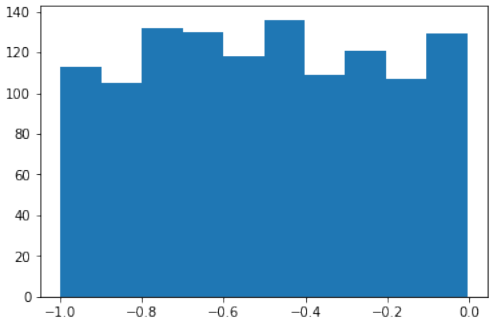
Tasodifiy ravishda vazn matritsalarini boshlashning turli usullari mavjud. Birinchisi, biz **numpy.random** dan birlik funksiyasini taklif qilamiz. U namunalarni yaratadi, ular yarim ochiq oraliqda bir tekis taqsimlanadi (past, yuqori), demak past va yuqori ko'rsatkichlar chiqarib tashlanadi. Belgilangan vaqt oralig'idagi har bir qiymat teng ravishda "uniform" bilan chiziladi.

```
1 import numpy as np
```

```
2
3 number_of_samples = 1200
4 low = -1
5 high = 0
6 s = np.random.uniform(low, high, number_of_samples)
7
8 # all values of s are within the half open interval [-1, 0) :
9 print(np.all(s >= -1) and np.all(s < 0))
10
11 True
```

Oldingi misolimizda bir jinsli (uniform) funksiya bilan yaratilgan namunalarning gistogrammasi quyidagicha ko'rinadi:

```
1 import matplotlib.pyplot as plt
2 plt.hist(s)
3 plt.show()
```



Rasm 3.21: *np.random.uniform bilan olingan taqsimot.*

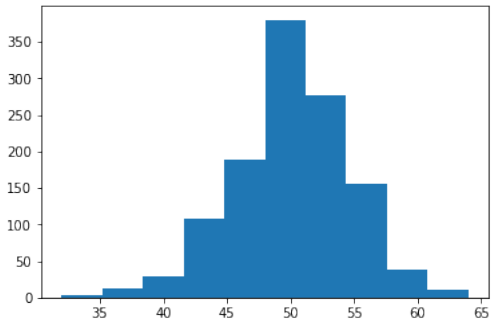
Keyingi funksiyani biz numpy.binomial-dan olamiz:

```
1 binomial(n, p, size = None)
```

Binomial taqsimot orqali tanlangan namunalarda parametrlar quyidagicha: n tanlashlar soni va muvaffaqiyat ehtimolligi p dan namunalar olinadi, bu yerda $n \geq 0$ va p - $[0, 1]$ intervalda yotuvchi o'nli kasr (inglizcha "float"). Qayd etish kerakki, n ham o'nli kasr sifatida kiritilishi mumkin, ammo u foydalanishda butun son deb qaraladi.

```
1 s = np.random.binomial(100, 0.5, 1200)
2 plt.hist(s)
3 plt.show()
```

Biz normal taqsimot bilan tasodifiy sonlarni yaratishni yaxshi ko'ramiz, ammo sonlar chegaralanishi kerak. Bu **np.random.normal()** bilan bog'liq emas, chunki u hech qanday chegaralash parametrini taklif qilmaydi.



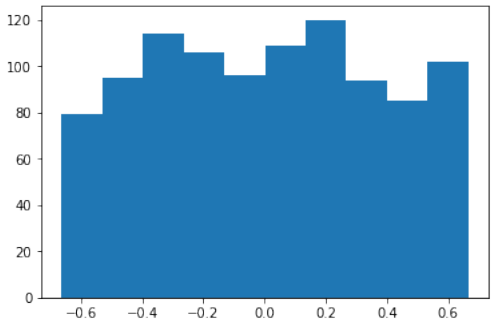
Rasm 3.22: *np.random.binomial bilan olingan taqsimot.*

Buning uchun biz **scipy.stats**-dagi **truncnorm**-dan foydalanishimiz mumkin.

Ushbu taqsimotning standart shakli $[a, b]$ oralig'ida kesilgan standart normaldir - e'tibor bering, a va b standart normal domen bo'yicha aniqlanadi. Muayyan o'rtacha va standart og'ish uchun klip qiymatlarini o'zgartirish uchun quyidagi ifodadan foydalanamiz:

$$a, b = (myclip\_a - my\_mean)/my\_std, (myclip\_b - my\_mean)/my\_std$$

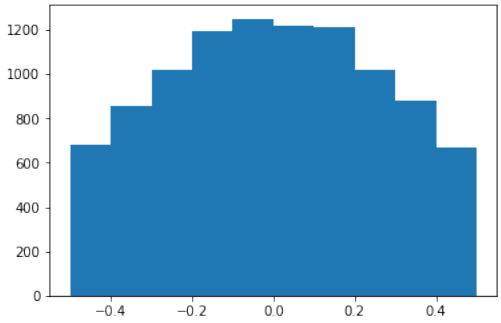
```
from scipy.stats import truncnorm

s = truncnorm(a=-2/3., b=2/3., scale=1, loc=0).rvs(size=1000)

plt.hist(s)
plt.show()
```



Rasm 3.23: *truncnorm bilan olingan taqsimot.*

truncnorm funksiyasidan foydalanish qiyin. Ishni osonlashtirish va ushbu vazifani yengillashtirish uchun ushbu **truncated_normal** funksiyani kiritamiz:

```
def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm( (low - mean) / sd, (upp - mean) / sd,
   loc=mean, scale=sd)

X = truncated_normal(mean=0, sd=0.4, low=-0.5, upp=0.5)
s = X.rvs(10000)

plt.hist(s)
plt.show()
```



Rasm 3.24: *truncated_normal bilan olingan taqsimot.*

Endi biz qatlamlar orasini bog'lovchi vaznlar matritsasini yaratamiz. Bunda eng optimal yo'l biror intervalda tasodifiy qiymatlarni tanlashdir:

$$(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}),$$

bu yerda n kirish tugunlarining sonini bildiradi.

Shunday qilib, biz ωih matritsamizni quyidagi qism-dastur orqali yaratamiz:

```
no_of_input_nodes = 3
no_of_hidden_nodes = 4
rad = 1 / np.sqrt(no_of_input_nodes)

X = truncated_normal(mean=2, sd=1, low=-rad, upp=rad)
wih = X.rvs((no_of_hidden_nodes, no_of_input_nodes))
wih

Output:

array([[-0.41379992, -0.24122842, -0.0303682 ],
[ 0.07304837, -0.00160437,  0.0911987 ],
[ 0.32405689,  0.5103896 ,  0.23972997],
[ 0.097932  , -0.06646741,  0.01359876]])
```

Xuddi shunday, biz ωho vazn matritsasini aniqlashimiz mumkin:

```
no_of_hidden_nodes = 4
no_of_output_nodes = 2
rad = 1 / np.sqrt(no_of_hidden_nodes)  # this is the input in
    this layer!

X = truncated_normal(mean=2, sd=1, low=-rad, upp=rad)
who = X.rvs((no_of_output_nodes, no_of_hidden_nodes))
who

Output:

array([[ 0.15892038,  0.06060043,  0.35900184,  0.14202827],
[-0.4758216 ,  0.29563269,  0.46035026, -0.29673539]])
```

3.3.2 Python-da neyron to'rini ishga tushirish

Neyron to'ri sinfini qurish

Biz neyron to'rlariga bag'ishlangan darsimizning oldingi bo'limida vazn bilan bog'liq eng muhim dalillarni bilib oldik. Biz ulardan qanday foydalanishimiz va ularni Pythonda qanday amalga oshirishimiz mumkinligini ko'rdik. Olingan qiymatlar bilan vaznlarni ko'paytirish Numpy-dan massivlar yordamida matritsali ko'paytmani qo'llash orqali amalga oshirilishini ko'rdik.

Biroq, biz qilmagan narsa ularni haqiqiy neyron to'r muhitida sinab ko'rish edi. Avval bu muhitni yaratishimiz kerak. Endi biz Python-da neyron to'rini yaratadigan sinf quramiz. Barchasini tushunish oson bo'lishi uchun biz kichik qadamlarni tashlaymiz.

Bizning sinfimiz talab qiladigan eng zarur usullar:

- sinfni boshlash uchun *__init__* dan foydalanamiz, ya'ni har bir qatlam uchun neyronlar sonini aniqlaymiz va vazn matritsalarini boshlaymiz
- **run** (ishga tushirish): Biz tasniflamoqchi bo'lgan namunaga qo'llaniladigan usul. Ushbu namunani neyron to'riga qo'llaydi. Biz natijani taxmin qilish uchun tarmoqni "ishga tushirdik" deyishimiz mumkin. Ushbu usul ko'pincha prognoz sifatida tanilgan boshqa dasturlarda mavjud.
- **train** (o'qitish): bu usul kiritma sifatida namuna va unga mos mo'ljal qiymatini oladi. Agar kerak bo'lsa, ushbu kiritma yordamida u vazn ko'rsatkichlarini o'zgartirishi mumkin. Bu shuni anglatadiki, neyron to'r kiritmadan o'rganadi. Foydalanuvchi nuqtai nazaridan biz tarmoqni "o'qitamiz". Masalan, **sklearn**-da bu usul **fit** deyiladi

Hozircha, **train** va **run** usullari bo'yicha mulohazalarimizni keyinroqqa qoldiramiz. Vazn matritsalari *__init__* usuli ichiga kiritilishi kerak. Buni

bilvosita qilamiz. Avvalo *create_weight_matrices* usulini aniqlaymiz va uni *_init_* da chaqiramiz. Shu tarzda, boshlash usuli aniq bo'lib qoladi.

Shuningdek, biz qatlamlarga bias tugunlarini qo'shishni kechiktiramiz.

Oldingi bo'limda o'rganilgan bilimlarimizga asoslanib, quyidagi Python kodi yordamida *NeuralNetwork* sinfini yaratamiz:

```
import numpy as np
from scipy.stats import truncnorm

def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm( (low - mean) / sd, (upp - mean) / sd,
   loc=mean, scale=sd)

class NeuralNetwork:

    def __init__(self,
                 no_of_in_nodes,
                 no_of_out_nodes,
                 no_of_hidden_nodes,
                 learning_rate):
        self.no_of_in_nodes = no_of_in_nodes
        self.no_of_out_nodes = no_of_out_nodes
        self.no_of_hidden_nodes = no_of_hidden_nodes
        self.learning_rate = learning_rate
        self.create_weight_matrices()

    def create_weight_matrices(self):
        rad = 1 / np.sqrt(self.no_of_in_nodes)
        X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
        self.weights_in_hidden = X.rvs(
        (self.no_of_hidden_nodes, self.no_of_in_nodes))
        rad = 1 / np.sqrt(self.no_of_hidden_nodes)
        X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
        self.weights_hidden_out = X.rvs(
        (self.no_of_out_nodes, self.no_of_hidden_nodes))


def train(self):
    pass

def run(self):
    pass
```

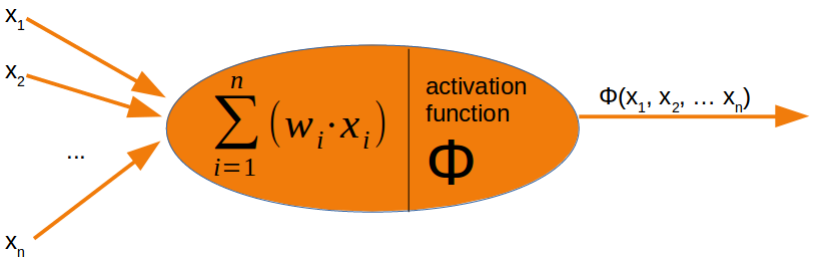
Ushbu kod yordamida biz ko'p ish qila olmaymiz, lekin hech bo'lmaganda uni ishga tushiramiz. Shuningdek, vazn matritsalarini ko'rib chiqamiz:

```
simple_network = NeuralNetwork(no_of_in_nodes = 3,
                               no_of_out_nodes = 2,
                               no_of_hidden_nodes = 4,
```

```
                                learning_rate = 0.1)
print(simple_network.weights_in_hidden)
print(simple_network.weights_hidden_out)

[[-0.3460287  -0.19427278 -0.19102916]
[ 0.56743476 -0.47164202 -0.06910573]
[ 0.53013469 -0.05117752 -0.430623  ]
[ 0.48414483  0.31263278 -0.08123676]]
[[-0.12645547  0.05260599 -0.36278102 -0.32649173]
[-0.20841352 -0.01456191 -0.13778649 -0.08920465]]
```

Faollashtirish funksiyalari, sigmoid va ReLU

Biz run usulini dasturlashdan oldin faollashtirish funksiyasi bilan shug'ullanishimiz kerak. Neyron to'rlari kirish qismida biz quyidagi diagrammani oldik:



Rasm 3.25: *Neyron to'ri diagrammasi.*

Perseptronning kiritma qiymatlari yig'ish funksiyasi tomonidan qayta ishlanadi va undan so'ng faollashtirish funksiyasidan o'tkazib, natijani kerakli va mos keladigan natijaga aylantiradi. Yig'ish funksiyasi vazn vektorlari va kiritma qiymatlarining matritsa ko'paytmasini anglatadi.

Avvalroq o'rganganimizdek, neyron to'rlarda juda ko'p turli xil faollashtirish funksiyalari mavjud.

Sigmoid funksiya tez-tez ishlatiladigan faollashtirish funksiyalaridan biridir. Biz foydalanadigan sigmoid funksiya, shuningdek, **Logistik funksiya** deb ham nomlanadi.

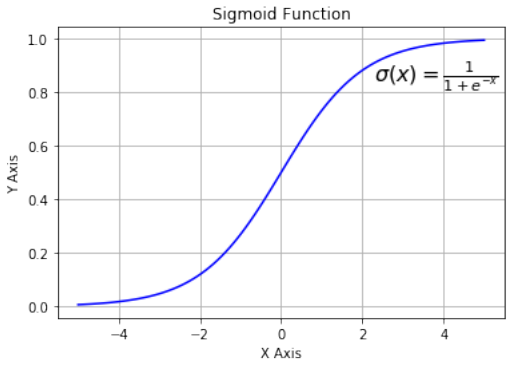
Uni quyidagicha aniqlaymiz:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Sigmoid funksiya grafigini ko'rib chiqaylik. Sigmoid funksiyani tuzishda biz matplotlib-dan foydalanamiz:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
def sigma(x):
    return 1 / (1 + np.exp(-x))

X = np.linspace(-5, 5, 100)

plt.plot(X, sigma(X),'b')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Sigmoid Function')

plt.grid()

plt.text(2.3, 0.84, r'$\sigma(x)=\frac{1}{1+e^{-x}}$',
   fontsize=16)
plt.show()
```



Rasm 3.26: *Sigmoid funksiya grafigi.*

Grafikka qarab, sigmoid funksiya berilgan x sonini 0 dan 1 gacha bo'lgan sonlar oralig'iga joylashtirganini ko'rishimiz mumkin! x ning kattalashishi bilan sigmoid funksiyaning qiymati 1 ga yaqinlashadi va x kichiklashganda sigmoid funksiya qiymati 0 ga yaqinlashadi.

Sigmoid funksiyani o'zimiz belgilashning o'rniga, sigmoid funksiyani amalga oshiruvchi **scipy.special**-dagi **expit** funksiyasidan foydalanishimiz mumkin. U **int**, **float**, **list**, **numpy**, **ndarray** va hokazolar kabi turli xil ma'lumot sinflarida qo'llanilishi mumkin.

```
from scipy.special import expit
print(expit(3.4))
print(expit([3, 4, 1]))
print(expit(np.array([0.8, 2.3, 8])))

0.9677045353015494
[0.95257413 0.98201379 0.73105858]
[0.68997448 0.90887704 0.99966465]
```

Logistik funksiya ko'pincha neyron to'rlarda modeldagi nochiziqlilikni kiritish va signallarni belgilangan diapazonga moslash uchun ishlatiladi, ya'ni 0 va 1 oraliqqa. Bundan tashqari, bu funksiya juda samarali bo'lib, uning hosilasi ortqa tarqalish jarayonida ishni osonlashtiradi. Biz yana bir bor bu funksiyani keltiramiz:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

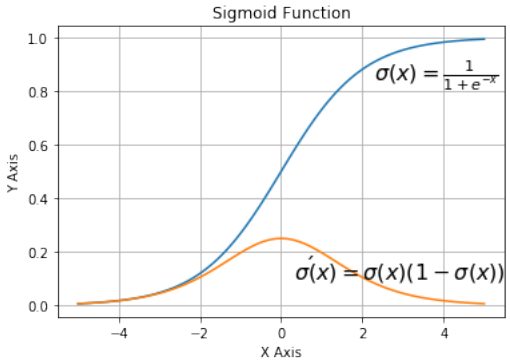
va uning hosilasi quyidagicha ifodalanadi:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)).$$

```
import numpy as np
import matplotlib.pyplot as plt

def sigma(x):
    return 1 / (1 + np.exp(-x))

X = np.linspace(-5, 5, 100)

plt.plot(X, sigma(X))

plt.plot(X, sigma(X) * (1 - sigma(X)))

plt.xlabel('X Axis')
plt.ylabel('Y Axis')

plt.title('Sigmoid Function')

plt.grid()

plt.text(2.3, 0.84, r'$\sigma(x)=\frac{1}{1+e^{-x}}$',
    fontsize=16)
plt.text(0.3, 0.1, r'$\sigma\'(x) = \sigma(x)(1 - \sigma(x))$
    ', fontsize=16)

plt.show()
```

Shuningdek, sigmoid funksiyani numpy-dan dekorator vektorizatsiya yordamida aniqlaymiz:

```
@np.vectorize
def sigmoid(x):
    return 1 / (1 + np.e ** -x)

#sigmoid = np.vectorize(sigmoid)
sigmoid([3, 4, 5])
```



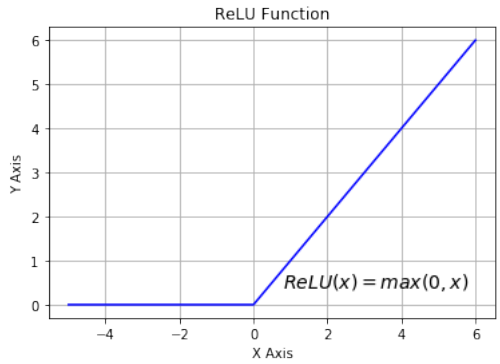
Rasm 3.27: *Sigmoid funksiya va uning hosilasi grafiklari.*

```
Output::

array([0.95257413, 0.98201379, 0.99330715])
```

Faollashtirishning yana bir qulay usuli bu ReLU funksiyasidir. ReLU - to'g'rilangan chiziqli birlik. Shuningdek, u "rump function" sifatida ham tanilgan. U o'z argumentining musbat qismi sifatida aniqlanadi, ya'ni $y = max(0, x)$. Bu "hozirda eng muvaffaqiyatli va keng ishlatiladigan faollashtirish funksiyasi - bu Rectified Linear Unit (ReLU)". ReLu funksiya Sigmoid singari funksiyalarga qaraganda ancha samaralidir, chunki Relu faqat 0 va argumentlar orasidagi x maksimalni tanlashni anglatadi. Holbuki sigmoid og'ir eksponensial operatsiyalarni bajarishi kerak.

```
# alternative activation function
def ReLU(x):
    return np.maximum(0.0, x)

# derivation of relu
def ReLU_derivation(x):
    if x <= 0:
        return 0
    else:
        return 1

import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-5, 6, 100)
plt.plot(X, ReLU(X),'b')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('ReLU Function')
```

```
plt.grid()
plt.text(0.8, 0.4, r'$ReLU(x)=max(0, x)$', fontsize=14)
plt.show()
```



Rasm 3.28: *ReLU funksiya grafigi.*

Run usulini qo'shish

NeuralNetwork sinfini to'liq shakllantirish va **run** (yoki **predict**) usulini qo'llash uchun bizda hamma narsa mavjud. Biz **scipy.special**-ni faollashtirish funksiyasi sifatida ishlatamiz va uni *activation_function* deb qayta nomlaymiz:

```
from scipy.special import expit as activation_function
```

Run usulida o'tilishi kerak bo'lgan bosqichlar quyidagilardan iborat:

1. Kiritma vektor va *weights_in_hidden* matritsani ko'paytirish.
2. 1-qadam natijalariga faollashtirish funksiyasini qo'llash.
3. 2-qadam natijasi bo'lgan vektorni va *weights_hidden_out* matritsani ko'paytirish.
4. Yakuniy natijani olish uchun 3-qadam natijasiga faollashtirish funksiyasini qo'llash.

```
import numpy as np
from scipy.special import expit as activation_function
from scipy.stats import truncnorm

def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm((low - mean) / sd, (upp - mean) / sd,
   loc=mean, scale=sd)
```

```
7
8
9  class NeuralNetwork:
10     def __init__(self,
11                  no_of_in_nodes,
12                  no_of_out_nodes,
13                  no_of_hidden_nodes,
14                  learning_rate):
15         self.no_of_in_nodes = no_of_in_nodes
16         self.no_of_out_nodes = no_of_out_nodes
17         self.no_of_hidden_nodes = no_of_hidden_nodes
18         self.learning_rate = learning_rate
19         self.create_weight_matrices()
20
21 def create_weight_matrices(self):
22     """ A method to initialize the weight matrices of the
       neural network"""
23     rad = 1 / np.sqrt(self.no_of_in_nodes)
24     X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
25     self.weights_in_hidden = X.rvs((self.no_of_hidden_nodes,
       self.no_of_in_nodes))
26     rad = 1 / np.sqrt(self.no_of_hidden_nodes)
27     X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
28     self.weights_hidden_out = X.rvs((self.no_of_out_nodes,
       self.no_of_hidden_nodes))
29
30
31 def train(self, input_vector, target_vector):
32     pass
33
34
35 def run(self, input_vector):
36     """
37     running the network with an input vector 'input_vector'.
38     'input_vector' can be tuple, list or ndarray
39     """
40     # turning the input vector into a column vector
41     input_vector = np.array(input_vector, ndmin=2).T
42     input_hidden = activation_function(self.weights_in_hidden
       @ input_vector)
43     output_vector = activation_function(self.
       weights_hidden_out @ input_hidden)
44     return output_vector
```

Neyron to'ri hisoblangan ushbu sinfni yurg'izishimiz mumkin. Quyidagi misolda biz ikkita kirish tuguni, to'rtta yashirin tugun va ikkita chiqish tugunlari bilan to'r yaratamiz:

```
1 simple_network = NeuralNetwork(no_of_in_nodes=2,
2                                no_of_out_nodes=2,
```

```
                                    no_of_hidden_nodes=4,
                                    learning_rate=0.6)
```

Run usulini shakli $(2,)$ bo'lgan barcha massivlarga, shuningdek ikkita sonli elementlarga ega list va "tuple"larga qo'llashimiz mumkin. Chaqirish natijasi vaznlarning tasodifiy qiymatlari bilan aniqlanadi:

```
simple_network.run([(3, 4)])

Output::

array([[0.54558831],
[0.6834667 ]])
```

3.3.3 Python-da neyron to'rini o'qitish

Avvalgi bo'limda biz Python kodi yordamida "NeuralNetwork" deb nomlangan sinfni dasturladik. Ushbu sinf holatiga ko'ra uchta qatlamli to'rdir. Biz ushbu sinfga doir sun'iy neyron to'rini chaqirganimizda, qatlamlar orasidagi vazn matritsalari avtomatik va tasodifiy tanlanadi. Hatto biron bir kiritma bilan sun'iy neyron to'rini ishlatish mumkin. Ammo, tabiiyki, bu sinov maqsadlaridan boshqa narsaga yaramaydi. Bunday sun'iy neyron to'ri to'g'ri tasniflash natijalarini bera olmaydi. Aslida, tasniflash natijalari kutilgan natijalarga moslashtirilmaydi. Vazn matritsalarining qiymatlari tasniflash vazifasiga muvofiq belgilanishi kerak. Vazn ko'rsatkichlarini yaxshilashimiz kerak, demak biz neyron to'rini o'qitishimiz kerak. Uni o'qitish uchun biz **train** usulida ortga tarqalishni amalga oshirishimiz kerak.

Ortga tarqalish usulini tushunganingizdan so'ng, siz **train** usulini to'liq tushunishga tayyor bo'lasiz.

Train usuli kiritma vektor ($input\_vector$) va mo'ljal vektor ($target\_vector$) bilan chaqiriladi. Vektorlarning shakli bir o'lchovli bo'lishi mumkin, ammo ular avtomatik ravishda to'g'ri ikki o'lchovli shaklga aylantiriladi:

$$reshape(input\_vector.size, 1),$$

$$reshape(target\_vector.size, 1).$$

Shundan so'ng biz to'r $output\_vector\_network = self.run(input\_vector)$ natijasini olishi uchun **run** usulini chaqiramiz. Ushbu chiqish $target\_vector$-dan farq qilishi mumkin. Neyron to'ri aniqlagan $output\_vector\_network$-ni $target\_vector$-dan ayirib, $output\_error$-ni hisoblab topamiz.

```
import numpy as np

@np.vectorize
```

```
4  def sigmoid(x):
5      return 1 / (1 + np.e ** -x)
6  activation_function = sigmoid
7
8  from scipy.stats import truncnorm
9
10 def truncated_normal(mean=0, sd=1, low=0, upp=10):
11     return truncnorm((low - mean) / sd, (upp - mean) / sd,
       loc=mean, scale=sd)
12
13 class NeuralNetwork:
14
15     def __init__(self,
16                  no_of_in_nodes,
17                  no_of_out_nodes,
18                  no_of_hidden_nodes,
19                  learning_rate):
20         self.no_of_in_nodes = no_of_in_nodes
21         self.no_of_out_nodes = no_of_out_nodes
22         self.no_of_hidden_nodes = no_of_hidden_nodes
23         self.learning_rate = learning_rate
24         self.create_weight_matrices()
25
26     def create_weight_matrices(self):
27         """ A method to initialize the weight matrices of the
       neural network"""
28         rad = 1 / np.sqrt(self.no_of_in_nodes)
29         X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
30         self.weights_in_hidden = X.rvs((self.
       no_of_hidden_nodes, self.no_of_in_nodes))
31         rad = 1 / np.sqrt(self.no_of_hidden_nodes)
32         X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
33         self.weights_hidden_out = X.rvs((self.no_of_out_nodes
       , self.no_of_hidden_nodes))
34
35
36     def train(self, input_vector, target_vector):
37         """
38         input_vector and target_vector can be tuples, lists
       or ndarrays
39         """
40         # make sure that the vectors have the right shape
41         input_vector = np.array(input_vector)
42         input_vector = input_vector.reshape(input_vector.size
       , 1)
43         target_vector = np.array(target_vector).reshape(
       target_vector.size, 1)
44
45         output_vector_hidden = activation_function(self.
```

```
    weights_in_hidden @ input_vector)
46        output_vector_network = activation_function(self.
    weights_hidden_out @ output_vector_hidden)
47
48        output_error = target_vector - output_vector_network
49        tmp = output_error * output_vector_network * (1.0 -
    output_vector_network)
50        self.weights_hidden_out += self.learning_rate  * (tmp
     @ output_vector_hidden.T)
51
52        # calculate hidden errors:
53        hidden_errors = self.weights_hidden_out.T @
    output_error
54        # update the weights:
55        tmp = hidden_errors * output_vector_hidden * (1.0 -
    output_vector_hidden)
56        self.weights_in_hidden += self.learning_rate * (tmp @
     input_vector.T)
57
58    def run(self, input_vector):
59        """
60        running the network with an input vector '
    input_vector'.
61        'input_vector' can be tuple, list or ndarray
62        """
63        # make sure that input_vector is a column vector:
64        input_vector = np.array(input_vector)
65        input_vector = input_vector.reshape(input_vector.size
    , 1)
66        input4hidden = activation_function(self.
    weights_in_hidden @ input_vector)
67        output_vector_network = activation_function(self.
    weights_hidden_out @ input4hidden)
68        return output_vector_network
```

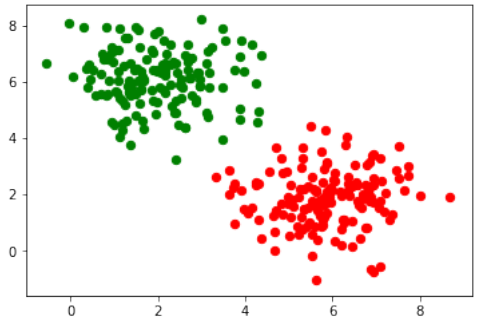
Yuqoridagi kodni *neural_networks*1.*py* deb nomlanuvchi faylda saqlaymiz. Kelgusi misollarda biz ushbu nom ostida foydalanamiz.

Ushbu neyron to'r sinfini sinab ko'rish uchun bizga kiritma ma'lumotlar va sinov ma'lumotlari kerak. Biz kiritma ma'lumotlarni *sklearn.datasets*-dan *make_blobs* yordamida yaratamiz.

```
1 from sklearn.datasets import make_blobs
2
3 n_samples = 300
4 samples, labels = make_blobs(n_samples=n_samples,
5                              centers=([2, 6], [6, 2]),
6                              random_state=0)
```

Keling, yaratilgan ma'lumotlarni grafikda aks ettiraylik:

```
import matplotlib.pyplot as plt

colours = ('green', 'red', 'blue',
           'magenta', 'yellow', 'cyan')
fig, ax = plt.subplots()

for n_class in range(2):
    ax.scatter(samples[labels==n_class][:, 0],
               samples[labels==n_class][:, 1],
               c=colours[n_class], s=40,
               label=str(n_class))

plt.show()
```



Rasm 3.29: *make_blobs yordamida yaratilgan kiritma ma'lumotlar grafigi.*

Biz train va test to'plamlarini yaratmoqchimiz:

```
size_of_learn_sample = int(n_samples * 0.8)
learn_data = samples[:size_of_learn_sample]
test_data = samples[-size_of_learn_sample:]
```

Endi esa, ikkita kirish tuguni, beshta yashirin tuguni va bitta chiqish tuguni bo'lgan neyron to'rni yarataylik:

```
from neural_networks1 import NeuralNetwork

simple_network = NeuralNetwork(no_of_in_nodes=2,
                               no_of_out_nodes=1,
                               no_of_hidden_nodes=5,
                               learning_rate=0.3)
```

Keyingi qadam neyron to'rini o'quv namunalariga o'rgatishdan iborat:

```
for i in range(size_of_learn_sample):
    simple_network.train(learn_data[i], labels[i])
```

Endi biz neyron to'ri qanchalik yaxshi o'rganganligini tekshirishimiz kerak. To'r faqat bitta chiqish neyroniga ega. Bu degani, qiymatlar 0 va 1 oralig'ida

bo'ladi. Agar chiqish qiymatlari ideal bo'lsa, garchi bunday bo'lmasa ham, unda 1-sinf uchun 1 qiymatlar va 0-sinf uchun 0 qiymatlar bo'ladi. Sigmoid funksiya ishlatilgani sababli 0 ni olib ham bo'lmaydi. Bu degani, 0 va 1 oraliqda bo'lgan qiymatlarni belgilashimiz kerak. Biz 0.5 ni chegara qiymat sifatida belgilaymiz: 0.5 dan katta yoki teng bo'lgan har bir narsa 1 deb hisoblanadi va undan kichik har bir narsa 0 sifatida qabul qilinadi. Endi biz natijalarni teglar ("label") bilan taqqoslashimiz mumkin:

```
from collections import Counter

evaluation = Counter()
for i in range(size_of_learn_sample):
    point, label = learn_data[i], labels[i]
    res = simple_network.run(point)
    if label == 1:
        if res >= 0.5:
            evaluation["correct"] += 1
        else:
            evaluation["wrong"] += 1
    elif label == 0:
        if res <= 0.5:
            evaluation["correct"] += 1
        else:
            evaluation["wrong"] += 1
print(evaluation)

# Natija:
Counter({'correct': 239, 'wrong': 1})
```

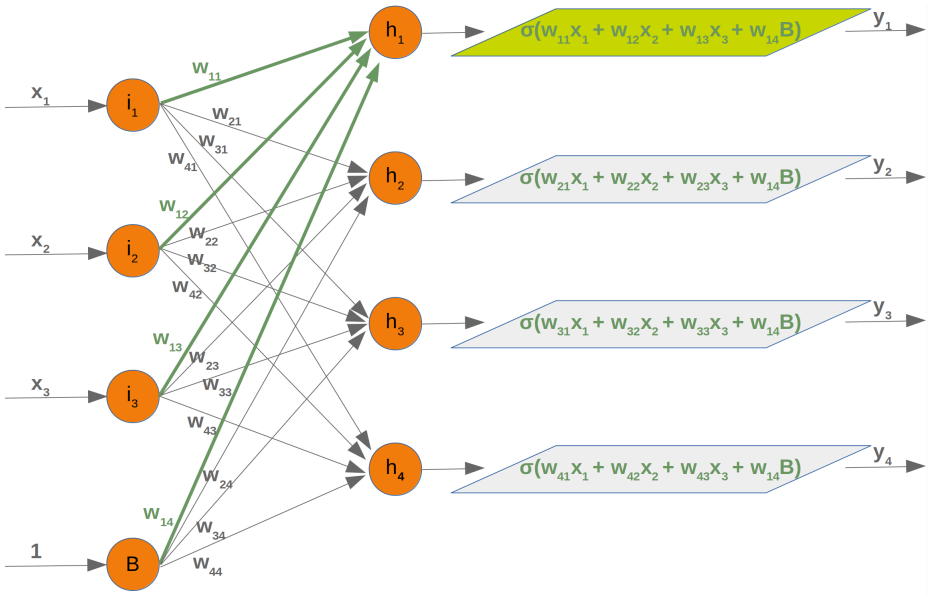
Yuqorida berilgan dizaynda kamchilik quyidagicha: Agar biz 0.5 ga teng yoki unga yaqinroq qiymatga ega bo'lsak, tasniflagich hal qilishdan to'siladi. Natija esa ikki mumkin bo'lgan natijalar o'rtasida qolib ketadi.

```
from collections import Counter

def evaluate(data, labels, threshold=0.5):
    evaluation = Counter()
    for i in range(len(data)):
        point, label = data[i], labels[i]
        res = simple_network.run(point)
        if threshold < res < 1 - threshold:
            evaluation["undecided"] += 1
        elif label == 1:
            if res >= 1 - threshold:
                evaluation["correct"] += 1
            else:
                evaluation["wrong"] += 1
        elif label == 0:
            if res <= threshold:
```

```
                evaluation["correct"] += 1
            else:
                evaluation["wrong"] += 1
    return evaluation


res = evaluate(learn_data, labels)
print (res)

# Output::
Counter({'correct': 238, 'wrong': 2})
```

Bias tugunlari bo'lgan Neyron to'rlari

Biz avvalgi bo'lmda bias tugunining zaruriyati haqida gaplashdik, unda biz juda oddiy chiziqli ajratiladigan ma'lumotlar to'plamiga e'tibor qaratdik. Bilamizki, bias tuguni har doim bir xil natijani jo'natuvchi tugun. Boshqacha



Rasm 3.30: *Uch qatlamli neyron to'rining dastlabki ikki qatlamlari.*

aytganda, bu qandaydir kirishga bog'liq bo'lmagan va hech qanday kirishga ega bo'lmagan tugun. Bias tugunining qiymati ko'pincha 1 bilan belgilanadi, lekin uni boshqa qiymatlarga ham tenglash mumkin. Muayyan sabablarga ko'ra, nolga tenglab bo'lmaydi – mantiqiy jihatdan bunday ish qilinmaydi, albatta. Agar neyron to'ridagi biror qatlamda bias tugun bo'lmasa, xususiyat qiymatlari 0 bo'lganda 0 dan farq qiluvchi keyingi qatlamda chiqish hosil qila

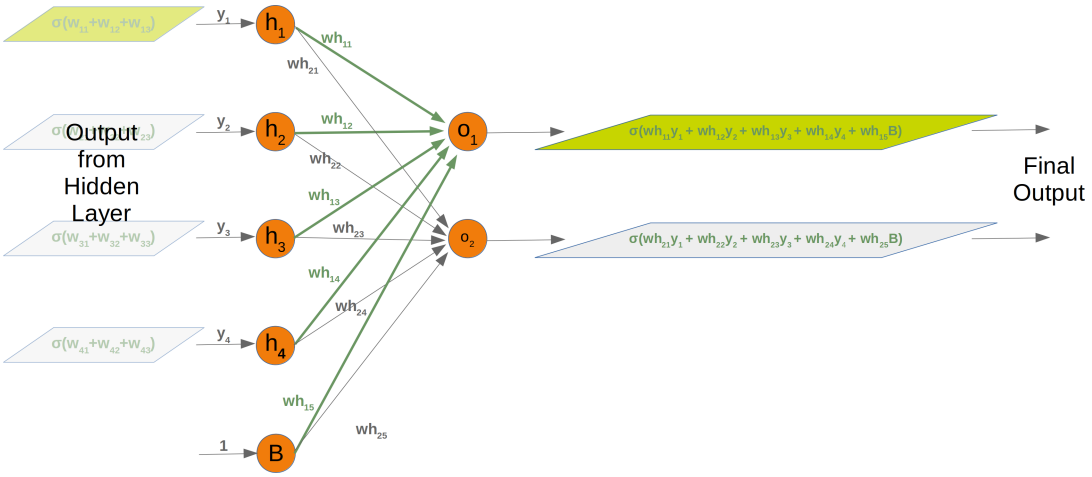
olmaydi. Umumiy qoida shundayki, bias tugunlari ma'lumotlarni moslashtirish uchun to'rning moslashuvchanligini oshiradi. Odatda, har bir qatlam uchun bittadan ko'p bo'lmagan tugun mavjud bo'ladi. Faqatgina istisno - bu chiqish qatlami, chunki bu qatlamga bias tugunini qo'shishning ma'nosi yo'q.

Yuqoridagi 3.30-rasmda ilgari ishlatilgan uch qatlamli neyron to'rimizning dastlabki ikki qatlami ko'rsatilgan.

Ushbu diagrammadan ko'rishimiz mumkinki, bizning vazn matritsamizga bitta qo'shimcha ustun kerak bo'ladi va bias qiymati kiritma vektorga qo'shilishi kerak:

$$\begin{pmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \omega_{14} \\ \omega_{21} & \omega_{22} & \omega_{23} & \omega_{24} \\ \omega_{31} & \omega_{32} & \omega_{33} & \omega_{34} \\ \omega_{41} & \omega_{42} & \omega_{43} & \omega_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix}$$

Yana, yashirin va chiqish qatlami o'rtasidagi vazn matritsasi uchun vaziyat o'xshashdir, 3.31:



Rasm 3.31: *Uch qatlamli neyron to'rining oxirgi ikki qatlamlari.*

Tegishli matritsa uchun ham xuddi shunday:

$$\begin{pmatrix} \omega h_{11} & \omega h_{12} & \omega h_{13} & \omega h_{14} & \omega_{15} \\ \omega h_{21} & \omega h_{22} & \omega h_{23} & \omega h_{24} & \omega_{25} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ 1 \end{pmatrix}$$

Quyida bias tugunlari ishtiroki bilan qurilgan neyron to'ri uchun to'liq Python sinfi ko'rsatilgan (*neural_networks2.py* deb nomlanuvchi yangi bir fayl ochib, kodni unga kiriting):

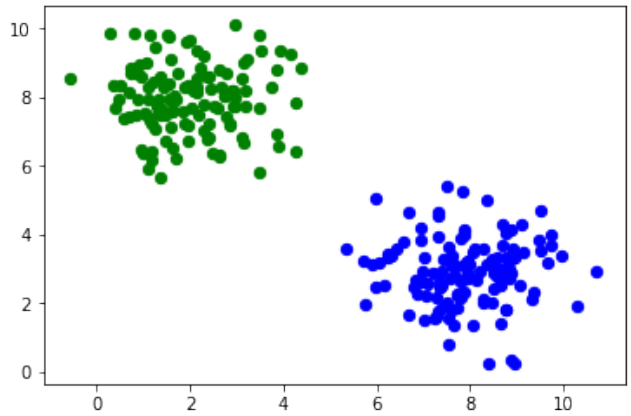
```
import numpy as np
from scipy.stats import truncnorm

@np.vectorize
def sigmoid(x):
    return 1 / (1 + np.e ** -x)
activation_function = sigmoid

def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm((low - mean) / sd,
                     (upp - mean) / sd,
                     loc=mean, scale=sd)



class NeuralNetwork:

    def __init__(self,
                 no_of_in_nodes,
                 no_of_out_nodes,
                 no_of_hidden_nodes,
                 learning_rate,
                 bias=None):
        self.no_of_in_nodes = no_of_in_nodes
        self.no_of_hidden_nodes = no_of_hidden_nodes
        self.no_of_out_nodes = no_of_out_nodes
        self.learning_rate = learning_rate
        self.bias = bias
        self.create_weight_matrices()


    def create_weight_matrices(self):
        """ A method to initialize the weight matrices of the
    neural
        network with optional bias nodes"""
        bias_node = 1 if self.bias else 0
        rad = 1 / np.sqrt(self.no_of_in_nodes + bias_node)
        X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
        self.weights_in_hidden =
        X.rvs((self.no_of_hidden_nodes,
               self.no_of_in_nodes + bias_node))
        rad = 1/np.sqrt(self.no_of_hidden_nodes + bias_node)
        X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
        self.weights_hidden_out =
        X.rvs((self.no_of_out_nodes,
               self.no_of_hidden_nodes + bias_node))


    def train(self, input_vector, target_vector):
```

```
49          """ input_vector and target_vector can be tuple, list
    or ndarray """
50
51          # make sure that the vectors have the right shap
52          input_vector = np.array(input_vector)
53          input_vector =
54               input_vector.reshape(input_vector.size, 1)
55
56          if self.bias:
57              # adding bias node to the end of the input_vector
58              input_vector =
59                np.concatenate( (input_vector, [[self.bias]]) )
60
61              target_vector =
62                np.array(target_vector).reshape(target_vector.
    size, 1)
63
64              output_vector_hidden = activation_function(self.
    weights_in_hidden @ input_vector)
65              if self.bias:
66                  output_vector_hidden = np.concatenate( (
    output_vector_hidden, [[self.bias]]) )
67              output_vector_network = activation_function(self.
    weights_hidden_out @ output_vector_hidden)
68
69              output_error = target_vector -
    output_vector_network
70              # update the weights:
71              tmp = output_error * output_vector_network * (1.0
     - output_vector_network)
72              self.weights_hidden_out += self.learning_rate  *
    (tmp @ output_vector_hidden.T)
73
74              # calculate hidden errors:
75              hidden_errors = self.weights_hidden_out.T @
    output_error
76              # update the weights:
77              tmp = hidden_errors * output_vector_hidden * (1.0
     - output_vector_hidden)
78              if self.bias:
79                  x = (tmp @input_vector.T)[:-1,:]     # last
    row cut off,
80              else:
81                  x = tmp @ input_vector.T
82              self.weights_in_hidden += self.learning_rate *  x
83
84
85      def run(self, input_vector):
86          """
```

```
        running the network with an input vector '
    input_vector'.
        'input_vector' can be tuple, list or ndarray
        """
        # make sure that input_vector is a column vector:
        input_vector = np.array(input_vector)
        input_vector = input_vector.reshape(input_vector.size
    , 1)
        if self.bias:
            # adding bias node to the end of the inpuy_vector
            input_vector = np.concatenate( (input_vector,
    [[1]]) )
        input4hidden = activation_function(self.
    weights_in_hidden @ input_vector)
        if self.bias:
            input4hidden = np.concatenate( (input4hidden,
    [[1]]) )
        output_vector_network = activation_function(self.
    weights_hidden_out @ input4hidden)
        return output_vector_network

    def evaluate(self, data, labels):
        corrects, wrongs = 0, 0
        for i in range(len(data)):
            res = self.run(data[i])
            res_max = res.argmax()
            if res_max == labels[i]:
                corrects += 1
            else:
                wrongs += 1
        return corrects, wrongs
```

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

data, labels = make_blobs(n_samples=250,
                          centers=([2.5, 3], [6.7, 7.9]),
                          random_state=0)

data, labels = make_blobs(n_samples=250,
                          centers=([2, 7.9], [8, 3]),
                          random_state=0)

colours = ('green', 'blue', 'red',
           'magenta', 'yellow', 'cyan')
fig, ax = plt.subplots()


for n_class in range(2):
```

```
    ax.scatter(data[labels==n_class][:, 0], data[labels==
n_class][:, 1],
                c=colours[n_class], s=40, label=str(n_class))
```



Rasm 3.32: *make_blobs yordamida yaratilgan kiritma ma'lumotlar grafigi.*

```
from neural_networks2 import NeuralNetwork

simple_network = NeuralNetwork(no_of_in_nodes=2,
                               no_of_out_nodes=2,
                               no_of_hidden_nodes=10,
                               learning_rate=0.1,
                               bias=1)

simple_network.__dict__

Output::

{'no_of_in_nodes': 2,
'no_of_hidden_nodes': 10,
'no_of_out_nodes': 2,
'learning_rate': 0.1,
'bias': 1,
'weights_in_hidden': array([[-0.38709697,  0.34216142,
   -0.53558474],
[-0.38265064,  0.07216054, -0.19346557],
[ 0.06640281, -0.22866835,  0.33205037],
[ 0.06057826,  0.49377468,  0.2012993 ],
[ 0.37061155, -0.12533196, -0.31096331],
[-0.26451103, -0.18483328,  0.34717534],
[-0.13736405, -0.11134647,  0.49381627],
[-0.29948127, -0.09301347, -0.33229527],
[ 0.52635705, -0.43268611,  0.36468496],
```

```
[-0.02565338,  0.19563129,  0.02560361]]),
'weights_hidden_out': array([[-0.2630956 , -0.28078972,
   -0.27659632,  0.28614888, -0.21546324,
-0.11180555, -0.04695778,  0.27741456, -0.04321395,
   0.24967042,
-0.03368758],
[-0.05675981, -0.28455935,  0.12732364,  0.07407335,
   -0.12645493,
0.07372685, -0.23219729, -0.02740009, -0.03702381,
   -0.23802179,
-0.28229358]])}
```

```
import numpy as np
labels_one_hot = (np.arange(2) == labels.reshape(labels.size,
    1))
labels_one_hot = labels_one_hot.astype(np.float)

for i in range(len(data)):
    simple_network.train(data[i], labels_one_hot[i])


simple_network.evaluate(data, labels)

Output:: (250, 0)

data

Output:: array([[ 1.61267318,  7.59769725],
                [ 3.13940068,  6.66517418],
                  ........
                  ........
                [ 0.99978465,  6.3552289 ],
                [ 6.81114074,  2.49318365]])
```

3.4 Xulosa

Ushbu bobda biz yillar davomida sun'iy neyron tarmoqlardan chuqur o'rganish qanday rivojlanganligini ko'rib chiqdik. Shuningdek, biz Perseptronni o'rganish usulini, uning cheklanishlarini va neyron to'rlarini o'qitishning hozirgi usulini muhokama qildik. Qavariq bo'lmagan qiymat funksiyalari, elliptik lokallashgan qiymat konturlari va egar nuqtalari bilan bog'liq muammolar, shuningdek, bunday muammolarni hal qilish uchun turli xil optimallashtirish vositalarining mavjudligi haqida batafsil muhokama qilindi. Bobning ikkinchi yarmida esa neyron to'rlarini Python-da qanday yaratish va ularni o'qitish masalalarini aniq dasturlar yordmida ko'rsatdik.

4

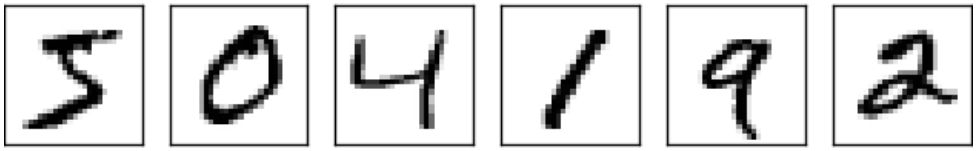
Qo'lyozma raqamlarni tasniflovchi dasturlar

4.1 Raqamlarni tasniflashda neyron to'rdan foydalanish

Avvalgi boblarda o'rgangan bilimlarimizni amalda qo'llab, qo'lda yozilgan raqamlarni tanishni sun'iy neyron to'rlari yordamida amalga oshiramiz. Bunda, stoxastik gradient tushish va MNIST ma'lumotlar bazasidan foydalanamiz. Buni oddiygina Python dasturi bilan amalga oshiramiz va shunchaki 74 ta satrdan iborat raqamli kod bilan kifoyalanamiz. Bizga kerak bo'lgan birinchi narsa - MNIST ma'lumotlarini yuklab olish. Agar siz git foydalanuvchisi bo'lsangiz, quyidagi buyruq orqali kod omborini klonlashtirib, ma'lumotlarni yuklab olishingiz mumkin:

```
git clone https://github.com/mnielsen/neural-networks-and-
    deep-learning.git
```

MNIST ma'lumotlar to'plamida o'n minglab qo'lda yozilgan raqamlarning skanerlangan rasmlari va ularning to'g'ri tasnifi mavjud. MNISTning nomi NIST, AQShning Standartlar va Texnologiyalar Milliy Instituti (NIST) tomonidan to'plangan ikkita ma'lumotlar to'plamining o'zgartirilgan to'plami ekanligidan kelib chiqadi. Mana MNISTning bir nechta rasmlari:



Rasm 4.1: *MNIST ma'lumotlar to'plamida keltirilgan rasmlardan ba'zi birlari.*

Ma'lumotlar to'plamidagi rasmlar 28×28 o'lchamda. Ular *csv* kengaytmali *mnist_train.csv* va *mnist_test.csv* ma'lumot fayllarida saqlanadi.

Ushbu fayllarning har bir satri rasmdan iborat, ya'ni 0 va 255 orasidagi 785 ta raqam.

Har bir satrning birinchi raqami bu yorliq ("label"), ya'ni rasmda ko'rsatilgan raqam. Yorliqdan keyin keluvchi 784 ta raqam esa 28×28 rasmning piksellari.

```
import numpy as np
import matplotlib.pyplot as plt

image_size = 28 # width and length
no_of_different_labels = 10 #  i.e. 0, 1, 2, 3, ..., 9
image_pixels = image_size * image_size
data_path = "data/mnist/"
train_data = np.loadtxt("mnist_train.csv", delimiter=",")
test_data = np.loadtxt("mnist_test.csv", delimiter=",")
```

```
print( test_data[:10] )

Output::
array([[7., 0., 0., ..., 0., 0., 0.],
[2., 0., 0., ..., 0., 0., 0.],
[1., 0., 0., ..., 0., 0., 0.],
...,
[9., 0., 0., ..., 0., 0., 0.],
[5., 0., 0., ..., 0., 0., 0.],
[9., 0., 0., ..., 0., 0., 0.]])

test_data[test_data==255]
print( test_data.shape )

Output::
(10000, 785)
```

MNIST ma'lumotlari ikki qismdan iborat:

- Birinchi qism o'quv ma'lumotlari sifatida ishlatilishi kerak bo'lgan 60000 ta rasmni o'z ichiga oluvchi *mnist_train.csv* fayl. Ushbu suratlarda 250 kishining qo'l yozuvi namunalari skanerlangan bo'lib, ularning yarmi AQSh aholini ro'yxatga olish byurosi xodimlari tomonidan yozilgan va yarmi o'rta maktab o'quvchilariga tegishli.

- MNIST ma'lumotlar to'plamining ikkinchi qismi sinov ma'lumotlari sifatida ishlatiluvchi 10000 rasmdan iborat *mnist_test.csv* fayl. Sinov ma'lumotlarini biz qurgan neyron to'rimiz raqamlarni aniqlashni qanchalik yaxshi o'rganganligini baholash uchun foydalanamiz.

MNIST ma'lumotlar to'plamining rasmlari kul rangda va piksellar 0 va 255 oralig'ida bo'lib, ikkala chegara qiymatlarini ham o'z ichiga oladi. Ushbu qiymatlarni har bir pikselni 0.99/255 ga ko'paytirish va natijaga 0.01 qo'shib, $[0.01, 1]$ oraliqda xaritani tuzamiz. Shunday qilib, kirish bo'limida ko'rganimizdek, vazn yangilanishining oldini olishga qodir bo'lgan kirish sifatida 0 qiymatidan qochamiz.

```
fac = 0.99 / 255
train_imgs = np.asfarray(train_data[:, 1:]) * fac + 0.01
test_imgs = np.asfarray(test_data[:, 1:]) * fac + 0.01

train_labels = np.asfarray(train_data[:, :1])
test_labels = np.asfarray(test_data[:, :1])
```

Birlik kodli ko'rinishdagi ("one-hot representation") hisob-kitoblarimizga yorliqlar kerak bo'ladi. Bizda 0 dan 9 gacha bo'lgan 10 ta raqam mavjud, ya'ni $lr = np.arange(10)$.

Yorliqni birlik kodli ko'rinishga aylantirish quyidagi buyruq bilan amalga oshiriladi: $(lr == label).astype(np.int)$

Buni quyidagicha qayd etamiz:

```
import numpy as np

lr = np.arange(10)

for label in range(10):
    one_hot = (lr==label).astype(np.int)
    print("label: ", label, " in one-hot representation: ",
    one_hot)

label:  0  in one-hot representation:  [1 0 0 0 0 0 0 0 0 0]
label:  1  in one-hot representation:  [0 1 0 0 0 0 0 0 0 0]
label:  2  in one-hot representation:  [0 0 1 0 0 0 0 0 0 0]
label:  3  in one-hot representation:  [0 0 0 1 0 0 0 0 0 0]
label:  4  in one-hot representation:  [0 0 0 0 1 0 0 0 0 0]
label:  5  in one-hot representation:  [0 0 0 0 0 1 0 0 0 0]
label:  6  in one-hot representation:  [0 0 0 0 0 0 1 0 0 0]
label:  7  in one-hot representation:  [0 0 0 0 0 0 0 1 0 0]
label:  8  in one-hot representation:  [0 0 0 0 0 0 0 0 1 0]
label:  9  in one-hot representation:  [0 0 0 0 0 0 0 0 0 1]
```

Biz yorliqli rasmlarimizni bir kodli ko'rinishga aylantirishimiz mumkin. Nol va bir o'rniga, biz 0.01 va 0.99 ni yaratamiz, bu bizning hisob-kitoblarimiz uchun yaxshiroq bo'ladi:

```
lr = np.arange(no_of_different_labels)

# transform labels into one hot representation
train_labels_one_hot = (lr==train_labels).astype(np.float)
test_labels_one_hot = (lr==test_labels).astype(np.float)

# we don't want zeroes and ones in the labels neither:
train_labels_one_hot[train_labels_one_hot==0] = 0.01
train_labels_one_hot[train_labels_one_hot==1] = 0.99
test_labels_one_hot[test_labels_one_hot==0] = 0.01
test_labels_one_hot[test_labels_one_hot==1] = 0.99
```

MNIST ma'lumotlar to'plamini neyron tarmog'imizdan foydalanishni boshlashdan oldin biz ba'zi rasmlarni ko'rib chiqamiz:

```
for i in range(10):
    img = train_imgs[i].reshape((28,28))
    plt.imshow(img, cmap="Greys")
    plt.show()
```

4.1.1 Tezroq qayta yuklash uchun ma'lumotlarni ixchamlash

Payqagan bo'lsangiz *csv*-fayllardan ma'lumotlarni o'qish juda sekin.

Biz ma'lumotlarni **dump** funksiyasi bilan **pickle** modul yordamida ikkilik formatda saqlaymiz:

```
import pickle

with open("pickled_mnist.pkl", "bw") as fh:
    data = (train_imgs,
            test_imgs,
            train_labels,
            test_labels,
            train_labels_one_hot,
            test_labels_one_hot)
    pickle.dump(data, fh)
```

Endi biz ma'lumotlarni *pickle.load*-dan foydalanib o'qiy olamiz. Bu *csv* fayllarida *loadtxt*-ni ishlatishdan ancha tezroq:

```
import pickle

with open("pickled_mnist.pkl", "br") as fh:
    data = pickle.load(fh)

train_imgs = data[0]
test_imgs = data[1]
train_labels = data[2]
test_labels = data[3]
train_labels_one_hot = data[4]
test_labels_one_hot = data[5]

image_size = 28 # width and length
no_of_different_labels = 10 #  i.e. 0, 1, 2, 3, ..., 9
image_pixels = image_size * image_size
```

4.1.2 Ma'lumotlarni tanniflash

Birinchi tasniflash uchun quyidagi neyron to'ri sinfidan foydalanamiz:

```
import numpy as np

@np.vectorize
def sigmoid(x):
    return 1 / (1 + np.e ** -x)
activation_function = sigmoid

from scipy.stats import truncnorm
```

```
9
10 def truncated_normal(mean=0, sd=1, low=0, upp=10):
11     return truncnorm((low - mean) / sd,
12                      (upp - mean) / sd,
13                      loc=mean,
14                      scale=sd)
15
16
17 class NeuralNetwork:
18
19     def __init__(self,
20                  no_of_in_nodes,
21                  no_of_out_nodes,
22                  no_of_hidden_nodes,
23                  learning_rate):
24         self.no_of_in_nodes = no_of_in_nodes
25         self.no_of_out_nodes = no_of_out_nodes
26         self.no_of_hidden_nodes = no_of_hidden_nodes
27         self.learning_rate = learning_rate
28         self.create_weight_matrices()
29
30     def create_weight_matrices(self):
31         """
32         A method to initialize the weight
33         matrices of the neural network
34         """
35         rad = 1 / np.sqrt(self.no_of_in_nodes)
36         X = truncated_normal(mean=0,
37                              sd=1,
38                              low=-rad,
39                              upp=rad)
40         self.wih = X.rvs((self.no_of_hidden_nodes,
41                           self.no_of_in_nodes))
42         rad = 1 / np.sqrt(self.no_of_hidden_nodes)
43         X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
44         self.who = X.rvs((self.no_of_out_nodes,
45                           self.no_of_hidden_nodes))
46
47
48     def train(self, input_vector, target_vector):
49         """
50         input_vector and target_vector can
51         be tuple, list or ndarray
52         """
53
54         input_vector = np.array(input_vector, ndmin=2).T
55         target_vector = np.array(target_vector, ndmin=2).T
56
57         output_vector1 = np.dot(self.wih,
```

```
                                    input_vector)
        output_hidden = activation_function(output_vector1)

        output_vector2 = np.dot(self.who,
                                output_hidden)
        output_network = activation_function(output_vector2)

        output_errors = target_vector - output_network
        # update the weights:
        tmp = output_errors * output_network \
        * (1.0 - output_network)
        tmp = self.learning_rate  * np.dot(tmp,
                                           output_hidden.T)
        self.who += tmp


    # calculate hidden errors:
    hidden_errors = np.dot(self.who.T,
                           output_errors)
    # update the weights:
    tmp = hidden_errors * output_hidden * \
          (1.0 - output_hidden)
    self.wih += self.learning_rate \
                    * np.dot(tmp, input_vector.T)

    def run(self, input_vector):
        # input_vector can be tuple, list or ndarray
        input_vector = np.array(input_vector, ndmin=2).T

        output_vector = np.dot(self.wih,
                               input_vector)
        output_vector = activation_function(output_vector)

        output_vector = np.dot(self.who,
                               output_vector)
        output_vector = activation_function(output_vector)

        return output_vector

    def confusion_matrix(self, data_array, labels):
        cm = np.zeros((10, 10), int)
        for i in range(len(data_array)):
            res = self.run(data_array[i])
            res_max = res.argmax()
            target = labels[i][0]
            cm[res_max, int(target)] += 1
        return cm

    def precision(self, label, confusion_matrix):
```

```
        col = confusion_matrix[:, label]
        return confusion_matrix[label, label] / col.sum()

    def recall(self, label, confusion_matrix):
        row = confusion_matrix[label, :]
        return confusion_matrix[label, label] / row.sum()


    def evaluate(self, data, labels):
        corrects, wrongs = 0, 0
        for i in range(len(data)):
            res = self.run(data[i])
            res_max = res.argmax()
            if res_max == labels[i]:
                corrects += 1
            else:
                wrongs += 1
        return corrects, wrongs
```

```
ANN = NeuralNetwork(no_of_in_nodes = image_pixels,
                    no_of_out_nodes = 10,
                    no_of_hidden_nodes = 100,
                    learning_rate = 0.1)


for i in range(len(train_imgs)):
    ANN.train(train_imgs[i], train_labels_one_hot[i])

for i in range(20):
    res = ANN.run(test_imgs[i])
    print(test_labels[i], np.argmax(res), np.max(res))
```

```
[7.] 7 0.9829245583409039
[2.] 2 0.7372766887508578
[1.] 1 0.9881823673106839
[0.] 0 0.9873289971465894
[4.] 4 0.9456335245615916
[1.] 1 0.9880120617106172
[4.] 4 0.976550583573903
[9.] 9 0.964909168118122
[5.] 6 0.36615932726182665
[9.] 9 0.9848677489827125
[0.] 0 0.9204097234781773
[6.] 6 0.8897871402453337
[9.] 9 0.9936811621891628
[0.] 0 0.9832119513084644
[1.] 1 0.988750833073612
[5.] 5 0.9156741221523511
[9.] 9 0.9812577974620423
```

```
[7.] 7 0.9888560485875889
[3.] 3 0.8772868556722897
[4.] 4 0.9900030761222965
```

```
corrects, wrongs = ANN.evaluate(train_imgs, train_labels)
print("accuracy train: ", corrects / ( corrects + wrongs))
corrects, wrongs = ANN.evaluate(test_imgs, test_labels)
print("accuracy: test", corrects / ( corrects + wrongs))

cm = ANN.confusion_matrix(train_imgs, train_labels)
print(cm)

for i in range(10):
    print("digit: ", i, "precision: ", ANN.precision(i, cm),
   "recall: ", ANN.recall(i, cm))

accuracy train:  0.9469166666666666
accuracy: test 0.9459
[[5802    0   53   21    9   42   35    8   14   20]
[   1 6620   45   22    6   29   14   50   75    7]
[   5   22 5486   51   10   11    5   53   11    3]
[   6   36  114 5788    2  114    1   35   76   72]
[   8   16   54    8 5439   41   10   52   25   90]
[   5    2    3   44    0 4922   20    3    5   11]
[  37    4   54   19   71   72 5789    3   41    4]
[   0    5   31   38    7    4    0 5762    1   32]
[  52   20  103   83    9  102   43   21 5535   38]
[   7   17   15   57  289   84    1  278   68 5672]]
digit:  0 precision:  0.9795711632618606 recall:
   0.9663557628247835
digit:  1 precision:  0.9819044793829724 recall:
   0.9637501819769981
digit:  2 precision:  0.9207787848271232 recall:
   0.9697719639384833
digit:  3 precision:  0.9440548034578372 recall:
   0.9269698910954516
digit:  4 precision:  0.9310167750770284 recall:
   0.9470659933832491
digit:  5 precision:  0.9079505626268216 recall:
   0.9814556331006979
digit:  6 precision:  0.978202095302467 recall:
   0.9499507712504103
digit:  7 precision:  0.9197126895450918 recall:
   0.9799319727891157
digit:  8 precision:  0.945992138096052 recall:
   0.9215784215784216
digit:  9 precision:  0.953437552529837 recall:
   0.87422934648582
```

Ko'p sonli takrorlashlar

Mashg'ulotlarni bir necha bor takrorlashimiz mumkin. Ushbu takrorlashlardagi har bir yurgizish "epoch", ya'ni "davr" deb nomlanadi.

```
epochs = 3

NN = NeuralNetwork(no_of_in_nodes = image_pixels,
                   no_of_out_nodes = 10,
                   no_of_hidden_nodes = 100,
                   learning_rate = 0.1)


for epoch in range(epochs):
    print("epoch: ", epoch)
    for i in range(len(train_imgs)):
        NN.train(train_imgs[i],
                 train_labels_one_hot[i])

    corrects, wrongs = NN.evaluate(train_imgs, train_labels)
    print("accuracy train: ", corrects / ( corrects + wrongs)
    )
    corrects, wrongs = NN.evaluate(test_imgs, test_labels)
    print("accuracy: test", corrects / ( corrects + wrongs))

epoch:  0
accruracy train:  0.94515
accruracy: test 0.9459
epoch:  1
accruracy train:  0.9626833333333333
accruracy: test 0.9582
epoch:  2
accruracy train:  0.96995
accruracy: test 0.9626
```

Biz neyron to'ri ichida ko'plab o'qitishlarni bajarishni xohlaymiz. Buning uchun biz **train** usulini qayta yozamiz va *train_single* usulini qo'shamiz. *train_single* biz ilgari **train** deb nomlagan usulga o'xshab ketadi. Shuningdek, yangi **train** usuli epoch-ni sanaydi. Sinov maqsadida har bir epochdan keyin vazn matritsalarini *intermediate_weights* ro'yxati bilan saqlaymiz. Ushbu ro'yxat train-ning chiqishi sifatida uzatiladi:

```
import numpy as np

@np.vectorize
def sigmoid(x):
    return 1 / (1 + np.e ** -x)
activation_function = sigmoid

```

```
from scipy.stats import truncnorm

def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm((low - mean) / sd,
                     (upp - mean) / sd,
                     loc=mean,
                     scale=sd)


class NeuralNetwork:

    def __init__(self,
                 no_of_in_nodes,
                 no_of_out_nodes,
                 no_of_hidden_nodes,
                 learning_rate):
        self.no_of_in_nodes = no_of_in_nodes
        self.no_of_out_nodes = no_of_out_nodes
        self.no_of_hidden_nodes = no_of_hidden_nodes
        self.learning_rate = learning_rate
        self.create_weight_matrices()

def create_weight_matrices(self):
    """ A method to initialize the weight matrices of the
   neural network"""
    rad = 1 / np.sqrt(self.no_of_in_nodes)
    X = truncated_normal(mean=0,
                         sd=1,
                         low=-rad,
                         upp=rad)
    self.wih = X.rvs((self.no_of_hidden_nodes,
                                    self.no_of_in_nodes))
    rad = 1 / np.sqrt(self.no_of_hidden_nodes)
    X = truncated_normal(mean=0,
                         sd=1,
                         low=-rad,
                         upp=rad)
    self.who = X.rvs((self.no_of_out_nodes,
                                    self.no_of_hidden_nodes))


    def train_single(self, input_vector, target_vector):
        """
        input_vector and target_vector can be tuple,
        list or ndarray
        """

        output_vectors = []
        input_vector = np.array(input_vector, ndmin=2).T
```

```
56          target_vector = np.array(target_vector, ndmin=2).T
57
58
59          output_vector1 = np.dot(self.wih,
60                                  input_vector)
61          output_hidden = activation_function(output_vector1)
62
63          output_vector2 = np.dot(self.who,
64                                  output_hidden)
65          output_network = activation_function(output_vector2)
66
67          output_errors = target_vector - output_network
68          # update the weights:
69          tmp = output_errors * output_network * \
70                  (1.0 - output_network)
71          tmp = self.learning_rate  * np.dot(tmp,
72                                             output_hidden.T)
73          self.who += tmp
74
75
76          # calculate hidden errors:
77          hidden_errors = np.dot(self.who.T,
78                                 output_errors)
79          # update the weights:
80          tmp = hidden_errors * output_hidden * (1.0 -
    output_hidden)
81          self.wih += self.learning_rate * np.dot(tmp,
    input_vector.T)
82
83
84      def train(self, data_array,
85                labels_one_hot_array,
86                epochs=1,
87                intermediate_results=False):
88          intermediate_weights = []
89          for epoch in range(epochs):
90              print("*", end="")
91              for i in range(len(data_array)):
92                  self.train_single(data_array[i],
93                                    labels_one_hot_array[i])
94              if intermediate_results:
95                  intermediate_weights.append((self.wih.copy(),
96                                               self.who.copy())
    )
97          return intermediate_weights
98
99      def confusion_matrix(self, data_array, labels):
100         cm = {}
101         for i in range(len(data_array)):
```

```
            res = self.run(data_array[i])
            res_max = res.argmax()
            target = labels[i][0]
            if (target, res_max) in cm:
                cm[(target, res_max)] += 1
            else:
                cm[(target, res_max)] = 1
        return cm


    def run(self, input_vector):
        """ input_vector can be tuple, list or ndarray """

        input_vector = np.array(input_vector, ndmin=2).T

        output_vector = np.dot(self.wih,
                               input_vector)
        output_vector = activation_function(output_vector)

        output_vector = np.dot(self.who,
                               output_vector)
        output_vector = activation_function(output_vector)

        return output_vector

    def evaluate(self, data, labels):
        corrects, wrongs = 0, 0
        for i in range(len(data)):
            res = self.run(data[i])
            res_max = res.argmax()
            if res_max == labels[i]:
                corrects += 1
            else:
                wrongs += 1
            return corrects, wrongs
```

```
epochs = 10

ANN = NeuralNetwork(no_of_in_nodes = image_pixels,
                               no_of_out_nodes = 10,
                               no_of_hidden_nodes = 100,
                               learning_rate = 0.15)



weights = ANN.train(train_imgs,
                    train_labels_one_hot,
                    epochs=epochs,
                    intermediate_results=True)
```

```
cm = ANN.confusion_matrix(train_imgs, train_labels)

print(ANN.run(train_imgs[i]))

[[2.60149245e-03]
[2.52542556e-03]
[6.57990628e-03]
[1.32663729e-03]
[1.34985384e-03]
[2.63840265e-04]
[2.18329159e-04]
[1.32693720e-04]
[9.84326084e-01]
[4.34559417e-02]]
```

```
cm = list(cm.items())
print(sorted(cm))

[((0.0, 0), 5853), ((0.0, 1), 1), ((0.0, 2), 3), ((0.0, 4),
   8), ((0.0, 5), 2), ((0.0, 6), 12), ((0.0, 7), 7), ((0.0,
   8), 27), ((0.0, 9), 10), ((1.0, 0), 1), ((1.0, 1), 6674),
   ((1.0, 2), 17), ((1.0, 3), 5), ((1.0, 4), 14), ((1.0, 5),
   2), ((1.0, 6), 1), ((1.0, 7), 6), ((1.0, 8), 15), ((1.0,
   9), 7), ((2.0, 0), 37), ((2.0, 1), 14), ((2.0, 2), 5791),
   ((2.0, 3), 17), ((2.0, 4), 11), ((2.0, 5), 2), ((2.0, 6),
   10), ((2.0, 7), 15), ((2.0, 8), 51), ((2.0, 9), 10),
   ((3.0, 0), 16), ((3.0, 1), 5), ((3.0, 2), 34), ((3.0, 3),
   5869), ((3.0, 4), 8), ((3.0, 5), 57), ((3.0, 6), 4),
   ((3.0, 7), 20), ((3.0, 8), 58), ((3.0, 9), 60), ((4.0, 0),
    14), ((4.0, 1), 6), ((4.0, 2), 8), ((4.0, 3), 1), ((4.0,
   4), 5678), ((4.0, 5), 1), ((4.0, 6), 14), ((4.0, 7), 5),
   ((4.0, 8), 11), ((4.0, 9), 104), ((5.0, 0), 7), ((5.0, 1),
    2), ((5.0, 2), 6), ((5.0, 3), 27), ((5.0, 4), 5), ((5.0,
   5), 5312), ((5.0, 6), 12), ((5.0, 7), 5), ((5.0, 8), 20),
   ((5.0, 9), 25), ((6.0, 0), 32), ((6.0, 1), 5), ((6.0, 2),
   1), ((6.0, 4), 10), ((6.0, 5), 52), ((6.0, 6), 5791),
   ((6.0, 8), 26), ((6.0, 9), 1), ((7.0, 0), 5), ((7.0, 1),
   11), ((7.0, 2), 22), ((7.0, 3), 2), ((7.0, 4), 17), ((7.0,
    5), 3), ((7.0, 6), 2), ((7.0, 7), 6074), ((7.0, 8), 26),
   ((7.0, 9), 103), ((8.0, 0), 20), ((8.0, 1), 18), ((8.0, 2)
   , 9), ((8.0, 3), 14), ((8.0, 4), 27), ((8.0, 5), 24),
   ((8.0, 6), 9), ((8.0, 7), 8), ((8.0, 8), 5668), ((8.0, 9),
    54), ((9.0, 0), 26), ((9.0, 1), 2), ((9.0, 2), 2), ((9.0,
    3), 16), ((9.0, 4), 69), ((9.0, 5), 14), ((9.0, 6), 7),
   ((9.0, 7), 19), ((9.0, 8), 15), ((9.0, 9), 5779)]
```

```
for i in range(epochs):
    print("epoch: ", i)
    ANN.wih = weights[i][0]
```

```
    ANN.who = weights[i][1]

    corrects, wrongs = ANN.evaluate(train_imgs, train_labels)
    print("accuracy train: ", corrects / ( corrects + wrongs)
    )
    corrects, wrongs = ANN.evaluate(test_imgs, test_labels)
    print("accuracy: test", corrects / ( corrects + wrongs))
```

Bias tugunlari bilan

```
import numpy as np

@np.vectorize
def sigmoid(x):
    return 1 / (1 + np.e ** -x)
activation_function = sigmoid

from scipy.stats import truncnorm

def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm((low - mean) / sd,
                     (upp - mean) / sd,
                     loc=mean,
                     scale=sd)


class NeuralNetwork:


    def __init__(self,
                 no_of_in_nodes,
                 no_of_out_nodes,
                 no_of_hidden_nodes,
                 learning_rate,
                 bias=None
                 ):

        self.no_of_in_nodes = no_of_in_nodes
        self.no_of_out_nodes = no_of_out_nodes
        self.no_of_hidden_nodes = no_of_hidden_nodes
        self.learning_rate = learning_rate
        self.bias = bias
        self.create_weight_matrices()



    def create_weight_matrices(self):
        """
```

```
39          A method to initialize the weight
40          matrices of the neural network with
41          optional bias nodes
42          """
43
44          bias_node = 1 if self.bias else 0
45
46          rad = 1 / np.sqrt(self.no_of_in_nodes + bias_node)
47          X = truncated_normal(mean=0,
48                               sd=1,
49                               low=-rad,
50                               upp=rad)
51          self.wih = X.rvs((self.no_of_hidden_nodes,
52                            self.no_of_in_nodes + bias_node))
53
54          rad = 1 / np.sqrt(self.no_of_hidden_nodes + bias_node
    )
55          X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
56          self.who = X.rvs((self.no_of_out_nodes,
57                            self.no_of_hidden_nodes + bias_node
    ))
58
59
60
61      def train(self, input_vector, target_vector):
62          """
63          input_vector and target_vector can
64          be tuple, list or ndarray
65          """
66
67          bias_node = 1 if self.bias else 0
68          if self.bias:
69              # adding bias node to the end of the inpuy_vector
70              input_vector = np.concatenate((input_vector,
71                                             [self.bias]) )
72
73
74          input_vector = np.array(input_vector, ndmin=2).T
75          target_vector = np.array(target_vector, ndmin=2).T
76
77
78          output_vector1 = np.dot(self.wih,
79                                  input_vector)
80          output_hidden = activation_function(output_vector1)
81
82          if self.bias:
83              output_hidden = np.concatenate((output_hidden,
84                                              [[self.bias]]) )
85
```

```
86
87          output_vector2 = np.dot(self.who,
88                                  output_hidden)
89          output_network = activation_function(output_vector2)
90
91          output_errors = target_vector - output_network
92          # update the weights:
93          tmp = output_errors * output_network * (1.0 -
    output_network)
94          tmp = self.learning_rate  * np.dot(tmp, output_hidden
    .T)
95          self.who += tmp
96
97
98          # calculate hidden errors:
99          hidden_errors = np.dot(self.who.T,
100                                 output_errors)
101         # update the weights:
102         tmp = hidden_errors * output_hidden * (1.0 -
    output_hidden)
103         if self.bias:
104             x = np.dot(tmp, input_vector.T)[:-1,:]
105         else:
106             x = np.dot(tmp, input_vector.T)
107         self.wih += self.learning_rate * x
108
109
110
111     def run(self, input_vector):
112         """
113         input_vector can be tuple, list or ndarray
114         """
115
116         if self.bias:
117             # adding bias node to the end of the inpuy_vector
118             input_vector = np.concatenate((input_vector, [1])
     )
119             input_vector = np.array(input_vector, ndmin=2).T
120
121             output_vector = np.dot(self.wih,
122                                    input_vector)
123             output_vector = activation_function(output_vector
    )
124
125             if self.bias:
126                 output_vector = np.concatenate( (
    output_vector,
127                                                  [[1]]) )
128
```

```
129
130             output_vector = np.dot(self.who,
131                                    output_vector)
132             output_vector = activation_function(output_vector
    )
133             return output_vector
134
135     def evaluate(self, data, labels):
136         corrects, wrongs = 0, 0
137         for i in range(len(data)):
138             res = self.run(data[i])
139             res_max = res.argmax()
140             if res_max == labels[i]:
141                 corrects += 1
142             else:
143                 wrongs += 1
144         return corrects, wrongs
```

```
ANN = NeuralNetwork(no_of_in_nodes=image_pixels,
                    no_of_out_nodes=10,
                    no_of_hidden_nodes=200,
                    learning_rate=0.1,
                    bias=None)


for i in range(len(train_imgs)):
    ANN.train(train_imgs[i], train_labels_one_hot[i])
for i in range(20):
    res = ANN.run(test_imgs[i])
    print(test_labels[i], np.argmax(res), np.max(res))

[7.] 7 0.9951478957895473
[2.] 2 0.9167137305226186
[1.] 1 0.9930670538508068
[0.] 0 0.9729093609525741
[4.] 4 0.9475097483176407
[1.] 1 0.9919906877733081
[4.] 4 0.9390079959736829
[9.] 9 0.9815469745110644
[5.] 5 0.23871278844097427
[9.] 9 0.9863859218561386
[0.] 0 0.9667234471027278
[6.] 6 0.8856024953669486
[9.] 9 0.9928943830319253
[0.] 0 0.96922568081586
[1.] 1 0.9899747475376088
[5.] 5 0.9595147911735664
[9.] 9 0.9958119066147573
[7.] 7 0.9883146384365381
```

```
[3.] 3 0.8706223167904136
[4.] 4 0.9912284156702522

corrects, wrongs = ANN.evaluate(train_imgs, train_labels)
print("accuracy train: ", corrects / ( corrects + wrongs))
corrects, wrongs = ANN.evaluate(test_imgs, test_labels)
print("accuracy: test", corrects / ( corrects + wrongs))

accruracy train:  0.9555666666666667
accruracy: test 0.9544
```

4.1.3 Bias va Epoch ishlatilgan versiya:

```
import numpy as np

@np.vectorize
def sigmoid(x):
    return 1 / (1 + np.e ** -x)
activation_function = sigmoid

from scipy.stats import truncnorm

def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm((low - mean) / sd,
                     (upp - mean) / sd,
                     loc=mean,
                     scale=sd)


class NeuralNetwork:

    def __init__(self,
                 no_of_in_nodes,
                 no_of_out_nodes,
                 no_of_hidden_nodes,
                 learning_rate,
                 bias=None
                 ):

        self.no_of_in_nodes = no_of_in_nodes
        self.no_of_out_nodes = no_of_out_nodes

        self.no_of_hidden_nodes = no_of_hidden_nodes

        self.learning_rate = learning_rate
        self.bias = bias
        self.create_weight_matrices()


```

```
37
38      def create_weight_matrices(self):
39          """
40          A method to initialize the weight matrices
41          of the neural network with optional
42          bias nodes"""
43
44          bias_node = 1 if self.bias else 0
45
46          rad = 1 / np.sqrt(self.no_of_in_nodes + bias_node)
47          X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
48          self.wih = X.rvs((self.no_of_hidden_nodes,
49                            self.no_of_in_nodes + bias_node))
50
51          rad = 1 / np.sqrt(self.no_of_hidden_nodes + bias_node
    )
52          X = truncated_normal(mean=0,
53                               sd=1,
54                               low=-rad,
55                               upp=rad)
56          self.who = X.rvs((self.no_of_out_nodes,
57                            self.no_of_hidden_nodes + bias_node
    ))
58
59
60      def train_single(self, input_vector, target_vector):
61          """
62          input_vector and target_vector can be tuple,
63          list or ndarray
64          """
65
66          bias_node = 1 if self.bias else 0
67          if self.bias:
68              # adding bias node to the end of the inpuy_vector
69              input_vector = np.concatenate( (input_vector,
70                                              [self.bias]) )
71
72          output_vectors = []
73          input_vector = np.array(input_vector, ndmin=2).T
74          target_vector = np.array(target_vector, ndmin=2).T
75
76
77          output_vector1 = np.dot(self.wih,
78                                  input_vector)
79          output_hidden = activation_function(output_vector1)
80
81          if self.bias:
82              output_hidden = np.concatenate((output_hidden,
83                                              [[self.bias]]) )
```

```
84
85
86          output_vector2 = np.dot(self.who,
87                                  output_hidden)
88          output_network = activation_function(output_vector2)
89
90          output_errors = target_vector - output_network
91          # update the weights:
92          tmp = output_errors * output_network * (1.0 -
    output_network)
93          tmp = self.learning_rate  * np.dot(tmp,
94                                             output_hidden.T)
95          self.who += tmp
96
97
98          # calculate hidden errors:
99          hidden_errors = np.dot(self.who.T,
100                                 output_errors)
101         # update the weights:
102         tmp = hidden_errors * output_hidden * (1.0 -
    output_hidden)
103         if self.bias:
104             x = np.dot(tmp, input_vector.T)[:-1,:]
105         else:
106             x = np.dot(tmp, input_vector.T)
107         self.wih += self.learning_rate * x
108
109
110     def train(self, data_array,
111         labels_one_hot_array,
112         epochs=1,
113         intermediate_results=False):
114         intermediate_weights = []
115         for epoch in range(epochs):
116             for i in range(len(data_array)):
117                 self.train_single(data_array[i],
118                                   labels_one_hot_array[i])
119             if intermediate_results:
120                 intermediate_weights.append((self.wih.copy(),
121                                              self.who.copy())
    )
122         return intermediate_weights
123
124
125
126
127     def run(self, input_vector):
128         # input_vector can be tuple, list or ndarray
129
```

```
        if self.bias:
            # adding bias node to the end of the inpuy_vector
            input_vector = np.concatenate( (input_vector,
                                            [self.bias]) )
        input_vector = np.array(input_vector, ndmin=2).T

        output_vector = np.dot(self.wih,
                               input_vector)
        output_vector = activation_function(output_vector)

        if self.bias:
            output_vector = np.concatenate( (output_vector,
                                             [[self.bias]]) )


        output_vector = np.dot(self.who,
                               output_vector)
        output_vector = activation_function(output_vector)

        return output_vector


    def evaluate(self, data, labels):
        corrects, wrongs = 0, 0
        for i in range(len(data)):
            res = self.run(data[i])
            res_max = res.argmax()
            if res_max == labels[i]:
                corrects += 1
           else:
                wrongs += 1
        return corrects, wrongs
```

```
epochs = 12

network = NeuralNetwork(no_of_in_nodes=image_pixels,
                        no_of_out_nodes=10,
                        no_of_hidden_nodes=100,
                        learning_rate=0.1,
                        bias=None)

weights = network.train(train_imgs,
                        train_labels_one_hot,
                        epochs=epochs,
                        intermediate_results=True)
for epoch in range(epochs):
    print("epoch: ", epoch)
    network.wih = weights[epoch][0]
    network.who = weights[epoch][1]
```

```
    corrects, wrongs = network.evaluate(train_imgs,
                                        train_labels)
    print("accuracy train: ", corrects / ( corrects + wrongs)
    )
    corrects, wrongs = network.evaluate(test_imgs,
                                        test_labels)
    print("accuracy test: ", corrects / ( corrects + wrongs))


epoch:  0
accruracy train:  0.9428166666666666
accruracy test:  0.9415
epoch:  1
accruracy train:  0.9596666666666667
accruracy test:  0.9548
epoch:  2
accruracy train:  0.9673166666666667
accruracy test:  0.9599
epoch:  3
accruracy train:  0.9693
accruracy test:  0.9601
epoch:  4
accruracy train:  0.97195
accruracy test:  0.9631
epoch:  5
accruracy train:  0.9750666666666666
accruracy test:  0.9659
epoch:  6
accruracy train:  0.97705
accruracy test:  0.9662
epoch:  7
accruracy train:  0.9767666666666667
accruracy test:  0.9644
epoch:  8
accruracy train:  0.9765666666666667
accruracy test:  0.9643
epoch:  9
accruracy train:  0.9771
accruracy test:  0.9643
epoch:  10
accruracy train:  0.9780333333333333
accruracy test:  0.9627
epoch:  11
accruracy train:  0.97875
accruracy test:  0.9638
```

```
epochs = 12


```

```
with open("nist_tests.csv", "w") as fh_out:
    for hidden_nodes in [20, 50, 100, 120, 150]:
        for learning_rate in [0.01, 0.05, 0.1, 0.2]:
            for bias in [None, 0.5]:
                network = NeuralNetwork(no_of_in_nodes=image_pixels,
                                        no_of_out_nodes=10,
                                        no_of_hidden_nodes=hidden_nodes,
                                        learning_rate=learning_rate,
                                        bias=bias)
                weights = network.train(train_imgs,
                                        train_labels_one_hot,
                                        epochs=epochs,
                                        intermediate_results=True)
                for epoch in range(epochs):
                    print("*", end="")
                    network.wih = weights[epoch][0]
                    network.who = weights[epoch][1]
                    train_corrects, train_wrongs = network.evaluate(train_imgs,
                                                                    train_labels)
                    test_corrects, test_wrongs = network.evaluate(test_imgs,
                                                                  test_labels)
                    outstr = str(hidden_nodes) + " " + str(learning_rate) + " " + str(bias)
                    outstr += " " + str(epoch) + " "
                    outstr += str(train_corrects / (train_corrects + train_wrongs)) + " "
                    outstr += str(train_wrongs / (train_corrects + train_wrongs)) + " "
                    outstr += str(test_corrects / (test_corrects + test_wrongs)) + " "
                    outstr += str(test_wrongs / (test_corrects + test_wrongs))

                    fh_out.write(outstr + "\n" )
                    fh_out.flush()
```

*nist_tests*_20_50_100_120_150.*csv* fayli oldingi dasturning natijalarini o'z ichiga oladi.

Bir nechta yashirin qatlamlarga ega bo'lgan to'rlar

Biz yangi neyron to'ri sinfini yozamiz, unda yashirin qatlamlarning tasodifiy sonini aniqlashimiz mumkin. Kod yana takomillashtirildi, chunki vazn matritsalari endi takrorlanuvchi protses ichiga kirtilgan:

```
import numpy as np
from scipy.special import expit as activation_function
from scipy.stats import truncnorm

def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm((low - mean) / sd,
                     (upp - mean) / sd,
                     loc=mean,
                     scale=sd)


class NeuralNetwork:


    def __init__(self,
                 network_structure, # ie. [input_nodes,
   hidden1_nodes, ... , hidden_n_nodes, output_nodes]
                 learning_rate,
                 bias=None
                 ):

        self.structure = network_structure
        self.learning_rate = learning_rate
        self.bias = bias
        self.create_weight_matrices()


    def create_weight_matrices(self):

        bias_node = 1 if self.bias else 0
        self.weights_matrices = []

        layer_index = 1
        no_of_layers = len(self.structure)
        while layer_index < no_of_layers:
            nodes_in = self.structure[layer_index-1]
            nodes_out = self.structure[layer_index]
            n = (nodes_in + bias_node) * nodes_out
            rad = 1 / np.sqrt(nodes_in)
            X = truncated_normal(mean=2,
                                 sd=1,
                                 low=-rad,
                                 upp=rad)
```

```
            wm = X.rvs(n).reshape((nodes_out, nodes_in +
bias_node))
            self.weights_matrices.append(wm)
            layer_index += 1



    def train(self, input_vector, target_vector):
        """
        input_vector and target_vector can be tuple,
        list or ndarray
        """

        no_of_layers = len(self.structure)
        input_vector = np.array(input_vector, ndmin=2).T
        layer_index = 0
        # The output/input vectors of the various layers:
        res_vectors = [input_vector]
        while layer_index < no_of_layers - 1:
            in_vector = res_vectors[-1]
            if self.bias:
                # adding bias node to the end of the 'input'
_vector
                in_vector = np.concatenate( (in_vector,
                                             [[self.bias]]) )
                res_vectors[-1] = in_vector
            x = np.dot(self.weights_matrices[layer_index],
                       in_vector)
            out_vector = activation_function(x)
            # the output of one layer is the input of the
next one:
            res_vectors.append(out_vector)
            layer_index += 1

        layer_index = no_of_layers - 1
        target_vector = np.array(target_vector, ndmin=2).T
        # The input vectors to the various layers
        output_errors = target_vector - out_vector
        while layer_index > 0:
            out_vector = res_vectors[layer_index]
            in_vector = res_vectors[layer_index-1]

            if self.bias and not layer_index==(no_of_layers
-1):
                out_vector = out_vector[:-1,:].copy()

            tmp = output_errors * out_vector * (1.0 -
out_vector)
            tmp = np.dot(tmp, in_vector.T)
```

```
87
88              #if self.bias:
89              #    tmp = tmp[:-1,:]
90
91              self.weights_matrices[layer_index-1] += self.
    learning_rate * tmp
92
93              output_errors = np.dot(self.weights_matrices[
    layer_index-1].T,
94                                     output_errors)
95              if self.bias:
96                  output_errors = output_errors[:-1,:]
97              layer_index -= 1
98
99
100
101
102     def run(self, input_vector):
103         # input_vector can be tuple, list or ndarray
104
105         no_of_layers = len(self.structure)
106         if self.bias:
107             # adding bias node to the end of the inpuy_vector
108             input_vector = np.concatenate( (input_vector,
109                                             [self.bias]) )
110         in_vector = np.array(input_vector, ndmin=2).T
111
112         layer_index = 1
113         # The input vectors to the various layers
114         while layer_index < no_of_layers:
115             x = np.dot(self.weights_matrices[layer_index-1],
116                        in_vector)
117             out_vector = activation_function(x)
118
119             # input vector for next layer
120             in_vector = out_vector
121             if self.bias:
122                 in_vector = np.concatenate( (in_vector,
123                                              [[self.bias]]) )
124
125             layer_index += 1
126
127
128         return out_vector
129
130     def evaluate(self, data, labels):
131         corrects, wrongs = 0, 0
132         for i in range(len(data)):
133             res = self.run(data[i])
```

```
            res_max = res.argmax()
            if res_max == labels[i]:
                corrects += 1
            else:
                wrongs += 1
        return corrects, wrongs
```

```
ANN = NeuralNetwork(network_structure=[image_pixels, 50, 50,
    10],
                                learning_rate=0.1,
                                bias=None)


for i in range(len(train_imgs)):
    ANN.train(train_imgs[i], train_labels_one_hot[i])
```

```
corrects, wrongs = ANN.evaluate(train_imgs, train_labels)
print("accuracy train: ", corrects / ( corrects + wrongs))
corrects, wrongs = ANN.evaluate(test_imgs, test_labels)
print("accuracy: test", corrects / ( corrects + wrongs))
```

Bir nechta yashirin qatlam va epochlar bo'lgan to'rlar

```
import numpy as np
from scipy.special import expit as activation_function
from scipy.stats import truncnorm

def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm((low - mean) / sd,
                     (upp - mean) / sd,
                     loc=mean,
                     scale=sd)


class NeuralNetwork:


    def __init__(self,
                 network_structure, # ie. [input_nodes,
    hidden1_nodes, ... , hidden_n_nodes, output_nodes]
                 learning_rate,
                 bias=None
                 ):

        self.structure = network_structure
        self.learning_rate = learning_rate
        self.bias = bias
        self.create_weight_matrices()
```

```
25
26
27
28      def create_weight_matrices(self):
29          X = truncated_normal(mean=2, sd=1, low=-0.5, upp=0.5)
30
31          bias_node = 1 if self.bias else 0
32          self.weights_matrices = []
33          layer_index = 1
34          no_of_layers = len(self.structure)
35          while layer_index < no_of_layers:
36              nodes_in = self.structure[layer_index-1]
37              nodes_out = self.structure[layer_index]
38              n = (nodes_in + bias_node) * nodes_out
39              rad = 1 / np.sqrt(nodes_in)
40              X = truncated_normal(mean=2, sd=1, low=-rad, upp=
    rad)
41              wm = X.rvs(n).reshape((nodes_out, nodes_in +
    bias_node))
42              self.weights_matrices.append(wm)
43              layer_index += 1
44
45
46
47      def train_single(self, input_vector, target_vector):
48          # input_vector and target_vector can be tuple, list
    or ndarray
49
50          no_of_layers = len(self.structure)
51          input_vector = np.array(input_vector, ndmin=2).T
52
53          layer_index = 0
54          # The output/input vectors of the various layers:
55          res_vectors = [input_vector]
56          while layer_index < no_of_layers - 1:
57              in_vector = res_vectors[-1]
58              if self.bias:
59                  # adding bias node to the end of the 'input'
    _vector
60                  in_vector = np.concatenate( (in_vector,
61                                               [[self.bias]]) )
62                  res_vectors[-1] = in_vector
63              x = np.dot(self.weights_matrices[layer_index],
    in_vector)
64              out_vector = activation_function(x)
65              res_vectors.append(out_vector)
66              layer_index += 1
67
68          layer_index = no_of_layers - 1
```

```
69          target_vector = np.array(target_vector, ndmin=2).T
70          # The input vectors to the various layers
71          output_errors = target_vector - out_vector
72          while layer_index > 0:
73              out_vector = res_vectors[layer_index]
74              in_vector = res_vectors[layer_index-1]
75
76              if self.bias and not layer_index==(no_of_layers
    -1):
77                  out_vector = out_vector[:-1,:].copy()
78
79              tmp = output_errors * out_vector * (1.0 -
    out_vector)
80              tmp = np.dot(tmp, in_vector.T)
81
82              #if self.bias:
83              #    tmp = tmp[:-1,:]
84
85              self.weights_matrices[layer_index-1] += self.
    learning_rate * tmp
86
87              output_errors = np.dot(self.weights_matrices[
    layer_index-1].T,
88                                     output_errors)
89              if self.bias:
90                  output_errors = output_errors[:-1,:]
91              layer_index -= 1
92
93
94
95
96      def train(self, data_array,
97                labels_one_hot_array,
98                epochs=1,
99                intermediate_results=False):
100         intermediate_weights = []
101         for epoch in range(epochs):
102             for i in range(len(data_array)):
103                 self.train_single(data_array[i],
    labels_one_hot_array[i])
104             if intermediate_results:
105                 intermediate_weights.append((self.wih.copy(),
106                                              self.who.copy())
    )
107         return intermediate_weights
108
109
110
111
```

```
    def run(self, input_vector):
        # input_vector can be tuple, list or ndarray

        no_of_layers = len(self.structure)
        if self.bias:
            # adding bias node to the end of the inpuy_vector
            input_vector = np.concatenate( (input_vector, [self.bias]) )
            in_vector = np.array(input_vector, ndmin=2).T

        layer_index = 1
        # The input vectors to the various layers
        while layer_index < no_of_layers:
            x = np.dot(self.weights_matrices[layer_index-1],
                       in_vector)
            out_vector = activation_function(x)

            # input vector for next layer
            in_vector = out_vector
            if self.bias:
                in_vector = np.concatenate( (in_vector,
                                             [[self.bias]]) )

            layer_index += 1


        return out_vector

    def evaluate(self, data, labels):
        corrects, wrongs = 0, 0
        for i in range(len(data)):
            res = self.run(data[i])
            res_max = res.argmax()
            if res_max == labels[i]:
                corrects += 1
            else:
                wrongs += 1
        return corrects, wrongs
```

```
epochs = 3

ANN = NeuralNetwork(network_structure=[image_pixels, 80, 80, 10],
                                learning_rate=0.01,
                                bias=None)


ANN.train(train_imgs, train_labels_one_hot, epochs=epochs)
```

```
corrects, wrongs = ANN.evaluate(train_imgs, train_labels)
print("accuracy train: ", corrects / ( corrects + wrongs))
corrects, wrongs = ANN.evaluate(test_imgs, test_labels)
print("accuracy: test", corrects / ( corrects + wrongs))
```

Bibliography

[1] Santanu Pattanayak. *Pro Deep Learning with TensorFlow*. Apress, Bangalore, Karnataka, India, 2017.

[2] Denis Rothman. *Artificial Intelligence By Example*. Packt, Birmingham, UK, 2018

[3] Sandro Skansi. *Introduction to Deep Learning*. Springer, Zagreb, Croatia, 2018

[4] Michael Nielsen. *Neural Networks and Deep Learning*. `http://neuralnetworksanddeeplearning.com`

[5] Wan, Li; Matthew Zeiler; Sixin Zhang; Yann LeCun; Rob Fergus. *Regularization of Neural Network using DropConnect*. International Conference on Machine Learning (ICML), 2013

[6] Ramachandran, Prajit; Barret, Zoph; Quoc, V. Le. *Searching for Activation Functions*. October 16, 2017

[7] Simon Haykin. *Neural Networks*. Pearson, Hamilton, Canada, 2nd edition, 1999

[8] Simon Haykin. *Neural Networks and Learning Machines* . Pearson, Hamilton, Canada, 3rd edition, 2009

[9] Simon Haykin. *Introduction to the Math of Neural Networks*. Heaton Research, 2012

[10] Andreas C. Müller & Sarah Guido. *Introduction to Machine Learning with Python*. O'Reilly Media, USA, 2017