ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

Ս. Գ. ՍԱՐԳՍՅԱՆ, Ա. Ս. ՀՈՎԱԿԻՄՅԱՆ,
Ս. Ռ. ՀԱԿՈԲՅԱՆ

ՄԵՔԵՆԱՅԱԿԱՆ ՈՒՍՈՒՑՈՒՄ PYTHON ԼԵԶՎԻ ԿԻՐԱՌՄԱՄԲ

ՈՒՍՈՒՄՆԱԿԱՆ ՁԵՌՆԱՐԿ

ԵՐԵՎԱՆ
ԵՊՀ ՀՐԱՏԱՐԱԿՉՈՒԹՅՈՒՆ
2025

ՀՏԴ 004.42(07)
ԳՄԴ 32.973գ7
Ս 259

Հրատարակության է երաշխավորել ԵՊՀ գիտական խորհուրդը:

Սիրանուշ Սարգսյան - գլուխ 1, գլուխ 2-ի 2.1-2.5 բաժիններ, գլուխ 3-ի 3.1 բաժին

Աննա Հովակիմյան - գլուխ 2-ի 2.6-2.9, գլուխ 3-ի 3.1, 3.2 բաժիններ

Սոնիկ Հակոբյան - Python լեզվով իրականացրել է ձեռնարկում ներառված մեքենայական ուսուցման մոդելների ծրագրերը:

Սարգսյան Ս. Գ., Հովակիմյան Ա. Ս., Հակոբյան Ս. Ռ.

Ս 259 Մեքենայական ուսուցում Python լեզվի կիրառմամբ: Ուսումնական ձեռնարկ / Ս. Գ. Սարգսյան, Ա. Ս. Հովակիմյան, Ս. Ռ. Հակոբյան: -Եր., ԵՊՀ հրատ., 2025, 162 էջ:

Ուսումնական ձեռնարկում նկարագրված են արհեստական բանականության հասկացություններ, ինչպես նաև նրա հիմնական մաս կազմող մեքենայական ուսուցման հավելվածների մշակման մեթոդներ: Ձեռնարկը նախատեսված է ինֆորմատիկայի և կիրառական մաթեմատիկայի ֆակուլտետի ուսանողների համար: Այն կարող է օգտակար լինել ինչպես սկսնակ ծրագրավորողների, այնպես էլ փորձառու մասնագետների համար, ովքեր նախատեսում են մշակել մեքենայական ուսուցման մոդելներ Python ծրագրավորման լեզվով: Ձեռնարկում նկարագրված են գրեթե բոլոր հիմնական մեթոդներն ու միջոցները, որոնք անհրաժեշտ են նեյրոնային ցանցերի մշակման համար: Բերված են Python լեզվով իրականացված 38 ծրագրեր, որտեղ օգտագործվել են հայտնի գրադարանների հիմնական դասերը:

ՀՏԴ 004.42(07)
ԳՄԴ 32.973գ7

ISBN 978-5-8084-2711-2
https://doi.org/10.46991/YSUPH/9785808427112

Բովանդակություն

Ներածություն

Արհեստական բանականությունը հնարավորություն է տալիս մտածել և խելամիտ որոշումներ կայացնել այնպես, ինչպես մարդիկ: Արհեստական բանականության մոդելների միջոցով ուսուցման արդյունքում ձեռք է բերվում փորձ, որից հետո իրականացվում են տարբեր առաջադրանքներ: Մեքենայական ուսուցումը, մասնավորապես նեյրոնային ցանցերը գնալով ավելի են տարածվում կիրառական տարբեր ոլորտներում: Այդ պատճառով ավելի ու ավելի շատ միջոցներ են ներդրվում այդ տեխնոլոգիաների զարգացման համար:

Համակարգիչների հայտնագործումից ի վեր՝ տարբեր առաջադրանքներ կատարելու նրանց կարողությունը անընդհատ ընդլայնվում է: Նրանց սովորեցրել են լսել և հասկանալ մարդու խոսքը, նկարներում և տեսաֆայլերում ճանաչել օբյեկտներ, գրել բանաստեղծություն կամ երաժշտություն, կառավարել անօդաչու սարքեր ու ինքնաթիռները, որոշել մարդկանց զգացմունքները և այլն:

Արհեստական բանականության համակարգերի մշակումը տարբերվում է սովորական տեղեկատվական համակարգերի մշակումից: Դրանց վրա աշխատելիս օգտագործվում են տեխնոլոգիական այլ մոտեցումներ՝ մոդելների համար ուսուցման տվյալների հավաքածուների և ալգորիթմների մշակում, դրանց ծրագրային իրականացումներ և այլն:

Արհեստական բանականության համակարգերի մշակման և ներդրման համար անհրաժեշտ է ունենալ ծրագրավորման պարզ ու հարմար լեզու, ինչպես նաև դրանք օգտագործելու պատրաստի գրադարաններ: Python-ը հենց այդպիսի լեզուներից մեկն է:

Նեյրոնային ցանցերի ստեղծման և մեքենայական ուսուցման նախագծերը հիմնականում գրվում են Python-ով: Այս դեպքում նեյրոնային ցանցի կառուցվածքի ձևավորման, դրա ուսուցման գործընթացում բարդ հաշվարկներ կատարելու համար օգտագործվում են Python-ի գրադարանների մոդուլները:

Նեյրոնային ցանցերի նախագծումը ներառում է երեք փուլ՝ ցանցի կառուցվածքի ձևավորում, ուսուցման տվյալների պատրաստում և ցանցի ուսուցում: Նեյրոնային ցանցի ուսուցումը սովորաբար պահանջում է հաշվողական հզոր ռեսուրսներ: Բնական է, որ այդ գործընթացի ծրագրային ապահովումը նաև պահանջում է ծրագրավորման C կամ C++ լեզուների օգտագործում: Այդ նպատակների համար կան ստանդարտ, լավ մշակված և արդյունավետ օգտագործվող գրադարաններ, որոնցից են keras, TensorFlow, Theano և այլն:

Ուսումնական ձեռնարկը ներառում է արհեստական բանականության (AI, Artificial Intelligence), տվյալագիտության, ծրագրավորման և հարակից ոլորտներում նախագծերի իրականացման համար անհրաժեշտ ժամանակակից մեթոդներ և գործիքներ:

ML (Machine Learning) ինժեներները, տվյալագիտության մասնագետները, AI հետազոտողները և ծրագրավորողները այժմ առավել պահանջված են ՏՏ ոլորտում: Խորը գիտելիքների տիրապետումը նրանց մեծ հնարավորություն է տալիս ստանալ մրցունակ աշխատատեղեր: Այս ձեռնարկը կարող է ապագա մասնագիտական զարգացման հիմք ծառայել:

Ներկայացված նյութը արդիական է մի քանի հիմնական պատճառներով.

- Մեքենայական ուսուցումը լայնորեն կիրառվում է բիզնեսում, բժշկության մեջ, ֆինանսական ոլորտում, ռոբոտատեխնիկայում, խաղերի ծրագրավորման ոլորտում, պատկերների մշակման և տեքստերի վերլուծության խնդիրներում: Ուստի, ձեռնարկում տրված գիտելիքներն այսօր անհրաժեշտ են տարբեր մասնագետների համար:
- Այժմ Python ծրագրավորման լեզուն դարձել է մեքենայական ուսուցման և արհեստական բանականության նախագծերի ծրագրավորման հիմնական լեզու: Scikit-learn, TensorFlow, Keras, NumPy, Pandas, Matplotlib գրադարանները լայնորեն կիրառվում են, և դրանց իմացությունը մեծ առավելություն է աշխատաշուկայում:

Ուսումնական ձեռնարկը կարևոր և օգտակար է հատկապես նրանց համար, ովքեր ցանկանում են խորանալ արհեստական բանականության և մեքենայական ուսուցման ոլորտներում: Այն տրամադրում է հիմնարար գիտելիքներ, որոնք անհրաժեշտ են նեյրոնային ցանցերը, դրանց կառուցվածքը, ուսուցման մեթոդները և ծրագրային իրականացումները հասկանալու համար:

Ձեռնարկում բերված են բազմաթիվ գործնական օրինակներ, որոնք ուսանողներին թույլ կտան իրենց գիտելիքները կիրառել Python լեզվի միջոցով ծրագրավորելիս: Ներկայացված են ծրագրային կոդեր, որոնք օգնում են հասկանալ, թե ինչպես են կառուցվում և աշխատում նեյրոնային ցանցերը:

Տրվում են AI-ի հայտնի գրադարանների (scikit-learn, Keras, TensorFlow) օգտագործման օրինակներ, ինչը հնարավորություն կտա ուսանողներին աշխատել արդիական գործիք-

ներով։ Ներկայացված են նեյրոնային ցանցերի կառուցման ինչպես դասական մեթոդներ, այնպես էլ խորը ուսուցման (Deep Learning) կիրառություններ։

Գլուխ 1-ում բերված են արհեստական բանականության ոլորտի հիմնական հասկացությունները։ Նկարագրված են նեյրոնային ցանցերի հիմք հանդիսացող արհեստական նեյրոնը և Python-ի միջոցով նրա գործառույթների ծրագրավորման ձևերը։ Նկարագրված են ակտիվացման ֆունկցիաները և դրանց տեսակները, միաշերտ և բազմաշերտ նեյրոնային ցանցերը։

Դիտարկվում են ցանցի կառուցվածքի ձևավորումը, նեյրոնային ցանցերի տարբեր տեսակները, ուսուցման և թեստային նմուշների ստեղծումը, ցանցի ուսուցումը և դրա գործնական օգտագործումը։ Նկարագրվում են նեյրոնային ցանցի կառուցման և ցանցի ուսուցման տարբեր տեխնոլոգիաներ։

Ներկայացվում է պերցեպտրոնը որպես արհեստական նեյրոնային ցանցի պարզ տեսակ։ Մանրամասն դիտարկվում է պերցեպտրոնի ուսուցման գործընթացը ուսուցման հատուկ ալգորիթմների միջոցով։

Գլուխ 2-ում բերված են նեյրոնային ցանցի տարրերի, մասնավորապես պերցեպտրոնի մշակման և օգտագործման օրինակներ։ Տրված են օբյեկտների դասակարգման խնդիրների լուծման սկզբունքները, պերցեպտրոնի միջոցով պատկերների ճանաչման ծրագրային մոդուլները։ Բերված են օբյեկտների դասակարգման ծրագրային կոդի օրինակներ։ Այդ օրինակներում օգտագործվում են Python-ը և որոշ գրադարաններ մաթեմատիկական ֆունկցիաների հետ աշխատելու համար։ Նշենք, որ այս դեպքում չեն կիրառվում մասնագիտացված

գրադարաններ, այդ պատճառով ծրագրի կոդը բավականին մեծ ու բարդ է ստացվում:

Այնուամենայնիվ, այս մոտեցումը թույլ է տալիս ավելի լավ հասկանալ բոլոր գործընթացները, որոնք տեղի են ունենում նեյրոնային ցանցերում, և ավելի խորությամբ ուսումնասիրել դրանց հիմնական մեխանիզմները:

Գլուխ 3-ում նկարագրվում են մասնագիտացված գրադարանները և դրանց օգտագործումը: Այս մոտեցումը արագացնում և պարզեցնում է նեյրոնային ցանցերի ծրագրավորման գործընթացը:

Օգտագործվում են Python-ի հետ աշխատող որոշ գրադարաններ (scikit-leam, Keras, TensorFlow), որոնք թույլ են տալիս լուծել գործնական խնդիրների բավականին մեծ դաս՝ զգալիորեն հեշտացնելով ծրագրավորողների աշխատանքը:

Նեյրոնային ցանցերի կառուցման համար կիրառվում են ուսուցման տվյալների պատրաստի հավաքածուներ: Սա թույլ է տալիս ավելի շատ կենտրոնանալ ուսուցման ալգորիթմների վրա՝ առանց ժամանակ վատնելու տվյալների հավաքածուն ինքնուրույն ստեղծելու համար: Օգտագործված գրադարաններից յուրաքանչյուրի հնարավորությունները նկարագրված են օրինակներում, որոնցում օգտագործվող պատրաստի ուսուցման տվյալների հավաքածուները կարելի է ներբեռնել համացանցից:

Այնուամենայնիվ, հնարավոր է ավելի պարզեցնել արհեստական բանականության հետ կապված ծրագրային գործիքների ստեղծումը: Կարելի է օգտագործել ոչ միայն պատրաստի տվյալների հավաքածուներ, այլև արդեն կառուցված նեյրոնային ցանցերի մոդելներ: Բացի այդ, նեյրոնային ցանցերի առավելությունն այն է, որ ցանցի նույն մոդելը կարող է լուծել

նաև բոլորովին այլ տիպի խնդիրներ: Դա կախված կլինի այն բանից, թե ինչ և ինչպես են նրան սովորեցրել:

Հետևաբար, կարևոր է ոչ միայն իմանալ, թե ինչպես նախագծել նեյրոնային ցանցեր, այլև պատրաստել դրանց ուսուցման համար տվյալների տարբեր հավաքածուներ:

Ձեռնարկում մեկնաբանված հարցերը լուծված են պարզ, հասկանալի և ծրագրորեն իրականացված օրինակներով:

Ցանկալի է ներկայացված թեմաները ուսումնասիրել համակարգչի մոտ աշխատելով և տեսնել գործողությունների արդյունքները: Դա կօգնի գրքում ներկայացված նյութերի ավելի լավ ըմբռնմանը և Python ծրագրավորման լեզվով աշխատելու հմտությունների ձևավորմանը:

Կարևորում ենք այն փաստը, որ գրքում շարադրված նյութը տրված է ուսուցման տարբեր մակարդակների ձևավորման համար: Առաջին մակարդակում նեյրոնը և նեյրոնային ցանցը կառուցվում են Python լեզվի հրամանների միջոցով: Դրանց աշխատանքը իրականացված է պարզ և հեշտ կիրառելի:

Երկրորդ մակարդակում նեյրոնային ցանցը կառուցվում է Python լեզվի գրադարանների օգտագործմամբ՝ համապատասխան գրադարանների մոդուլների միջոցով:

Երրորդ մակարդակում նեյրոնային ցանցը կառուցվում է պատրաստի հարթակի գործիքաշարի միջոցով, որի դեպքում պետք չէ ժամանակ վատնել մոդելի ուսուցման համար: Փորձարկումներում կարևորվում է օգտագործվող ալգորիթմների արդյունավետությունը:

Տվյալների ծավալն ու բարդությունը առաջացնում են նեյրոնային ցանցը մշակելու և վերլուծելու անհրաժեշտություն: Ի վերջո, նեյրոնային ցանցն ապահովում է ստանալ շատ ավելի լավ գնահատականներ և կատարել ավելի ճշգրիտ կանխա-

տեսումներ: Դա զգալիորեն բարելավում է ցանցի աշխատանքի արդյունավետությունը և բարձրացնում նրա արտադրողականությունը:

Ուսումնական ձեռնարկը կազմված է այնպես, որ սկսնակները կարողանան սովորել նեյրոնային ցանցերի հիմունքները, իսկ առաջնակարգ ուսանողները կարողանան խորանալ տվյալների վերլուծության և նեյրոնային ցանցերի բարդ կառուցվածքների մեջ: Տրվում են տարբեր ծրագրային մոտեցումներ՝ սկսած պարզ օրինակներից մինչև բարդ ալգորիթմներ: Ուսանողները ոչ միայն տեսականորեն կյուրացնեն մեքենայական ուսուցման սկզբունքները, այլև ձեռք կբերեն պրակտիկ հմտություններ:

Գլուխ 1
Արհեստական բանականության տարրեր

Արհեստական բանականությունը (AI, Artificial Intelligence) համակարգչային գիտության ոլորտ է, որի շրջանակում մշակվում են համակարգչային ծրագրեր այնպիսի խնդիրներ իրականացնելու համար, որոնք ինչ-որ չափով իմիտացնում են մարդու մտավոր գործունեությունը: Այդպիսի ծրագրերը կարող են ընդհանրացումներ և եզրակացություններ անել, բացահայտել տվյալների միջև կապեր, ինչպես նաև սովորել կուտակված փորձի հիման վրա: Արհեստական բանականության համակարգերը չեն փոխարինում մարդուն, դրանք ընդլայնում և լրացնում են նրա հնարավորությունները:

Արհեստական բանականության հիմնական ուղղություններից են նեյրոնային ցանցերը: Նեյրոնային ցանցը մաթեմատիկական մոդել է, որը իրականացնում է մարդու մտավոր գործունեության որոշ տարրեր:

Մեքենայական ուսուցումը (ML, Machine Learning), ըստ Արթուր Սամուելի (1959 թ.), ուսումնասիրության ոլորտ է, որը համակարգիչներին տալիս է սովորելու ունակություն՝ առանց հստակ ծրագրավորված լինելու: Ըստ Թոմ Միթչելի (1998 թ.)՝ համակարգչային ծրագիրը սովորում է E փորձառությունից T առաջադրանքի նկատմամբ P արդյունավետության ցուցանիշով, եթե T առաջադրանքի արդյունավետությունը P ցուցանիշով լավանում է E փորձառություններից:

Մեքենայական ուսուցումը հատուկ ալգորիթմների հավաքածու է, որի շնորհիվ տրված տվյալների հիման վրա մոդելը ձեռք է բերում ինքնուրույն սովորելու ունակություն: Ուսուցանող տվյալների քանակից կախված՝ ուսուցանող

ալգորիթմը տվյալների մեջ հայտնաբերում է օրինաչափություններ և կատարում ընդհանրացումներ:

Ուսումնական ձեռնարկում դիտարկվում են արհեստական բանականության հիմնական տարրերը, մասնավորապես՝

- արհեստական բանականության հիմնական հասկացություններն ու սահմանումները,
- արհեստական նեյրոնը որպես նեյրոնային ցանցի հիմք,
- ակտիվացման տարբեր ֆունկցիաների նշանակությունն ու տեսակները,
- նեյրոնային ցանցերի տարբեր կառուցվածքները,
- նեյրոնային ցանցերի ուսուցման հիմունքները,
- ուսուցանող տվյալների հավաքածուների տեսակներն ու նշանակությունը,
- նեյրոնային ցանցերի ուսուցման տեսակները (ուսուցչով կամ առանց ուսուցչի ուսուցում):

1.1 Արհեստական բանականության հիմնական հասկացությունները

Արհեստական բանականության հիմնական տեսակներն են.

- Սահմանափակ արհեստական բանականություն (ANI, Artificial Narrow Intelligence): Այն ծրագրային և ապարատային մասնագիտացված համալիր է բացառապես մեկ կոնկրետ ոլորտի համար: Օրինակ՝ համակարգչային ծրագիր, որը կարող է հաղթել շախմատի աշխարհի

չեմպիոնին, բայց դա այն ամենն է, ինչ նա կարող է անել:

- Ընդհանուր արհեստական բանականություն (AGI, Artificial General Intelligence): Այն ծրագրային և ապարատային համալիր է, որը նման է մարդու բանականությանը: Ընդհանուր արհեստական բանականությունը թույլ է տալիս որոշ չափով պատճենել մարդու մտածողության հնարավորությունները: Այն հնարավորություն է տալիս ստանալ նոր տվյալներ, դրանց հոսքից առանձնացնել անհրաժեշտ ինֆորմացիա, տվյալ խնդրի լուծման տարբերակները համեմատել իրար հետ, օգտագործել կուտակված փորձը որոշումներ կայացնելու համար և այլն:
- Արհեստական գերբանականություն (ASI, Artificial SuperIntelligence): Այն բանականություն է, որը գերազանցում է մարդուն գրեթե բոլոր բնագավառներում, այդ թվում՝ գիտական, ճանաչողական և հաղորդակցման հմտություններում:

Ներկայումս մարդկությունը արդյունավետորեն օգտագործում է արհեստական բանականության բաղադրիչները տարբեր ոլորտներում, ինչպիսիք են՝

- անօդաչու մեքենաները, որոնք կարող են ճանաչել ճանապարհին հանդիպող խոչընդոտները և արձագանքել դրանց,
- անօդաչու թռչող սարքերը, որոնք կարող են ինքնուրույն տեղաշարժվել տրված երթուղով,
- նավիգատորները, որոնք ձայնային հրամանների միջոցով երթուղով տեղաշարժվելու հրահանգներ են տալիս,

- էլեկտրոնային փոստում սպամ-ֆիլտրը, որը սկզբում սովորում է նույնականացնել սպամը, այնուհետև, հիմնվելով իր փորձի վրա, սպամ նամակները առանձնացնում և տեղափոխում է որոշակի թղթապանակ,
- թարգմանիչները, որոնք որակյալ թարգմանություններ են կատարում,
- տեքստի ճանաչման, ձայնի ճանաչման, տեքստից ձայնի փոխակերպման համակարգերը և այլն:

Արհեստական բանականությունը ինտելեկտուալ (խելացի) մեքենաներ ստեղծելու գիտություն և տեխնոլոգիա է: AI–ն իրականացվում է ծրագրային ապահովման միջոցով, որը կարող է աշխատել հզոր համակարգիչներում, սմարթֆոններում և հաշվողական այլ միջավայրերում: Բոլոր դեպքերում այն ծրագրաապարատային միավորում է: AI համակարգերը կատարում են ոչ միայն հաշվողական, այլ նաև ստեղծագործական որոշ գործառույթներ, որոնք հատուկ են մարդուն:

Մեքենայական ուսուցումը արհեստական բանականության ենթաբաժին է, որը ուսուցանող ալգորիթմների միջոցով կառուցում է տարբեր մոդելներ: Ուսուցանող ալգորիթմների շնորհիվ մուտքային տվյալներից (Data Set) մոդելը ձեռք է բերում ընդհանրացման հատկություն՝ պարբերաբար լավացնելով այն:

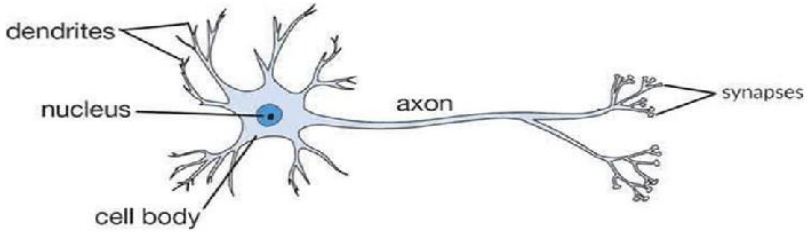
Մեքենայական ուսուցման առավել տարածված, հետաքրքիր և լայն կիրառության ոլորտ է արհեստական նեյրոնային ցանցերը (ANN, Artificial Neural Networks), որոնք մարդու ուղեղի կենսաբանական նեյրոնային ցանցի պարզեցված մոդել են:

1.2 Արհեստական նեյրոնը որպես նեյրոնային ցանցերի հիմք

Մարդու ուղեղը կենսաբանական-նեյրոնային բարդ ցանց է, որը ինֆորմացիան ստանում է զգայական օրգաններից (աչք, ականջ, քիթ) և ինչ-որ ձևով մշակում է այն (ճանաչում է դեմքը, հասկանում է խոսքը, զգում է հոտը և այլն)։ Ստացված ինֆորմացիայի հիման վրա ուղեղը տալիս է հրամաններ (սեղմել ծանոթ մարդու ձեռքը, ուսուցչի խնդրանքով գնալ գրատախտակի մոտ, տհաճ հոտի դեպքում տարածքից հեռանալ)։

Մարդու ուղեղում նեյրոնային ցանցը կազմված է մոտ 90 մլրդ նեյրոններից, որոնք մեկը մյուսին կապված են միլիարդավոր կապերով։ Սա մարդկությանը հայտնի ամենաբարդ օբյեկտն է։ Նեյրոնը այս ցանցի հիմքն է։

Կենսաբանական նեյրոնի պարզ կառուցվածքը ցույց է տրված նկ.1.1- ում։

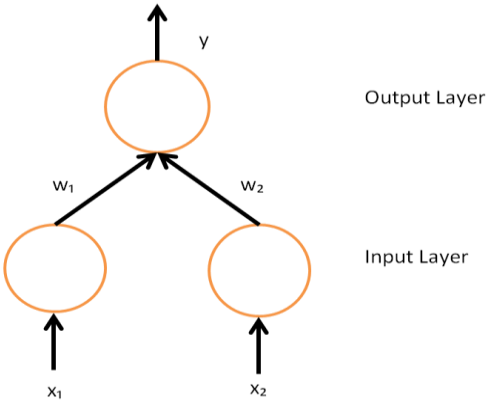


Նկ. 1.1 Կենսաբանական նեյրոնի կառուցվածքը

Կենսաբանական նեյրոնը նյարդային բջիջ է, որը բաղկացած է մարմնից (body), դենդրիտներից (dendrites) և աքսոնից (axon)։ Նեյրոնը մուտքային ազդանշաններ է ստանում բազմաթիվ դենդրիտների միջոցով։ Ազդանշանները մշակվում են նեյրոնի մարմնում։ Ձևավորված ելքային ազդանշանը աքսոնի

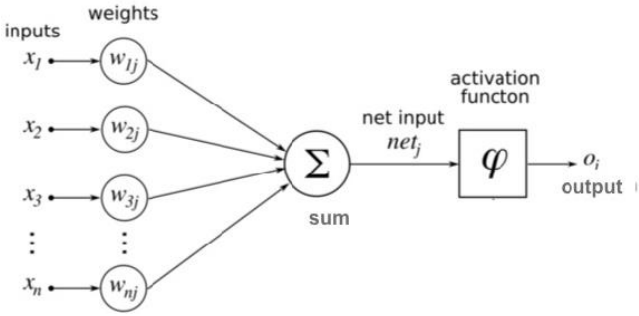
միջոցով փոխանցվում է հաջորդ նեյրոններին: Կենսաբանական նեյրոնը բավականին բարդ համակարգ է:

Արհեստական նեյրոն իրականացնելու համար հարկավոր է կառուցել նրա մաթեմատիկական մոդելը: Նկ. 1.2-ում ներկայացված է արհեստական նեյրոնի պարզեցված սխեմա, որն ունի x1, x2 երկու մուտք և մեկ y ելք: Նեյրոնը ներկայացվում է որպես «սև արկղ», որը ստանում է x1, x2 մուտքային տվյալներ, դրանց հետ կատարում է որոշ գործողություններ և ձևավորում y արդյունք:



Նկ. 1.2 Արհեստական նեյրոնի պարզ սխեմա

Արհեստական նեյրոնի առաջին մոդելը և նրա հիման վրա կառուցված առաջին նեյրոնային ցանցի մոդելը 1943 թվականին առաջարկել են ամերիկացի գիտնականներ նեյրոֆիզիոլոգ Ուորեն Մաք-Քալոկը և մաթեմատիկոս Ուոլթեր Փիթսը: Արհեստական նեյրոնի մաթեմատիկական մոդելը ներկայացված է նկ. 1.3-ում:



Նկ. 1.3 Արհեստական նեյրոնի մաթեմատիկական մոդել

Այս դեպքում նեյրոնը մուտքում ստանում է $x_1, x_2, \dots, x_n$ մուտքային պարամետրեր, որոնցից յուրաքանչյուրն ունի կշիռ՝ համապատասխանաբար $w_1, w_2, \dots, w_n$: Արհեստական նեյրոնում մուտքային ազդանշանը բազմապատկվում է իր կշռային գործակիցով, և ձևավորվում է S գումարը:

$$S = x_1 * w_1 + x_2 * w_2 + \cdots + x_n * w_n$$

S-ի արժեքը փոխանցվում է ակտիվացման ֆունկցիային: Եթե այդ արժեքը մեծ է որոշակի շեմից, ապա նեյրոնի ելքային ազդանշանը հավասար է մեկի: Այս դեպքում նեյրոնն ակտիվանում է, և համապատասխան ազդանշանը փոխանցվում է հաջորդ նեյրոնին: Եթե ակտիվացման ֆունկցիան S-ի վրա տալիս է շեմից ցածր արժեք, ապա նեյրոնի ելքային ազդանշանը հավասար է զրոյի, իսկ նեյրոնը համարվում է չակտիվացված:

Օրինակ 1.1-ում ներկայացված է արհեստական նեյրոնի S գումարիչի (summator) ծրագրային իրականացումը Python լեզվով: Ստեղծվում է Neuron դասը, որտեղ հաշվվում է մուտքային պարամետրերի և դրանց համապատասխան կշիռների արտադրյալների գումարը:

Օրինակ 1.1

```
# Մոդուլ Neuron
import numpy as np

# Նկարագրել Neuron դասը
class Neuron:
    def __init__(self, w):
        self.w = w

    def y(self, x):  # Գումարիչ
        s = np.dot(self.w, x)
        return s

X = np.array([2, 3])  # Մուտքային արժեքներ
W = np.array([1, 1])  # Մուտքային արժեքների կշիռներ
n = Neuron(W)  # Neuron դասի օբյեկտի ստեղծում
print('S1= ', n.y(X))  # Նեյրոնի ելքը X=[2,3] մուտքային արժեքների դեպքում
X = np.array([5, 6])  # Մուտքային արժեքներ
print('S2= ', n.y(X))  # Նեյրոնի ելքը X=[5,6] մուտքային արժեքների դեպքում
```

Այս ծրագրի գործարկման արդյունքում կունենանք հետևյալ պատասխանը.

```
S1=  5
S2=  11
```

Այժմ անցնենք արհեստական նեյրոնի հաջորդ բաղադրիչին՝ **ակտիվացման ֆունկցիային**: Այս բաղադրիչի աշխատանքի սկզբունքը հասկանալու համար դիտարկենք պարզ օրինակ: Ենթադրենք ունենք մեկ արհեստական նեյրոն, որի խնդիրն է որոշել՝ գնա՞լ ձկնորսության, թե՞ ոչ: Սա տիպիկ առաջադրանք է, որտեղ պետք է վերլուծել բազմաթիվ գործոնների համադրություն և ըստ այդ վերլուծության կայացնել վերջնական որոշում: Պարզության համար դիտարկենք ձկնորսության գնալու պայմանների չորս գործոն՝ **քամու արագություն** (ուժեղ քամի-0, չափավոր քամի-1), **մթնոլորտային ճնշում** (բարձր ճնշում-0, ցածր ճնշում-1), **արևի պայծառություն** (պայծառ-0, ամպամած-1), **ջրի ջերմաստիճանի կայունություն** (ոչ կայուն-0, կայուն-1): Նեյրոնի մուտքեր համարենք

- x_1 - քամու արագությունը,
- x_2 – մթնոլորտային ճնշումը,
- x_3 – արևի պայծառությունը,

- x_4 – ջրի ջերմաստիճանի կայունությունը:

Դիտարկենք այս գործոնների ազդեցությունը ձկնորսության գնալու համար: Բնական է, որ ձկնորսության կարելի է գնալ, եթե քամին չափավոր է, մթնոլորտային ճնշումը ցածր է, արևի պայծառությունը թույլ է, ջրի ջերմաստիճանը կայուն է:

Եթե նեյրոնն ունի չորս մուտք, ապա պետք է ունենա չորս կշռային գործակից: Մեր դեպքում կշռային գործակիցները պետք է ցույց տան յուրաքանչյուր մուտքի կարևորությունը նեյրոնի կողմից որոշում կայացնելու հարցում: Որքան ավելի մեծ է մուտքային պարամետրի կշիռը, այնքան մեծ է դրա կարևորությունը: Օրինակ, բաշխենք մուտքերի կշիռները հետևյալ կերպ:

- $w_1 = 5$
- $w_2 = 4$
- $w_3 = 1$
- $w_4 = 1$

Ելնելով տրված կշռման գործակիցներից՝ հեշտ է նկատել, որ ավելի կարևոր դեր են խաղում քամու արագության և մթնոլորտային ճնշման գործոնները: Մյուսներն ավելի քիչ ազդեցություն ունեն:

Նեյրոնի մուտքերին տանք հետևյալ ազդանշանները $x_1=1$ (քամին՝ չափավոր), $x_2=0$ (մթնոլորտային ճնշումը՝ բարձր), $x_3=0$ (արևի պայծառությունը՝ պայծառ), $x_4=1$ (ջրի ջերմաստիճանը՝ կայուն):

Այս տվյալների դեպքում S գումարը կունենա հետևյալ արժեքը.

$$S = x_1 * w_1 + x_2 * w_2 + \cdots + x_n * w_n$$
$$= 1 * 5 + 0 * 4 + 0 * 1 + 1 * 1 = 5 + 0 + 0 + 1 = 6$$

Օրինակ 1.1–ում x, w պարամետրերին տալով նոր արժեքներ՝

```
X= np.array( [1, 0, 0, 1])#Մուտքային արժեքներ
W= np.array( [5, 4, 1, 1 ])#Մուտքային արժեքների կշիռներ
```

կունենանք՝

```
S= 6
```

Այսպիսով, գումարիչը տալիս է S=6 արժեք։ Բայց ինչպե՞ս պետք է նեյրոնը որոշի՝ գնա՞լ ձկնորսության, թե՞ ոչ։ Ակնհայտ է, որ պետք է ինչ-որ ձևով S գումարը ազդեցություն ունենա վերջնական որոշման համար։ Այս խնդիրը լուծում է ակտիվացման ֆունկցիան։ Նեյրոնը պետք է ինչ-որ կերպ մշակի S-ի արժեքը և գեներացնի համապատասխան ելքային ազդանշան։ Այլ կերպ ասած՝ պետք է ըստ S-ի արժեքի ձևավորել նեյրոնի Y ելքային ազդանշանը և դրանով որոշել ձկնորսության գնալու հարցը։

Արհեստական նեյրոնները օգտագործում են ակտիվացման տարբեր ֆունկցիաներ։ Ակտիվացման ֆունկցիան (Activation Function) որպես մուտքային պարամետր ստանում է S գումարը, իսկ ելքում ձևավորվում է նեյրոնի ելքային Y արժեքը։ Ընդհանուր դեպքում այդ ֆունկցիան ունի հետևյալ տեսքը .

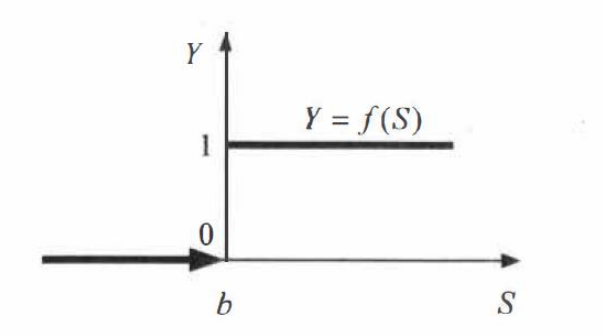
$$Y = f(S)$$

Դիտարկենք մի քանի ֆունկցիաներ, որոնք կարող են օգտագործվել որպես ակտիվացման ֆունկցիաներ։

1.3 Ակտիվացման ֆունկցիաներ

Onestep ակտիվացման ֆունկցիա (մեկ քայլով ակտիվացման ֆունկցիա)

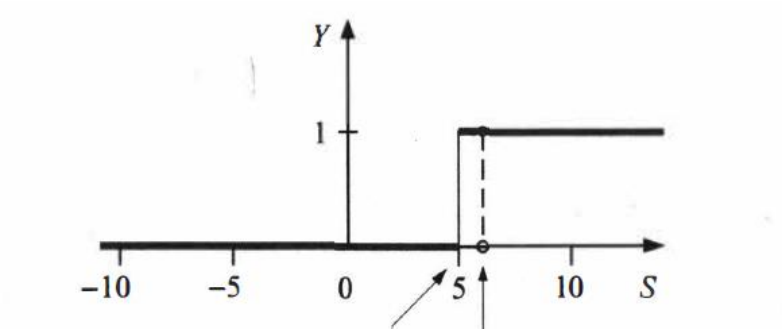
Onestep ֆունկցիայի ընդհանուր պատկերը հետևյալն է (նկ.1.4).



Նկ. 1.4 Onestep ֆունկցիայի տեսքը

Հորիզոնական առանցքի վրա S գումարի արժեքներն են, իսկ ուղղահայաց առանցքի վրա՝ Y ելքային ազդանշանի արժեքները: Ֆունկցիան ելքում կարող է ունենալ երկու արժեք՝ 0 կամ 1, որը կախված է b շեմային արժեքից: Եթե S-ը մեծ է b-ից, ապա նեյրոնի ելքը կստանա մեկ արժեք (Y=1), հակառակ դեպքում նեյրոնի ելքը կստանա զրո արժեք (Y=0):

b=5 շեմային արժեքի դեպքում, երբ S գումարի արժեքը 6 է, ակտիվացման ֆունկցիայի ելքում կստացվի Y=1 (նկ. 1.5): Այսպիսով, ելնելով տրված մուտքային ազդանշաններից, կշռային գործակիցներից և շեմային արժեքից կստացվի Y=1, այսինքն՝ ձկնորսը կարող է գնալ որսի:



Շեմը b = 5 S = 6

Նկ. 1.5 Onestep ակտիվացման ֆունկցիա

Onestep ակտիվացման ֆունկցիայի աշխատանքը իրականացնող ծրագրային կոդը բերված է օրինակ 1.2-ում:

Օրինակ 1.2

```
# Սողուլ onestep
import numpy as np

def onestep(x):
    b = 5
    if x >= b:
        return 1
    else:
        return 0


# Նկարագրել Neuron դասը
class Neuron:
    def __init__(self, w):
        self.w = w

    def y(self, x):  # Գումարիչ
        s = np.dot(self.w, x)
        return onestep(s)  # Ակտիվացման ֆունկցիա


X = np.array([1, 0, 0, 1])  # Մուտքային արժեքներ
W = np.array([5, 4, 1, 1])  # Մուտքային արժեքների կշիռներ
n = Neuron(W)  # Neuron դասի օբյեկտի ստեղծում
print('Y= ', n.y(X))
```

```
Y=  1
```

Եթե ծրագրում մուտքային տվյալները վերցնենք՝

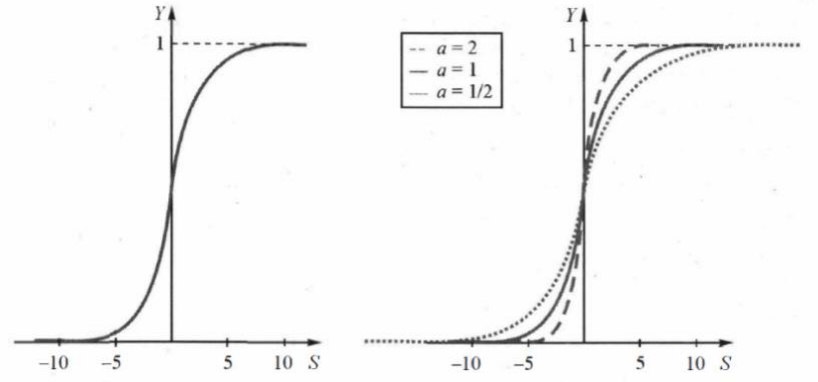
```
X = np.array([0, 0, 1, 1])  # Մուտքային արժեքներ
```

ապա կշռային գումարը՝ S=2, որը փոքր է b=5 շեմային արժեքից, հետևաբար Y=0, և ձկնորսության գնալը նպատակահարմար չէ։

Ակտիվացման սիգմոիդ ֆունկցիա

Ընդհանրապես կա սիգմոիդ ֆունկցիաների ընտանիք, որոնց մի մասը օգտագործվում է արհեստական նեյրոնների ակտիվացման համար։ Նեյրոնային ցանցերում առավել հաճախ կիրառվում է սիգմոիդ (լոգիստիկ) ֆունկցիան (նկ. 1.6)։ Այս ֆունկցիայի տեսքն է՝

$$Y = \frac{1}{1 + \exp(-aS)}:$$



Նկ. 1.6 Սիգմոիդ ֆունկցիան a պարամետրի տարբեր արժեքների համար

Վերադառնանք մեր արհեստական նեյրոնին, որը որոշում է ձկնորսության գնալու հաջող փորձը: Onestep ֆունկցիայի դեպքում պատասխանը 1 կամ 0 է (գնալ կամ չգնալ ձկնորսության): Սիգմոիդ ֆունկցիայի դեպքում նեյրոնի վերջնական արդյունքը ընկած է 0 և 1 թվերի միջև: Որքան մեծ է S գումարը, այնքան ավելի մոտ կլինի Y ելքը 1-ին, այսինքն՝ ձկնորսության գնալու հավանականությունը մեծ է: Հակառակ դեպքում գնալու հավանականությունը փոքր է: Վերջնական լուծումը որոշելու համար անհրաժեշտ է սահմանել շեմային արժեք: Օրինակ, եթե սահմանվել է շեմային արժեքը b=0.6, և Y≥0.6, կարելի է գնալ ձկնորսության, Y< 0.6 դեպքում՝ ոչ: Մեր դեպքում, եթե Y=0.8, ապա արժի գնալ ձկնորսության, իսկ Y= 0.2 դեպքում՝ ոչ:

Լոգիստիկ ֆունկցիան ունի հետևյալ հատկությունները.

- այն «սեղմող» ֆունկցիա է, այսինքն՝ անկախ S պարամետրից՝ ելքային Y ազդանշանը 0-ից 1 միջակայքում է,
- այն ավելի ճկուն է, քան onestep ֆունկցիան, նրա արդյունքը կարող է լինել ոչ միայն 0 կամ 1, այլև դրանց միջև եղած ցանկացած թիվ,
- բոլոր կետերում ֆունկցիան ունի ածանցյալ, որն արտահայտվում է նույն ֆունկցիայի միջոցով:

Լոգիստիկ ակտիվացման ֆունկցիայի աշխատանքը իրականացնող ծրագրային կոդը բերված է օրինակ 1.3-ում:

Օրինակ 1.3

```
# Մոդուլ sigmoid
import numpy as np
# Ակտիվացման ֆունկցիա: f(x) = 1 / (1 + exp(-x))
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Նկարագրել Neuron դասը
class Neuron:
    def __init__(self, w):
        self.w = w

    def y(self, x):  # Գումարիչ
        s = np.dot(self.w, x)
        return sigmoid(s)  # Ակտիվացման ֆունկցիա


X = np.array([0, 0, 1, 1])  # Մուտքային արժեքներ
W = np.array([5, 4, 1, 1])  # Մուտքային արժեքների կշիռներ
n = Neuron(W)  # Neuron դասի օբյեկտի ստեղծում
print('Y= ', n.y(X))
```

Ելքում ստանում ենք՝

```
Y=  0.8807970779778823
```

Քանի որ y> 0.6, հետևաբար կարելի է գնալ ձկնորսության։
Փոխենք մուտքային տվյալները՝

```
X = np.array([0, 0, 0, 0])  # Մուտքային արժեքներ
```

Կստանանք՝

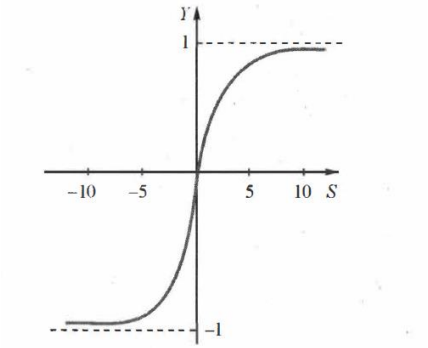
```
Y=  0.5
```

հետևաբար՝ y< 0.6 չի կարելի է գնալ ձկնորսության։

Ակտիվացման հիպերբոլիկ տանգենս ֆունկցիա

Դիտարկենք ևս մեկ սիգմոիդ ֆունկցիա՝ հիպերբոլիկ տանգենսը, որի արժեքները (-1,1) միջակայքից են (նկ. 1.7)։ Այս ֆունկցիան ավելի հաճախ է օգտագործվում կենսաբանների կողմից։ Ֆունկցիան տրվում է հետևյալ կերպ.

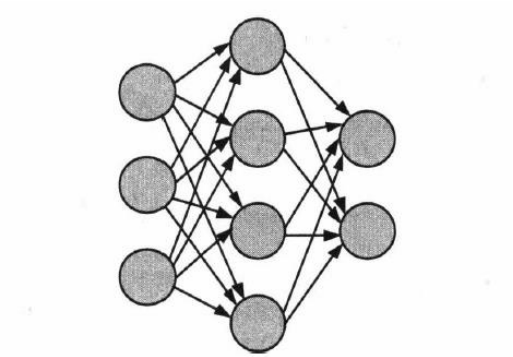
$$Y = th(S)$$



Նկ. 1.7 Հիպերբոլիկ տանգենս ֆունկցիայի գրաֆիկը

1.4 Նեյրոնային ցանցեր (միաշերտ և բազմաշերտ)

Եթե կենսաբանական նեյրոնային ցանցը կենսաբանական նեյրոնների հավաքածու է, ապա արհեստական նեյրոնային ցանցը իրար հետ փոխազդող արհեստական նեյրոնների հավաքածու է։ Արհեստական նեյրոնային ցանցի կառուցվածքը ներկայացված է նկ. 1.8-ում։



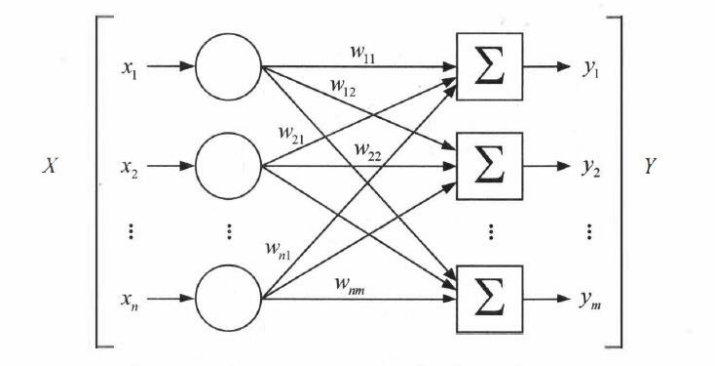
Նկ. 1.8 Արհեստական նեյրոնային ցանցի կառուցվածքը

Նեյրոնային ցանցերը և նեյրոնները գրաֆիկորեն ներկայացնելու մի քանի եղանակ կա: Այստեղ արհեստական նեյրոնները պատկերված են շրջանների տեսքով: Օգտագործվում են սլաքներ ազդանշանի շարժման ուղղությունը նշելու համար: Այսպիսով, արհեստական նեյրոնային ցանցը կարող է ներկայացվել շրջանների (արհեստական նեյրոնների) հավաքածուի տեսքով, որոնք իրար միացված են սլաքներով: Բոլոր նեյրոնային ցանցերն ունեն մուտքային շերտ, որը մուտքային ազդանշանները փոխանցում է մյուս նեյրոններին: Մուտքային շերտի նեյրոնները ոչ մի հաշվարկ չեն կատարում: Նեյրոնային ցանցերի կառուցվածքները կարող են տարբեր լինել:

Ինչպես գիտենք, նեյրոնների միջև յուրաքանչյուր կապ բնութագրվում է որոշակի թվով, որը կոչվում է կշիռ: Երբ մեկ նեյրոնի ելքային ազդանշանը փոխանցվում է մեկ այլ նեյրոնի որպես մուտք, ապա մուտքային արժեքը բազմապատկվում է այդ կապի կշռով: Ձկնորսության գնալու օրինակում այս ամենին արդեն առնչվել ենք: Դիտարկենք նեյրոնային ցանցերի տեսակներ:

Միաշերտ նեյրոնային ցանցեր

Միաշերտ նեյրոնային ցանցի մի տարբերակ բերված է նկ. 1.9-ում:

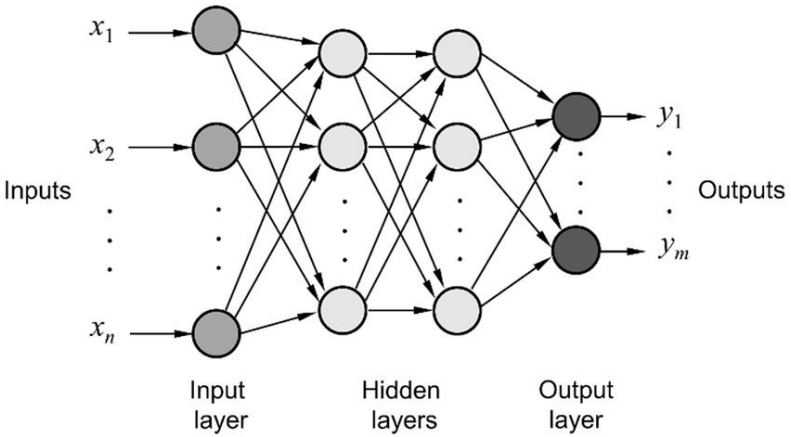


Նկ. 1.9 Միաշերտ նեյրոնային ցանցի կառուցվածքը

Մուտքային շերտը նշված է շրջաններով, իսկ աջ կողմում ելքային շերտն է: Նեյրոնները իրար հետ կապված են համապատասխան կշիռներով (կշռային գործակիցներ):

Բազմաշերտ նեյրոնային ցանցեր

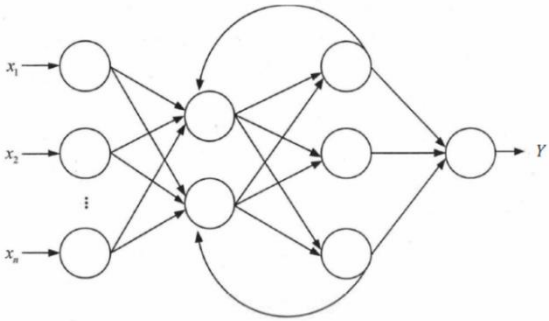
Բազմաշերտ նեյրոնային ցանցը բաղկացած է մուտքային շերտից, մի քանի թաքնված շերտերից և ելքային շերտից (նկ. 1.10): Որոշակի աղբյուրներից (օրինակ՝ ռադարներից, անօդաչու մեքենայի տեսախցիկներից և այլն) մուտքային շերտում ձևավորվում և ցանցին են փոխանցվում ազդանշաններ: Նեյրոնային ցանցի մուտքային շերտը փոխանցում է մուտքային ինֆորմացիա: Ելքային շերտը վերջնական որոշումները տալիս է ղեկավարող հրամանների տեսքով (ինքնակառավարվող մեքենայի համար ղեկի կառավարումը, ոտնակի արագացուցիչի կառավարումը, արգելակները և այլն):



Նկ. 1.10 Բազմաշերտ նեյրոնային ցանցի կառուցվածքը

Նեյրոնային ցանցի կառուցվածքում բոլոր կապերն ունեն մեկ ուղղություն՝ ձախից աջ: Նման ցանցերում ազդանշանը մուտքային շերտից գալիս է դեպի ելք: Այդպիսի ցանցերը կոչվում են առաջընթաց նեյրոնային ցանցեր (Feedforward Neural Network): Այս ցանցերը բավականին տարածված են: Նրանք հաջողությամբ լուծում են կանխատեսման, դասակարգման և օբյեկտի ճանաչման խնդիրներ:

Սակայն ոչինչ չի խանգարում ազդանշաններն ուղարկել հակառակ ուղղությամբ, այսինքն՝ ելքային շերտից նեյրոնի ազդանշանը փոխանցվի նախորդ շերտի նեյրոնին որպես մուտք: Հետևաբար, ազդանշանը կարող է նաև փոխանցվել հակառակ ուղղությամբ: Այդպիսի ցանցերը կոչվում են հետադարձ կապով ցանցեր (RNN, Recurrent Neural Network):



Նկ. 1.11 Հետադարձ կապով բազմաշերտ նեյրոնային ցանցի կառուցվածքը

Հետադարձ կապով նեյրոնային ցանցերում նեյրոնի ելքերը կարող են վերադառնալ դեպի մուտք: Սա նշանակում է, որ նեյրոնի ելքը որոշվում է ոչ միայն իր կշռով և մուտքային ազդանշանով, այլ նաև հետագա շերտերի որոշ նեյրոնների ելքերով: Ցանցում ազդանշանների շրջանառության հնարավորությունը բացում է նեյրոնային ցանցերի նոր հեռանկարներ: Նման ցանցերի օգնությամբ հնարավոր է ստեղծել նեյրոնային ցանցեր, որոնք վերականգնում կամ լրացնում են ազդանշանները: Այլ կերպ ասած, նման նեյրոնային ցանցերը կարող են ունենալ կարճաժամկետ հիշողության ունակություն:

1.5 Կորստի ֆունկցիա (Loss Function)

Նախքան նեյրոնային ցանցերի ուսուցման մասին խոսելը սահմանենք կորստի ֆունկցիա հասկացությունը: Կորուստի ֆունկցիան ցույց է տալիս, թե որոշակի առաջադրանքի կատարման համար որքան արդյունավետ է աշխատում նեյրոնային ցանցը: Կորուստի ֆունկցիան կարող է որոշվել հետևյալ ձևով՝

$$L(y,\hat{y}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y_i})^2,$$

որտեղ m-ը ուսուցման օրինակների քանակն է, y_i-ն՝ իրական ելքային արժեքը ուսուցման i-րդ օրինակի դեպքում, $\hat{y}i$-ն՝ ցանցի կանխատեսած արժեքը: Եթե կորստի ֆունկցիան մեծ է, ապա մեր ցանցն այնքան էլ լավ չի աշխատում, ուրեմն այն պետք է հնարավորինս փոքրացնել:

1.6 Նեյրոնային ցանցի ուսուցում և թեստավորում

Նեյրոնային ցանցի ուսուցում

Դիտարկենք նեյրոնային ցանց՝ կազմված որոշ քանակի նեյրոններից, կապերից, թաքնված շերտերից: Նեյրոնային ցանցի ուսուցումը կշռային գործակիցների այնպիսի հավաքածուի որոնումն է, որի դեպքում մուտքային ազդանշանը ցանցով անցնելիս ձևավորում է անհրաժեշտ ելքը:

Սկզբից կշիռներին տրվում են պատահական արժեքներ: Ակնհայտ է, որ դա լավ արդյունք չի տա: Ուսուցման գործընթացը սովորաբար սկսվում է վատ աշխատող նեյրոնային ցանցից և ավարտվում ավելի բարձր ճշգրտությամբ նեյրոնային ցանցով: Ինչ վերաբերում է կորստի ֆունկցիային, ապա այն պետք է շատ ավելի փոքր լինի ուսուցման վերջում: Ցանցի բարելավումը հնարավոր է կշռային գործակիցների վերանայմամբ:

Ուսուցման խնդիրը համարժեք է կորստի ֆունկցիան նվազագույնի հասցնելու խնդրին:

Նեյրոնային ցանցի թեստավորում

Ուսուցման հավաքածուն (Training Set) մուտքային ազդանշանների հավաքածու է, որտեղ ներառված են ճիշտ ելքային ազդանշանները (պիտակները): Ուսուցման հավաքածուի վրա ուսուցանվում է նեյրոնային ցանցը: Ցանցը ուսուցումից հետո, այսինքն՝ երբ ցանցը ճիշտ արդյունքներ է տալիս ուսուցման հավաքածուի բոլոր մուտքային ազդանշանների համար, կարելի է համարել, որ այն անցել է ուսուցման փուլը: Այս գործընթացը կարելի է համեմատել սովորողների ուսուցման գործընթացի հետ: Հաջորդ փուլում պետք է ստուգել ցանցի աշխատանքը թեստային մուտքային ազդանշանների համար: Նեյրոնային ցանցի աշխատանքի որակը ստուգելու համար կատարվում է փորձարկում թեստային հավաքածուի վրա:

Թեստային հավաքածուն (Testing Set) մուտքային ազդանշանների հավաքածու է ճիշտ ելքային ազդանշանների հետ միասին, որոնց միջոցով գնահատվում է ուսուցումից հետո ցանցի աշխատանքի որակը: Այդ փուլում ստուգվում է նեյրոնային ցանցի աշխատանքը, այն է՝ ելքային արժեքները արդյոք ունեն ընդունելի շեղումներ իրական ելքային արժեքների համեմատ: Այսինքն՝ կատարվում է ցանցի աշխատանքի կայունության ստուգում:

Այսպիսով, նեյրոնային ցանցի ուսուցումը տեղի է ունենում երկու փուլով՝ ուսուցում՝ օգտագործելով իդեալական տվյալներով օրինակներ, և թեստավորում՝ օգտագործելով իրական տվյալներով օրինակներ:

Կան նեյրոնային ցանցերի ուսուցման երկու մոտեցում՝ ուսուցչով և առանց ուսուցչի ուսուցում:

1.7 Ուսուցչով և առանց ուսուցչի ուսուցում

Ուսուցչով ուսուցում (Supervised Learning)

Ուսուցչով ուսուցումը նեյրոնային ցանցի ուսուցման տեսակ է, որտեղ նեյրոնների կապերի կշիռներն ընտրվում են այնպես, որ ցանցի ելքային պատասխանները ճիշտ պատասխաններից նվազագույն չափով տարբերվեն: Այս մոտեցման էությունն այն է, որ ուսուցման հավաքածուի ազդանշաններից ստացվում է ցանցի ելքային արժեքը, և դա համեմատվում հայտնի ճիշտ պատասխանի հետ: Այնուհետև որոշ ալգորիթմների միջոցով փոխվում են նեյրոնային ցանցի կապերի կշիռները, և նեյրոնային ցանցը գործարկվում է նույն մուտքային ազդանշանների համար: Կրկնում ենք այս գործընթացը այնքան ժամանակ, մինչև ցանցի տված արդյունքները լինեն ընդունելի ճշտությամբ:

Օրինակ, եթե ցանկանում ենք, որ ցանցը ճանաչի լուսանկարներում եղած դեմքերը, կարող ենք ստեղծել լուսանկարներից բաղկացած ուսուցման հավաքածու, ուսուցանելով դրանք՝ որոշել լուսանկարներում եղած դեմքերը:

Եթե ցանկանում ենք, որ ցանցը կանխատեսի եղանակը, ապա մուտքային ազդանշանները կարող են ներառել պարամետրեր, ինչպիսիք են՝ օդի ջերմաստիճան, մթնոլորտային ճնշում, քամու ուժ, քամու ուղղություն, խոնավություն և այլն: Այս դեպքում ուսուցման հավաքածուն (մուտքային տվյալները) պետք է ձևավորվի իրական տարբեր տվյալների հիման վրա:

Հարկ է նշել, որ նեյրոնային ցանցը երբեմն պետք է ժամերով ուսուցանել՝ կատարելով տասնյակ կամ հարյուրհազարավոր ուսուցման փուլեր:

Առանց ուսուցչի ուսուցում (Unsupervised Learning)

Առանց ուսուցչի ուսուցումը նեյրոնային ցանցի ուսուցման տեսակ է, երբ ցանցն ինքնուրույն դասակարգում է մուտքային ազդանշանները: Ճիշտ ելքային ազդանշանները նրան չեն ցուցադրվում: Առանց ուսուցչով ուսուցումն օգտագործվում է, երբ չենք կարող ապահովել ցանցի մուտքային ազդանշանների ճիշտ ելքերը: Այս դեպքում ուսուցման հավաքածուն բաղկացած է մուտքային ազդանշանների հավաքածուից, և ցանցն ինքն է դասակարգում ստացված մուտքային ազդանշանները: Նա, ըստ էության, ինքնուրույն է սովորում:

Դիտարկենք, թե ինչ դասի խնդիրներ կարող է լուծել ցանցը ինքնուսուցման գործընթացում: Օրինակ՝ ենթադրենք՝ նեյրոնային ցանցին պետք է վարժեցնել, որ տարբերի որոշ առարկաներ լուսանկարներում: Դա անելու համար ցանցի մուտքին տալիս ենք մեքենաների, ճանապարհային նշաններ, հետիոտների բազմաթիվ լուսանկարներ: Ինքնուսուցման փուլերից հետո ցանցը կգտնի յուրաքանչյուր օբյեկտի տարբերակիչ գծերը, կսովորի տարբերել դրանք և կսկսի տալ երեք տարբեր տեսակի ազդանշաններ, որոնք կհամապատասխանեն մուտքում տրված յուրաքանչյուր օբյեկտին: Այս տիպի ուսուցումով ցանցը մուտքային ազդանշանները տարբեր խմբերի բաժանելու ընդունակություն է ձեռք բերում: Այս գործընթացը կոչվում է խմբավորում (clustering):

ԳԼՈՒԽ 2
Նեյրոնային ցանցի ծրագրային իրականացում

Նախորդ գլխում տրվեցին տեղեկություններ արհեստական նեյրոնի և նեյրոնային ցանցերի կառուցվածքի, դրանց ուսուցման ընդհանուր մոտեցումների վերաբերյալ: Այս գլխում նկարագրվում են արհեստական նեյրոնային ամենապարզ ցանցի՝ պերցեպտրոնի աշխատանքի սկզբունքները: Կներկայացվեն հետևյալ թեմաները.

- պերցեպտրոններ և դրանց դասակարգումը,
- նեյրոնային ցանցերում պերցեպտրոնների դերը,
- օբյեկտների գծային բաժանելիություն,
- օբյեկտների դասակարգման խնդիրների լուծման եղանակները,
- ինչպես սովորեցնել պերցեպտրոնին ճանաչել պատկերները,
- ինչպես սովորեցնել պերցեպտրոնին ընտրել կապերի կշիռները,
- դելտա-կանոնը և պերցեպտրոնի ուսուցման մեջ դրա օգտագործումը,
- գծային մոտարկում և դրա օգտագործումը օբյեկտների դասակարգման համար:

2.1 Պերցեպտրոններ (Perceptron)

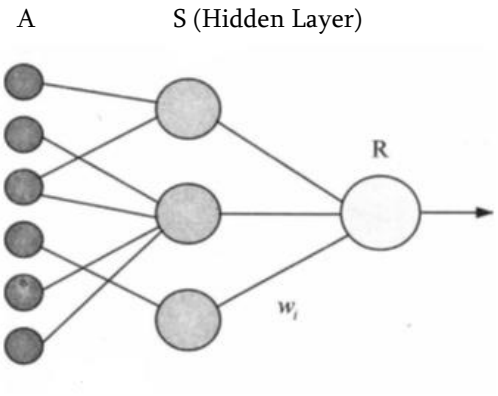
Արհեստական նեյրոն և արհեստական նեյրոնային ցանց հասկացությունները ի հայտ են եկել 1943 թ.: Մաք Քալոկը փորձեր է արել մոդելավորել ուղեղի աշխատանքը: Նրա գաղափարները զարգացեց նեյրոֆիզիոլոգ Ֆրենկ Ռոզենբլատը:

Նա առաջարկեց մարդու կողմից տեղեկատվության ընկալման գործընթացը մոդելավորող սարքավորման սխեմա, որը անվանեց պերցեպտրոն (լատիներեն perceptio-ընկալում): 1960 թվականին Ֆ. Ռոզենբլատը ներկայացրեց «Mark-1» անունով առաջին նեյրոհամակարգիչը, որն ունակ էր ճանաչելու անգլերեն այբուբենի որոշ տառեր: Պերցեպտրոնը նեյրոնային ցանցերի առաջին մոդելն է, իսկ Mark-1-ը՝ աշխարհի առաջին նեյրոհամակարգիչը:

Պերցեպտրոնները սկսեցին շատ ակտիվորեն հետազոտվել, քանի որ դրանց հետ մեծ հույսեր էին կապում: Սակայն, ինչպես պարզվեց, դրանք լուրջ սահմանափակումներ ունեին: Դրանց թերությունների մասին 1971 թվականին ամերիկացի գիտնական Մարվին Մինսկին գիրք է գրել: Այդ ժամանակից ի վեր գիտնականների խանդավառությունը պերցեպտրոնների և արհեստական նեյրոնային ցանցերի ուսումնասիրության ոլորտում թուլացել էր: Որոշ ժամանակ անց իրականացվեցին նեյրոնային ցանցերի ուսուցման ալգորիթմներ, ինչը նորից հետաքրքրություն առաջացրեց այդ ոլորտում: Հետագայում Մինսկին խոստովանեց, որ իր գիրքը լուրջ հարված հասցրեց պերցեպտրոնի և նեյրոնային ցանցերի կիրառման ոլորտի զարգացմանը: Նա հետագայում լրջորեն զբաղվել է այդ ուղղության զարգացմամբ և դարձել Մասաչուսեթսի տեխնոլոգիական ինստիտուտի արհեստական բանականության լաբորատորիայի հիմնադիրներից մեկը:

Նկարագրենք պերցեպտրոնի կառուցվածքը:

Պերցեպտրոնային մոդելում առանձնացվում են երեք տիպի տարրեր՝ սենսորային (S-տարրեր), ասոցիատիվ (A-տարրեր), արձագանքող (R-տարրեր) (նկ. 2.1):



Նկ. 2.1 Պերցեպտրոնի կառուցվածքը

S տարրերը կազմում են սենսորային շերտը (ընկալիչներ, ռեցեպտորներ) և առաջինն են ակտիվանում: Յուրաքանչյուր S տարր կարող է գտնվել երկու վիճակներից մեկում՝ արգելակված (0) կամ գրգռված (1), և միայն վերջին դեպքում է այն փոխանցում 1 ազդանշանը հաջորդ՝ ասոցիատիվ A շերտին: Սենսորային շերտը ուղարկում է ազդակներ, որոնք փոխանցվում են հաջորդ շերտեր:

Հաջորդ շերտի A տարրերը կոչվում են ասոցիատիվ տարրեր: Յուրաքանչյուր այդպիսի տարրին սովորաբար համապատասխանում է S տարրերի մի ամբողջ խումբ: A տարրը ակտիվանում է, երբ իր մուտքի S ազդանշանների թիվը գերազանցում է որոշակի շեմային արժեքը: Օրինակ, եթե S շերտում տրվում է որևէ պատկեր, ապա A տարրը ակտիվանում է (ունենում է 1 արժեք) S շերտում պատկերի հետ կապված որոշակի քանակությամբ սենսորների միջոցով: A տարրը չի ակտիվանում (ունենում է 0 արժեք) S շերտում պատկերով չզբաղեցված տարածքի սենսորներից:

Գրգռված A տարրերից ստացված ազդանշանները հավաքվում են R տարրում, որտեղ որոշակի գործողություններով ձևավորվում է վերջնական արդյունքը: S տարրերից ազդանշանների փոխանցումը A տարրերին կատարվում է S-A կապերով, որոնք ունեն կշիռներ: S-A կապերի կշիռները կարող են ունենալ -1, + 1 կամ 0 արժեքներ: Նշենք, որ մեկ A տարրը կարող է կապված լինել բազմաթիվ S տարրերի հետ: Եթե A տարրին տրված արժեքը գերազանցում է որոշակի շեմը, ապա A տարրը գրգռվում է և ունենում 1 արժեք: Հակառակ դեպքում այն ունենում է 0 արժեք:

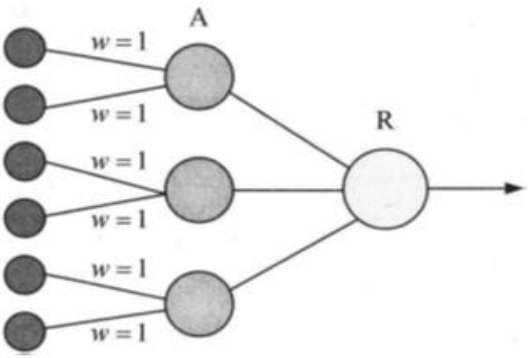
Հաջորդ քայլում գրգռված A տարրերից ազդանշանները որոշակի w կշռային գործակցով փոխանցվում են գումարիչ (R տարր): Այդ գործակիցները կոչվում են A-R կապերի կշիռներ և կարող են ունենալ ցանկացած արժեքներ: Այնուհետև R տարրը հաշվարկում է մուտքային արժեքների գումարը՝ ազդանշանները բազմապատկելով իրենց կշռային w գործակիցներով: Ստացված գումարը համեմատվում է b որոշակի շեմի հետ և փոխակերպվում հատուկ ֆունկցիայի միջոցով ելքային ազդանշանի, որն ունի ընդամենը երկու արժեք (-1 կամ +1): Մուտքային պատկերը ճանաչվում է, երբ ելքային ազդանշանը +1 է, երբ այն -1 է, պատկերը չի ճանաչվել:

Այսպիսով, պերցեպտրոնը նեյրոնային ցանցերի ամենապարզ մոդելն է: Պերցեպտրոնի հիմքում ընկած է ուղեղի կողմից տեղեկատվության ընկալման մաթեմատիկական մոդելը, որը բաղկացած է սենսորներից, ասոցիատիվ և արձագանքող տարրերից: Գոյություն ունեն պերցեպտրոնների մի քանի տեսակներ: Դիտարկենք դրանցից մի քանիսը:

2.2 Պերցեպտրոնների դասակարգում

Տարբեր հեղինակներ տարբեր կերպ են դասակարգել պերցեպտրոնները: Առավել հաճախ օգտագործվող ճարտարապետությամբ պերցեպտրոնները լինում են միաշերտ և բազմաշերտ: Դրանց հիմնական տարբերությունը S տարրերի և A տարրերի կապերի միացման, միացումների կշիռների տրման և A տարրերի շեմի որոշման ձևեր են:

Միաշերտ պերցեպտրոնում բոլոր S-A կապերը միշտ ունեն մեկին հավասար կշիռ (w=l), իսկ A տարրերի շեմը միշտ +1 է: Այս դեպքում սենսորները կարող են ուղարկել ազդանշան, որը հավասար է միայն 0-ի կամ 1-ի (նկ. 2.2):



Նկ. 2.2 Միաշերտ պերցեպտրոն

Նկարագրենք R տարրի ելքում 1-ի հավասար ազդանշանի գեներացումը: Սենսորային ազդանշանն անցնում է S-A կապով և չի փոխվում, քանի որ ցանկացած թիվ բազմապատկվում է 1-ով: Յուրաքանչյուր A տարրի շեմը 1 է: Եթե սենսորային ազդանշանը հավասար է 1-ի, ապա, ինչպես գիտենք, A տարրը գրգռվում է: Սա նշանակում է, որ նա ելքում տալիս է 1

ազդանշան (քանի որ այն կարող է միայն 1 կամ 0 գեներացնել իր ելքում):

Այնուհետև այդ արժեքը բազմապատկվում է A-R կապի կշռով և տրվում է R տարրին, որտեղ հաշվարկվում է կշռված բոլոր ազդանշանների գումարը: Եթե կշռված գումարը R տարրի որոշակի շեմից մեծ է, ապա R տարրի ելքը 1 է, հակառակ դեպքում՝ -1:

Նշենք, որ դա կատարվում է Onestep ակտիվացման ֆունկցիայով, որը քննարկվել է 1-ին գլխում: Տարբերությունն այն է, որ Onestep ֆունկցիան տալիս է 0, եթե շեմը չի գերազանցվում, այստեղ այն տալիս է -1, բայց դա էական չէ:

Հիմա նկարագրենք պերցեպտրոնի ծրագրային իրականացումը Python-ով: Այս ծրագրում ակտիվացման ֆունկցիայի շեմը վերցված է 7, սենսորների քանակը շերտում վերցված է 15, ելքային շերտի 1 արժեքին համապատասխանեցված է True, իսկ -1 արժեքին՝ False:

```
# Արհեստական նեյրոն (պերցեպտրոն)
weights=[1 for i in range(15)]
# Արհեստական նեյրոն (պերցեպտրոն)
def perceptron(Sensor):
  b = 7 # Ակտիվացման ֆունկցիայի շեմ
  s = 0 # Գումարի սկզբնարժեքավորում
  for i in range(15): # Սենսորների ազդանշանների գումարման ցիկլ
    s += int(Sensor[i]) * weights[i]
  if  s >= b:
    return True # Գումարը գերազանցում է շեմը
  else:
    return False # Գումարը փոքր է շեմից
```

Ծրագրի աշխատանքը ստուգենք 15 սենսորային մուտքային ազդանշանների (զրոներ և մեկեր) զանգվածի համար: Չնավորենք հետևյալ երկու մուտքային զանգվածները:

```
num1 = list('001001001001001')
num2 = list('111001111100111')
```

Բոլոր կապերի կշիռներին տանք 1 արժեքներ, քանի որ դիտարկում ենք պերցեպտրոնային պարզ մոդելը (բոլոր կապերի կշիռները նույնն են): Դա կատարվում է մեկ ցիկլի օպերատորի միջոցով:

```
weights = [1 for i in range(15)] # Բոլոր կշռային գործակիցներին վերագրել 1
```

Փորձարկենք ծրագիրը և ստանանք արհեստական նեյրոնի արդյունքը սենսորային արժեքների երկու՝ num1, num2 զանգվածների համար:

Ծրագրի կոդում ավելացվել են հրամաններ, որոնք ստուգում են պերցեպտրոնի աշխատանքը:

Օրինակ 2.1

```
# Արհեստական նեյրոն (պերցեպտրոն)
def perceptron(Sensor):
  b = 7 # Ակտիվացման ֆունկցիայի շեմ
  s = 0 # Գումարի սկզբնարժեքավորում
  for i in range(15): # Սենսորների ազդանշանների գումարման ցիկլ
    s += int(Sensor[i]) * weights[i]
  if  s >= b:
    return True # Գումարը գերազանցում է շեմը
  else:
    return False # Գումարը փոքր է շեմից

# Պերցեպտրոնի աշխատանքի ստուգում
num1 = list('001001001001001')
num2 = list('111001111100111')
weights = [1 for i in range(15)] # Բոլոր կշռային գործակիցներին վերագրել 1
print(num1) # Պերցեպտրոնի ստացած ազդանշանները num1 սենսորներից
print(perceptron(num1)) # Պերցեպտրոնի ելք
print(num2) # Պերցեպտրոնի ստացած ազդանշանները num2 սենսորներից
print(perceptron(num2)) # Պերցեպտրոնի ելք
```

Ծրագրի կատարման արդյունքում կարտածվի

```
['0', '0', '1', '0', '0', '1', '0', '0', '1', '0', '0', '1', '0', '0', '1']
False
['1', '1', '1', '0', '0', '1', '1', '1', '1', '1', '0', '0', '1', '1', '1']
True
```

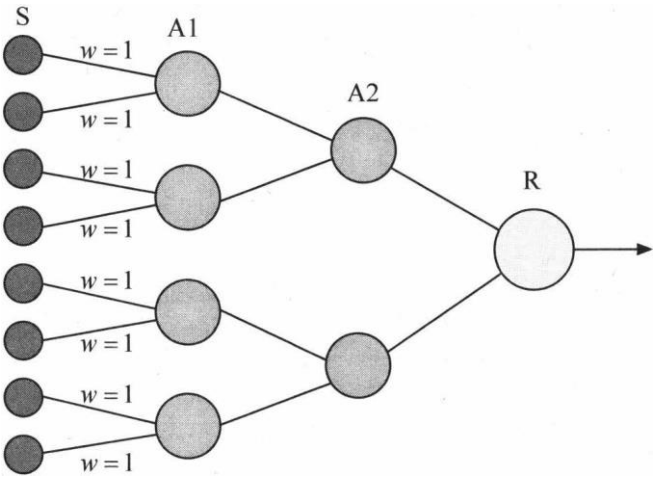
Առաջին դեպքում բոլոր սենսորներից ստացված ազդանշանների գումարը հավասար է 5-ի, որը փոքր է 7 արժեքով շեմից: Այս դեպքում ելքում ստացվել է False, քանի որ շեմը չի գերազանցվել: Երկրորդ դեպքում բոլոր սենսորներից ստացված ազդանշանների գումարը 11 է, որը մեծ է 7 արժեքով շեմից: Այս դեպքում ելքում ստացվել է True, քանի որ շեմը գերազանցվել է:

Այսպիսով մշակվեց ծրագիր, որը իմիտացնում է ուղեղի բջիջների ամենապարզ մոդելի աշխատանքը:

2.3 Բազմաշերտ պերցեպտրոններ

Դիտարկենք բազմաշերտ պերցեպտրոնների տեսակներ: Բազմաշերտ պերցեպտրոնները հիմնականում երկու տեսակի են՝ բազմաշերտ պերցեպտրոն ըստ Ռոզենբլատի և բազմաշերտ պերցեպտրոն ըստ Ռումելհարտի:

Ռոզենբլատի բազմաշերտ պերցեպտրոնն ունի A տարրերի մեկից ավել շերտեր (նկ. 2.3):



Նկ. 2.3 Ռոզենբլատի բազմաշերտ պերցեպտրոն

Բազմաշերտ պերցեպտրոնը, ըստ Ռումելհարթի, բազմաշերտ պերցեպտրոն է, որտեղ ուսուցման ենթակա են նաև S-A կապերը, և ուսուցումն իրականացվում է ըստ սխալի ետ տարածման մեթոդի (Backpropagation): Այսինքն՝ բազմաշերտ պերցեպտրոնը ըստ Ռումելհարթի բազմաշերտ պերցեպտրոնի հատուկ տեսակ է, որը Ռոզենբլատի բազմաշերտ պերցեպտրոնից տարբերվում է հետևյալ հատկանիշներով.

- S-A կապերը կարող են ունենալ կամայական կշիռներ և ուսուցանվում են նույն հիմունքներով, ինչ A-R կապերը,
- ուսուցումն իրականացվում է հատուկ ալգորիթմի միջոցով, որը կոչվում է ուսուցում օգտագործելով սխալի ետ տարածման մեթոդը:

Այս մեթոդը հաճախ կիրառվում է բազմաշերտ նեյրոնային ցանցերի ուսուցման համար, որի շնորհիվ նեյրոնային ցանցերի կիրառման ոլորտը զգալիորեն ընդլայնվել է:

2.4 Պերցեպտրոնների դերը նեյրոնային ցանցերում

Ի՞նչ խնդիրներ է լուծում պերցեպտրոնը: Նախքան նեյրոնային ցանցի տարրերի ծրագրային կոդի իրականացումը պետք է հստակեցնել, թե որ խնդիրները կարող է լուծել պերցեպտրոնը և որոնք՝ ոչ: Կդիտարկենք պարզ գործնական խնդիրներ՝ հասկանալու համար, թե արդյոք հնարավոր է օգտագործել պերցեպտրոն դրանց լուծման համար: Սա չափազանց կարևոր է, քանի որ պերցեպտրոնը նեյրոնային ցանցի ամենապարզ տեսակն է և շատ բան չի կարող անել:

Պերցեպտրոնները լավ են լուծում դասակարգման խնդիրները (օբյեկտների խմբերի տարանջատումը): Ի՞նչ է սա նշանակում: Եթե ունեք օբյեկտների խմբեր (օրինակ՝ կատուներ և շներ, լուսացույցներ և ճանապարհային նշաններ), ապա պերցեպտրոնը ուսուցանվելուց հետո կկարողանա որոշել, թե որ խմբին է պատկանում իրեն տրված օբյեկտը: Այսինքն՝ եթե պերցեպտրոնի մուտքին տանք շան նկար (անկախ նրանից, թե ինչ ցեղատեսակի, ինչ դիրքում է այն նկարվել), ելքում կստանանք պատասխան, որ դա շուն է: Եթե անօդաչու մեքենայի տեսախցիկի պատկերում հայտնվի ճանապարհային նշան, այն կտարբերակվի այլ օբյեկտներից: Ավելին, կարելի է որոշել, թե ինչ նշան է դա, և ինչ է պետք անել այդ դեպքում: Ռոզենբլատն ապացուցել է մի շարք թեորեմներ, որոնց էությունը փորձենք հասկանալ հնարավոր ամենապարզ օրինակների վրա:

Ռոզենբլատի առաջին թեորեմն ապացուցում է տարրական պերցեպտրոնի գոյությունը, որը ի վիճակի է կատարել պատկերների տրված հավաքածուի դասակարգում, այսինքն՝ ցույց է տալիս, որ պերցեպտրոնը պատկերների դասակարգ-

ման խնդիրների լուծման համար է: Այս արդյունքի էությունը մեկնաբանենք օրինակով:

Թվային ֆոտոխցիկի մատրիցը բաղկացած է բազմաթիվ սենսորներից, որոնք վերցնում են լուսային ազդանշանը: Եթե կան սենսորների դաշտ (մատրիցա) և որևէ դասակարգում՝ կախված այն բանից, թե ինչ ազդանշան է ստացվել յուրաքանչյուր մատրիցային տարրից, ապա դասակարգման միջոցով կարելի է որոշել, թե որ օբյեկտը հայտնվեց թվային տեսախցիկի տեսադաշտում:

Եկեք ավելի մանրամասն դիտարկենք ասվածը, որի համար տանք երկու հասկացություն՝ սենսորային դաշտ և օբյեկտների դասակարգում: Սենսորների դաշտը կհասկանանք որպես S տարրերի բազմություն: Օբյեկտները դասակարգվում են ըստ մեր կողմից տրվող դասերի (օրինակ՝ կատուներ կամ շներ):

Մեկ դասի բոլոր օբյեկտները ունեն որոշ ընդհանուր հատկանիշներ, որոնց ամբողջությունը համապատասխանում է S տարրերի որոշակի համադրությանը:

Իսկ ինչպե՞ս են մարդիկ տարբերում տարբեր առարկաներ: Նրանք դա սովորում են մանկուց: Եկեք դիտարկենք, թե ինչպես ծնողները կսովորեցնեն փոքրիկ երեխային տարբերել կատվին շնից: Նորածինը, ի տարբերություն իր ծնողների, գաղափար չունի կատուների կամ շների մասին: Որպեսզի երեխային սովորեցնեն տարբերել այս երկու օբյեկտները միմյանցից, ծնողները նրան ցույց կտան տարբեր նկարներ և կասեն. «Այս նկարում կա կատու, բայց այս նկարում կա շուն»: Նման դասերը կրկնելով՝ արդյունքում տասնյակ նկարներ դիտելուց հետո (դրանք անվանենք ուսուցման հաջորդականություն) երեխան կհասկանա, որ կենդանիները ունեն մի շարք հատ-

կանիշներ, որոնցով կարելի է տարբերել կատվին շնից, և կիիշի այդ հատկանիշները:

Նախնական ուսուցումից հետո ծնողները կստուգեն երեխայի ստացած գիտելիքը: Նրան կցուցադրեն շների և կատուների նկարների այլ հավաքածու (դրանք անվանենք թեստային նմուշներ), որը երեխան պետք է ինքնուրույն բաժանի երկու խմբի՝ մեկի մեջ դնի բոլոր կատուների նկարները, մյուսում՝ շների: Դա էլ հենց օբյեկտների դասակարգման գործընթացն է, այսինքն՝ օբյեկտները խմբերի բաժանելը, որտեղ յուրաքանչյուրում բոլոր օբյեկտները ունեն ընդհանուր հատկանիշներ: Եթե երեխան անցավ այս թեստային առաջադրանքը, ինչը նշանակում է, որ նրա ուսուցումը հաջող է անցել, նա կարող է այդ գիտելիքները օգտագործել:

Որոշ ժամանակ անց կատու կամ շուն տեսնելիս (նույնիսկ այն, որը նա չի տեսել ուսուցման կամ թեստային նկարների մեջ), նա կնկատի դրանում եղած հատկանիշները և կգտնի այս հատկությունների համապատասխանությունը այս կամ այն օբյեկտին և եզրակացություն կանի:

Նմանատիպ գործընթաց իրականացնենք պերցեպտրոնի միջոցով: Ենթադրենք՝ պետք է ստեղծել պերցեպտրոն, որը կտարբերի կատուներին և շներին: Ի սկզբանե պերցեպտրոնը, ինչպես փոքր երեխան, չի կարող դա անել: Նոր ստեղծվող պերցեպտրոնի համար պետք է արվի նույն գործողությունները, ինչը անում են ծնողները փոքր երեխայի հետ, այսինքն՝ պետք է նրան սովորեցնել: Առաջին հայացքից այս առաջադրանքը բավականին բարդ է թվում կամ նույնիսկ անհնարին: Բայց դա հնարավոր է լուծել, եթե էապես պարզեցնենք խնդիրը և բաժանեք այն պարզ քայլերի հաջորդականության:

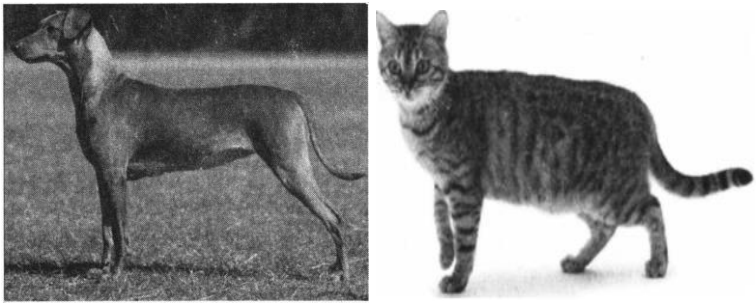
Պերցեպտրոնը, ինչպես երեխան, պետք է իմանա, թե ինչ հայտանիշերով կարելի է կատուներին առանձնացնել շներից: Պերցեպտրոնի առաջադրանքը պարզեցնելու համար առանձնացնենք երեք հայտանիշներ: Ուսուցման ժամանակ երեխան նկատել է, որ շները շատ ավելի բարձր են և ունեն երկար թաթեր, քան կատուները:

Ցուցադրված պատկերներում բոլոր շներն ունեին միագույն, իսկ կատուները՝ բազմերանգ մորթի (կային բազմերանգ բծեր ու զոլեր): Կատուների գլուխները կլոր էին, իսկ շներինը՝ երկարուկ: Ըստ այս երեք բնորոշ հայտանիշների՝ պերցեպտրոնը կսովորի տարբերակել կատուներին (1-ն խումբ) և շներին (2-րդ խումբ):

Պերցեպտրոնում ձևավորենք երեք սենսոր (երեք S տարր)՝ Sl-թաթի երկարություն, S2- բրդի գույն, S3-գլխի ձև: Քանի որ S տարրերը կարող են ընդունել 0 կամ 1 արժեքներ, ապա ընդունենք, որ սենսորները ունեն հետևյալ արժեքները.

- Sl=1 (երկար ոտքեր), Sl=0 (կարճ ոտքեր),
- S2=1 (միագույն բուրդ), S2=0 (բազմագույն բուրդ),
- S3=1 (երկարավուն գլուխ), S3=0 (կլոր գլուխ):

Այսպիսով, ձևավորեցինք սենսորային դաշտ: Այն կարող է ներկայացվել S տարրերի հնարավոր 0 և 1 արժեքների բազմության տեսքով: Մեր օրինակում 2-րդ խմբի օբյեկտը (շունը) պերցեպտրոնի ելքում ճիշտ դասակարգելու համար S տարրերի (1,1,1) հավաքածուն իդեալական տարբերակ է, այսինքն՝ երկար ոտքեր, բուրդը՝ միագույն, երկարավուն գլուխ: Իդեալական հավաքածու կատվի (1-ին խումբ) համար (0,0,0)-ն է, այսինքն՝ կարճ ոտքեր, բազմագույն բուրդ, կլոր գլուխ (նկ. 2.4):



Նկ. 2.4 S տարրերի իդ. հավաքածու (1,1,1) S-տարրերի իդ. հավաքածու (0,0,0)

Բերված օրինակում սենսորների տարբեր հավաքածուներ պերցեպտրոնի ելքում որոշում են տարբեր դասերին պատկանող օբյեկտներ: Իրականում կատարվել է օբյեկտների որոշ դասակարգում, այսինքն՝ ձևավորվել է օբյեկտների երկու դաս: Մաթեմատիկական տեսանկյունից պերցեպտրոնի էությունը կարելի է բնութագրել հետևյալ կերպ: Դա ֆունկցիա է, որը, մուտքում ունենալով S տարրերի արժեքները, ելքում ստանում է 0 կամ 1 արժեք: Մեր օրինակում այդ ֆունկցիայի արժեքներն ունեն ֆիզիկական իմաստ. 0-ն՝ 1-ին խումբ, 1-ը՝ 2-րդ խումբ:

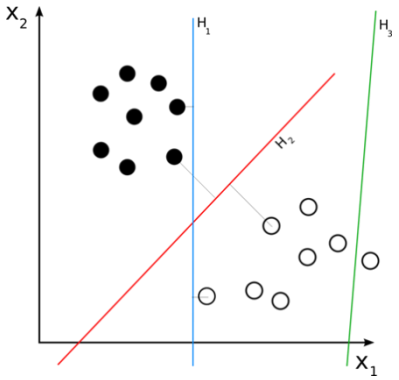
Ռոզենբլատի առաջին թեորեմից հետևում է, որ ճիշտ դասակարգող պերցեպտրոնների բազմությունը դատարկ չէ:

Գործնականում կարելի է ձևավորել S տարրերի ցանկացած հավաքածու և դրանց հիման վրա ստեղծել ցանկացած դասակարգում: Իսկ «լուծումների» բազմությունը դատարկ չի լինի, այսինքն՝ միանշանակ արդյունք միշտ էլ կստացվի: Սա նշանակում է, որ տեսականորեն պերցեպտրոններն ի վիճակի են լուծելու օբյեկտների դասակարգման հետ կապված ցանկացած խնդիր, այսինքն՝ տարբեր օբյեկտներ բաժանել խմբերի: Դիտարկված օրինակում օբյեկտները բաժանեցինք երկու խմբերի:

Այնուամենայնիվ, կան երկու կարևոր սահմանափակումներ.

1. Դիտարկվում են տարրական պերցեպտրոնները։
2. Այն օբյեկտները, որոնք մենք ցանկանում ենք դասակարգել, պետք է ունենան գծային բաժանելիության հատկություն, այսինքն՝ դրանք պետք է ունենան տարբեր հայտանիշների հավաքածուներ, որոնց հիման վրա հնարավոր լինի տարբերակել մի օբյեկտը մյուսից։

Ի՞նչ է օբյեկտների գծային բաժանելիությունը: Երկչափ տարածության մեջ կետերի երկու բազմություններ կոչվում են գծորեն բաժանելի, եթե դրանք կարող են ամբողջությամբ առանձնացվել միմյանցից ուղիղ գծով (նկ. 2.5)։



Նկ. 2.5 H_1 և H_2 ուղիղներով առանձնացված տվյալներ

Երկչափ տարածության մեջ բազմության կետերը բնութագրվում են երկու հայտանիշներով: Երեք հայտանիշների դեպքում ունենք եռաչափ տարածություն: Այդ դեպքում կետերի բազմությունը դասակարգելու համար գծվելու են հարթություններ: Եվ այսպես շարունակ. ընդհանուր առմամբ n-չա-

փանի տարածության մեջ կառուցվում են n-1 չափի հիպերհարթություններ:

Ֆրենկ Ռոզենբլատը երկրորդ թեորեմում ապացուցում է հետևյալը: Եթե կա պերցեպտրոն (այսինքն՝ սենսորների դաշտ և ինչ-որ դասակարգում կապված դրա հետ), ապա պերցեպտրոնի ուսուցման գործընթացում օգտագործելով սխալների ուղղման մեթոդը, անկախ կշռային գործակիցների սկզբնական վիճակից և ազդակների հանդես գալու հաջորդականությունից, միշտ վերջավոր ժամկետում կունենանք լուծում:

Կամայական սկզբնական վիճակ այստեղ նշանակում է պերցեպտրոն S-A և A-R կապեր կամայական կշիռներով: Թեորեմում լուծում ասելով հասկացվում է, որ տրված պայմաններում որոշակի կշիռներով պերցեպտրոնում հաջողությամբ լուծվում է դասակարգման խնդիրը:

Այս թեորեմը երաշխավորում է դասակարգման խնդիրների լուծման հաջողությունը: Հիմա մենք գիտենք, որ միշտ կարելի է լուծել դասակարգման խնդիրը որոշակի ժամանակահատվածում: Երկու թեորեմներն էլ ունեն ապացույցներ:

Նշենք, որ այս եզրակացությունները հիմնավորված են մաթեմատիկական ապացույցներով, ինչպես նաև գործնականում իրականացված: Ռոզենբլատը ստեղծել է աշխարհում առաջին նեյրոնային համակարգիչը՝ Mark-1, որը կարողացել է սովորեցնել լուծել գործնական խնդիրներ, որոնք չեն ենթարկվել ավանդական ալգորիթմացման:

Այսպիսով, տարրական պերցեպտրոնները երաշխավորված են լուծելու գծորեն բաժանվող օբյեկտների դասակարգման խնդիրները: Դրանց ուսուցման համար օգտագործվում է սխալների ուղղման մեթոդը:

2.5 Պերցեպտրոնին սովորեցնում ենք հասկանալ պատկերներ

Նկարագրենք նեյրոնային ցանցի ուսուցման գործընթացը: Ի՞նչ է դա, և ինչպես է այն տեղի ունենում:

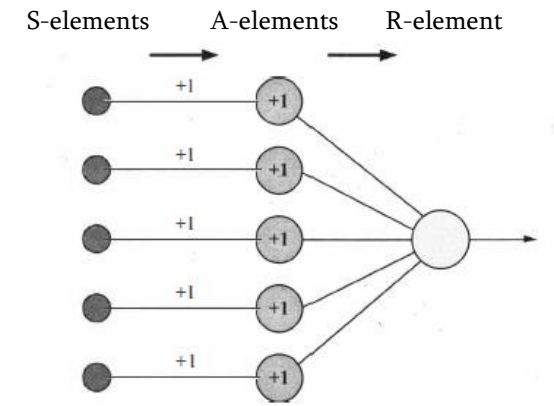
Արհեստական նեյրոնային ցանցը արհեստական նեյրոնների հավաքածու է: Ծրագրային իրականացման միջոցով նկարագրենք նեյրոնների և դրանց կապերի ստեղծումը, ցանցի ուսուցումը և օգտագործումը:

Ցանցի մուտքին կտանք մուտքային ազդանշանների հավաքածու և կնկարագրենք ցանցի ելքի ստացումը: Նախօրոք պետք է որոշել թե ինչ տիպի խնդիր պետք է լուծինեյրոնային ցանցը, և միայն դրանից հետո նրան պետք է ուսուցանել:

Նեյրոնային ցանցի ուսուցումը ցանցի տարրերի միջև կապերի կշռային գործակիցների ճշգրտումն է, այնպես, որ ցանցի մուտքին տրվող ազդանշանների տվյալ հավաքածուի դեպքում այն տա ճիշտ պատասխան:

Ստեղծենք պարզ միաշերտ պերցեպտրոն, որը ունի հետևյալ կառուցվածքը և պարամետրերը (նկ. 2.6).

- յուրաքանչյուր S տարր կապված է միայն մեկ A տարրի հետ,
- բոլոր S-A միացումների կշիռները հավասար են + 1-ի,
- բոլոր A տարրերի շեմերը հավասար են + 1-ի,
- կա միայն մեկ R տարր,
- բոլոր A-R կապերը կարող են ընդունել միայն ամբողջ արժեքներ (..., -2, -1, 0, 1, 2, ...):



Նկ. 2.6 Միաշերտ պարզագույն պերցեպտրոն մեկ թաքնված շերտով

2.5.1 Թվանշանների ճանաչում

Այժմ փորձենք սովորեցնել պերցեպտրոնին ճանաչել 0–9 թվանշանները:

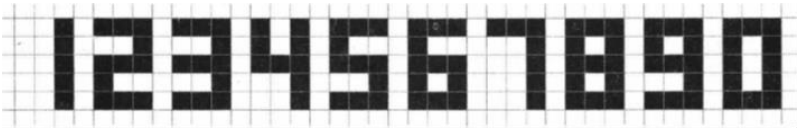
Ինչպե՞ս են երեխաներին սովորեցնում ճանաչել թվանշանները: Նրանց ցուցադրում են թվանշանների նկարները համապատասխան բացատրություններով: Օրինակ՝ «Այս նկարում 1 թվանշանն է», «Այս նկարում 2 թվանշանն է» և այլն: Նույնն անենք պերցեպտրոնի համար: Նրան ցույց տանք պատկեր՝ «բացատրելով», թե որ թվանշանն է այն: Ինչպե՞ս կարող ենք պատկերի իմաստը փոխանցել պերցեպտրոնին: Նկարը, ինչպես գիտեք, բաղկացած է պիքսելներից: Յուրաքանչյուր պիքսելի արժեք կարող է համարվել պերցեպտրոնի մեկ սենսոր: Այսինքն՝ քանի պիքսել կա նկարում, այդքան S տարր պետք է ստեղծվի պերցեպտրոնի համար:

Ի՞նչ ազդանշաններ կուղարկվեն յուրաքանչյուր S տարրին: Որպեսզի պարզեցվի պերցեպտրոնի ուսուցման գործըն-

թացի ըմբռնումը, պարզեցնենք կատարվելիք առաջադրանքը: Այսինքն՝

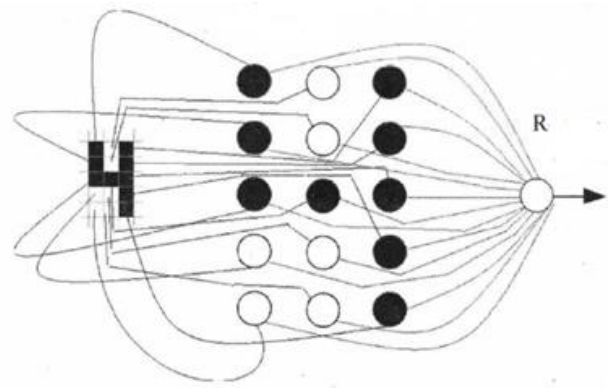
- սովորեցնենք նրան ճանաչել միայն սև-սպիտակ 0-ից մինչև 9 թվանշանները,
- բոլոր թվանշանները գրված կլինեն աղյուսակներում,
- ուսուցման փուլում պերցեպտրոնը պետք է սովորի բոլոր թվանշանները:

Սկզբում ձևավորենք ուսուցման հաջորդականություն: Ստեղծենք նկարների հավաքածու, որը կօգտագործենք պերցեպտրոնի ուսուցման փուլում: Յուրաքանչյուր թվանշանի նկարը պատկերենք 15 քառակուսուց կազմված աղյուսակում՝ 5 տող, 3 սյուն (նկ. 2.7):



Նկ. 2.7 Ուսուցման հաջորդականություն

Խնդրի լուծման համար անհրաժեշտ կլինի 15 հատ S-տարր, որոնք նույնացվում են A տարրերի հետ: Յուրաքանչյուրը S տարր ազդանշան կստանա 3 x 5 չափի աղյուսակի մեկ քառակուսուց: Քառակուսու սև գույնը կհամապատասխանի S տարրի գրգռված վիճակին, իսկ սպիտակ գույնը՝ սենսորի արգելակված վիճակին: Նկ. 2.8-ում ցույց է տրված, թե ինչ վիճակում կլինեն բոլոր 15 հատ S տարրերը այն դեպքում, եթե նրանց տրվի 4 թվանշանը ուսուցման հավաքածուից:

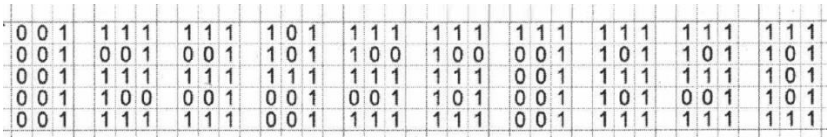


Նկ. 2.8 S տարրերի վիճակը 4 թվանշանի դեպքում

Պերցեպտրոնի հետ աշխատելու համար պետք է նրա մուտքային ազդանշանները տրվեն թվանշանի պատկերի տեսքով: Քանի որ քառակուսու վանդակի սև գույնին համապատասխանում է S տարրի գրգռված վիճակը, ապա այդպիսի ազդանշանի արժեքը հավասար կլինի 1: Իսկ սպիտակ գույնին համապատասխանում է S տարրի արգելակված վիճակը, ուստի ազդանշանի արժեքը հավասար կլինի 0: Չորս թվանշանի համար սենսորների վիճակը թվային տեսքով կներկայացվի նկ. 2.9-ում տրված աղյուսակով:

Նկ. 2.9 4 թվանշանի սենսորների վիճակը թվային տեսքով

0-9 թվանշանների թվային աղյուսակները կունենան նկ. 2.10-ում բերված տեսքը:



Նկ. 2.10 Պերցեպտրոնի սենսորների վիճակը 0-9 թվանշանների համար

Պերցեպտրոնում սենսորներից ստացված տվյալները A տարրերի միջոցով անցնում են գումարիչ (R տարր): Մեր դիտարկված պերցեպտրոնում բոլոր S-A միացումների կշիռները հավասար են +1-ի: Գումարիչում ստացվում է սենսորներից փոխանցված գումարը: 0-9 թվանշանների համար R տարրի արժեքները ներկայացված են նկ.2.11-ում:

Թվանշան	Սենսորներից եկած ազդանշաններ	R-տարրի արժեք
	0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 1	5
2	1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 1 + 1 + 0 + 0 + 1 + 1 + 1	11
3	1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 1	11
4	1 + 0 + 1 + 1 + 0 + 1 + 1 + 1 + 1 + 0 + 0 + 1 + 0 + 0 + 1	9
5	1 + 1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 1	11
6	1 + 1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 1 + 0 + 1 + 1 + 1 + 1	12
7	1 + 1 + 1 + 0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 1	7
8	1 + 1 + 1 + 1 + 0 + 1 + 1 + 1 + 1 + 1 + 0 + 1 + 1 + 1 + 1	13
9	1 + 1 + 1 + 1 + 0 + 1 + 1 + 1 + 1 + 0 + 0 + 1 + 1 + 1 + 1	12
0	1 + 1 + 1 + 1 + 0 + 1 + 1 + 0 + 1 + 1 + 0 + 1 + 1 + 1 + 1	12

Նկ. 2.11 Գումարիչի աշխատանքի արդյունքները սենսորներից եկած ազդանշանների համար

Նկատենք, որ, ունենալով միայն R տարրի արժեքը, հնարավոր չէ որոշել, թե այդ արժեքը որ թվանշանին է համապատասխանում: Նկ.2.10-ում երևում է, որ. օրինակ, R տարրի 11

արժեքը համապատասխանում է երեք թվանշանի՝ 2, 3 և 5: R տարրի 12 արժեքը նույնպես համապատասխանում է երեք թվանշանների՝ 6, 9 և 0: Սա նշանակում է, որ սենսորների միջոցով հաշվարկված գումարով ստացված R տարրի արժեքով հնարավոր չէ միարժեք ճանաչել մուտքային թվանշանը:

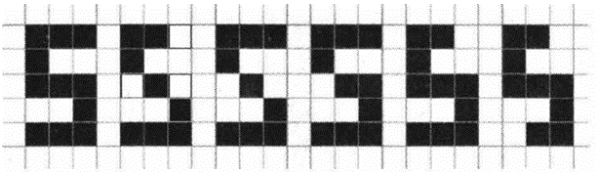
Նշենք, որ աղյուսակի յուրաքանչյուր տողի գրառումները իրար կցելուց ստացված տողը, որը զրոների և մեկերի հաջորդականություն է (նիշերի կցում), կլինի եզակի և կհամապատասխանի տվյալ թվանշանին: Հետևաբար այդ տողերը կարող ենք օգտագործել պերցեպտրոնում մուտքային պատկերներին համապատասխան թվանշանները ճանաչելու համար: R տարրում նիշերի կցումուվ ստացված տողերը բերված են նկ. 2.12-ում:

Թվանշան	Սենսորներից եկած ազդանշաններ	R-տարրի արժեք
1	001 + 001 + 001 + 001 + 001	001001001001001
2	111 + 001 + 111 + 100 + 111	111001111100111
3	111 + 001 + 111 + 001 + 111	111001111001111
4	101 + 101 + 111 + 001 + 001	101101111001001
5	111 + 100 + 111 + 001 + 111	111100111001111
6	111 + 100 + 111 + 101 + 111	111100111101111
7	111 + 001 + 001 + 001 + 001	111001001001001
8	111 + 101 + 111 + 101 + 111	111101111101111
9	111 + 101 + 111 + 001 + 111	111101111001111
0	111 + 101 + 101 + 101 + 111	111101101101111

Նկ. 2.12 Գումարիչի աշխատանքի արդյունքները ազդանշանների նիշերի կցումով

Այսպիսով, պերցեպտրոնը սովորեց ճանաչել թվանշանները։ Այդ արդյունքները (նկ.2 12) պետք է պահպանել հիշողությունում։ Հետագայում մեր ուսուցանված պերցեպտրոնը, մուտքում տեսնելով, օրինակ, 4 թվանշանի պատկերը և ազդանշաններ ստանալով սենսորներից, դրանք վերածում է 101101111001001 տողի։ Այնուհետև իր հիշողության մեջ կգտնի նույն տողը և կտա պատասխան՝ «Զրոների և մեկերի այս տողին համապատասխանում է 4 թվանշանը»։

Այնուամենայնիվ, պետք է հիշենք, որ մենք զգալիորեն պարզեցրել ենք խնդրի պայմանները։ Պերցեպտրոնի մուտքում տալիս ենք նույն չափի թվանշաններ (3 x 5 չափի աղյուսակներ) և աղյուսակները լրացնում ըստ նմուշի։ Նման պերցեպտրոնը հաջողությամբ կաշխատի միայն այն դեպքում, եթե ցուցադրվեն թվանշանների պատկերներ, որոնք նույնությամբ համապատասխանում են ուսուցման նմուշին։ Բայց իրական կյանքում թվանշաններն ունեն տարբեր չափսեր, գրված են տարբեր տառատեսակներով և նույնիսկ ավելին՝ նույն թվանշանը, որը գրված է տարբեր մարդկանց կողմից, ինչ-որ չափով տարբերվում է։ Նկ. 2.13-ում ցույց է տրված 5 թվանշանը պատկերելու տարբերակներ։



Նկ. 2.13 Թվանշան 5-ի հնարավոր պատկերներ

Մարդը կարող է հեշտությամբ անտեսել թվանշանի գրառման մեջ առկա տարբերությունները և ճիշտ որոշել այդ թվանշանի արժեքը։ Բայց եթե այդ պատկերները տանք պեր-

ցեպտրոնի մուտքին, այն չի գտնի ճիշտ պատասխան, քանի որ նրա հիշողության մեջ չկան այնպիսի տողեր, որոնք համապատասխանում են սև և սպիտակ պիքսելների նման դասավորությանը:

Այս պերցեպտրոնը ինչ-որ բան սովորեց, բայց շարունակենք ուսուցումը, որովհետև. պարզեցված պերցեպտրոնային մոդելում սահմանել ենք պայմաններ, որոնց դեպքում բոլոր կապերի կշիռների արժեքները նույնն են և հավասար են 1-ի: Բայց եթե սովորեցնենք պերցեպտրոնին ինքնուրույն ընտրել կապերի կշիռները սենսորային արժեքների տարբեր կոնֆիգուրացիաների համար, ապա իրականում կսովորեցնենք նրան տարբերել նույն 5 թվանշանը գրելու տարբեր ձևերը և ոչ միայն այդ թվանշանը: Անցնենք ուսուցման հաջորդ փուլին:

2.6 Պերցեպտրոնին սովորեցնում ենք ընտրել կապի կշիռները

Նախորդ բաժիններում արդեն նշվել է, որ ուսուցման գործընթացում կարևոր է սենսորային տարրերի և գումարիչի միջև կապերի կշիռների ընտրությունը:

Գիտենք, որ որոշակի մուտքերի (S-տարրերի) կարևորությունը տրվում է կշիռներով, որոնք դրանք կապում են R-տարրի հետ: Որքան ուժեղ է որոշ կապի կշիռը, այնքան ավելի ուժեղ է այս սենսորը ազդում վերջնական արդյունքի վրա: Բայց ինչպե՞ս փոխել կշիռը, ի՞նչ չափով: Եթե պատահականորեն փոխենք կշիռները, ուսուցման գործընթացը կարող է ձգվել տարիներ:

Հիմա փորձենք ուսուցանել նեյրոնային ցանցը, դրա համար պետք է ունենալ ուսուցանող ծրագիր:

1949 թվականին կանադացի ֆիզիոլոգ Դոնալդ Հեբը ստեղծել է նեյրոնային ցանցերի ուսուցման ալգորիթմ, որն օգտագործում է հետևյալ կանոնները.

Կանոն 1. Եթե պերցեպտրոնի ելքային ազդանշանը սխալ է և հավասար 0-ի, ապա անհրաժեշտ է մեծացնել այն մուտքերի կշիռները, որոնցով տրվել է 1:

Կանոն 2. Եթե պերցեպտրոնի ազդանշանը սխալ է և հավասար է 1-ի, ապա անհրաժեշտ է նվազեցնել այն մուտքերի կշիռները, որոնցով տրվել է 1:

Սրանք, ըստ էության, այն երկու կանոններն են, որոնց վրա հիմնված է պերցեպտրոնի ուսուցման ալգորիթմը: Այս դեպքում մուտքերը հավասար են 0 կամ 1, և լուծվում է դասակարգման ամենապարզ խնդիր:

Այժմ ներկայացնենք ծրագիր, որը նեյրոնային ցանցին կուսուցանի ճանաչել թվանշանները՝ օգտագործելով Հեբի կանոնները: Նկարագրենք քայլերը, որոնք պետք է կատարվեն այնպես, որ նեյրոնային ցանցը սովորի ճիշտ ճանաչել, օրինակ՝ 5 թվանշանը:

1. Եթե նեյրոնային ցանցը ճիշտ է ճանաչել թվանշանը (այսինքն՝ 5 թվանշանը ընդունվել է որպես 5), ապա պետք է մեծացնել բոլոր կշիռները, որոնց միջոցով դրական ազդանշաններ են անցել: Այսինքն՝ պետք է մեծացնել այն սենսորների ազդեցությունը, որոնք գրգռված էին և հանգեցրին ճիշտ եզրակացության:

2. Եթե նեյրոնային ցանցը սխալ է ճանաչել ներկայացված թվանշանը (օրինակ՝ 3 թվանշանը ճանաչել է որպես 5), պետք է նվազեցնել այն կապերի կշիռները, որոնց միջոցով դրական ազդանշան է անցել: Այսինքն՝ պետք է նվազեցնել այն սենսոր-

ների ազդեցությունը, որոնք ակտիվ էին և հանգեցրել են սխալ եզրակացության։

Այժմ նկարագրենք թվանշաների ուսուցման ալգորիթմը և կիրառենք այն 5 թվանշանի ճանաչման համար։

Կան Python-ի պատրաստի գրադարաններ, որոնք պետք է ներբեռնել։ Սա մեծապես հեշտացնում է ծրագրավորողների աշխատանքը։ Python ծրագրերում առաջին քայլերը անհրաժեշտ գրադարանների ներառումն է։ Քանի որ ծրագրի համար օգտագործում ենք պատահական թվերի գեներատոր, ապա ներառվում է random մոդուլը։

Հիմա ստեղծենք 0-ից 9-ը թվանշանների իդեալական պատկերներով ուսուցման նմուշներ և բոլոր թվանշանները գրենք նիշերի տողերի տեսքով (վերցնենք այդ տողերը նկ. 2.12-ի աղյուսակի R-տարրերի արժեքների սյունից)։

ուսուցման հաջորդականություն (0-ից 9-ը թվանշանների իդեալական պատկերներ)

```
num0 = list('111101101101111')
num1 = list('001001001001001')
num2 = list('111001111100111')
num3 = list('111001111001111')
num4 = list('101101111001001')
num5 = list('111100111001111')
num6 = list('111100111101111')
num7 = list('111001001001001')
num8 = list('111101111101111')
num9 = list('111101111001111')
```

Բոլոր 10 թվանշանները հավաքենք ընդհանուր զանգվածում։

nums = [num0, numl, num2, num3, num4, num5, num6, num7, num8, num9]

Նշենք ուսուցման թեման, այսինքն՝ թե որ թվանշանը ճանաչելու համար ենք ուսուցանում ցանցը և սենսորների քանակը (մուտքային ազդանշանները ստանում ենք 15 սենսորներից)։

```
tema = 5 # որ թվանշանն ենք ուսուցանում
n_sensor = 15 # Սենսորների քանակը
```

Ստեղծենք կշիռների վեկտոր։ Քանի որ ունենք 15 սենսոր, որոնք միացված են մեկ R տարրի, ապա անհրաժեշտ է 15 կապ։ Ծրագրի սկզբում կշիռների վեկտորի բոլոր տարրերի արժեքները հավասար են 0-ի։

```
# Սենսորների կապերի կշիռների սկզբնարժեքավորում
weights = []
for i in range(15):
  weights.append(0)
```

Նույնը կարելի է գրել հետևյալ կերպ.

```
weights = [0 for i in range(15)] # Կշիռների զրոյացում
```

Գրենք պերցեպտրոնի ֆունկցիան։ Այն կհաշվի կշռված գումարը, կհամեմատի շեմի հետ և կորոշի՝ պատկերը 5 թվանշա՞նն է, թե՞ ոչ։ Օրինակ 2.2-ում պերցեպտրոնի ֆունկցիան վերադարձնում է True , եթե ճանաչվել է 5, և False՝ հակառակ դեպքում։

Օրինակ 2.2

```
# Արհեստական նեյրոն (պերցեպտրոն)
def perceptron(Sensor):
  b = 7 # Ակտիվացման ֆունկցիայի շեմ
  s = 0 # Գումարի սկզբնարժեքավորում
  for i in range(n_sensor): # Սենսորների ազդանշանների գումարման ցիկլ
    s += int(Sensor[i]) * weights[i]
  if  s >= b:
        return True # Գումարը գերազանցում է շեմը
  else:
       return False # Գումարը փոքր է շեմից
```

Քանի որ օգտագործվում է ակտիվացման շեմային ֆունկցիա, ապա պետք է սահմանվի շեմ: Եթե կշռված գումարը մեծ է կամ հավասար շեմին, ապա ցանցի ելքը կլինի «True» (նշանակում է՝ ցանցը համարում է, որ տրված թվանշանը 5 է): Կարելի է ընտրել շեմային տարբեր արժեքներ և ստանալ տարբեր ելքեր:

Այժմ սահմանենք երկու օժանդակ ֆունկցիա՝ «պատիժ» և «խրախուսում»: Առաջին ֆունկցիան կանչվում է, երբ ցանցն որպես 5 հասկանում է մեկ այլ թվանշան: Այդ դեպքում մեկով նվազեցվում են գրգռված մուտքերի հետ կապված բոլոր կշիռները (օրինակ՝ 2.3): Հիշեցնենք, որ մուտքը համարում ենք գրգռված (1), եթե այն ստացվել է սև պիքսելից:

Օրինակ 2.3

Կշռային արժեքների փոքրացում, եթե ցանցը պատասխանել է True հինգից տարբեր մուտքային թվանշանի համար

```
def decrease(number):
  for i in range(n_sensor):
    if int(number[i]) == 1: # Մուտքը գրգռվա՞ծ է
        weights[i] -= 1 # Նվազեցնել մուտքային արժեքի հետ կապված կշռային
                        # գործակիցը մեկով
```

Երկրորդ ֆունկցիան կանչվում է, երբ ցանցը ճանաչել է 5 այն դեպքում, երբ մուտքում 5 է: Այդ ֆունկցիան մեծացնում է 1-ով բոլոր կշիռները, որոնք կապված են գրգռված մուտքերի հետ (օրինակ 2.4):

Օրինակ 2.4

Կշռային արժեքների ավելացում, եթե ցանցը պատասխանել է True մուտքային հինգ թվանշանի համար

```
def increase(number):
  for i in range(n_sensor):
    if int(number[i]) == 1: # Մուտքը գրգռվա՞ծ է
        weights[i] += 1 # Ավելացնել մուտքային արժեքի հետ կապված կշռային
                        # գործակիցը մեկով
```

Օրինակ 2.5-ում ուսուցանենք նեյրոնային ցանցը 5 թվանշանի համար (tema=5): Ուսուցումը իրականացվում է n դասերի միջոցով: Յուրաքանչյուր դաս համապատասխանում է ցիկլի մեկ իտերացիայի, որի ընթացքում պատահականորեն գեներացվում է մի թվանշան 0-9 միջակայքից: Այդ թվանշանը տալով պերցեպտրոնի մուտքին, կախված պերցեպտրոնի ելքից, փոփոխության է ենթարկվում 5 թվանշանի կշռային վեկտորը:

Օրինակ 2.5

Ցանցի ուսուցում, մուտքային տվյալներ

```
import random
tema = 5 # որ թվանշանն ենք ուսուցանում
n_sensor = 15 # Սենսորների քանակ

weights = [0 for i in range(n_sensor)] # Բոլոր կշռային գործակիցներին վերագրել 0
num0 = list('111101101101111')
numl = list('001001001001001')
num2 = list('111001111100111')
num3 = list('111001111001111')
num4 = list('101101111001001')
num5 = list('111100111001111')
num6 = list('111100111101111')
num7 = list('111001001001001')
num8 = list('111101111101111')
num9 = list('111101111001111')

nums = [num0, numl, num2, num3, num4, num5, num6, num7, num8, num9]
```

Ցանցի ուսուցում, ֆունկցիաների նկարագրություններ

```
# Արհեստական նեյրոն (պերցեպտրոն)
def perceptron(Sensor):
  b = 7 # Ակտիվացման ֆունկցիայի շեմ
  s = 0 # Գումարի սկզբնարժեքավորում
  for i in range(n_sensor): # Սենսորների ազդանշանների գումարման ցիկլ
    s += int(Sensor[i]) * weights[i]
  if  s >= b:
        return True # Գումարը գերազանցում է շեմը
  else:
       return False # Գումարը փոքր է շեմից

def decrease(number):
  for i in range(n_sensor):
    if int(number[i]) == 1: # Մուտքը զրգռվա՞ծ է
        weights[i] -= 1 # Նվազեցնել մուտքային արժեքի հետ կապված կշռային
                        # գործակիցը մեկով

def increase(number):
  for i in range(n_sensor):
    if int(number[i]) == 1: # Մուտքը զրգռվա՞ծ է
        weights[i] += 1 # Ավելացնել մուտքային արժեքի հետ կապված կշռային
                        # գործակիցը մեկով
```

Ցանցի ուսուցում

```
n = 10 # Դասերի քանակը
for i in range(n):

  j = random.randint(0, 9) # Գեներացվում է j պատահական թիվ 0-9 միջակայքից
  r = perceptron(nums[j])  # Պերցեպտրոնի ելք

  if j != tema:
    if r:

      decrease(nums[j]) # Ցանցի պատժում (կշռային գործակիցների փոքրացում)
  else:
    if not r:

      increase(nums[tema]) # Ցանցի խրախուսում (կշռային գործակիցների մեծացում)

print(weights)
```

Յուրաքանչյուր թվանշանի ուսուցման ընթացքում պերցեպտրոնին փոխանցվում են սենսորային արժեքները, որոնք բնութագրում են գեներացված թվանշանը: Հետագայում, կախված ստացված ելքից, ուսուցումը կամ պատժվում, կամ խրախուսվում է մշակված ֆունկցիաներով: 5 թվանշանի ուսուցման բոլոր դասերի ավարտից հետո ստացվում է ուսուցման արդյունքը, այսինքն՝ 5 թվանշանի կշռային վեկտորը, որն արտածվում է print (weights) հրամանով:

Ստորև բերված են 5 թվանշանի կշռային վեկտորները դասերի տարբեր քանակների դեպքում.

n=10

```
[0, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 1, 1, 0]
```

n=100

```
[0, 0, 0, 3, 0, -5, 0, 0, 0, -5, 0, 0, 0, 0, 0]
```

n=1000

```
[1, 1, 1, 2, 0, -7, 1, 2, 1, -7, 0, 1, 1, 1, 1]
```

n=10000

```
[1, 1, 1, 2, 0, -6, 1, 1, 1, -6, 0, 1, 1, 1, 1]
```

n=100000

```
[1, 2, 1, 1, 0, -9, 1, 2, 1, -9, 0, 1, 2, 2, 1]
```

Օրինակ 2.5-ում բերված ծրագիրը թեստավորելու նպատակով աշխատեցնենք այն՝ դասերի քանակը սահմանելով 1, իսկ ելքային արդյունքները դուրս բերենք հետևյալ հրամաններով՝

print(j)

print(weights)

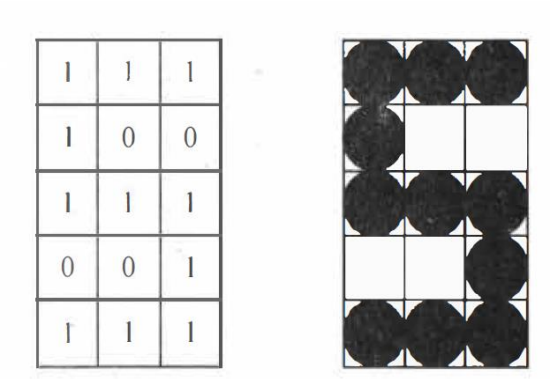
Ծրագրի յուրաքանչյուր թողարկման ժամանակ կգենեըրացվի 0-9 միջակայքից որևէ մի թվանշան:

```
2
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
5
[1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1]
```

5-ից տարբեր բոլոր թվանշանների համար 15 սենսորների կշիռները կլինեն զրո:

Կշռային վեկտորի իմաստը մեկնաբանելու նպատակով կշիռների այս արժեքները տեղադրենք 3 x 5 չափի աղյուսակում, այնուհետև նույն տիպի աղյուսակում 1-երի փոխարեն դնենք սև շրջանակներ, իսկ 0-ների փոխարեն՝ սպիտակ (նկ. 2.14):



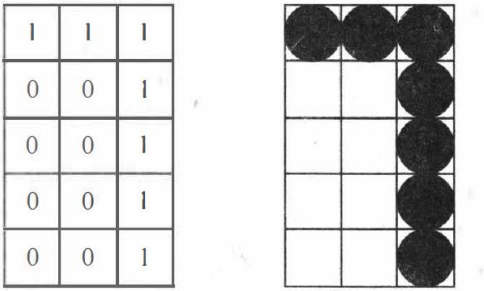
Նկ. 2.14 Կշռային արժեքների հիման վրա 5 թվանշանի պատկերը սենսորային դաշտում

Այսպիսով, կշռային գործակիցների արժեքներով պատկերվեց 5 թվանշանը։

Այժմ կշռման գործակիցների արժեքների կիրառմամբ փորձենք նկարել 7 թվանշանը։ Դա անելու համար ծրագրում փոխենք թեման՝ tema=7: Ծրագրի աշխատանքի արդյունքում 7 թվանշանի համար կստանանք կշռային գործակիցների հետևյալ արժեքները.

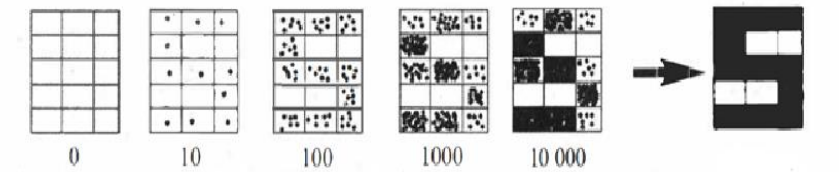
```
7
[1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1]
```

3 x 5 չափի աղյուսակում տեղադրելով կշիռների այս արժեքները՝ կստանանք 7 թվանշանի պատկերը (նկ. 2.15)։

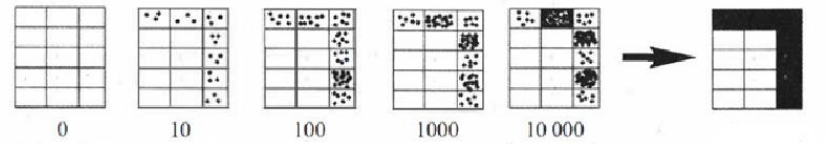


Նկ. 2.15 Կշռային արժեքների հիման վրա 7 թվանշանի պատկերը սենսորային դաշտում

Փորձարկումները ցույց են տալիս, որ որքան ուսուցման ցիկլերը շատ լինեն (ավելի շատ դասեր տանք), այնքան ցանցը ավելի խելացի կդառնա (առավել նշանակալի սենսորների կշիռները կավելանան, մյուսներինը՝ կնվազեն): 5 և 7 թվանշանների համար դա ցուցադրված է նկ. 2.16-ում: Ծրագիրը թողարկված է 5 անգամ՝ յուրաքանչյուր դեպքում դասերի քանակը վերցնելով համապատասխանաբար 10, 100, 1000, 10000, 100000:



Մինչ ուսուցում Դասերի քանակ (n=10, 100, 1000, 10000) Նմուշ



Մինչ ուսուցում Դասերի քանակ (n=10, 100, 1000, 10000) Նմուշ

Նկ. 2.16 Դասերի քանակի ազդեցությունը ուսուցման արդյունքների վրա

Այսպիսով ցանցը վարժեցվեց երկու թվանշան ճանաչելու համար։ Յուրաքանչյուր թվանշանի համար ընտրվեցին սենսորների կապերի կշիռներ (նկ. 2.17, 2.18)։ Ինչպես տեսանք, ուսուցանող ծրագիրը մեկն էր, որը ցանցին տվեց երկու թվանշան ճանաչելու հմտություններ։ Այսպիսով պետք է ցանցը ուսուցանել բոլոր թվանշանները ճանաչելու համար՝ փոխելով ուսուցման թեման։

Ուսուցման ցիկլեր	S-R կապերի կշռային գործակիցները 15 սենսորների համար														
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15
10	0	0	0	1	0	-2	0	0	0	-1	0	1	0	0	0
100	0	0	0	1	0	-4	0	0	0	-3	0	0	0	0	0
1000	1	1	1	1	0	-6	1	2	1	-6	0	1	1	1	1
10 000	1	1	1	4	0	-9	1	2	1	-9	0	1	1	1	1
100 000	1	1	1	3	0	-8	1	2	1	-8	0	1	1	1	1

Նկ.2.17 S-R կապերի կշռային գործակիցների արժեքները տարբեր թվով ուսուցման ցիկլերի համար (5 թվանշանի ճանաչում)

Ուսուցման ցիկլեր	S-R կապերի կշռային գործակիցները 15 սենսորների համար														
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15
10	0	0	0	0	0	0	-1	-1	0	0	0	0	-1	-1	0
100	1	1	1	-1	0	1	-1	0	1	-1	0	1	-1	-1	1
1000	1	1	1	-1	0	1	-1	-1	1	-1	0	1	-1	-1	1
10 000	1	1	1	-1	0	1	-1	0	1	-1	0	1	-1	-1	1
100 000	1	1	1	-1	0	1	-1	-1	1	0	0	1	-1	-1	1

Նկ.2.18 S-R կապերի կշռային գործակիցների արժեքները տարբեր թվով ուսուցման ցիկլերի համար (7 թվանշանի ճանաչում)

Ուսուցանված ցանցը թեստավորելու համար ստուգենք նրա աշխատանքը ինչպես ուսուցման տվյալների (իդեալական տեսքի թվանշաններ), այնպես էլ թեստային տվյալների (աղավաղված թվանշաններ) համար:

Թեստավորումը իրականացնենք հետևյալ կերպ: Ծրագիրը գործարկենք tema=5 դեպքի համար տարբեր քանակով ուսուցման ցիկլերով (10, 100, 1000, 10000, 100000), ինչի արդյունքում 5 թվանշանի համար կստացվեն բոլոր 15 սենսորային տարրերի կշռային արժեքները:

Դրանից հետո ծրագրին տրվում է առաջադրանք՝ համեմատել 5 թվանշանի հետ ուսուցման հավաքածուի 0-ից 9-ը բոլոր թվանշանների իդեալական պատկերները: Հետո ստուգվում է, թե ինչպես է մշակված ծրագրային մոդուլը իրականացրել այդ թեստային առաջադրանքը, և արտածվում են արդյունքները:

```
# պերցեպտրոնի աշխատանքի ստուգում ուսուցման տվյալների վրա
print("0-ն 5-է՞", perceptron(num0))
print("1-ն 5-է՞", perceptron(num1))
print("2-ն 5-է՞", perceptron(num2))
print("3-ն 5-է՞", perceptron(num3))
print("4-ն 5-է՞", perceptron(num4))
print("5-ն 5-է՞", perceptron(num5))
print("6-ն 5-է՞", perceptron(num6))
print("7-ն 5-է՞", perceptron(num7))
print("8-ն 5-է՞", perceptron(num8))
print("9-ն 5-է՞", perceptron(num9))
```

Թեստավորման արդյունքները բերված են նկ. 2.19-ում:

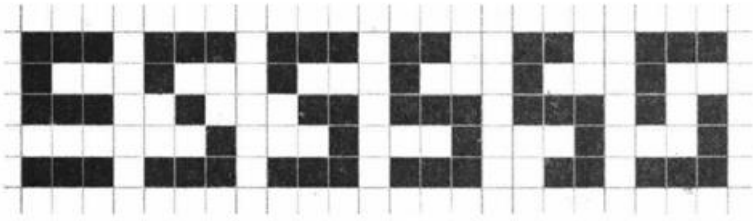
Թվանշաններ	Ուսուցման ցիկլեր				
	10	100	1000	10 000	100 000
0	0-ն 5-է՞ False	0-ն 5-է՞ False	0-ն 5-է՞ False	0-ն 5-է՞ False	0-ն 5-է՞ False
1	1-ն 5-է՞ False	1-ն 5-է՞ False	1-ն 5-է՞ False	1-ն 5-է՞ False	1-ն 5-է՞ False
2	2-ն 5-է՞ False	2-ն 5-է՞ False	2-ն 5-է՞ False	2-ն 5-է՞ False	2-ն 5-է՞ False
3	3-ն 5-է՞ False	3-ն 5-է՞ False	3-ն 5-է՞ False	3-ն 5-է՞ False	3-ն 5-է՞ False
4	4-ն 5-է՞ False	4-ն 5-է՞ False	4-ն 5-է՞ False	4-ն 5-է՞ False	4-ն 5-է՞ False
5	5-ն 5-է՞ False	**5-ն 5-է՞ True**	**5-ն 5-է՞ True**	**5-ն 5-է՞ True**	**5-ն 5-է՞ True**
6	6-ն 5-է՞ False	6-ն 5-է՞ True	6-ն 5-է՞ False	6-ն 5-է՞ False	6-ն 5-է՞ False
7	7-ն 5-է՞ False	7-ն 5-է՞ False	7-ն 5-է՞ False	7-ն 5-է՞ False	7-ն 5-է՞ False
8	8-ն 5-է՞ False	8-ն 5-է՞ False	8-ն 5-է՞ False	8-ն 5-է՞ False	8-ն 5-է՞ False
9	9-ն 5-է՞ False	9-ն 5-է՞ True	9-ն 5-է՞ False	9-ն 5-է՞ False	9-ն 5-է՞ False

Նկ.2.19 Ստուգման աշխատանքի արդյունքները («Դա 5՞ է» հարցին)

Ինչպես երևում է աղյուսակից, պերցեպտրոնը սկսեց ճշգրիտ տարբերակել հինգը մնացած բոլոր թվերից 1000-ից ավելի ուսուցման դասերից հետո:

Երկրորդ թեստում բարդացնենք պերցեպտրոնի առաջադրանքը: Ստուգենք՝ կկարողանա՞ արդյոք նա ճանաչել 5 թվանշանի տարբեր գրելաձևերը: Ստուգենք թեստային նմուշների համար, որտեղ կլինեն 5 թվանշանները՝ չնչին աղավաղումներով:

Ստեղծենք թեստային նմուշներ, որտեղ բոլոր պատկերները նույն չափսի են, ինչպես ուսուցման հավաքածուում (նկ. 2.20):



Նկ. 2.20 Թվանշան 5 ի աղավաղված պատկերներ

Այս պատկերները ձևափոխենք տողային ձևաչափի (նկ. 2.21): Աղյուսակից պարզ է դառնում, որ տողային ձևաչափում 5 թվանշանը կարող է ունենալ զրոների և մեկերի արժեքների տարբեր հավաքածուներ: Իրավիճակը նույնն է նաև այլ թվանշանների համար:

	Թվան-շան	Սենսորներից եկած ազդա-նշաններ	R-տարրի արժեք
1	5	111+100+111+000+111	111100111000111
2	5	111+100+010+001+111	111100010001111
3	5	111+100+011+001+111	111100011001111
4	5	110+100+111+001+111	110100111001111
5	5	110+100+111+001+011	110100111001011
6	5	111+100+101+001+111	111100101001111

Նկ. 2.21 5 թվանշանի պատկերների տարբերակներ տողային ֆորմատով

Այժմ գրենք 5 թվանշանը՝ պատկերելով վեց հնարավոր տարբերակները սիմվոլային տողի տեսքով (տողերի տեսքը վերցնում ենք նկ.2.21-ից): Ծրագրում դրանք կներկայացվեն հետևյալ կերպ

```
num51 = list('111100111000111')
num52 = list('111100010001111')
num53 = list('111100011001111')
num54 = list('110100111001111')
num55 = list('110100111001011')
num56 = list('111100101001111')
```

Ծրագրի թեսատավորումը աղավաղված պատկերների վրա կիրականացնենք հետևյալ կերպ: Ծրագիրը թողարկենք տարբեր թվով ուսուցման ցիկլերով (10, 100, 1000, 10000, 100000) և ստանանք բոլոր սենսորային տարրերի կշիռները: Դրանից հետո ծրագիրը 5 թվանշանը պետք է համեմատի

թեստում այդ թվի բոլոր վեց գրվածքներով նմուշների հետ: Ստուգենք մշակված ծրագրային մոդուլի աշխատանքը:

```
# Կատարենք թեստավորում
print ( "ճանաչե՞լ եք 5-ը 51-ում", perceptron (num51))
print("ճանաչե՞լ եք 5-ը 52-ում", perceptron(num52))
print ("ճանաչե՞լ եք 5-ը 53-ում", perceptron (num53))
print ( "ճանաչե՞լ եք 5-ը 54-ում", perceptron (num54))
print("ճանաչե՞լ եք 5-ը 55-ում", perceptron(num55))
print ("ճանաչե՞լ եք 5-ը 56-ում", perceptron (num56))
```

Թեստավորման արդյունքները բերված են նկ. 2.22-ում:

5 թվանշանի տարբերակ-ներ	Ուսուցման ցիկլեր				
	10	100	1000	10 000	100 000
51	Ճանաչե՞լ եք 5-ը 51-ում False	Ճանաչե՞լ եք 5-ը 51-ում False	Ճանաչե՞լ եք 5-ը 51-ում False	**Ճանաչե՞լ եք 5-ը 51-ում True**	**Ճանաչե՞լ եք 5-ը 51-ում True**
52	Ճանաչե՞լ եք 5-ը 52-ում False	Ճանաչե՞լ եք 5-ը 52-ում False	**Ճանաչե՞լ եք 5-ը 52-ում True**	**Ճանաչե՞լ եք 5-ը 52-ում True**	**Ճանաչե՞լ եք 5-ը 52-ում True**
53	Ճանաչե՞լ եք 5-ը 53-ում False	Ճանաչե՞լ եք 5-ը 53-ում False	**Ճանաչե՞լ եք 5-ը 53-ում True**	**Ճանաչե՞լ եք 5-ը 53-ում True**	**Ճանաչե՞լ եք 5-ը 53-ում True**
54	Ճանաչե՞լ եք 5-ը 54-ում False	Ճանաչե՞լ եք 5-ը 54-ում False	**Ճանաչե՞լ եք 5-ը 54-ում True**	**Ճանաչե՞լ եք 5-ը 54-ում True**	**Ճանաչե՞լ եք 5-ը 54-ում True**
55	Ճանաչե՞լ եք 5-ը 55-ում False	Ճանաչե՞լ եք 5-ը 55-ում False	Ճանաչե՞լ եք 5-ը 55-ում False	**Ճանաչե՞լ եք 5-ը 55-ում True**	**Ճանաչե՞լ եք 5-ը 55-ում True**
56	Ճանաչե՞լ եք 5-ը 56-ում False	Ճանաչե՞լ եք 5-ը 56-ում False	Ճանաչե՞լ եք 5-ը 56-ում False	**Ճանաչե՞լ եք 5-ը 56-ում True**	**Ճանաչե՞լ եք 5-ը 56-ում True**

Նկ.2.22 Ստուգման աշխատանքի արդյունքները («Դա 5՞ է» հարցին)

Ինչպես երևում է աղյուսակից, պերցեպտրոնը 10 դասից հետո չկարողացավ հաղթահարել առաջադրանքը (չկարողացավ ճանաչել ոչ մի 5)։ 100 դասից հետո երեք սխալ թույլ տվեց, 1000 դասից հետո՝ երկու սխալ; Բայց 10000 ուսուցման ցիկլերից հետո այն կարողացավ անսխալ լուծել առաջադրանքը։

Այսպիսով, համոզվեցինք, որ նույնիսկ մեկ թաքնված շերտով պերցեպտրոնում, որը, ըստ էության, իրականում մեկ արհեստական նեյրոն է, կարողացանք հասնել ընդհանրացման։ Արհեստական նեյրոնը երբեք չէր տեսել 5 թվանշանի աղավաղված պատկերները, բայց կարողացավ դրանք ճանաչել։

Համոզվեցինք, որ ուսուցման ցիկլերի աճի հետ նեյրոնը դառնում է ավելի ուսուցանված և ի վերջո դադարում է սխալվել։

Դիտարկեցինք ցանցի շատ պարզեցված տարբերակ, որի կապերն ու մուտքերը կարող են լինել միայն ամբողջ թվեր։ Մնացած դեպքերը, երբ այդ մեծությունները կարող են լինել կամայական թվեր, կդիտարկենք հաջորդ բաժնում։

2.7 Դելտա կանոն

Այժմ փորձենք ընդլայնել Հեբբի կանոնները մուտքային և նեյրոնային ցանցի տարրերի միջև կապերի կամայական արժեքների (ոչ միայն ամբողջ) համար։ Ենթադրենք՝ գիտենք ցանցի ճիշտ ելքը (d) և ցանցի տված փաստացի պատասխանը (y)։ Եթե ուսուցման գործընթացում ցանցը սխալվել է, կհաշվենք սխալի արժեքը (ցանցի սխալանքը)։ Ցանցի սխալանքը (δ) կարող է որոշվել որպես տարբերություն ճիշտ ելքի և փաստացի պատասխանի։

$$\delta = d - y$$

Գիտենք, որ ազդանշանների փոխանցման մեջ որոշիչ դեր են խաղում կապերի կշիռները: Նշանակում է, դրանք պետք է ինչ-որ կերպ փոփոխվեն: Կապերի փոփոխման դասական ալգորիթմ է համարվում դելտա կանոնը կամ պերցեպտրոնի ուսուցման ալգորիթմը:

Դելտա կանոնը պերցեպտրոնի ուսուցման մեթոդ է, որը հիմնված է սխալի գրադիենտային իջեցման սկզբունքի վրա: Դրա հետագա զարգացումը հանգեցրեց սխալանքի ետ տարածման մեթոդի ստեղծմանը:

Դելտա կանոնը արտահայտվում է հետևյալ բանաձևով.

$$w_i(t+1) = w_i(t) + \delta \cdot x_i \cdot \eta,$$

որտեղ w_i-ն i կապի կշիռն է, t-ն՝ ուսուցման քայլը (դասի համարը), δ- ն՝ սխալի մեծությունը, x_i-ն՝ i-րդ սենսորի մուտքային ազդանշանի արժեքը, η-ն՝ համաչափության գործակիցը (ուսուցման արագությունը, նորմը):

Ըստ այդ բանաձևի, պերցեպտրոնի ուսուցման գործընթացի (t+1)-րդ քայլում հաշվվում է i-րդ կապի կշռի նոր արժեքը:

δ-ն նեյրոնային ցանցի սխալանքի արժեքն է: Հեբի կանոնի համաձայն, եթե նեյրոնային ցանցը տվել է ճիշտ արդյունք, ապա δ = 0 և կապերի կշիռները պետք չէ փոխել:

Երբ δ > 0, իսկ դա կլինի d > y դեպքում (այսինքն՝ ճիշտ պատասխանը ավելի մեծ է, քան ցանցի տվածը), այս դեպքում կշիռի ավելացման արժեքը դրական կլինի: Կապի կշիռը պետք է մեծացվի (Հեբի առաջին կանոն): Սա համապատասխանում է այն դեպքին, երբ ցանցը ստացել է 5 թիվը որպես մուտք, բայց չճանաչեց այն:

Եթե սխալի արժեքը բացասական է δ < 0, ապա d < y (այսինքն՝ ճիշտ պատասխանը փոքր է ցանցի տվածից), այս

դեպքում անհրաժեշտ է փոքրացնել կապի կշիռը (Հեբի երկրորդ կանոն)։ Սա համապատասխանում է այն դեպքին, երբ ցանցն իրեն տրված 5-ից տարբեր թվանշանը ճանաչել է որպես 5։

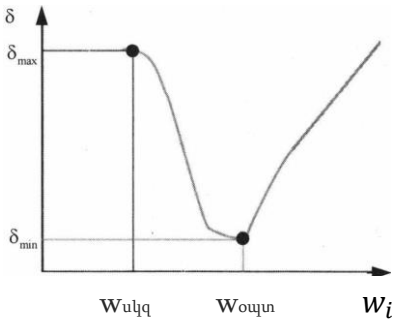
x_i-ն ցանցի i-րդ մուտքին սենսորից տրվող ազդանշանի արժեքն է։ Որքան ուժեղ ազդանշան է տրվել մուտքին, այնքան ավելի շատ է փոխվում այդ մուտքի հետ կապված կապի կշիռը։ Եթե մուտքն ընդհանրապես ազդանշան չի ստացել (x_i=0), ապա համապատասխան կշիռը չպետք է փոխվի։

η պարամետրը բնութագրում է ուսուցման արագությունը (ուսուցման արագության գործակից)։ Ցանցի ճիշտ աշխատանքը կախված է ճիշտ ընտրված կապի կշիռներից։ Դա նշանակում է, որ սխալանքի մեծությունը կախված է կշիռներից։ Որքան շատ է ցանցի ստեղծած պատասխանը շեղվում ճիշտ պատասխանից, այնքան մեծ է սխալանքը։ Գրաֆիկի տեսքով ներկայացնենք ցանցի δ սխալանքի կախվածությունը w_i կապի կշռի արժեքից (նկ. 2.23.)։

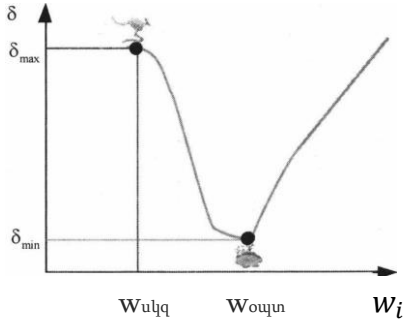
Ինչպես երևում է նկարից, որոշ $w_{սկզ}$ սկզբնական արժեքի համար ունենք ցանցի աշխատանքի δ_{max} առավելագույն սխալ։ Պետք է գտնել կշռի այնպիսի օպտիմալ $w_{օպտ}$ արժեք, որի դեպքում սխալի արժեքը լինի նվազագույն δ_{min}։ Այդ դեպքում պետք է որոշ քայլով մեծացնել i–րդ կշռի $w_{սկզ}$ արժեքը։ Այդ քայլի մեծությունը որոշվում համաչափության գործակցով։ Ուսուցման արագությունը և ընդհանրապես կշռի $w_{օպտ}$ օպտիմալ արժեքը որոշելը կախված է դրա արժեքից։

Գրադիենտային իջեցման սկզբունքը բացատրենք օրինակով, թե ինչպես է կենգուրուն բլուրից իջնում դեպի ջրի փոսը՝ ջուր խմելու։ Ենթադրենք, որ կենգուրուն գտնվում է բլրի վրա, իսկ ամենաներքևում ջրի փոսն է (նկ. 2.24)։

Կենգուրուի գտնվելու դիրքը կապենք որոշակի կշռի արժեքի հետ, որն ունի նեյրոնային ցանցը վարժեցնելուց առաջ: Կենգուրուն պետք է իջնի բլուրից մինչև ամենաներքևը, քանի որ այդ կետում է գտնվում ջրի փոսը (այդ դիրքում ցանցի սխալը նվազագույնն է): Նշենք, որ կենգուրուն կարող է միայն ցատկել: Ցատկը կարող է լինել կարճ կամ երկար: Եվ հենց «ցատկի երկարության» հետ է կապվում η գործակիցը կշռի փոփոխման բանաձևում:



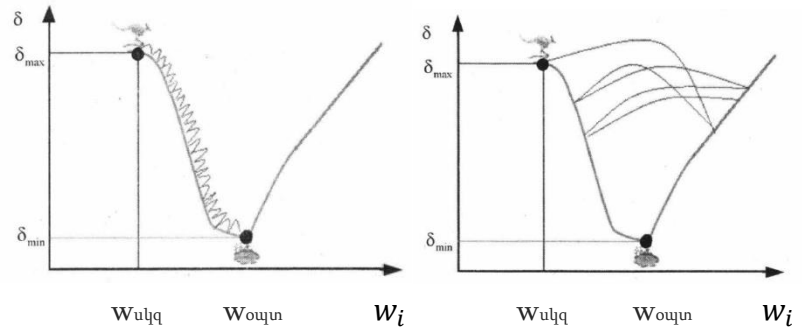
Նկ. 2.23 δ-ի կախվածությունը w_i ից



Նկ. 2.24 Կենգուրուի սկզբնական դիրքը

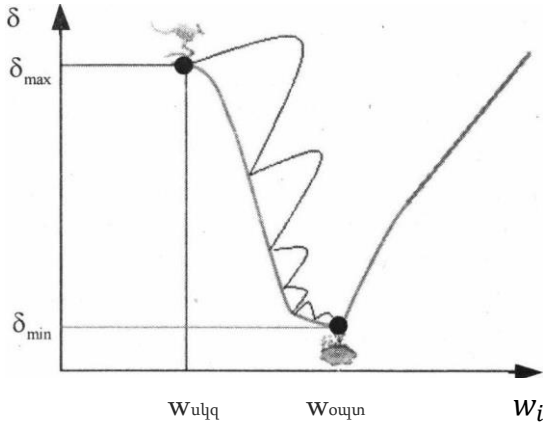
Ենթադրենք, η գործակցի արժեքը շատ փոքր է, այսինքն՝ կենգուրուն առաջ կգնա դեպի ջուրը շատ փոքր թռիչքներով, և այդ դեպքում ճանապարհը շատ երկար կլինի:

Երբ η գործակցի արժեքը դարձնենք շատ մեծ, ապա կհամարենք, որ կենգուրուն մեծ թռիչքներով ավելի արագ կհասնի ներքևում գտնվող ջրին: Այնուամենայնիվ դա այդպես չէ: Թռիչքի մեծ արժեքով (ուսուցման մեծ արագություն) գոյություն ունի հենց ներքև չհասնելու վտանգ, քանի որ կենգուրուն անընդհատ ցատկելու է աջ ու ձախ՝ գործնականում մնալով նույն բարձրության վրա (նկ. 2.25):



Նկ. 2.25 η գործակցի մեծության ազդեցությունը նեյրոնային ցանցի սխալները նվազագույնի հասցնելու գործընթացում

Հետևաբար, կենգուրուն պետք է ռացիոնալ կերպով ցատկի դեպի ջուրը՝ նախ մեծ երկարությամբ ցատկերով, և երբ մոտենում է ջրին, կրճատի ցատկի երկարությունը: Այսինքն՝ անհրաժեշտ է օգտագործել որոշակի ֆունկցիա, որը նվազեցնի η պարամետրի արժեքը՝ մոտենալով սխալանքի նվազագույն արժեքին (նկ. 2.26):



Նկ. 2.26 η գործակցի արժեքի փոփոխությունները նեյրոնային ցանցի սխալները նվազագույնի հասցնելու գործընթացում

Այսպիսով, պարզեցինք ուսուցման արագության գործակցի նպատակը: Կապի կշռի արժեքը կարող ենք փոխել η գործակցի միջոցով:

2.8 Գծային մոտարկում

Այժմ դիտարկենք մեկ այլ ոչ ստանդարտ խնդիր, որը վերաբերում է ոչ թե օբյեկտների դասակարգմանը (ինչպես թվանշանների ճանաչման օրինակում), այլ մոտարկմանը: Այսինքն՝ տրված հավաքածուի որոշակի արժեքների դեպքում պետք է կատարել որոշ արժեքների որոնում: Այստեղ նորից օգտակար է դելտա կանոնը. այն կիրառենք օրինակի վրա:

Գործնականում հաճախ ստիպված ենք լինում բախվել այնպիսի խնդրի, ինչպիսին է հանցագործի կոշիկի թողած հետքից նրա հասակի որոշումը: Եթե իմանանք հասակի կախվածությունը կոշիկի հետքի երկարությունից, ապա, ունենալով հանցագործի կոշիկի հետքը, մոտավորապես կարող ենք գուշակել նրա հասակը: Վերցնենք հասակի տարբեր չափս ունեցող 10 հոգու (տղամարդկանց), որոշենք նրանց կոշիկների թողած հետքերի երկարությունների արժեքները և մուտքագրեք աղյուսակում (նկ. 2.27.):

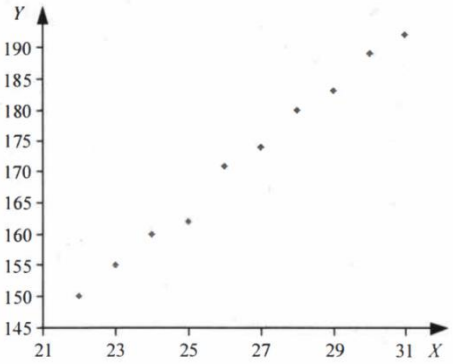
x կոշիկի հետքի երկարություն, սմ	y հասակ, սմ
22	150
23	155
24	160
25	162
26	171
27	174
28	180

29	183
30	189
31	192

Նկ.2.27 Տղամարդու հասակի կախվածությունը կոշիկի հետքի երկարությունից

Ինչո՞ւ ընտրանքում վերցրինք միայն տղամարդկանց, իսկ կանանց չներառեցինք: Որովհետև կանանց հասակի և կոշիկի չափսի միջև հարաբերակցությունը այլ է:

Այս տվյալները ներկայացնենք գրաֆիկի տեսքով (նկ. 2.28):



Նկ.2.28 Տղամարդու հասակի կախվածությունը կոշիկի հետքի երկարությունից

Տեսնում ենք, որ ստացված տվյալների դասավորվածությունը ուղիղ գիծ է հիշեցնում:

Մեր խնդիրն է հնարավորինս ճշգրիտ ուղիղ գիծ ստանալ, որը կներառի տրված արժեքների բազմությունը: Սա մոտավորության սկզբունքն է, երբ, չունենալով Y կոորդինատի արժեքը որևէ X կետում, հնարավոր է կանխատեսել Y-ի արժեքը այդ կետում: Օրինակ, մենք չունենք X=23.5 կետի Y կոորդինատի

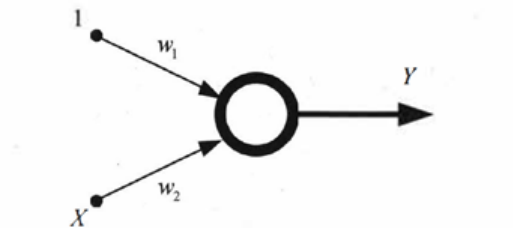
մասին տվյալներ: Պետք է կանխատեսել Y-ի արժեքը այդ կետում: Այսինքն՝ որոշել՝ ինչ հասակ ունի նա, ով թողել է 23.5 սմ չափսի հետք:

Այս դասի խնդիրները լուծելու համար սովորաբար օգտագործվում են նվազագույն քառակուսիների մեթոդը: Այստեղ կարևոր է խնդիրն իրականացնել նեյրոնային ցանցի միջոցով, որը պետք է որոշի այն գծի հավասարումը, որի միջոցով հնարավորինս լավ մոտարկվում են տրված տվյալնները (անձի հասակի կախվածությունը կոշիկի չափսից): Հիշեցնենք, որ ուղիղ գծի հավասարումն է

$$Y = k \cdot X + C,$$

որտեղ k գործակիցը որոշում է գծի թեքությունը, C-ն ցույց է տալիս y առանցքի այն կետը, որով անցնում է ուղիղ գիծը:

Որոշենք ցանցի մուտքային և ելքային պարամետրերը: Մուտքային պարամետրերից մեկը X փոփոխականն է: Քանի որ ուղիղ գծի հավասարման մեջ կա C հաստատուն արժեքը, ապա ցանցին կավելացնենք երկրորդ մուտքը (պարամետրը), որը մեր դեպքում հավասար կլինի 1-ի: Այս դեպքում ակտիվացման ֆունկցիա չունենք: Կշռված գումարը կլինի ցանցի Y արդյունքը: Կառուցված պարզ պերցեպտրոնը ներկայացված է նկ. 2.29-ում.



Նկ. 2.29 Պարզ պերցեպտրոնի տեսքը

Պերցեպտրոնն այս դեպքում արհեստական նեյրոնն է, որի ելքը որոշվում է $Y = w_2X + w_1$ կշռված գումարով: Սա հենց Y=kX+C ուղիղ գծի հավասարումն է, որտեղ $w_2=k$, $w_1=C$:

Նշենք, որ պերցեպտրոնի կառուցվածքը այնպիսին է, որ ուսուցման ընթացքում նրա կշռները դառնում են փնտրվող ուղղի գործակիցները:

Գրենք պերցեպտրոնի ծրագիրը, որը որոշի մարդու հասակի կախվածությունը նրա կոշիկի չափսից: Մուտքային տվյալները մարդկանց կոշիկների չափսերն են, իսկ հասակները պերցեպտրոնի ելքային սպասվող արժեքներն են: Կապերի կշիռները սկզբնարժեքավորենք [-5, 5] միջակայքից պատահական իրական թվերով:

```
import random
k=random.uniform(-5, 5)
C=random.uniform(-5, 5)
```

Երկրորդ մուտքի w_1 կշիռը ուղղի հավասարման C ազատ անդամն է: Քանի որ երկու պարամետրերն էլ որոշվել են պատահականորեն, ուրեմն ունենք պատահական ուղիղ գիծ, որի դիրքը կարևոր չէ նախքան պերցեպտրոնի ուսուցումը:

Արտածենք ստեղծված փոփոխականները, որ տեսնենք, թե k-ի և C-ի ինչ արժեքներից ենք սկսել ուսուցանել նեյրոնը:

```
print ('Սկզբնական ուղիղ գիծ ', k, ' * X + ' ,C)
```

Որոշենք X:Y տվյալների հավաքածուն որպես բառարան (dictionary):

```
data = {22: 150, 23: 155, 24: 160, 25: 162, 26: 171, 27: 174, 28:180, 29: 183, 30: 189, 31: 192 }
```

Այս բառարանում X-ը բանալին է (կոշիկի չափս), Y-ը արժեքն է (հասակ):

Սահմանենք ուսուցման արագության գործակիցը: Դա արվում է փորձարկման միջոցով: Սկզբում այս խնդրի համար ուսուցման արագությունը տրվել էր 0.1, բայց այդ դեպքում ցանցը չսովորեց: Քայլը մեծացնելիս ցանցի սխալը միայն ավելանում էր: Այս դեպքը համապատասխանում է նկ. 2.25-ում կենգուրուի ցատկի երկարության չափի մեծ լինելուն: Ուսուցման արագության գործակցի մինչև 0.0001 արժեքը փոքրացնելուց հետո պերցեպտրոնը սկսեց սովորել, այսինքն՝ սխալանքը սկսեց նվազել և կայունացավ:

Պերցեպտրոնի ելքային արժեքի ձևավորման ֆունկցիան հետևյալն է.

```
# y-ի հաշվում
def proceed(x):
  return x*k+C
```

Ինչպես տեսնում եք, այստեղ ակտիվացման ֆունկցիա չկա: Ցանցի ելքը հաշվարկված գումար է, որտեղ k-ն X մուտքի կապի կշիռն է, իսկ C-ն՝ միշտ 1-ի հավասար մուտքի կապի կշիռն է: Ցանցը ուսուցանենք օրինակ 2.6-ում: Նպատակահարմար է կատարել տարբեր փորձարկումներ: Ուսուցման համար վերցնենք, օրինակ, 100000 քայլ:

Մեր բառարանից 100000 անգամ ընտրում ենք պատահական բանալի (X-ի արժեք):

Դրանից հետո ստեղծվում է true_result փոփոխականը, որտեղ պահվում է X-ին համապատասխան ճիշտ Y-ը: out փոփոխականում ստացվում է ցանցի ելքը տրված X-ի համար: delta կանոնի միջոցով հաշվարկվում է սխալանքը, փոփոխ-

վում են երկու կապի կշիռները: Վերջնական ուղիղ գծի համար ստացվում են հետևյալ տվյալները.

```
print(f"կառուցված ուղիղ գիծ: y = {k} * x + {C}")
```

Օրինակ 2.6-ում բերված է ծրագրի ամբողջական կոդը:

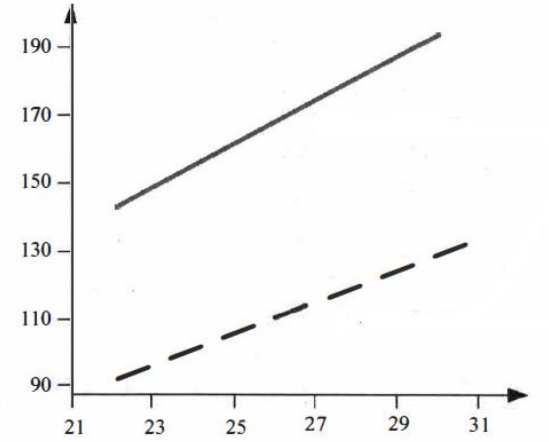
Օրինակ 2.6

```
import random
k = random.uniform(-5,5) # x-ի գործակից
C = random.uniform(-5,5) # Ուղղի հավասարման ազատ անդամ
print(f"Սկզբնական ուղիղ գիծ: y = {k} * x + {C}") # Սկզբնական ուղղի
                                                  # տվյալների արտածում
rate = 0.0001  # Ուսուցման արագություն
data = { 22: 150,
         23: 155,
         24: 160,
         25: 162,
         26: 171,
         27: 174,
         28: 180,
         29: 183,
         30: 189,
         31: 192 }
# y-ի ուսուցում
def proceed(x):
  return x*k+C

# Ցանցի ուսուցում
for i in range(100000):
  x = random.choice(list(data.keys())) # Պատահականորեն ստանալ կետի x կոորդինատը
  true_result = data[x] # Համապատասխանաբար ստանալ կետի y կոորդինատը
  out = proceed(x)  # Ստանալ ցանցից պատասխան
  delta = true_result - out # Ցանցի սխալի հաշվում
  k += delta*rate*x # Դելտա կանոնով փոխում ենք x-ի կշիռը
  C += delta*rate  #  Դելտա կանոնով փոխում ենք հաստատուն մուտքի կշիռը
print(f"կառուցված ուղիղ գիծ: y = {k} * x + {C}")
```

Ծրագրի գործարկումից հետո ստացվում են հետևյալ արդյունքները.

```
Սկզբնական ուղիղ գիծ: y = 3.9 * x + 4.7
Կառուցված ուղիղ գիծ: y = 6.074695039633236 * x + 9.185469221896618
```

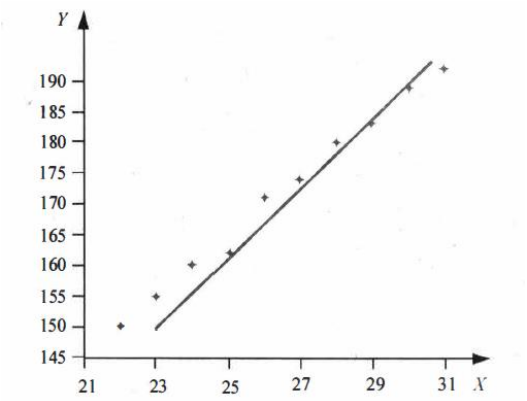
Գրաֆիկի վրա պատկերենք երկու գիծ: Կետագծով պատկերենք ուսուցումից առաջ ունեցած գիծը, իսկ հոծ գծով՝ ուսուցումից հետո (նկ. 2.30).



Նկ.2.30 Ուղիղ գծերի հավասարումների գրաֆիկները
(պերցեպտրոնի ուսուցումից առաջ և հետո)

Նկարում կետագծով պատկերված է այն գիծը, որը պատահականորեն տրվեց ծրագրի սկզբում: Հոծ ուղիղ գիծը ստացվել է պերցեպտրոնի աշխատանքի արդյունքում: Դա տրված կետերի բազմության պերցեպտրոնի միջոցով գծային մոտարկումն է: Այդ ուղիղով որոշվում է հասակի չափսի կախվածությունը կոշիկի չափսից:

Հիմա համեմատենք հոծ ուղիղ գիծը մեր սկզբնական տվյալների հետ (նկ. 2.31):



Նկ. 2.31 Հասակի և կոշիկի չափսի միջև կախվածության ուղիղ գծի մոտարկում

Ուղիղ գիծը ստացվել է պերցեպտրոնի աշխատանքի արդյունքում. կարելի է ասել, որ դա կետերի տվյալ բազմության գծային մոտարկումն է: Հետևաբար, պերցեպտրոնները կարող են իրականացնել ոչ միայն դասակարգման խնդիրներ:

2.9 Դասակարգում պերցեպտրոնի միջոցով

Նախորդ բաժնում ծանոթացանք պերցեպտրոնի աշխատանքի սկզբունքին և դրա ծրագրային իրականացմանը: Հիմա պերցեպտրոնի միջոցով կատարենք օբյեկտների դասակարգում: Կօգտագործենք Python-ի NumPy, Pandas, Matplotlib գրադարանները: Կստեղծենք երկու մեթոդ՝ fit() մեթոդը, որը իրականացնում է ուսուցման գործընթացը ուսուցման տվյալների հավաքածուի վրա, և predict() մեթոդը, որը հնարավորություն կտա կանխատեսումներ անել այլ մուտքային տվյալների վրա: Օրինակ 2.7-ում պերցեպտրոնի իրականացման կոդն է Perceptron դասի տեսքով:

Օրինակ 2.7

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
# Պերցեպտրոն դասի նկարագրություն
class Perceptron(object):
    def __init__(self, eta=0.01, n_iter=10):
      self.eta = eta # Ուսուցման արագություն (0-ից 1)
      self.n_iter = n_iter # Իտերացիաների քանակ (դասեր)
#Մոդելը համապատասխանեցնել վերապատրաստման տվյալներին,
# X - ուսուցման տվյալներ՝զանգված,
# չափս՝ X[n_samples, n_feature],n_samples - նմուշների քանակը,
#n _feature - հատկանիշների քանակը,
#y - թիրախային արժեքներ՝ զանգված, չափս՝ y[n_samples]
    def fit(self, X, y):
      self.w_ = np.zeros(1 + X.shape[1]) # w : միաչափ զանգված-
                                         #կշիռները ուսուցումից հետո
      self.errors_ = [] # սխալանք : սխալների ցուցակի դասակարգում
                        #յուրաքանչյուր էպոխում
      for _ in range(self.n_iter):
        errors = 0
        for xi, target in zip(X, y):
            update = self.eta * (target - self.predict(xi))
            self.w_[1:] += update * xi
            self.w_[0] += update
            errors += int(update != 0.0)

        self.errors_.append(errors)
```

```
      return self


    def net_input(self, X): #
          return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X): # Վերադարձնել դասի պիտակը
          return np.where(self.net_input(X) >= 0.0, 1, -1)
```

Պերցեպտրոնային մոդելի մուտքային տվյալները X[n_samples, n_features] երկչափ զանգվածն է, որտեղ n_samples-ը ուսուցման տվյալների քանակն է, n_features-ը՝ օբյեկտի հայտանիշների քանակը: Ուսուցման հավաքածուի մեջ պիտակները ներկայացված են y[n_samples] միաչափ զանգվածով: Ծրագրում տրվում են ուսուցման eta արագությունը, n_i_ter

էպոխների (դասերի, ցիկլերի) քանակը, ուսուցման տվյալների հավաքածուն: fit() ուսուցման մեթոդը ձևավորում է self.w կշիռները:

Այսպիսով, ունենք պերցեպտրոն: Այն կօգտագործենք հիրիկ ծաղկի տեսակը որոշելու համար՝ օգտվելով նրա ծաղկաթերթիկի երկարության և լայնության չափսերից:

Պերցեպտրոնը ուսուցանելու համար ուսուցման տվյալների հավաքածուի համար վերցնենք հիրիկ ծաղկի երկու տեսակ (setosa/սրածայր, versicolor/բազմագույն): Դրանք ուսուցման նմուշի հայտանիշներն են: Կօգտագործենք տվյալների պատրաստի հավաքածու, որը կբեռնենք DataFrame օբյեկտի մեջ և կստուգենք նրա ճշտությունը: Օրինակ 2.8 -ում ծրագրի կոդի այդ հատվածն է:

Օրինակ 2.8

```
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
df = pd.read_csv(url, header=None)
print('Տվյալներ իրիս ծաղկի մասին')
print(df.to_string())
data = df.to_csv('C:\CSV\Iris.csv')
```

url փոփոխականում տվյալների ինտերնետային հղումն է: Դրանք ներբեռնելուց հետո տվյալները պահվում են DataFrame տիպի df օբյեկտում, արտածվում և պահվում համակարգչում՝ lris.csv ֆայլում: Ծրագրի կոդի արդյունքները բերված են 2.32 նկարի աղյուսակում:

	0	1	2	3	4
	Բաժակաթերթիկ		Ծաղկաթերթիկ		Հիրիկի ծաղկի տեսակը
	Երկարություն, սմ	Լայնություն, սմ	Երկարություն, սմ	Լայնություն, սմ	
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
…	…	…	…	…	…
50	7.0	3.2	4.7	1.4	Iris-versicolor
51	6.4	3.2	4.5	1.5	Iris-versicolor
52	6.9	3.1	4.9	1.5	Iris-versicolor
53	5.5	2.3	4.0	1.3	Iris-versicolor
54	6.5	2.8	4.6	1.5	Iris-versicolor
55	5.7	2.8	4.5	1.3	Iris-versicolor
…	…	…	…	…	…
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

Նկ.2.32 Հիրիկ ծաղկի տարբեր տեսակների պարամետրեր

Առաջին չորս սյուներում այն պարամետրերի արժեքներ են, որոնք բնութագրում են հիրիկ ծաղկի այս կամ այն տեսակը: 2.32 նկարի աղյուսակի տվյալներով ձևավորենք ուսուցման հաջորդականություն: Առաջին և երրորդ սյուների արժեքները վերագրենք հատկանիշների մատրիցին (x երկչափ զանգված): Հինգերրորդ սյան արժեքները վերագրենք y վեկտորին: Այս քայլերը կարող են իրականացվել՝ օգտագործելով օրինակ 2.9-ի կոդը:

Օրինակ 2.9

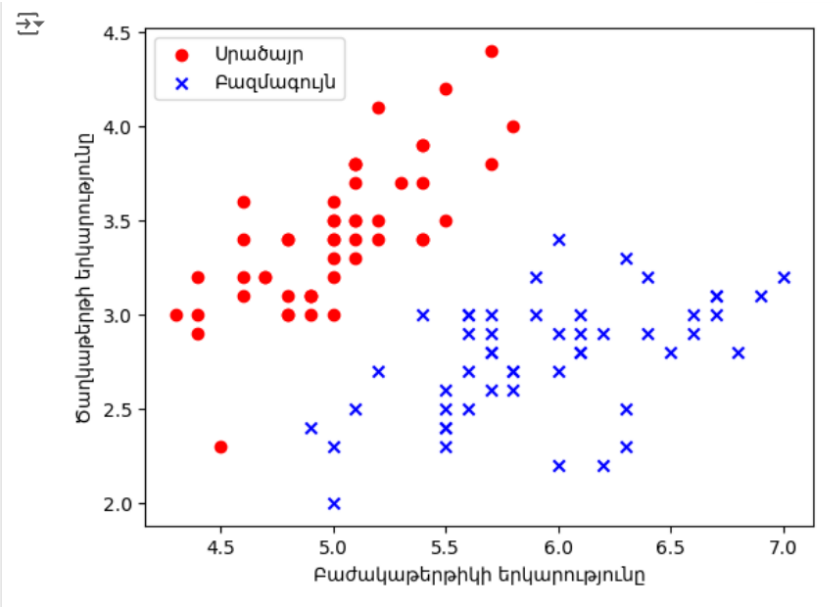
```
#DF-ից վերցնել 100 տող(0 և 2 սյունակները), բեռնել դրանք X զանգվածում
X = df.iloc[0:100,[0, 2]].values
print('X արժեքները - 100')
# print(X)

# DF-ից վերցնել100 (ծաղիկների անվանումը 4-րդ սյուն) և բեռնել Y զանգվածում
y = df.iloc[0:100,4].values

# Ծաղիկների անունները (սյունակ 4) փոխակերպել -1 և 1 թվերի զանգվածի
y = np.where(y == 'Iris-setosa',-1, 1)
print('Ծաղիկների անունների արժեքները -1 և 1 տեսքով, Y - 100')
# print(y)
```

Օգտագործելով հետևյալ ծրագրային կոդը՝ ստացված տվյալները ներկայացնենք գրաֆիկորեն:

Օրինակ 2.10

```
# Ուսուցման հավաքածուի առաջին 50 տարրերը (տողեր 0-50, սյունակներ 0,1)
plt.scatter(X[0:50, 0], X[0:50, 1], color='red', marker='o', label='Սրածայր')
# Ուսուցման հավաքածուի հաջորդ 50 տարրերը (տողեր 50-100, սյունակներ 0,1)
plt.scatter(X[50:100, 0], X[50:100, 1], color='blue', marker='x', label='Բազմագույն')

# Առանցքների անունների ստեղծում և գրաֆիկի ցուցադրում
plt.xlabel('Բաժակաթերթիկի երկարությունը')
plt.ylabel('Ծաղկաթերթիկի երկարությունը')
plt.legend(loc='upper left')
plt.show()
```

Այս ծրագրի կոդի արդյունքը ներկայացված է նկ. 2.33-ում:



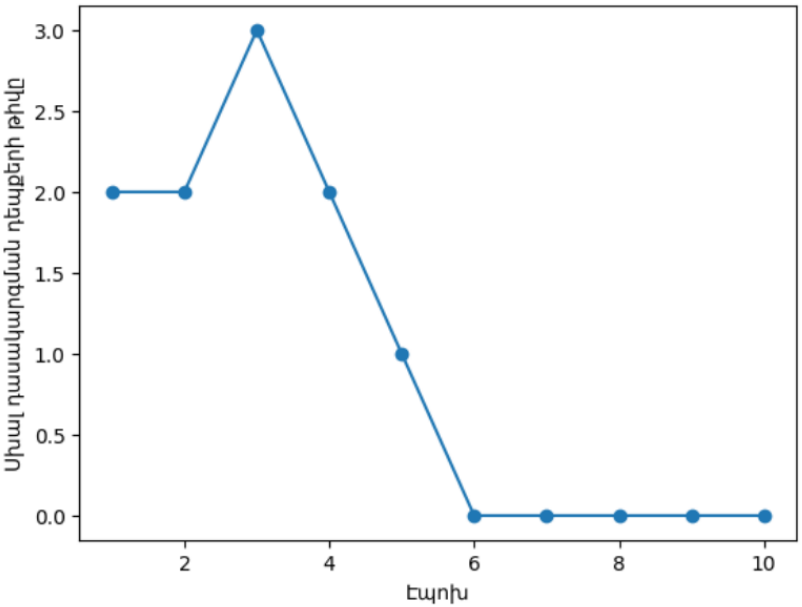
Նկ. 2.33 Ծրագրային կոդի աշխատանքի արդյունքը

Ինչպես երևում է նկ. 2.33-ում, այդ երկու հայտանիշները բավականին լավ խմբավորում են ծաղիկները ըստ իրենց տեսակի:

Այժմ այդ տվյալների վրա ուսուցանենք պերցեպտրոնը և կառուցենք սխալների փոփոխման գրաֆիկը հետևյալ ծրագրի կոդով (Օրինակ 2.11).

Օրինակ 2.11

```
# Ուսուցում (j,extybt)
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Էպոխ')
plt.ylabel('Սխալ դասակարգման դեպքերի թիվը')
plt.show()
```



Նկ.2.34 Սխալ դասակարգման դեպքերի քանակի կախվածությունը ուսուցման ցիկլերի քանակից

Ինչպես երևում է նկ. 2.34-ից, պերցեպտրոնը կայունացել է ուսուցման վեցերորդ ցիկլից (epoch) հետո, և արդեն կարելի է դրանով դասակարգել նոր նմուշներ:

Սա ստուգենք հետևյալ կոդով.

Օրինակ 2.12

```
il=[5.5, 1.6]
i2=[6.4, 4.5]
R1 = ppn.predict(il)
R2 = ppn .predict(i2)
print('R1=', R1,' R2=', R2)
```

```
R1= -1  R2= 1
```

Այստեղ il և i2 փորձարկվող նմուշների համար R1 և R2-ի ստացված արժեքները ճիշտ պատասխան են:

```
if (Rl==1):
    print('Rl= Iris setosa տեսակ')
else:
    print('Rl= Iris versicolor տեսակ')
```

```
Rl= Iris versicolor տեսակ
```

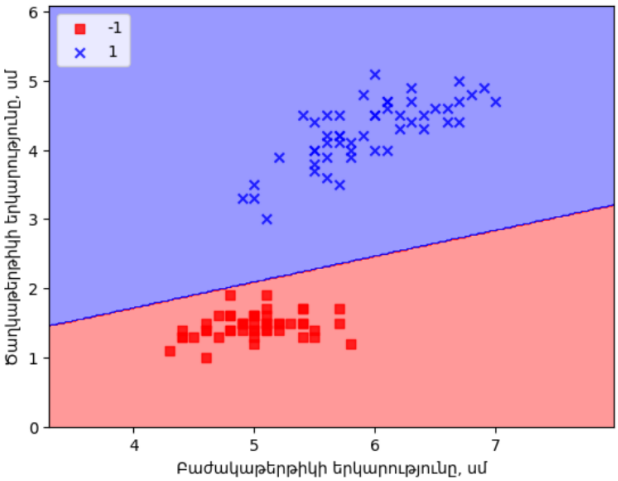
Փաստորեն կառուցվեց պերցեպտրոն, այն ուսուցանվեց և փորձարկվեց:

Այժմ գտնենք ուսուցման հավաքածուի երկու խմբերը բաժանող ֆունկցիան, որի միջոցով կիրականացվի ծաղիկները երկու խմբերի բաժանման վիզուալիզացիան:

Օրինակ 2.13

```
# Բաժանարար սահմանի վիզուալիզացիա
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, resolution=0.02):
    # Կարգավորել մարկերների գեներատորը և գունապակը
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ( 'red', 'blue', 'green', 'gray', 'cyan')
    cmap= ListedColormap(colors[:len(np.unique(y))])

    # Սահմանել մակերեսը
    xl_min, xl_max = X[:, 0] .min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1] .min() - 1, X[:, 1].max() + 1

    xxl, xx2 = np.meshgrid(np.arange(xl_min, xl_max, resolution),
    np.arange(x2_min, x2_max, resolution))

    Z = classifier.predict(np.array([xxl.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xxl.shape)

    plt.contourf(xxl, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim (xxl.min (), xxl.max ())
    plt.ylim(xx2.min(), xx2.max())
    # Ցույց տալ դասերի նմուշները
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8,
            c=cmap(idx), marker=markers[idx], label=cl)

# Նկարել պատկերը
plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('Բաժակաթերթիկի երկարությունը, սմ')
plt.ylabel('Ծաղկաթերթիկի երկարությունը, սմ')
plt.legend(loc='upper left')
plt.show()
```

Այս ծրագրի կոդը գործարկելուց հետո ծաղիկների երկու տեսակները կբաժանվեն երկու խմբերի (նկ. 2.35):



Նկ. 2.35 Ծաղիկների բաժանումը խմբերի

Նկ. 2.35-ից երևում է, որ պերցեպտրոնը ուսուցման գործընթացում գտել է այն ուղիղը, որը առանձնացնում է ծաղիկների ուսուցման հավաքածուի բոլոր նմուշները: Դա իդեալական դեպք է: Սակայն գործնականում նման արդյունքի միշտ հնարավոր չէ հասնել: Տարբեր դասերի օբյեկտների միջև հստակ սահմաններ որոշելու ունակությունը պերցեպտրոնի համար դժվար խնդիրներից է:

Ֆրենկ Ռոզենբլատը մաթեմատիկորեն ապացուցել է, որ պերցեպտրոնի ուսուցման գործընթացը զուգամիտում է միայն այն դեպքում, եթե երկու դասերը կարող են բաժանվել գծային հիպերհարթությամբ: Սակայն, եթե անհնար է ամբողջությամբ առանձնացնել դասերը նման գծային սահմանով, ապա ուսուցումը երբեք չի ավարտվի:

Այս գլխում ցույց տրվեց, որ պերցեպտրոնը արհեստական նեյրոնային ցանցի պարզ տեսակ է: Նկարագրվեց պերցեպտրոնի ուսուցման և փորձարկման գործընթացները:

ԳԼՈՒԽ 3
Python միջավայրում նեյրոնային ցանցերի ստեղծման հիմնական գրադարանները

Python ծրագրավորման լեզուն իր գրադարաններով աջակցում է ծրագրավորողներին նեյրոնային ցանցեր ստեղծելու և կիրառելու գործընթացներում:

Արհեստական բանականության և մեքենայական ուսուցման հետ կապված նախագծեր մշակելու համար Python-ում կան բազմաթիվ հնարավորություններ: Դրանցից են ներկառուցված գրադարանները, ինտեգրման և նախատիպեր ստեղծելու հարմարավետությունը և այլն:

Python ծրագրավորման լեզուն իր պարզ շարահյուսության և ընթեռնելիության շնորհիվ կիրառելի է դարձել շատերի համար: Նրա գրադարանները արհեստական բանականության և մեքենայական ուսուցման համար զգալիորեն պարզեցնում և արագացնում են նախագծերի մշակման և փորձարկման գործընթացները:

Արհեստական բանականության և մեքենայական ուսուցման խնդիրների լուծման համար հիմնականում օգտագործվում են Python-ի հետևյալ գրադարանները.

- scikit-learn-տվյալների հետազոտման, մոդելի ուսուցման և մոդելի որակի գնահատման համար,
- Keras, TensorFlow-նեյրոնային ցանցերի կառուցման, օբյեկտների դասակարգման համար,
- Theano - օգտագործում է կենտրոնական (CPU) և գրաֆիկական (GPU) պրոցեսորներ՝ ծրագրերի մշակման արդյունավետությունը բարձրացնելու նպատակով: Այն ներառում է կոդի օպտիմալացման և թեստավորման

ներդրված մեխանիզմներ և այլն: Theano-ն մաթեմատիկական բարձր ճշտության հաշվարկներ կատարելու շնորհիվ լայնորեն օգտագործվում է նեյրոնային բարդ կառուցվածքի ցանցերում և մեքենայական ուսուցման այլ ալգորիթմներում:

Օգտագործվում են նաև ընդհանուր նշանակության հետևյալ գրադարանները.

- NumPy-օգտագործվում է n-չափանի զանգվածների և մատրիցների հիմնական գործողությունները՝ գումարում, հանում, բաժանում, բազմապատկում կատարելու համար:
- Pandas-հիմնականում օգտագործվում է տվյալների մշակման համար:
- Matplotlib-օգտագործվում է տվյալների և դրանց վերլուծության արդյունքների վիզուալիացիայի համար: Այդ գրադարանը հնարավորություն է տալիս ստեղծելու գծային գրաֆիկներ, տվյալների ցրվածության դիագրամներ, հիստոգրամներ, շրջանաձև գծապատկերներ, սպեկտրոգրամներ և այլն:

3.1 scikit-learn գրադարանը նեյրոնային ցանցերի ստեղծման և ուսուցման համար

scikit-learn-ը մեքենայական ուսուցման հայտնի գրադարան է, որն աշխատում է Python-ի հետ և իրականացնում ուսուցչով և առանց ուսուցչի ուսուցման ալգորիթմներ: Այդ գրադարանը հնարավորություն է տալիս լուծելու տվյալների դասակարգման, ռեգրեսիայի, կլաստերավորման և այլ խնդիրներ:

scikit-learn-ը բաց կողով գրադարան է, այսինքն՝ այն կարելի է օգտագործել և տարածել: Դրա օգտագործման համար սովորաբար պահանջվում են նաև NumPy, SciPy և Matplotlib գրադարանները:

NumPy-ը Python-ում գիտական հաշվարկների հիմնական փաթեթներից մեկն է: Այն պարունակում է միաչափ և բազմաչափ զանգվածների հետ աշխատելու ֆունկցիաներ, գծային հանրահաշվի գործողություններ, վիճակագրական ֆունկցիաներ, պատահական թվերի գեներատոր և այլն:

scikit-learn-ը տվյալները ընդունում է NumPy զանգվածների տեսքով: NumPy-ի հիմնական օբյեկտը np.array-ն է, որը n-չափանի զանգված է: Զանգվածի բոլոր տարրերը պետք է լինեն նույն տիպի:

Ստեղծենք երկու տողով և երեք սյունով x մատրիցը:

```
import numpy as np
x = np.array( [[1, 2, 3], [4, 5, 6]])
print(x)
```

Ծրագրային կոդը կատարելուց հետո կարտածվի

```
[[1 2 3]
 [4 5 6]]
```

np.array դասի օբյեկտները կանվանենք NumPy զանգվածներ կամ պարզապես զանգվածներ:

SciPy գրադարանը ունի բազմաթիվ ֆունկցիաներ գիտական հաշվարկներ կատարելու համար: Այն պարունակում է գծային հանրահաշվի գործողություններ, ֆունկցիաների օպ-

տիմիզացիայի մեթոդներ, ազդանշանների մշակման, մաթեմատիկական և վիճակագրական ֆունկցիաներ։

Մեքենայական ուսուցման համար SciPy-ի կարևոր մաս է կազմում scipy.sparse փաթեթը, որի օգնությամբ ստանում ենք scikit-learn-ում օգտագործվող տվյալների նոր ձևաչափ՝ նոսր մատրիցներ (Sparse matrices)։

Sparse մատրիցները նրանք են, որոնց տարրերի մեծ մասը զրոներ են։ Հիշողության տնտեսման և հաշվարկների արդյունավետության բարձրացման նպատակով այս մատրիցները հատուկ ձևաչափերով են ներկայացվում, որտեղ պահվում են միայն ոչ զրոյական արժեքները։ Օրինակ, եթե ունենք մատրից, որտեղ 95%-ից ավել արժեքները զրոներ են, ապա խնայողության նպատակով պահում են միայն ոչ զրոյական արժեքները իրենց դիրքերի համարներով։

Նոսր մատրիցների համար հիմնականում կիրառվում են երկու ձևաչափեր՝ CSR (Compressed Sparse Row) և COO (Coordinate List)։

CSR ձևաչափի հիմնական բաղադրիչներն են՝

- Data-պահում է մատրիցի բոլոր ոչ զրոյական արժեքները,
- Indices-պահում է յուրաքանչյուր ոչ զրոյական արժեքի սյան ինդեքսը,
- Indptr-թվերի զանգված, որոնք ցույց են տալիս, թե Data զանգվածում որ դիրքում է գտնվում մատրիցի հերթական տողի առաջին ոչ զրոյական տարրը։ Եթե տողը զրոյական է, ապա Indptr զանգվածում երկու հարևան տարրերը համընկնում են։ Indptr զանգվածի վերջին տարրը հավասար է մատրիցի ոչ զրոյական տարրերի քանակին։

Ենթադրենք՝ ունենք հետևյալ 4x4 մատրիցան.

[[0, 0, 3, 0],
[22, 0, 0, 0],
[7, 5, 0, 0],
[0, 0, 0, 8]]

CSR ձևաչափում մատրիցան կներկայացվի հետևյալ կերպ՝

- Data-[3, 22, 7, 5, 8] (ոչ զրոյական արժեքները),
- Indices-[2, 0, 0, 1, 3] (ոչ զրոյական արժեքների սյուների համարները),
- Indptr-[0, 1, 2, 4, 5] (զանգվածի տարրը ցույց է տալիս մատրիցի յուրաքանչյուր տողի առաջին ոչ զրոյական տարրի դիրքի համարը Data զանգվածում. վերջին տարրը հավասար է մատրիցի ոչ զրոյական տարրերի քանակին):

COO ձևաչափը ավելի պարզ է, քան CSR-ը և հաճախ օգտագործվում է սկզբնական տվյալները հավաքելու համար:

COO ձևաչափի հիմնական բաղադրիչներն են՝

- Data-պահպանում է մատրիցի ոչ զրոյական արժեքները,
- Row- ոչ զրոյական արժեքների տողերի ինդեքսները,
- Col-ոչ զրոյական արժեքների սյուների ինդեքսները:

Օրինակ,

[[0, 0, 3, 0],
[22, 0, 0, 0],
[7, 5, 0, 0],
[0, 0, 0, 8]]

մատրիցը COO ձևաչափում կներկայացվի այսպես՝

- Data: [3, 22, 7, 5, 8] (ոչ զրոյական արժեքները),

- Row: [0, 1, 2, 2, 3] (ոչ զրոյական արժեքների համապատասխան տողերը),
- Col: [2, 0, 0, 1, 3] (ոչ զրոյական արժեքների համապատասխան սյուները):

Նշենք, որ CSR ձևաչափը ավելի օպտիմալ է հիշողության և հաշվարկների համար հատկապես մեծ մատրիցների դեպքում: COO ձևաչափը հաճախ կիրառվում է տվյալների հավաքման փուլում: Վերջնական հաշվարկների համար այն վերածվում է CSR կամ այլ ձևաչափի: Այս երկու ձևաչափերն էլ հաճախ կիրառվում են մեքենայական ուսուցման մեջ, որտեղ տվյալների կառուցվածքը սովորաբար նոսրացված է:

Ստորև բերված ծրագրային կոդի միջոցով արտածվում են CSR-ի և COO-ի բաղադրիչները երկու մատրիցների համար: Առաջին մատրիցը Numpy-ի eye օբյեկտն է (միավոր անկյունագծային մատրից), երկրորդը՝ ծրագրում տրված dense_matrix մատրիցը:

Օրինակ 3.1

```
import numpy as np
from scipy import sparse

eye = np.eye(4)
dense_matrix = np.array([
    [1, 0, 0, 0, 2],
    [0, 0, 3, 0, 0],
    [0, 0, 5, 4, 0],
    [0, 0, 0, 0, 0],
    [0, 7, 0, 3, 9]
])
print("Numpy-qանqված eye: \n{}".format(eye))
print("Numpy-qանqված dense_matrix: \n{}".format(dense_matrix))
```

```
# Ձևափոխել eye Numpy-զանգվածը CSR-ձևաչափի sparse զանգվածի
eye_csr_matrix = sparse.csr_matrix(eye)
print("\nCSR-ձևաչափով eye sparse զանգված: \n")
print("Տվյալներ:", eye_csr_matrix.data)
print("Սյան ինդեքսներ:", eye_csr_matrix.indices)
print("Տողի ցուցիչներ:", eye_csr_matrix.indptr)

# Ձևափոխել dense_matrix Numpy-զանգվածը CSR-ձևաչափի sparse զանգվածի
dense_csr_matrix = sparse.csr_matrix(dense_matrix)
print("\nCSR-ձևաչափով  dense_matrix sparse զանգված: \n")
print("Տվյալներ:", dense_csr_matrix.data)
print("Սյան ինդեքսներ:", dense_csr_matrix.indices)
print("Տողի ցուցիչներ:", dense_csr_matrix.indptr)
```

```
# Ձևափոխել eye Numpy-զանգվածը COO-ձևաչափի  sparse զանգվածի
eye_coo_matrix = sparse.coo_matrix(eye)
print("\nCOO-ձևաչափով eye sparse զանգված:")
print("Տվյալներ:", eye_coo_matrix.data)
print("Տողի ինդեքսներ:", eye_coo_matrix.row)
print("Սյան ինդեքսներ:", eye_coo_matrix.col)

# Ձևափոխել dense_matrix Numpy-զանգվածը COO-ձևաչափի  sparse զանգվածի
dense_coo_matrix = sparse.coo_matrix(dense_matrix)
print("\nCOO-ձևաչափով dense_matrix sparse մատրիցա:")
print("Տվյալներ:", dense_coo_matrix.data)
print("Տողի ինդեքսներ:", dense_coo_matrix.row)
print("Սյան ինդեքսներ:", dense_coo_matrix.col)
```

Կոդի կատարման արդյունքում կստանանք՝

```
Numpy-զանգված eye:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
Numpy-զանգված dense_matrix:
[[1 0 0 0 2]
 [0 0 3 0 0]
 [0 0 5 4 0]
 [0 0 0 0 0]
 [0 7 0 3 9]]

CSR-ձևաչափով eye sparse զանգված:

Տվյալներ: [1. 1. 1. 1.]
Սյան ինդեքսներ: [0 1 2 3]
Տողի ցուցիչներ: [0 1 2 3 4]

CSR-ձևաչափով  dense_matrix sparse զանգված:

Տվյալներ: [1 2 3 5 4 7 3 9]
Սյան ինդեքսներ: [0 4 2 2 3 1 3 4]
Տողի ցուցիչներ: [0 2 3 5 5 8]
```

```
COO-ձևաչափով eye sparse զանգված:
Տվյալներ: [1. 1. 1. 1.]
Տողի ինդեքսներ: [0 1 2 3]
Սյան ինդեքսներ: [0 1 2 3]

COO-ձևաչափով dense_matrix sparse մատրիցա:
Տվյալներ: [1 2 3 5 4 7 3 9]
Տողի ինդեքսներ: [0 0 1 2 2 4 4 4]
Սյան ինդեքսներ: [0 4 2 2 3 1 3 4]
```

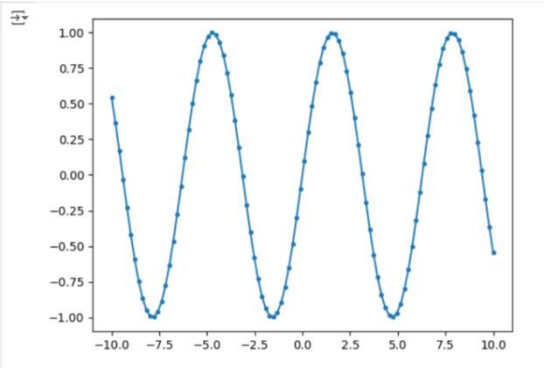
Տվյալների վերլուծության խնդիրներում զգալիորեն օգնում են տվյալների վիզուալիզացիայի արդյունքները։

Matplotlib գրադարանը նախատեսված է տարբեր տիպի գրաֆիկների տեսքով տվյալների վիզուալիզացիան իրականացնելու համար: Այդ գրադարանի մեթոդների օգնությամբ կարելի է կառուցել գծային դիագրամներ, հիստոգրամներ, տվյալների ցրվածության դիագրամներ և այլն:

Օրինակ՝ 3.2-ում կառուցվում է $y = \sin(x)$ ֆունկցիայի գրաֆիկը [-10,10] միջակայքում: Դիտարկվում է միջակայքի 100 կետ:

Օրինակ 3.2

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-10, 10, 100)
y = np.sin(x)
plt.plot(x, y, marker=".")
plt.show()
```



Python-ում տվյալների մշակման և վերլուծության համար նախատեսված է նաև pandas գրադարանը: Նրա ֆունկցիաներով մշակվում են DataFrame կառուցվածքի օբյեկտներ, որոնք երկչափ աղյուսակներ են: pandas-ը կարող է աշխատել տարբեր ձևաչափի ֆայլերի հետ՝ SQL, Excel, CSV:

Օրինակ՝ 3.3-ում Dictionary-օբյեկտի համար կառուցվում է DataFrame-օբյեկտ:

Օրինակ 3.3

```
import pandas as pd
data = {'Անուն': ["Տիգրան", "Անի", "Գրիգոր", "Շաքե"],
            'Քաղաք' : ["Երևան", "Գյումրի", "Բարսելոնա", "Գորիս"] ,
            'Տարիք': [24, 13, 23, 33]}
data_pandas = pd.DataFrame(data)
print(data_pandas)
```

```
     Անուն       Քաղաք  Տարիք
0  Տիգրան       Երևան      24
1     Անի     Գյումրի      13
2  Գրիգոր  Բարսելոնա      23
3    Շաքե       Գորիս      33
```

3.1.1 Տվյալների հավաքածուներ scikit-learn գրադարանում

Մենք, արդեն օգտագործելով հիրիկ ծաղիկների օրինակը (տես բաժին 2.9), քննարկել ենք պերցեպտրոնի միջոցով օբյեկտների դասակարգման խնդիրը: Հիմա նույն առաջադրանքը կատարենք scikit-learn գրադարանի միջոցով: Այս դեպքում կգտագործենք պերցեպտրոնի ուսուցման ալգորիթմից տարբերվող ալգորիթմ:

Կառուցենք դասակարգման մոդել ըստ k-մոտակա հարևանների մեթոդի, ուսուցանենք այդ մոդելը, ապա այն օգտագործենք գործնական խնդիրներում: Այսպիսով, մեր խնդիրն է ստեղծել և ուսուցանել դասակարգման մոդել: Այդ մոդելը պետք է ուսուցման հավաքածուի միջոցով ուսուցանվի հիրիկ

ծաղկի դասակարգման համար: Այսինքն՝ այն պետք է կանխատեսի ցանկացած նոր հիրիկ ծաղկի տեսակը:

scikit-learn գրադարանի ֆունկցիաները օգտագործենք հիրիկ ծաղիկը (օբյեկտ) ըստ տեսակի (setosa, versicolor, virginica) երեք տարբեր դասերում դասակարգման նպատակով: Օբյեկտը բնութագրվում է petal length, petal width, sepal length, sepal width հայտանիշների միջոցով: Տվյալների հավաքածուի յուրաքանչյուր օբյեկտ պատկանում է երեք դասերից մեկին: Տարբեր դասերին համապատասխանում են տարբեր պիտակներ: Այսպիսով, լուծվող խնդիրը եռադաս դասակարգման է: Այս օրինակի համար կօգտագործենք մեքենայական ուսուցման հիրիկ ծաղկի տվյալների պատրաստի հավաքածու, որն առկա է scikit-leam գրադարանի datasets մոդուլում: Հավաքածուն ներբեռնենք load_iris () ֆունկցիայով:

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

iris_dataset-ը Bunch օբյեկտ է (կապոց, կույտ): Այն պարունակում է բանալիներ (տվյալների զանգվածի բաղադրիչների անուններ) և արժեքներ (տվյալների զանգվածներ): iris_dataset-ի տվյալների վերաբերյալ տեղեկատվություն (բանալիների անուններ, զանգվածի տիպ և այլն) կարելի է ստանալ ստորև ներկայացված կոդի միջոցով:

Օրինակ 3.4

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
print ( " iris _ dataset-ի բանալիները:\n{}". format (iris_dataset. keys ()))
print ( " data զանգվածի տիպը: {}". format (type (iris_dataset [ 'data'])))
print ("data զանգվածի ֆորմատը: {}" .format (iris_dataset [ 'data'] .shape))
print ("Թիրախային փոփոխականի արժեքները: {}". format (iris_dataset [ 'target']))
print("դասերի անվանումները: {}".format(iris_dataset['target_names']))
print (iris_dataset ['DESCR'] [: 193] + "\n ... ")  # iris _ dataset-ի նկարագրությունը
print("Հատկանիշերի անվանումները: \n{}".format(iris_dataset['feature_names']))
print("Ֆայլի հասցեն: \n{}".format(iris_dataset['filename']))
```

Ստորև բերված է կոդի աշխատանքի արդյունքը.

```
 iris _ dataset-ի բանալիները:
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
 data զանգվածի տիպը: <class 'numpy.ndarray'>
data զանգվածի ֆորմատը: (150, 4)
Թիրախային փոփոխականի արժեքները: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
դասերի անվանումները: ['setosa' 'versicolor' 'virginica']
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, pre
 ...
Հատկանիշերի անվանումները:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Ֆայլի հասցեն:
iris.csv
```

iris_dataset-ի բանալիներն են data-ն, target-ը, target_names-ը, DESCR-ը, feature_names-ը, որոնց համապատասխան տվյալները նկարագրած են ստորև.

- data-NumPy զանգված, որը պարունակում է չորս հատկանիշ (feature)՝ ծաղկի ծաղկաթերթիկի երկարությունը և լայնությունը, ծաղկի բաժակաթերթիկի երկարությունը և լայնությունը;
- target-NumPy զանգված, որը պարունակում է թիրախային արժեքներ՝ ծաղկի տեսակի թվայնացված արժեքները (0 - setosa, 1- versicolor, 2- virginica);

- target_names - ծաղկի տեսակների անունների զանգված;
- DESCR - տվյալների հավաքածուի համառոտ նկարագրություն;
- feature_names - հայտանիշերի զանգված;
- filename - ֆայլի անունը, որը պարունակում է տվյալների հավաքածուն։

data զանգվածի տողը համապատասխանում է մեկ օբյեկտի, իսկ սյուները օբյեկտի հայտանիշներն են։ Զանգվածը պարունակում է 150 տող (150 օբյեկտների տվյալներ), յուրաքանչյուր օբյեկտ բնութագրվում է չորս հայտանիշով։ Զանգվածի չափսը (shape) բնութագրվում է օբյեկտների և հայտանիշների քանակով (150*4)։ data զանգվածի առաջին հինգ տողերը կարելի է ստանալ

```
print("data զանվածի առաջին հինգ տողերը:\n{}".format(iris_dataset['data'][:5]))
```

հրամանով։

```
data զանվածի առաջին հինգ տողերը:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

target-ը NumPy-զանգված է, որը պարունակում է հավաքածուի ծաղիկների տեսակները՝ կոդավորված 0, 1, 2 թվերով (ճիշտ պատասխաններ)։ Այդ զանգվածի տարրերը կարելի է ստանալ հետևյալ հրամանով՝

```
print("ճիշտ պատասխաններ:\n{}".format(iris_dataset['target']))
```

```
Ճիշտ պատասխաններ:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Այստեղ թվերը համապատասխանում են ծաղիկների տեսակներին. 0-setosa, 1-versicolor, 2- virginica:

3.1.2 Ուսուցման և թեստավորման տվյալների հավաքածուներ scikit-learn գրադարանում

Նախորդ բաժնում ստեղծված տվյալների միջոցով կառուցենք մեքենայական ուսուցման մոդել, որը պետք է ծաղկի նոր նմուշի համար կանխատեսի նրա տեսակը: Նախքան մոդելը նոր նմուշի վրա կիրառելը պետք է համոզվել, որ մոդելն իրականում աշխատում է, և դրա կանխատեսումները կարող են վստահելի լինել:

Մոդելի աշխատանքի արդյունավետությունը գնահատելու համար նրան պետք է տալ նոր տվյալներ, որոնք նա նախկինում չի տեսել: Սովորաբար դա արվում է ունեցած տվյալները (150 օբյեկտները) բաժանելով երկու մասի: Տվյալների մի մասը օգտագործվում է նեյրոնային ցանցը ուսուցանելուու համար և կոչվում է ուսուցման հավաքածու (training set): Մնացած տվյալները օգտագործվում են ուսուցման որակը գնահատելու համար, այն կոչվում է թեստային հավաքածու (testnig set):

scikit-learn գրադարանն ունի train_test_split() ֆունկցիա, որը խառնում է տվյալների հավաքածուն և այն բաժանում երկու մասի: Այդ ֆունկցիան պատահականորեն ընտրում է տվյալների 75%-ը որպես ուսուցման հավաքածու: Մնացած

տվյալները (25%) վերցվում են որպես թեստային հավաքածու: Հարցը, թե որքա՞ն տվյալներ պետք է ընտրել ուսուցման հավաքածուի համար և որքանը՝ թեստային հավաքածուի համար, վիճելի է: Այնուամենայնիվ, տվյալների 25%-ը որպես թեստային հավաքածու օգտագործելը գործնականորեն լավ արդյունքներ է ցուցաբերել:

Տվյալների երկչափ զանգվածը նշանակենք X-ով, իսկ թիրախային փոփոխականների միաչափ զանգվածը՝ y-ով: scikit-learn-ում տվյալները սովորաբար նշանակվում են մեծատառերով, իսկ պիտակները (ճիշտ պատասխանները)՝ փոքրատառերով: Ուսուցման տվյալների հավաքածուն երկու մասի բաժանելու համար օգտագործվում է train_test_split() ֆունկցիան:

```
from sklearn .model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
                                                    iris_dataset['target'],
                                                      random_state=0)
```

train_test_split() ֆունկցիան վերադարձնում է չորս զանգված՝X_train, X _test, y_train և y_test: X_train զանգվածը պարունակում է սկզբնական տվյալների հավաքածուի 75%-ը, իսկ X_test զանգվածը՝ 25%-ը: Ստեղծված զանգվածների չափսերը կարելի է ստանալ՝ օգտագործելով հետևյալ հրամանները.

```
print(" X_train: {} զանգվածի չափը".format(X_train.shape))
print(" y_t rain: {} զանգվածի չափը". format (y_train.shape))
print(" X_test: {} զանգվածի չափը".format(X_test.shape))
print ("y_ test: {} զանգվածի չափը". format (y_test.shape))
```

Տվյալների հավաքածուների համար ծրագրի կոդը գործարկելիս ստացվում է հետևյալ արդյունքը.

```
X_train: (112, 4) զանգվածի չափը
y_t rain: (112,) զանգվածի չափը
X_test: (38, 4) զանգվածի չափը
y_ test: (38,) զանգվածի չափը
```

3.1.3 Տվյալների հավաքածուների նախնական վերլուծություն

Նախքան մոդել կառուցելը և ուսուցանելը պետք է ուսումնասիրել տվյալները՝ հասկանալու համար, թե արդյոք հնարավոր է լուծել խնդիրը, և արդյոք անհրաժեշտ տեղեկատվությունը առկա է տվյալների մեջ: Բացի այդ, տվյալների ուսումնասիրումը անոմալիաները և սխալները հայտնաբերելու լավ միջոց է: Օրինակ, հնարավոր է, որ ծաղիկների վերաբերյալ տվյալների մի մասը տրվել են ոչ թե սանտիմետրերով, այլ դյույմերով: Իրականում տվյալների մեջ անհամապատասխանություններ հաճախ են լինում:

Ինչպես նշել ենք, տվյալների ուսումնասիրության լավագույն միջոցներից մեկը դրանց վիզուալիզացնելն է: Դա կարելի է անել՝ օգտագործելով ցրվածության դիագրամներ (scatter plot): Երկչափ դիագրամի վրա հայտանիշներից մեկը կարելի է վերցնել x առանցքի երկայնքով, իսկ մյուսը՝ y առանցքի: Դիագրամի վրա յուրաքանչյուր օբյեկտի համապատասխանում է մեկ կետ: Այսպիսի դիագրամում երեքից ավել հայտա-նիշներով տվյալների հավաքածուներ չեն պատկերվում:

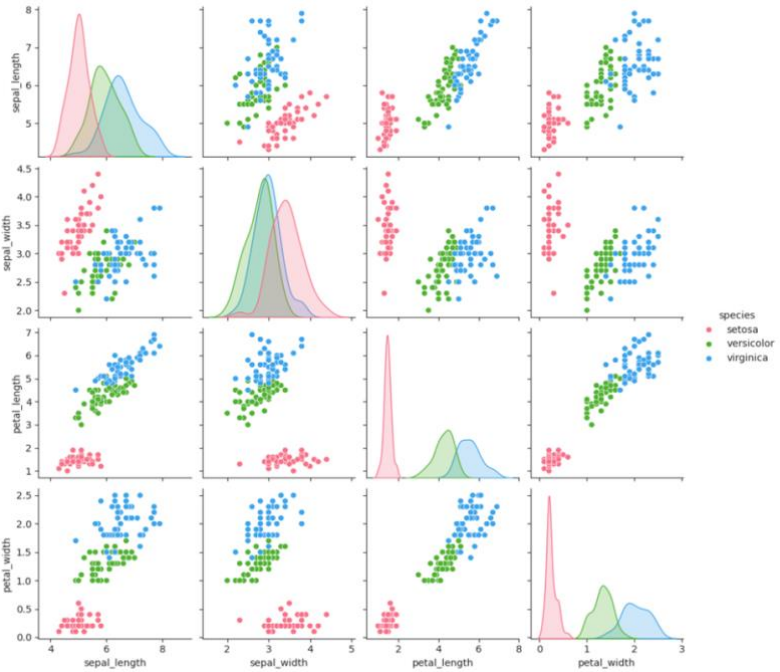
Բազմաչափ տվյալների դեպքում այս խնդիրը լուծելու եղանակներից մեկը ցրվածության մատրից (scatterplot matrix) կառուցելն է: Նշենք, որ ցրվածության մատրիցը ցույց չի տալիս բոլոր հայտանիշների կապը, ուստի տվյալների որոշ կարևոր հայտանիշներ չեն բացահայտվի այդ եղանակով:

Ցրվածության մատրից կարելի է կառուցել օգտագործելով seaborn գրադարանը, որը աշխատում է pandas և matplotlib գրադարանների հետ: Պետք է ներմուծել այդ գրադարանները, այնուհետև ստեղծել ցրվածության մատրից:

Օրինակ 3.5

```
import pandas as pd
import seaborn as sb
from matplotlib import pyplot as plt

df = sb.load_dataset('iris')
sb.set_style("ticks")
sb.pairplot(df, hue='species', diag_kind="kde", kind="scatter", palette="husl")
plt.show()
```

Կոդի աշխատանքի արդյունքը բերված է նկ. 3.1-ում:



Նկ. 3.1 Հիրիկ ծաղկի հայտանիշների ցրվածության մատրից

Ստացված ցրվածության մատրիցն ունի չորս տող և չորս սյուն, որոնք համապատասխանում են ծաղիկների հայտանիշներին: Մատրիցի տողի և սյան հատման վանդակում ներկայացված են հայտանիշների զույգերով բնութագրվող տվյալները: Մատրիցի անկյունագծային վանդակներում պատկերված են երեք տեսակի ծաղիկների համար յուրաքանչյուր հայտանիշի արժեքների բաշխվածության խտության ֆունկցիաները: Ինչպես տեսնում ենք, բաշխվածությունները նորմալ տեսքի են: Ցրվածության մատրիցից երևում է, որ երեք դասի ծաղիկները հնարավոր է բաժանել ըստ նրանց հայտանիշների: Սա նշանակում է, որ այս տվյալներով կարելի է ուսուցանել նեյրոնային ցանց, որը կլուծի դասակարգման խնդիրը:

3.1.4 Մոդելի ուսուցումը և օգտագործումը scikit-learn գրադարանի միջոցով

scikit-learn գրադարանն ունի դասակարգման ալգորիթմներ, որոնք կարող են օգտագործվել մոդելների ուսուցման համար: Առաջարկված օրինակում կօգտագործենք դասակարգիչ ըստ k-մոտակա հարևանների (K-Nearest Neighbors) մեթոդի: Այս դեպքում ուսուցման նպատակը ուսուցման տվյալների հավաքածուն հիշելն է: Նոր տվյալի համար կանխատեսում կատարելիս ալգորիթմը (k=1 դեպքում) ուսուցման տվյալների հավաքածուի մեջ գտնում է օբյեկտ, որը ամենամոտն է տվյալ օբյեկտին: Այնուհետև ուսուցման հավաքածուի մեջ գտած օբյեկտի պիտակը վերագրում է նոր տվյալին:

Այս մեթոդը ոչ միայն օգտագործում է տվյալների նոր կետին մոտ մեկ հարևան կետ, այլև կարող է դիտարկել հարևանների ցանկացած ֆիքսված քանակ՝ k (օրինակ, հաշվի առնել

մոտակա երեք, հինգ կամ տասը հարևանները): Այդ դեպքում կարելի է տվյալների նոր կետի համար կանխատեսել այն դասը, որին պատկանում է նրա հարևանների մեծ մասը:

Մեր դեպքում կօգտագործենք միայն մեկ հարևան: K ամենամոտ հարևանների մեթոդով դասակարգման ալգորիթմը իրականացված է KNeighborsClassifier դասում: Այս դասակարգիչը օգտագործելուց առաջ պետք է ստեղծենք դասի օբյեկտ և օբյեկտին տանք անհրաժեշտ պարամետրերը: KNeighborsClassifier-ի կարևոր պարամետրերից է հարևանների քանակը, որը կվերցնենք 1:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

Ստեղծված knn օբյեկտը պարունակում է ալգորիթմ, որը կօգտագործվի ուսուցման տվյալների վրա մոդելը կառուցելու համար, և ալգորիթմ, որը կանխատեսումներ կանի տվյալների նոր կետերի համար: Այն նաև պարունակում է տեղեկատվություն, որը ուսուցման ընթացքում ալգորիթմը ստացել է ուսուցման հավաքածուից: KNeighborsClassifier-ի դեպքում այն պարզապես կպահի ուսուցման հավաքածուն: Մոդելը ուսուցանելու համար օգտագործում է knn օբյեկտի fit() մեթոդը: Այն որպես արգումենտ ընդունում է ուսուցման տվյալները պարունակող X_train և պիտակներին համապատասխանող y_train զանգվածները.

```
z = knn.fit(X_train, y_train)
print(z)
```

fit() մեթոդը վերադարձնում է knn օբյեկտ: print(z) հրամանով կտպվեն մոդելի ստեղծման համար օգտագործված պարամետրերը:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric= 'minkowski ', metric_params=None,
n_jobs=None, n_neighbors=1, p=2, weights='uniform')
```

Ինչպես երևում է, բոլոր պարամետրերն ունեն լռելյայն արժեքներ, բացի մեր սահմանածից. դա n_neighbors=1 է: scikit-learn գրադարանի մոդելների մեծ մասն ունի մի շարք պարամետրեր, որոնք կապված են հաշվողական արագության օպտիմալացման հետ կամ նախատեսված են հատուկ օգտագործման դեպքերի համար:

Այժմ կարող ենք կանխատեսումներ անել՝ կիրառելով ստեղծված մոդելը այնպիսի տվյալների համար, որոնց ճիշտ պիտակները չգիտենք: Ենթադրենք՝ ունենք հիրիկ տեսակի ծաղիկ՝ բաժակաթերթիկի 5 սմ երկարությամբ, 2.9 սմ լայնությամբ և ծաղկաթերթիկի 1սմ երկարությամբ և 0.2 սմ լայնությամբ: Նրա դասը կանխատեսելու համար պետք է ունենալ չորս տարրերով NumPy զանգված:

```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new զանգվածի չափը: {}".format(X_new.shape))
```

Ստեղծվեց մեկ տող և չորս սյուն պարունակող զանգված: Զանգվածի տողում օբյեկտի հայտանիշներն են: Այդ օբյեկտի համար կանխատեսումը իրականացվում է knn օբյեկտի predict() մեթոդով.

```
pr = knn.predict(X_new)
print("Ծաղկի տեսակի պիտակը: {}".format(pr))
print("Ծաղկի տեսակը: {}".format(iris_dataset['target_names'][pr]))
```

Ստորև բերված է knn մոդելի ուսուցման, մոդելի կիրառմամբ հիրիկ ծաղկի նոր նմուշի կանխագուշակման ամբողջական ծրագրային կոդը:

Օրինակ 3.6

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris_dataset = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
                                                    iris_dataset['target'],
                                                    random_state=0)
# Դասակարգիչի ստեղծում և ուսուցում
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

# Դասակարգիչի օգտագործում
X_new = np.array([[5, 2.9, 1, 0.2]])
pr = knn.predict(X_new)
print("Ծաղկի տեսակի պիտակը: {}".format(pr))
print("Ծաղկի տեսակը: {}".format(iris_dataset['target_names'][pr]))
```

Կոդի աշխատանքի շնորհիվ ստացվել է հետևյալ արդյունքը՝

```
Ծաղկի տեսակի պիտակը: [0]
Ծաղկի տեսակը: ['setosa']
```

Ուսուցանված մոդելը կանխատեսել է, որ նոր օբյեկտը պատկանում է 0 դասին. դա նշանակում է, որ նոր օբյեկտը setosa տեսակի է: Այսպիսով, օգտագործելով scikit-learn գրադարանը, ուսուցանվեց դասակարգիչ և իրականացվեց կանխատեսում:

Այս օրինակը ցույց է տալիս, թե որքան արդյունավետ է կանխատեսման մոդել ստեղծելու և ուսուցանելու համար

Python-ում պատրաստի գրադարանների օգտագործումը: Բայց, արդյո՞ք կարող ենք վստահել ստեղծված մոդելին: Արդյո՞ք այն ճիշտ է որոշել ծաղկի տեսակը, որի չափսերը տրվում են նրա մուտքում: Ճիշտ կանխատեսումներ անելը մոդելի կառուցման հիմնական նպատակն է: Հաջորդ բաժնում ցույց կտանք, թե ինչպես կարելի է գնահատել մոդելի աշխատանքի որակը, և որքանով կարող ենք վստահել ուսուցանված մոդելի որոշումներին:

3.1.5 Մոդելի ուսուցման որակի գնահատում scikit-learn գրադարանում

Ուսուցանված մոդելի որակը գնահատելու համար կօգտագործենք նախապես ստեղծված թեստային հավաքածուն: Ինչպես գիտենք, ծաղկի ճիշտ տեսակը թեստային հավաքածուում հայտնի է: Թեստային հավաքածուի յուրաքանչյուր օբյեկտի համար կանխատեսած պիտակը իրական պիտակի հետ համեմատելով՝ կարող ենք գնահատել մոդելի աշխատանքի որակը:

```
pr = knn.predict(X_test)
print("Թեստային հավաքածուում ծաղկի տեսակի կանխատեսում:\n {}".format(pr))
print("Կանխատեսման ճշտությունը թեստային հավաքածուի վրա:{:.2f}".
                                format(np.mean(pr == y_test)))
```

Այս կոդը գործարկելուց հետո ստացվում է հետևյալ արդյունքը.

```
Թեստային հավաքածուում ծաղկի տեսակի կանխատեսում:
 [2 1 0 2 0 2 0 1 1 2 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]
Կանխատեսման ճշտությունը թեստային հավաքածուի վրա:0.97
```

Այս մոդելի ճշտությունը թեստային հավաքածուի վրա 0.97 է: Դա նշանակում է, որ դասակարգման մոդելը ուսուցումից հետո 97%-ով ճիշտ է կանխատեսել թեստային հավաքածուն: Հետևաբար, կարող ենք ակնկալել, որ մոդելը 97%-ով ճիշտ կաշխատի նաև նոր օբյեկտների համար: Այսինքն՝ կառուցած մոդելը օգտագործման համար կարող է բավականին հուսալի լինել:

3.1.6 Perceptron և scikit-learn գրադարան

scikit-learn գրադարանը համատեղում է օգտագործողի համար հարմար միջոցները դասակարգման օպտիմիզացված ալգորիթմների իրականացման հետ:

scikit-learn գրադարանն ունի ոչ միայն ուսուցման ալգորիթմների լայն տեսականի, այլ նաև տվյալների նախնական մշակման, մոդելները ճշգրտելու և գնահատելու հարմար ֆունկցիաներ:

Կառուցենք և ուսուցանենք մոդել պերցեպտրոնային հիմքով: Պարզության համար կօգտագործենք Iris տվյալների բազան, որին մենք արդեն ծանոթ ենք: Ինչպես նշվեց, այն հասանելի է scikit-learn գրադարանում. այդ տվյալների բազան հաճախ օգտագործվում է ալգորիթմների փորձարկման և ուսուցման ժամանակ: Կօգտագործենք scikit-leam գրադարանի datasets մոդուլում առկա հիրիկ ծաղիկների հավաքածուն: Վիզուալիզացիայի համար կվերցնենք օբյեկտի հայտանիշներից երկուսը (ծաղկաթերթիկի երկարությունը և լայնությունը): Հայտանիշների X մատրիցին տրվում են 150 ծաղկի նմուշների երկու հայտանիշները, իսկ պիտակների y վեկտորին՝ համապատասխան դասերի համարները:

Օրինակ 3.7-ում ստեղծենք և ուսուցանենք պերցեպտրոն, որը պետք է իրականացնի հիրիկ ծաղկի դասակարգում: Հավաքածուն ներբեռնում ենք՝ կանչելով load_iris() ֆունկցիան:

Օրինակ 3.7

```
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y= iris.target
print (y)
```

Կոդի կատարմամբ արտածվում են iris.target–ում պահվող դասերի պիտակները:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Iris setosa, Iris virginica և Iris versicolor ծաղկատեսակների անունները ներկայացված են ամբողջ թվերով՝ 0, 1, 2: Մեքենայական ուսուցման շատ գրադարաններում դա արվում է հաշվարկները օպտիմալացնելու համար:

Պերցեպտրոնը ուսուցանելու և թեստավորելու նպատակով տվյալների հավաքածուն բաժանվել է ուսուցման և թեստավորման հավաքածուների: Այդ բաժանումը կատարվել է հետևյալ ձևով:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)
```

scikitlearn գրադարանի model_selection մոդուլի train_test_split ֆունկցիան պատահականորեն բաժանում է X և y զանգվածները ընդհանուր ծավալի 30% (45 նմուշ) թեստային տվյալների և 70% ուսուցման տվյալների (105 նմուշ):

Մեքենայական ուսուցման շատ ալգորիթմներ ուսուցման որակը բարելավելու համար պահանջում են հայտանիշների մասշտաբավորում: Այստեղ նախնական մշակման համար հայտանիշները ստանդարտացվում են scikit-learn գրադարանի preprocessing մոդուլի standardScaler դասի միջոցով:

Օրինակ 3.8

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train) # յուրաքանչյուր հայտանիշի համար հաշվարկվում է միջին արժեքը՝ μ և
                #շեղումը՝ σ, որոնց հիման վրա transform(տվյալներ) մեթոդով կատարվում է
                #ուսուցման և թեստային տվյալների ստանդարտավորում
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

Օրինակում standardScaler դասը բեռնվել է նախնական մշակման մոդուլից, որից հետո սկզբնարժեքավորվել է StandardScaler օբյեկտ և այն վերագրվել sc փոփոխականին: Օգտագործվել է StandardScaler օբյեկտի fit() մեթոդը՝ ուսուցման հավաքածուի յուրաքանչյուր հայտանիշի μ (էմպիրիկ միջին) և σ (ստանդարտ շեղում) պարամետրերը հաշվելու համար: Օգտագործելով μ և σ պարամետրերը՝ transform() մեթոդի միջոցով ստանդարտացվել են ուսուցման տվյալները:

Նշենք, որ թեստային հավաքածուն ստանդարտացնելու համար օգտագործվել են նույն պարամետրերը, որոնց շնորհիվ ուսուցման և թեստային հավաքածուների արժեքները դարձել են համեմատելի: Ստանդարտացնելով ուսուցման

տվյալները՝ կարելի է մոդելը ուսուցանել պերցեպտրոնի հիմքով:

scikit-learn գրադարանի ալգորիթմների մեծ մասը լռելյայն աջակցում է բազմադաս դասակարգմանը: Linear_model մոդուլից Perceptron դասը բեռնելուց հետո սկզբնարժեքավորվել է Perceptron օբյեկտ, և մոդելը ուսուցանվել fit() մեթոդով: eta0 պարամետրը համարժեք է պերցեպտրոնի ուսուցման արագության eta պարամետրին, իսկ max_iter պարամետրը սահմանում է փուլերի (էպոխների) քանակը: Պերցեպտրոնի ուսուցումը scikit-leam գրադարանի միջոցով կատարվում է ստորև ներկայացված կոդով:

Օրինակ 3.9

```
from sklearn.linear_model import Perceptron
              # սահմանել ուսուցման արագությունը 0.1, իսկ իտերացիաների քանակը՝ 40
              # random_state օգտագործվում է ուսուցման հավաքածուն ամեն իտերացիայից
              # հետո խառնելու համար
ppn = Perceptron(eta0=0.1, random_state=0, max_iter=40)
ppn.fit(X_train_std, y_train)
```

Ուսուցանված մոդելում թեստային տվյալների կանխատեսման համար օգտագործում ենք predict () մեթոդը:

Օրինակ 3.10

```
y_pred = ppn.predict(X_test_std)
print('Սխալ դասակարգված նմուշների քանակ. % d' % (y_test != y_pred) .sum())
```

Կոդի աշխատանքի արդյունքում ստանում ենք՝

```
Սխալ դասակարգված նմուշների քանակ.  5
```

Կանխատեսող մոդելի ճշտությունը գնահատելու համար օգտագործվում է accuracy հայտանիշը, որի արժեքը կարելի է ստանալ accuracy_score() մեթոդով։

Օրինակ 3.11

```
from sklearn.metrics import accuracy_score
print('Մոդելի ճշտությունն. %.2f' % accuracy_score(y_test, y_pred))
```

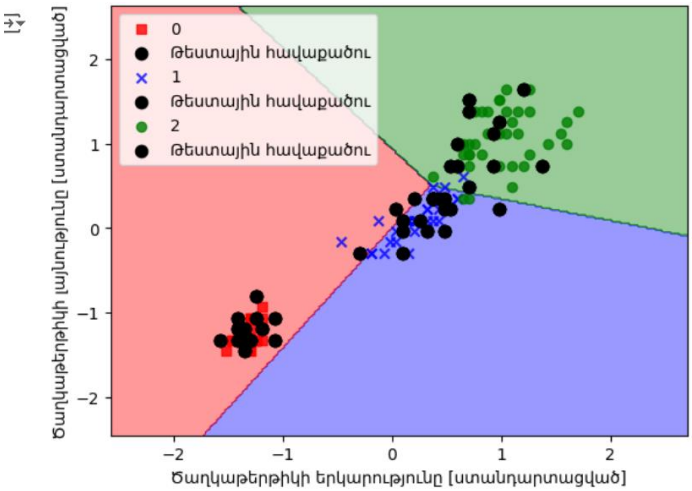
```
Մոդելի ճշտություն. 0.89
```

Օրինակ 3.12-ում բերված է ծաղիկների դասերի բաժանման տիրույթների գրաֆիկի (նկ.3.2) ստացման ծրագրային հատվածը։

Օրինակ 3.12

```
from matplotlib.colors import ListedColormap
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
    # Կարգավորել մարկերների գեներատորը և ներկապնակը
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'green', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # Ցույց տալ բոլոր նմուշները
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)
```

```
# Ընտրել թեստային նմուշները
        if test_idx:
             X_test, y_test = X[test_idx, :], y[test_idx]
             plt.scatter(X_test[:, 0], X_test[:, 1], c='0', alpha=1.0,
                linewidths=1, marker='o', s=55, label='թեստային հավաքածու')

X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
plot_decision_regions(X=X_combined_std, y=y_combined, classifier=ppn,
                      test_idx=range(105, 150))

plt.xlabel('Ծաղկաթերթիկի երկարությունը [ստանդարտացված]')
plt.ylabel('Ծաղկաթերթիկի լայնությունը [ստանդարտացված]')
plt.legend(loc='upper left')
plt.show()
```



Նկ. 3.2 Ծաղիկների հավաքածուի բաժանումը դասերի

Ինչպես երևում է նկարից, հնարավոր չէ ծաղիկների երեք դասերը ամբողջությամբ բաժանել ուղիղ գծերով։ Հայտնի է, որ պերցեպտրոնի ուսուցման ալգորիթմը չի զուգամիտում տվյալների այն հավաքածուների վրա, որոնք գծայնորեն ամբողջովին բաժանելի չեն։

Կան գծային ավելի լավ դասակարգիչներ, որոնք նվազագույնի են հասցնում կորուստների արժեքը, նույնիսկ եթե դասերը գծայնորեն լիովին բաժանելի չեն: Հաջորդ բաժնում կծանոթանանք այդ դասակարգիչներից մեկին:

3.1.7 Լոգիստիկ ռեգրեսիայի մեթոդով դասակարգիչներ scikit-learn գրադարանում

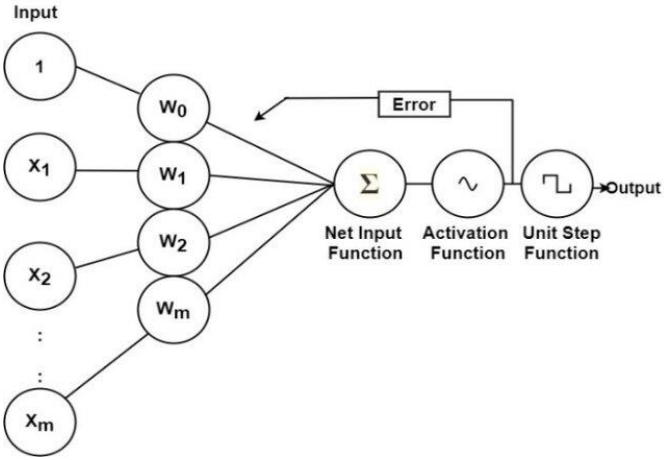
Ինչպես տեսանք, պերցեպտրոնը տվյալների դասակարգման պարզ և հարմարավետ մոդել է, սակայն նրա թերությունն այն է, որ այն չի ապահովում դասակարգման խնդրի լուծումը, եթե տվյալները գծայնորեն բաժանելի չեն:

Նախորդ բաժնում դիտարկված խնդիրը հենց այդպիսին է: Ինտուիտիվ դա կարելի է հիմնավորել հետևյալ կերպ. կապերի կշիռները անընդհատ փոփոխվում են, քանի որ ուսուցման յուրաքանչյուր իտերացիայում առկա է լինում առնվազն մեկ սխալ դասակարգված օբյեկտ: Բնականաբար կարելի է փոփոխել ուսուցման արագությունը և իտերացիաների քանակը, սակայն դիտարկվող տվյալների հավաքածուի վրա պերցեպտրոնը, միևնույնն է, չի զուգամիտի:

Ուսումնասիրենք մեկ այլ՝ պարզ և միաժամանակ ավելի հաճախ օգտագործվող ալգորիթմ՝ լոգիստիկ ռեգրեսիան, գծային և բինար դասակարգման խնդիրների համար: Նշենք, որ, չնայած իր անվանմանը, այդ ալգորիթմը լուծում է դասակարգման, այլ ոչ թե ռեգրեսիայի խնդիրներ:

Լոգիստիկ ռեգրեսիան հեշտ է իրականացնել որպես դասակարգման մոդել: Այն հիմնականում լավ է աշխատում գծայնորեն բաժանվող դասերի համար: Սա մեքենայական ուսուցման դասակարգման խնդիրներում առավել հաճախ օգ-

տագործվող ալգորիթմներից է: Պերցեպտրոնի նմանությամբ լոգիստիկ ռեգրեսիայի մոդելը բինար դասակարգման գծային մոդել է, որը կարելի է ընդլայնել բազմադաս դասակարգման խնդիրների համար (նկ. 3.3):



Նկ. 3.3 Լոգիստիկ ռեգրեսիայի մոդելի կառուցվածքը

Լոգիստիկ ռեգրեսիայի մոդելում որպես ակտիվացման ֆունկցիա օգտագործվում է սիգմոիդ ֆունկցիան: Վերջինիս ելքը մեկնաբանվում է որպես դասերից մեկին տվյալ օբյեկտի պատկանելիության հավանականություն: Օրինակ, եթե հիրիկ ծաղկի նմուշի համար սիգմոիդ ֆունկցիայի ելքը 0.8 է, ապա օբեկտը 80% ճշտությամբ պատկանում է տվյալ դասին: Կանխատեսած հավանականությունը One Step Function-ով կարելի է ձևափոխել բինար y արդյունքի: Օրինակ, եթե սիգմոիդ ֆունկցիայի ելքը մեծ է 0.5, ապա y=1, այլապես՝ y=0:

Բազմաթիվ հավելվածներում կարևոր է ոչ միայն որոշել դասերի պիտակները, այլ նաև գնահատել տվյալ դասին օբյեկտի պատկանելիության հավանականությունը:

Օրինակ՝ եթե լոգիստիկ ռեգրեսիան օգտագործվի եղանակի կանխատեսման համար, ապա պետք է կանխատեսել ոչ միայն անձրև կգա, թե ոչ, այլև տեղեկություն տալ անձրև գալու հավանականության մասին: Նմանատիպ ձևով լոգիստիկ ռեգրեսիան կարող է օգտագործվել որոշակի ախտանիշերի առկայության դեպքում բուժառուի մոտ հիվանդության կանխատեսման նպատակով:

scikit-learn գրադարանում իրականացված է լոգիստիկ ռեգրեսիայի օպտիմալ տարբերակ, որն աջակցում է նաև բազմադասային դասակարգմանը: Մենք կօգտվենք sklearn.linear_model.LogisticRegression դասից և դասի fit() մեթոդից՝ հիրիկ ծաղկի տվյալների հավաքածուի վրա մոդելը ուսուցանելու նպատակով:

Օրինակ 3.13

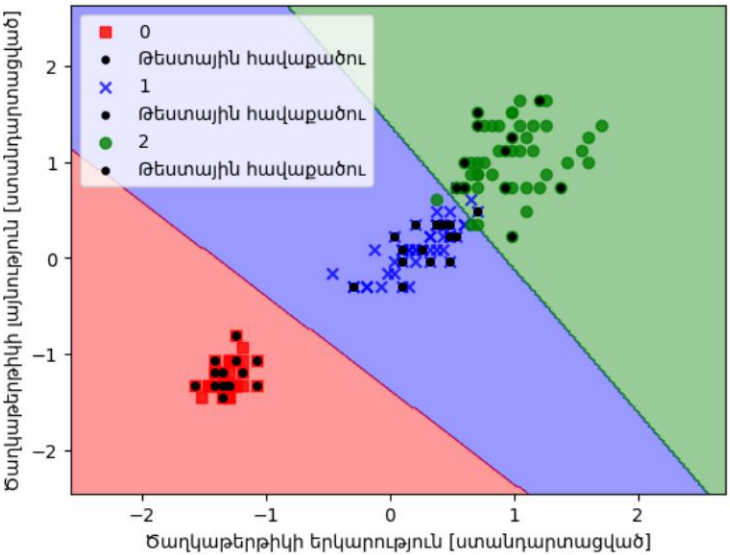
```
from matplotlib.colors import ListedColormap
from sklearn import datasets
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

Lr = LogisticRegression(C=1000.0, random_state=0)
Lr.fit(X_train_std, y_train)

y_pred = Lr.predict(X_test_std)
print('Սխալ դասակարգված նմուշների քանակը: %d' % (y_test != y_pred).sum())
```

```
print('Ճշտությունը: %.2f' % accuracy_score(y_test, y_pred))

def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
    # կարգավորել մարկերների գեներատորը և ներկապնակը
    markers = ('s', 'x', 'o', '>', 'v')
    colors = ('red', 'blue', 'green', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    #Սահմանել լուծման մակերեսը
    xl_min, xl_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xxl, xx2 = np.meshgrid(np.arange(xl_min, xl_max, resolution), np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xxl.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xxl.shape)
    plt.contourf(xxl, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xxl.min(), xxl.max())
    plt.ylim(xx2.min(), xx2.max())
    # Ցուցադրել բոլոր նմուշները
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8, c=cmap(idx), marker=markers[idx], label=cl)
        # Ցուցադրել թեստային նմուշները
        if test_idx:
            X_test, y_test = X[test_idx, :], y[test_idx]
            plt.scatter(X_test[:, 0], X_test[:, 1], c='0', alpha=1.0, linewidths=1, marker='.', s=55,
                        label='0testային հավաքածու')
```

```
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
plot_decision_regions(X_combined_std, y_combined, classifier=Lr, test_idx=range(105, 150))
plt.xlabel('Ծաղկաթերթիկի երկարություն [ստանդարտացված]')
plt.ylabel('Ծաղկաթերթիկի լայնություն [ստանդարտացված]')
plt.legend(loc='upper left')
plt.show()
```

Բերված ծրագրային կոդի աշխատանքի արդյունքը ներկայացված է նկ. 3.4-ում:

```
Սխալ դասակարգված նմուշների քանակը:  1
ճշտությունը: 0.98
```



Նկ. 3.4 Ծաղիկների դասերը

Նկարից երևում է, որ լոգիստիկ ռեգրեսիայի կիրառմամբ հաջողվեց օբյեկտները բաժանել երեք դասի: Սակայն ծաղիկների երկու տեսակները հստակ չեն բաժանվում, այլ ունեն հատման տիրույթ: Այդ տիրույթում գտնվող ծաղիկները պատկանում են առաջին կամ երկրորդ դասին որոշակի հավանականությամբ:

Կոնկրետ նմուշի պատկանելիությունը որևէ դասի կարելի է կանխատեսել predict_proba() մեթոդի միջոցով: Հիրիկ ծաղկի նմուշների պատկանելիության հավանականությունը որևէ դասի կարելի է որոշել օրինակ 3.14-ում բերված ծրագրի միջոցով:

Օրինակ 3.14

```
X1 = np.array([[1.5, 1.5]])
X2 = np.array([[0.0, 0.0]])
X3 = np.array([[-1, -1]])
p1 = Lr.predict_proba(X1)
p2 = Lr.predict_proba(X2)
p3 = Lr.predict_proba(X3)

print(X1)
print(p1)
print(X2)
print(p2)
print(X3)
print(p3)
```

Ծրագիրը կարտածի հետևյալ արդյունքները՝

```
[[1.5 1.5]]
[[2.48365285e-21 2.60361206e-07 9.99999740e-01]]
[[0. 0.]]
[[3.66607755e-05 9.99835075e-01 1.28264583e-04]]
[[-1 -1]]
[[9.89007732e-01 1.09922678e-02 1.46252130e-13]]
```

Ծրագրի փորձարկման արդյունքները բերված են աղյուսակ 1-ում:

Աղյուսակ 1

Նմուշի համարը	Մուտքային պարամետրեր	Դասին պատկանելիության հավանականությունը		
		setosa	virginica	versicolor
1	[1.5, 1.5]	0.000	0.000	0.999
2	[0.0, 0.0]	0.000	0.999	0.000
3	[-1. 0, -1.0]	0.991	0.008	0.000

Աղյուսակից երևվում է, որ ծաղիկի առաջին նմուշը 99% ճշտությամբ versicolor տեսակի է, երկրորդ նմուշը՝ virginica տեսակի, երրորդը՝ setosa տեսակի:

3.2 Keras գրադարանը

Keras-ը Python-ի հիմնական գրադարաններից մեկն է, որը ստեղծվել է 2015 թվականին նեյրոնային ցանցերի և մեքենայական ուսուցման նախագծերի իրականացման համար: Keras-ը կարող է աշխատել Deepleaming, MXNet, Microsoft Cognitive Toolkit (CNTK), Theano, TensorFlow մոդուլների հետ: Այդ գրադարանում իրականացված են նեյրոնային ցանցերի հետ կապված գրեթե բոլոր մոդուլները, ներառյալ օպտիմիզատորները, ակտիվացման և կորստի ֆունկցիաները, սկզբնարժեքավորման ձևերը, ռեգուլյարիզացիայի մոդելները: Keras-ը հնարավորություն է տալիս, ավելացնելով նոր դասեր, կառուցել նեյրոնային ցանցի նոր մոդուլներ:

Keras-ը օգտագործվում է փաթույթային նեյրոնային ցանցերի (CNNs, Convolutional Neural Networks) կառուցման և օգտագործման համար: Գրադարանը տրամադրում է նաև ցանցի շերտերի օպտիմալացման, նորմալացման և ակտիվացման տարբեր ալգորիթմներ: Keras-ի միջոցով ստեղծված հավելվածները կարող են հեշտությամբ տեղակայվել տարբեր հարթակներում.

- iOS օպերացիոն համակարգում Apple CoreML-ի միջոցով,
- Android օպերացիոն համակարգում Android-ի TensorFlow միջավայրի շրջանակում,
- բրաուզերի միջավայրում,

- Google Cloud-ում TensorFlow-Serving-ի միջոցով,
- Python-ի Web հավելվածներում,
- Raspberry Pi-ում և այլն:

Keras գրադարանը հիմնականում օգտագործվում է փաթույթային նեյրոնային ցանցի միջոցով պատկերների ճանաչման խնդիրներ լուծելու համար:

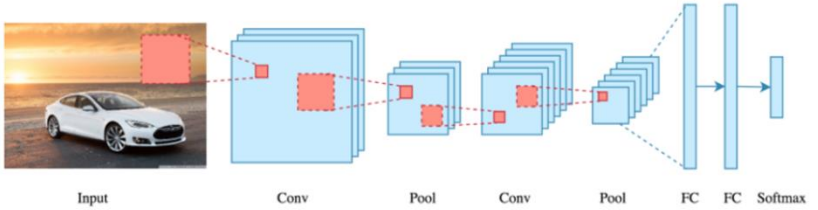
Ստորև նկարագրված են փաթույթային նեյրոնային ցանցի կառուցվածքը և աշխատանքի սկզբունքը:

3.2.1 Փաթույթային նեյրոնային ցանցեր

Փաթույթային նեյրոնային ցանցերը հիմնականում օգտագործվում են պատկերների դասակարգման համար: CNN ցանցում յուրաքանչյուր պատկեր ներկայացվում է պիքսելային արժեքների զանգվածով: Սև-սպիտակ նկարներում պիքսելն ունի 0 կամ 1 արժեք, գունավոր նկարներում յուրաքանչյուր պիքսել ներկայացվում է գունային բաղադրիչների տեսքով՝ RGB սանդղակում:

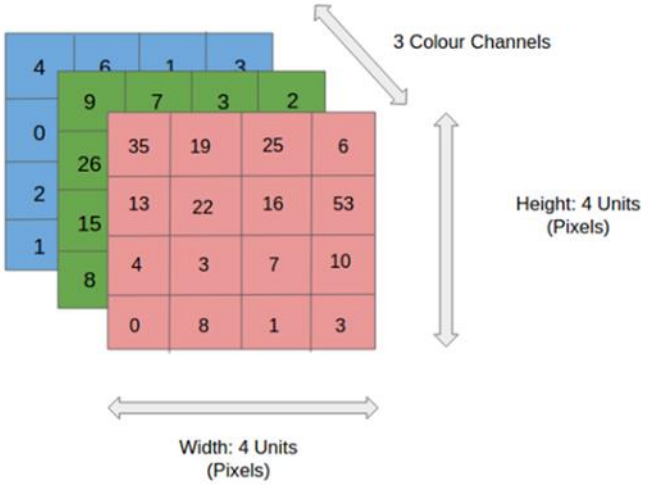
Սովորական նեյրոնային ցանցերի թերություններից է այն, որ այնտեղ սովորաբար անտեսվում է մուտքային տվյալների կառուցվածքը: Ցանցի մուտքային տվյալները բերվում են մեկչափանի զանգվածի (single dimensional array): CNN ցանցի միջոցով պատկերում հայտնաբերվում են պատկերի որոշակի առանձնահատկություններ՝ եզրեր, անկյուններ, գույնի փոփոխություն և այլն: Կարելի է ասել, որ փորձ է արվում նմանակել մարդու տեսողությանը: CNN ցանցերը համակարգչային տեսողության (Computer Vision) խնդիրների լուծման կարևոր մասն են:

CNN ցանցը պարունակում է մի քանի շերտ՝ մուտքային (Input layer), փաթույթային (Convolutional layer), ակտիվացման (ReLU layer), միավորող (Pooling layer), հարթեցնող (Flattening), լրիվ կապված (Fully connected layer) (նկ. 3.5):



Նկ. 3.5 CNN ցանցի կառուցվածքը

Ցանցի մուտքային շերտին փոխանցվում է գունավոր նկար երեք գունային բաղադրիչներով (նկ. 3.6):

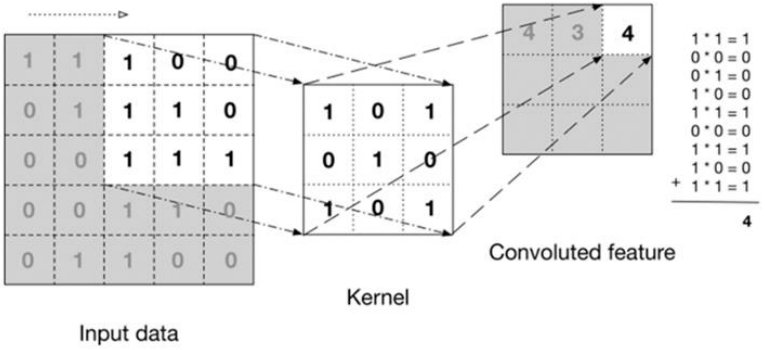


Նկ. 3.6 Գունավոր նկարի գունային բաղադրիչները

CNN-ում առաջին շերտը՝ փաթույթայինը, միշտ սեղմող է: Փաթույթային շերտի տեղեկատվությունը վերցվում է նախորդ

շերտից: Կատարվում է փաթույթի գործողություն, և ստացված արդյունքը պատկերվում է հայտանիշների քարտեզի վրա (Feature map): Փաթույթի գործողությունը կատարվում է մուտքային պատկերի և ֆիլտրի (միջուկ, kernel) միջև:

Ֆիլտրը մի մատրիցա է, որը շարժվում է նկարի վրայով վերևից ներքև, ձախից աջ: Ֆիլտրի տարրերը բազմապատկվում են նկարի համապատասխան դիրքերի պիքսելների արժեքներով և գումարվում: Հաջորդ քայլը ֆիլտրի՝ մեկ պիքսելով շարժումն է դեպի աջ: Ֆիլտրի տեղափոխումը հորիզոնական ուղղությամբ կատարվում է մինչև պատկերի աջ եզրին հասնելը: Այնուհետև ֆիլտրը կրկին տեղադրվում է պատկերի ձախ եզրում՝ մեկ պիքսել դեպի ներքև: Նշված գործողությունները կրկնվում են՝ մինչև ֆիլտրը հասնի պատկերի ներքևի աջ եզրին: Փաթույթի գործողության արդյունքում ստացված արժեքները գրվում են հայտանիշների քարտեզում: Եթե մուտքային նկարի չափերն են 32x32x1, իսկ ֆիլտրի չափը՝ 5x5x1, ապա կստացվի 28x28x1 չափսի մատրիցա (28=32-5+1) (նկ. 3.7):

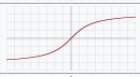
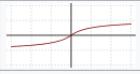
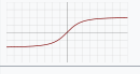
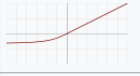
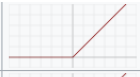
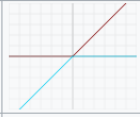


Նկ. 3.7 Հայտանիշների քարտեզ

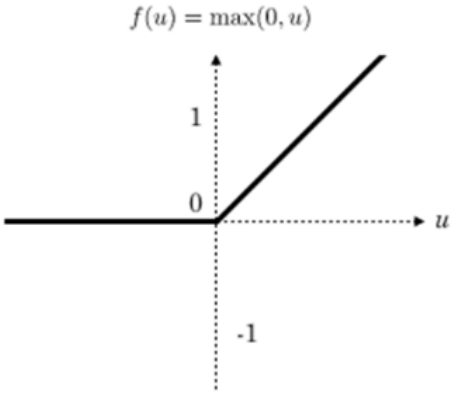
Ֆիլտրը, տեղափոխվելով նկարի վրայով, պատկերում առանձնացնում է հայտանիշներ, որոնք կախված են ֆիլտրի չափից։ Այս գործընթացի արդյունքում ստացված ելքային տվյալների համախումբը տեղեկություն է տալիս պատկերում առկա անկյունների, եզրերի, ուղիղների, կորերի և այլ պարզ մասնիկների մասին։ Որքան շատ ֆիլտրեր օգտագործվեն, այդքան շատ ինֆորմացիա կունենանք նկարում առկա մասնիկների վերաբերյալ։

Ակտիվացման ֆունկցիա (Activation Function)

Հայտանիշների քարտեզի վրա կիրառվում են ակտիվացման ֆունկցիաներ։ Ակտիվացման ֆունկցիան օգտագործում են տվյալները նորմավորելու համար։ Նրա արժեքը որևէ միջակայքից է։ Սովորաբար վերցվում են [0,1] կամ [-1,1] միջակայքերը։ Հաճախ օգտագործվող ակտիվացման ֆունկցիաները, դրանց գրաֆիկները և արժեքների տիրույթը բերված են ստորև։

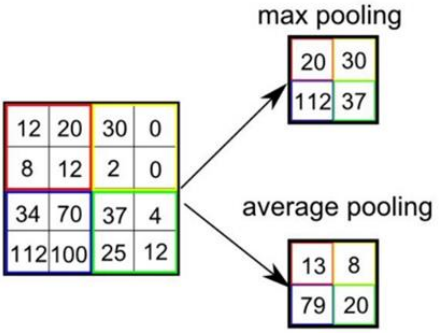
Name	Plot	Equation	Range
Identity		$f(x) = x$	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$\{0, 1\}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$ [1]	$(0, 1)$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$(-1, 1)$
ArcTan		$f(x) = \tan^{-1}(x)$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$
ElliotSig[9][10][11] Softsign[12][13]		$f(x) = \frac{x}{1 + \lvert x \rvert}$	$(-1, 1)$
Inverse square root unit (ISRU)[14]		$f(x) = \frac{x}{\sqrt{1 + \alpha x^2}}$	$\left(-\frac{1}{\sqrt{\alpha}}, \frac{1}{\sqrt{\alpha}}\right)$
Inverse square root linear unit (ISRLU)[14]		$f(x) = \begin{cases} \frac{x}{\sqrt{1 + \alpha x^2}} & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$\left(-\frac{1}{\sqrt{\alpha}}, \infty\right)$
Square Nonlinearity (SQNL)[11]		$f(x) = \begin{cases} 1 & : x > 2.0 \\ x - \frac{x^2}{4} & : 0 \leq x \leq 2.0 \\ x + \frac{x^2}{4} & : -2.0 \leq x < 0 \\ -1 & : x < -2.0 \end{cases}$	$(-1, 1)$
Rectified linear unit (ReLU)[15]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Bipolar rectified linear unit (BReLU)[16]		$f(x_i) = \begin{cases} ReLU(x_i) & \text{if } i \bmod 2 = 0 \\ -ReLU(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$(-\infty, \infty)$
Leaky rectified linear unit (Leaky ReLU)[17]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Parameteric rectified linear unit (PReLU)[18]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$ [2]
Randomized leaky rectified linear unit (RReLU)[19]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \text{ [3]} \\ x & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Exponential linear unit (ELU)[20]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$(-\alpha, \infty)$

Հաճախ օգտագործվում է ReLU (Rectified Linear Units) ոչ գծային ֆունկցիան, որը բոլոր բացասական թվերը փոխարինում է 0-ներով (նկ. 4):

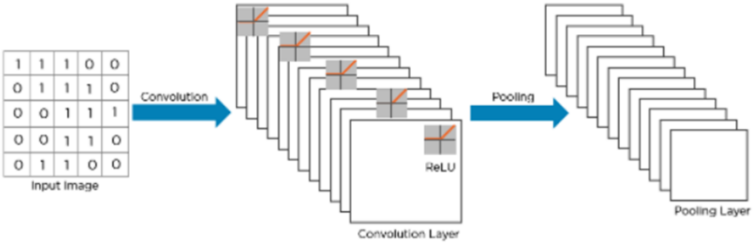


Նկ. 3.8 ReLU(x)=max(0,x)

Հայտանիշների քարտեզը փոխանցվում է ցանցի մյուս շերտին հետագա մշակման համար: Միավորման շերտը (Pooling Layer) իրականացնում է միավորելու գործողություն՝ հայտանիշների քարտեզի վրա դիտարկելով 2*2 չափի ենթամատրիցներ: Միավորման գործողությունը կատարվում է Max Pooling կամ Average Pooling եղանակներով (նկ. 3.9, 3.10): Max Pooling-ը ենթամատրիցներից վերցնում է ամենամեծ արժեքները, Average Pooling-ը՝ միջին արժեքները, այնուհետև ենթամատրիցները փոխարինվում են այդ արժեքներով:



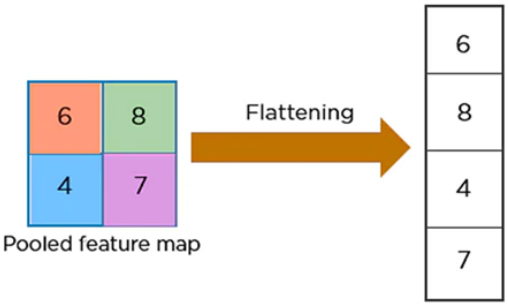
Նկ. 3.9 Pooling գործողության տեսակները



Նկ. 3.10 CNN ցանցի շերտերը

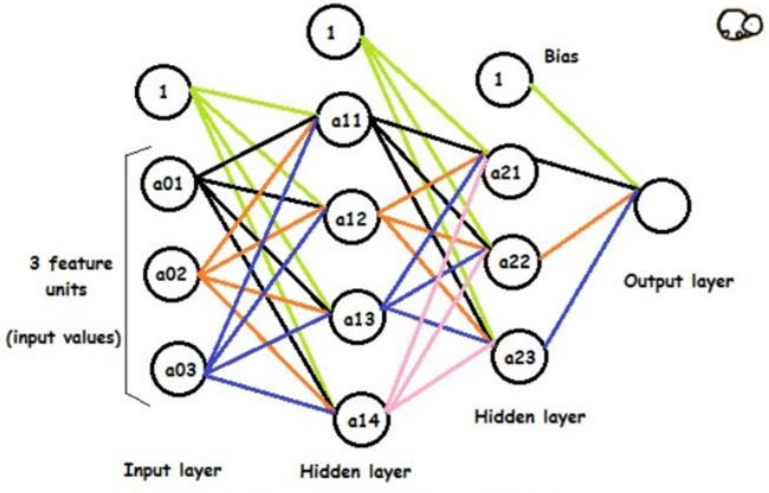
Convolution-ը պատկերում հայտնաբերում է առկա փոքր մասնիկները, իսկ Pooling-ը կարողանում է ճանաչել ավելի խոշոր մասնիկները: Convolution-ի միջոցով ստացված հայտանիշների քարտեզներից այն անցնում է այլ հայտանիշների, օրինակ՝ կետերի փոխարեն հայտնաբերում է գծիկներ և այլն:

Pooling-ի արդյունքի մատրիցը ներկայացվում է մեկչափանի զանգվածի տեսքով (Flattening) (նկ. 3.1) և փոխանցվում ցանցի հաջորդ՝ լրիվ կապված շերտին (Fully-Connected layer):



Նկ. 3.11 Pooling-ի արդյունքների ներկայացումը միաչափ զանգվածի տեսքով

Fully-Connected layer-ը CNN ցանցի վերջին բաղադրիչն է, որը կազմված է որոշ քանակությամբ թաքնված շերտերից: Այն ունի հետևյալ տեսքը (նկ. 3.12).

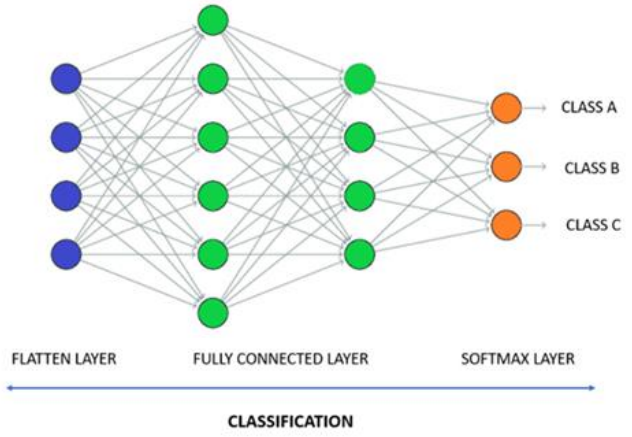


Նկ. 3.12 Լրիվ կապված շերտ

Flattening-ի փուլում ստացված վեկտորը մուտքային վեկտոր է Fully-Connected layer -ի համար: Այստեղ օգտագործվում են գծային ֆունկցիա՝ ազդանշանների կշռված գումարը հաշվելու համար, և ոչ գծային Sigmoid, TanH կամ ReLu ֆունկցիաները՝ որպես ակտիվացման ֆունկցիաներ:

Կախված խնդրից՝ Fully-Connected layer-ին կարելի է ավելացնել ևս մի քանի թաքնված շերտեր: Վերջին թաքնված շերտից ստացված ելքային տվյալները տրվում են Softmax կամ Sigmoid ֆունկցիաներին:

CNN ցանցի Fully-Connected layer բաղադրիչը դասակարգում է պատկերը: Ցանցի ելքային շերտը պարունակում է n նեյրոն: Դասակարգման խնդրում n-ը դասերի քանակն է: i-րդ նեյրոնը որոշում է նկարի՝ i-րդ դասին պատկանելու հավանականությունը: Ակտիվացման ֆունկցիան կիրառվում է ազդանշանների կշռված գումարի համար (նկ. 3.13): sigmoid ֆունկցիան օգտագործվում է, երբ n=2, իսկ n>2 դեպքում կիրառվում է softmax ֆունկցիան:



Նկ. 3.13 Պատկերի դասակարգում

3.2.2 Թվանշանների դասակարգում CNN ցանցում

Այժմ CNN-ը օգտագործենք 0-ից 9 ձեռագիր թվանշանների պատկերները ճանաչելու համար: Ցանցի ուսուցման և թեստավորման համար կօգտվենք MNIST տվյալների հավաքածուից, որը պարունակում է 0-ից 9 ձեռագիր թվանշանների 28x28 չափանի 60 հազար պատկեր (ուսուցման հավաքածու) և 10 հազար պատկեր (թեստային հավաքածու):

Ստորև բերված կոդի միջոցով կարելի է արտածել MNIST հավաքածուի ուսուցման և թեստային տվյալների չափերը:

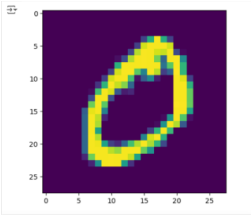
Օրինակ 3.15

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print('Ուսուցման տվյալների հավաքածու', X_train.shape)
print('Ուսուցման տվյալների նշիչների հավաքածու', y_train.shape)
print('Թեստային  տվյալների հավաքածու', X_test.shape)
print('Թեստային տվյալների նշիչների հավաքածու', y_test.shape)
```

```
Ուսուցման տվյալների հավաքածու (60000, 28, 28)
Ուսուցման տվյալների նշիչների հավաքածու (60000,)
Թեստային  տվյալների հավաքածու (10000, 28, 28)
Թեստային տվյալների նշիչների հավաքածու (10000,)
```

Արտածենք X_train հավաքածուի առաջին թվանշանի տեսքը հետևյալ հրամանների միջոցով (նկ. 3.14):

```
import matplotlib.pyplot as pl
plt.imshow(X_train[1])
plt.show()
```

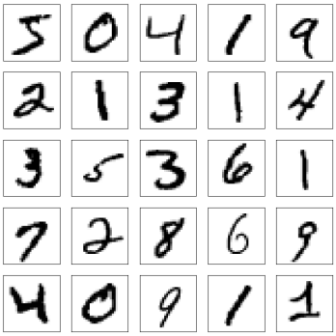


Նկ. 3.14 X_train հավաքածուի առաջին թվանշանը

Հետևյալ կոդի օգնությամբ արտածվում են ուսուցման հավաքածուի առաջին 25 թվանշանների պատկերները (նկ. 3.15)։

```
plt.figure(figsize=(10, 10))
for i in range(25):
  plt.subplot(5, 5, i + 1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(X_train[i], cmap=plt.cm.binary)
plt.show()
```

Կոդի կատարման արդյունքում ստացվում է՝



Նկ. 3.15 Ուսուցման հավաքածուի առաջին 25 թվանշանների պատկերները

Քանի որ մենք դիտարկում ենք թվանշանների սև/սպիտակ պատկերները, ուստի պետք է X_train և X_test հավաքա-

ծուներում պիքսելների գույնի խորությունը տանք հավասար մեկի՝ ի տարբերություն RGB պատկերների (գունավոր պատկերներ), որտեղ գույնի խորությունը երեք է: Դա արվում է հետևյալ հրամաններով.

```
X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)
```

reshape ֆունկցիաներում առաջին պարամետրը պատկերների քանակներն են X_train և X_test հավաքածուներում՝ 60000 և 10000, երկրորդ և երրորդ պարամետրերը նշում են պատկերի չափը (28*28), վերջին պարամետրով տրվում է գույնի խորությունը:

Անհրաժեշտ է նաև ուսուցման և թեստային հավաքածուներում ձևափոխել տվյալների պիտակները (labels)՝ կոդավորելով դրանք տասը տարրից կազմված պիտակի կոդերով (նկ. 3.16):

Թվանշանի պիտակ	Թվանշանի պիտակի կոդ
0	1000000000
1	0100000000
2	0010000000
3	0001000000
4	0000100000
5	0000010000
6	0000001000
7	0000000100
8	0000000010
9	0000000001

Նկ. 3.16 Թվանշանների պիտակները և դրանց կոդերը

Այդ ձևափոխությունը կատարվում է հետևյալ հրամաններով.

```
from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Արտածենք y_train և y_test հավաքածուների չափերը և X_train հավաքածուի առաջին երեք թվանշանների պիտակների կոդերը:

```
print('Ուսուցման տվյալների պիտակները ձևափոխությունից հետո', y_train.shape)
print('Թեստային  տվյալների պիտակները ձևափոխությունից հետո', y_test.shape)
y0 = y_train[0]
y1 = y_train[1]
y2 = y_train[2]
print(y0, y1, y2, sep='\n')
```

```
Ուսուցման տվյալների պիտակները ձևափոխությունից հետո (60000, 10)
Թեստային  տվյալների պիտակները ձևափոխությունից հետո (10000, 10)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

Ինչպես երևում է արտածման արդյունքից, X_train հավաքածուի առաջին երեք թվանշաններն են 5, 0 և 4:

Այժմ կառուցենք CNN ցանց՝ օգտագործելով Keras գրադարանի Sequential հաջորդական մոդելի ճարտարապետությունը, որը հնարավորություն է տալիս ցանցը կառուցել շերտ առ շերտ: Ցանցին նոր շերտ ավելացնում ենք add() ֆունկցիայով:

Օրինակ 3.16

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
input_shape = (28, 28, 1)
batch_size = 64
# Մոդելի ստեղծում
model= Sequential()
# առաջին փաթույթային շերտ
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
# երկրորդ փաթույթային շերտ
model.add(Conv2D(32, kernel_size=3, activation='relu'))
# լրիվ կապակցված ցանցի համար մուտքային վեկտորի ստեղծում
model.add(Flatten())
# միաշերտ պերցեպտրոնի ստեղծում
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Բերված ծրագրի կատարման շնորհիվ ստեղծվում է CNN ցանց՝ հետևյալ բնութագրիչներով: Ցանցն ունի երկու փաթույթային Conv2D շերտ: Առաջին շերտը պարունակում է 64 հանգույց, երկրորդը՝ 32: Այդ քանակները կարելի է փոխել: kernel_size պարամետրի արժեքը տալիս է ֆիլտրի չափը՝ 3 x 3:

Առաջին երկու շերտերը ակտիվացնելու համար օգտագործվում է ReLU ֆունկցիան: Առաջին շերտի համար input_shape=(28, 28, 1) պարամետրով տրվում է մուտքային պատկերի չափը՝ 28*28 և գույնի խորությունը՝ 1:

Փաթույթային (Conv2D) շերտերից հետո հարթեցման (Flatten) շերտն է, որը երկչափ զանգվածը ձևափոխում է միաչափի:

Dense-ը կիրառվում է ելքային տվյալներ ստանալու համար: Ըստ էության դա պերցեպտրոն է, որը ելքային շերտում ունի 10 նեյրոն՝ յուրաքանչյուր թվանշանի համար մեկական : softmax–ը պերցեպտրոնի ակտիվացման ֆունկցիան է:

Մոդելի կառուցվածքը տալուց հետո այն կոմպիլացվում է model.compile() ֆունկցիայով: Կոմպիլացիայի համար օգտա-

գործվում է երեք պարամետր՝ օպտիմիզատոր (optimizer), կորստի ֆունկցիա (loss) և գնահատման չափանիշներ (metrics): Այստեղ adam օպտիմիզատորը հետևում է ուսուցման արագությանը, որը որոշում է, թե մոդելի համար որքան արագ են հաշվարկվում օպտիմալ կշիռները: Ուսուցման արագության փոքր արժեքների դեպքում կշիռները որոշվում են ավելի ճշգրիտ (մինչև որոշակի սահման), սակայն դրա համար ծախսվում է ավելի շատ ժամանակ: Որպես կորստի ֆունկցիա օգտագործվում է categorical_crossentropy ֆունկցիան: Մոդելի որակը գնահատվում է ըստ ճշտության ցուցանիշի (accuracy):

Մոդելի բնութագրիչները կարող ենք ստանալ հետևյալ հրամանի միջոցով.

```
[28] model.summary()
```

```
Model: "sequential_17"
```

Layer (type)	Output Shape	Param #
conv2d_32 (Conv2D)	(None, 26, 26, 64)	640
conv2d_33 (Conv2D)	(None, 24, 24, 32)	18,464
flatten_16 (Flatten)	(None, 18432)	0
dense_16 (Dense)	(None, 10)	184,330

```
Total params: 203,434 (794.66 KB)
Trainable params: 203,434 (794.66 KB)
Non-trainable params: 0 (0.00 B)
```

Կառուցված ցանցի ուսուցումը կատարվում է fit() ֆունկցիայով, իսկ արդյունքը պահպանվում է hist փոփոխականում:

```
hist = model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size = 64, epochs = 1)
print(hist.history)
```

fit() ֆունկցիան ունի մի շարք պարամետրեր: Դրանցից են՝ X_train ուսուցման պատկերները, y_train կոդավորված պիտակները, X_test թեստային պատկերները, y_test կոդավորված պիտակները, ուսուցման ցիկլերի քանակը (epochs):

Կառուցված մոդելի աշխատանքի որակը գնահատվում է թեստային հավաքածուի վրա հետևյալ կոդի միջոցով.

```
score = model.evaluate(X_test, y_test, verbose = 0)
print("Test loss: ", score[0])
print("Test accuracy: ", score[1])
```

```
Test loss:  0.10581579804420471
Test accuracy:  0.9717000126838684
```

Նկատենք, որ նույնիսկ ուսուցման մեկ ցիկլի դեպքում թեստային տվյալների վրա ստացվում են բավարար արդյունքներ՝ 97.17% ճշտություն։ Որքան մեծ լինի ուսուցման ցիկլերի քանակը, այնքան ավելի լավ կլինի ուսուցման արդյունքը։ Սակայն լավացումը տեղի կունենա մինչև որոշակի մակարդակ։

Այժմ ուսուցանենք մոդելը երեք ցիկլով և կրկին գնահատենք նրա աշխատանքի որակը։

Օրինակ 3.17

```
input_shape = (28, 28, 1)
batch_size = 64

model= Sequential()
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.summary()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])


hist = model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size = 64, epochs = 3)
print(hist.history)

score = model.evaluate(X_test, y_test, verbose = 0)
print("Test loss: ", score[0])
print("Test accuracy: ", score[1])
```

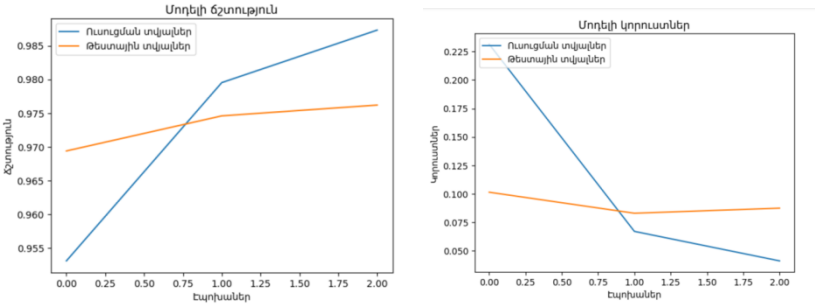
```
Test loss:  0.08748841285705566
Test accuracy:  0.9761999845504761
```

Նկատենք, որ այս դեպքում կորստի արժեքը փոքրացել է, իսկ ճշտությունը՝ մեծացել:

Ուսուցման արդյունքը վիզուալիզացվում է հետևյալ ծրագրային կոդի միջոցով (նկ. 3.17):

Օրինակ 3.18

```
# Կառուցել կանխատեսման ճշտության գրաֆիկը
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Մոդելի ճշտություն')
plt.ylabel('ճշտություն')
plt.xlabel('էպոխաներ')
plt.legend(['Ուսուցման տվյալներ', 'Թեստային տվյալներ'], loc = 'upper left')
plt.show()

# Կորուստների գրաֆիկի կառուցում (սխալների)
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Մոդելի կորուստներ')
plt.ylabel('Կորուստներ')
plt.xlabel('էպոխաներ')
plt.legend(['Ուսուցման տվյալներ', 'Թեստային տվյալներ'], loc = 'upper left')
plt.show()
```



Նկ. 3.17 Մոդելի որակի կախվածությունը ուսուցման փուլերի քանակից

Ուսուցման երեք ցիկլից հետո մոդելը դրսևորել է 98% ճշտություն ուսուցման հավաքածուի վրա, 97%՝ թեստային

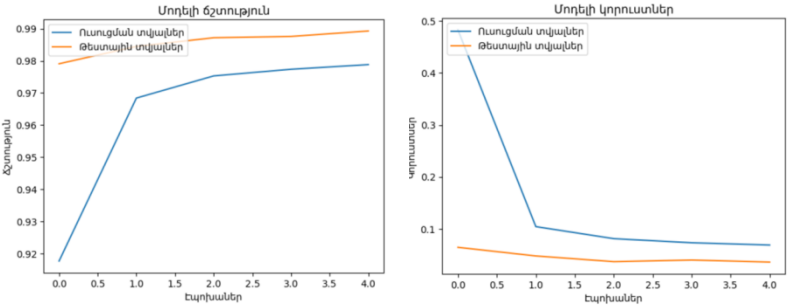
հավաքածուի: Այդ ցուցանիշները ավելի բարձր են, քան ուսուցման մեկ ցիկլի դեպքում:

Գրաֆիկներից երևում է, որ մոդելը չի զուգամիտում, այսինքն՝ կանխատեսման ճշտությունը ուսուցման և թեստային հավաքածուների վրա զգալիորեն տարբերվում է: Հետևաբար պետք է նեյրոնային ցանցի մոդելում կատարել փոփոխություններ: Առաջին և երկրորդ փաթույթային շերտերից հետո ավելացնենք MaxPooling2D շերտ: Դա անհրաժեշտ է մոդելի պարամետրերի քանակը նվազեցնելու համար: Երկրորդ փաթույթային շերտում նեյրոնների քանակը սահմանենք 128, ուսուցման ցիկլերի քանակը՝ 5: Ուսուցանենք ցանցը և կառուցենք ճշտության ու կորստի ֆունկցիաների գրաֆիկները:

Օրինակ 3.19

```
from keras.models import Sequential
#from keras import layers
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
input_shape=(28,28,1)
batch_size = 64
model = Sequential(
  [
      Conv2D(64, kernel_size=(3,3), activation='relu', input_shape = input_shape),
      MaxPooling2D(pool_size = (2, 2)),
      Conv2D(128, kernel_size=(3,3), activation='relu'),
      MaxPooling2D(pool_size = (2, 2)),
      Flatten(),
      Dropout(0.5),
      Dense(10, activation="softmax"),
  ]
)

model.summary()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

hist = model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size = 64, epochs = 5)
print(hist.history)
```

Փոփոխված մոդելի աշխատանքի որակի գնահատման գրաֆիկները կունենան հետևյալ տեսքը (նկ. 3.18):



Նկ. 3.18 Մոդելի որակի կախվածությունը ուսուցման փուլերի քանակից

Ինչպես երևում է գրաֆիկներից, մոդելի զուգամիտությունը երևում է ուսուցման չորրորդ ցիկլից հետո։ Հինգերորդ ցիկլում մոդելի ճշտությունն ուսուցման և թեստային հավաքածուների վրա համարյա նույնն է։ Այդպիսի մոդելը արդեն կարող է օգտագործվել։ Բնական է, որ այն հետագայում օգտագործելու համար պետք է պահել որևէ ֆայլում։ Ուսուցանված մոդելը մեր դեպում պահվում է **my_model.h5** ֆայլում։

```
model.save ( 'my_model.h5' )
```

Ֆայլից կարելի է ուսուցանված մոդելը ներբեռնել հետևյալ հրամաններով։

```
from keras.models import load_model
my_new_model = load_model('my_model.h5')
```

Այդ ֆայլը պարունակում է մոդելի ճարտարապետությունը, ուսուցման ընթացքում ձևավորված կշռային գործակիցները, ուսուցման կոնֆիգուրացիան (կորուստը, օպտիմիզատորը), օպտիմիզատորի վիճակը, որոնք թույլ են տալիս շարունակել մոդելի ուսուցումը ընդհատման կետից։

Այժմ կարելի է ուսուցանված մոդելը օգտագործել ձեռագիր թվանշանների պատկերները ճանաչելու համար: Մոդելի օգտագործման համար կարող ենք ձեռագիր թվանշանների պատկերները վերցնել MNIST հավաքածուից: Թվանշանների ճանաչումը կատարվում է predict() ֆունկցիայի միջոցով: Ստորև բերված օրինակում կատարվել է 5, 0, 4 թվանշանների ճանաչումը:

Օրինակ 3.20

```
from tensorflow.keras.models import load_model
import numpy as np

# Ուսուցանված մոդելի բեռնումը ֆայլից
model_New = load_model('my_model.h5')
y_train_pr = np.argmax(model_New.predict(X_train[:3], verbose=0), axis=-1)

# Արդյունքների արտածում
print('Առաջին 3 նիշերը: ', y_train[:3])
print('Առաջին 3 կանխատեսումները:', y_train_pr[:3])
```

Ծրագրի կոդը գործարկելիս արտածվում են 5, 0, 4 թվանշանների դասերի պիտակները կոդավորված և ամբողջ թվերի տեսքով:

```
Առաջին 3 նիշերը:  [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
Առաջին 3 կանխատեսումները: [5 0 4]
```

Նշենք, որ, փոխելով ուսուցման արագությունը, թաքնված շերտերի քանակը, փաթույթային գործողությունները, ակտիվացման ֆունկցիաները, կարելի է ընդլայնել Keras գրադարանի միջոցով կառուցված նեյրոնային ցանցի մոդելը այլ խնդիրների լուծման համար:

3.3 TensorFlow գրադարան

TensorFlow գրադարանը մշակել է Google-ը: TensorFlow-ի միջոցով ստեղծվում են արհեստական բանականության, մեքենայական ուսուցման հետ կապված բարդ նախագծեր: TensorFlow գրադարանն աշխատում է այլ գրադարանների հետ, մասնավորապես Keras-ի: TensorFlow-Keras համադրությունը զգալիորեն հեշտացնում է նեյրոնային ցանցերի կառուցումը և ուսուցումը:

Նեյրոնային ցանցը շերտ առ շերտ կառուցելու համար օգտագործվում է tensorflow.keras փաթեթը: Այդ փաթեթով կառուցված ցանցերը կարելի է կիրառել խորը ուսուցմամբ մոդելներում, պատկերների և ձեռագիր տեքստի ճանաչման համար, բնական լեզուների մշակման (NLP-Natural Languages Processing) խնդիրներում, դեմքի արտահայտությունը կամ ձայնի ինտոնացիան ճանաչելու համար: tensorflow.keras փաթեթում կան նաև բառերի վեկտորացման (embedding), մասնակի դիֆերենցիալ հավասարումների լուծման մոդուլներ:

TensorFlow գրադարանը թույլ է տալիս կատարել հաշվարկներ տարբեր հարթակներում: Այդ գրադարանը հեշտությամբ ինտեգրվում է այլ պլատֆորմներին՝ սերվերներ, մոբայլ սարքեր և այլն:

Դիտարկենք նեյրոնային ցանց և սովորեցնենք նրան կատարել xor ֆունկցիան: Իհարկե, նեյրոնային ցանցի միջոցով xor-ի հաշվարկը գործնականում կիրառական իմաստ չունի, բայց դա կօգնի հասկանալ նեյրոնային ցանցի ուսուցման և օգտագործման հիմնական սկզբունքները:

Օրինակ 3.18-ում ներկայացված է TensorFlow-ի և Keras-ի կիրառմամբ xor ֆունկցիան հաշվող նեյրոնային ցանցի կա-

ռուցման ծրագրային կոդը: Կոդի մեջ անհրաժեշտ գրադարանները ներառելուց հետո կառուցվում է նեյրոնային ցանց Sequential դասի օբյեկտի տեսքով, որն ունի երկու հանգույցով մուտքային շերտ, մեկ նեյրոնով ելքային շերտ և չորս նեյրոնով մեկ թաքնված շերտ:

Օրինակ 3.21

```
import tensorflow as tf
from tensorflow import keras
import numpy as np

# Նեյրոնային ցանցի կառուցումը
model = keras.Sequential(
    [
    keras.layers.Dense(4, input_shape=(2,), activation='relu'),
    # թաքնված շերտ 4 նեյրոնով և մուտքային շերտ 2 նեյրոնով
    keras.layers.Dense(1, activation='sigmoid')  # ելքային շերտ մեկ նեյրոնով
    ])
# Մոդելի կոմպիլյացիա
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Ցանցում օգտագործվում է adam օպտիմիզատորը, որը որոշում է ուսուցման արագությունը (learning rate), binary_crossentropy կորստի ֆունկցիան, որը կիրառվում է բինար դասակարգման խնդիրների համար: Մոդելի գնահատման համար ընտրված է ճշտության (accuracy) չափանիշը (metrics=['accuracy']):

Ցանցի ուսուցումը օգնում է որոշելու ցանցի պարամետրերի արժեքները: Տանք ցանցի X մուտքային և y ելքային տվյալները Numpy զանգվածների տեսքով:

```
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])
```

fit ֆունկցիայի միջոցով ուսուցանվում է մոդելը՝ կատարելով ուսուցման 500 փուլ: Մոդելի աշխատանքը գնահատենք՝ օգտագործելով X հավաքածուն:

Օրինակ 3.22

```
# Մոդելի ուսուցում
model.fit(X, y, epochs=500, verbose=0)  # Ուսուցում 500 էպոխով
# Կանխատեսում X հավաքածուի վրա
predictions = model.predict(X)
print("XOR ցանցի կանխատեսումները")
print(predictions)
# Մոդելի գնահատում
score = model.evaluate(X,y)
print("Test loss: ", score[0])
print("Test accuracy: ", score[1])
```

Ծրագրի կատարման արդյունքում կստանանք

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
XOR ցանցի կանխատեսումները
[[0.3284252 ]
 [0.68580425]
 [0.6756161 ]
 [0.3172804 ]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 1.0000 - loss: 0.3873
Test loss:  0.38727355003356934
Test accuracy:  1.0
```

Ցանցը XOR ֆունկցիայի համար կանխատեսել է արժեքներ (0,1) միջակայքից, որոնք round ֆունկցիայով կարելի է դարձնել 0 կամ 1:

Եթե կառուցված ցանցի աշխատանքի արդյունքները գոհացուցիչ չեն, ապա պետք է փոխել դրա ճարտարապետությունը՝ ավելացնել թաքնված շերտերի քանակը, շերտում նեյրոնների քանակը, պարամետրերի արժեքները և այլն:

Նշենք, որ tensorflow գրադարանի ալգորիթմների օգտագործման դեպքում հաճախ դժվար է նեյրոնային ցանցն ուսուցանել կարճ ժամկետում: tensorflow.keras փաթեթը ցանցի պա-

րամետրերը սկզբնարժեքավորում է պատահական թվերով, և ցանցի ծրագիրը կրկին գործարկելիս արդյունքը կարող է տարբեր լինել։ Ճիշտ ուսուցանված ցանցը կարելի է օգտագործել առանց սահմանափակումների։

Կառուցված ցանցը նոր շերտերի և նեյրոնների ավելացմամբ կարող է ընդլայնվել և կիրառվել բարդ խնդիրների լուծման համար։ Ինչպես արդեն նշվել է, ցանցն ուսուցանելուց հետո նրա պարամետրերը կարելի է պահել և հետագայում օգտագործել արդեն իսկ ուսուցանված նեյրոնային ցանցը։

Այսպիսով ուսումնական ձեռնարկում ներկայացվեցին մեքենայական ուսուցման հիմնական մոդելները՝ նեյրոնային ցանցերը։ Դիտարկվեցին նեյրոնային ցանցերի ճարտարապետությունը, մոդելավորումը և իրականացումը Python ծրագրավորման լեզվով։ Ձեռնարկում նեյրոնային ցանցերի կառուցման համար նաև օգտագործվել են Python-ի մասնագիտացված գրադարանները։ Նշենք, որ գոյություն ունեն նաև նեյրոնային ցանցերի այլ մոդելներ, որոնց ուսումնասիրությունը դուրս է սույն ձեռնարկի շրջանակից։ Դրանց մասին տեղեկատվություն կարելի է ստանալ այլ աղբյուրներից։

РЕЗЮМЕ

МАШИННОЕ ОБУЧЕНИЕ С ПРИМЕНЕНИЕМ ЯЗЫКА PYTHON

Саркисян С.Г., Овакимян А.С., Акопян С.Р.

В учебном пособии описываются концепции искусственного интеллекта (ИИ) и методы разработки приложений машинного обучения, которые являются основными компонентами ИИ. Пособие предназначено для студентов факультета информатики и прикладной математики. Также оно может быть полезным как для начинающих программистов, так и для опытных специалистов, планирующих разрабатывать модели машинного обучения на языке программирования Python.

Предоставленный в пособии учебный материал необходим для понимания нейронных сетей, их структуры, методов обучения и программных реализаций. В пособии описаны основные методы и инструменты, необходимые для разработки нейронных сетей.

В пособии приведены многочисленные практические примеры, которые позволят студентам применить свои знания при программировании на языке Python. Представлены программные коды, помогающие понять, как устроены и работают нейронные сети. В примерах также использованы основные классы популярных библиотек ИИ, таких как Scikit-Learn, Keras, TensorFlow, которые помогут студентам пользоваться современными программными инструментами.

SUMMARY

MACHINE LEARNING USING PYTHON

Sargsyan S.G., Hovakimyan A.S., Hakobyan S.R.

This tutorial introduces concepts and methods related to artificial intelligence (AI), focusing on the development of machine learning applications, which are a fundamental aspect of AI. It is designed for students in the Faculty of Computer Science and Applied Mathematics. Additionally, it can serve as a valuable resource for both novice programmers and experienced professionals who aim to create machine learning models using the Python programming language.

The educational material in the manual is essential for understanding neural networks, their structure, training methods, and implementations. It describes the key methods and tools necessary for developing neural networks.

The manual includes a variety of practical examples that enable students to apply their programming knowledge in Python. It presents program codes that illustrate how neural networks are structured and work. Additionally, the examples incorporate the primary classes from popular artificial intelligence libraries such as scikit-learn, Keras, and TensorFlow, providing students with the opportunity to utilize modern software tools effectively.

Գրականություն

1. Джоши Пратик, Искусственный интеллект с примерами на Python."Диалектика", 2019, 448 с.

2. Мюллер А. П., Гвидо С. Введение в машинное обучение с помощью Python: пер. с англ. М.: Вильяме, 2017, 480 с.

3. Aurelien Geron, Hands-on Machine Learning with Scikit-Learn, Keras&TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems. 2-nd Edition, O'Relly, 2019.

4. Н.Ю. Золотых, Машинное обучение и анализ данных (Machine Learning and Data Mining), 2016, http://www.uic.unn.ru/~zny/ml

5. И.В.Левченко и др.,Основы искусственного интеллекта :учеб.пособие. М. : Образование и информатика, 2019, 95 с.

6. В.А. Касторнова, Системы искусственного интеллекта как технологическая основа решения задач обучения на примере предметной области, 2018.

7. Уэс Маккини. Python и анализ данных.Изд-во ДМК Пресс, М., 2020.

8. Коул Анирад, Ганджу Сиддха, Казам Мехер, Искусственный интеллект и компьютерное зрение. Реальные проекты на Python, Keras и TensorFlow, пер. с англ. СПб.: Питер, 2023, 624 с.

9. Прохоренок Н. А., Дронов В. А. Python 3 и PyQt 5. Разработка приложений. -СПб:: БХВ-Петербург, 2016, 832 с.

10. Рашка С. Python и машинное обучение: пер. с англ. -М.: ДМК Пресс, 2017. 418 с.

11. Реза Босаг З. Р., Бхарат Р. TensorFlow для глубокого обучения: пер. с англ. СПб.: БХВ-Петербург, 2020, 256 с.

12. Шолле Ф. Глубокое обучение на Python: пер. с англ. - СПб.: Питер, 2018, 400 с.

13. www.williamspublishing.com Введение в машинное обучение с помощью Python.

ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

ՍԻՐԱՆՈՒՇ ԳԵՂԱՄԻ ՍԱՐԳՍՅԱՆ,
ԱՆՆԱ ՍԵԴՐԱԿԻ ՀՈՎԱԿԻՄՅԱՆ,
ՍՈՆԻԿ ՌՈՒԲԵՆԻ ՀԱԿՈԲՅԱՆ

ՄԵՔԵՆԱՅԱԿԱՆ ՈՒՍՈՒՑՈՒՄ PYTHON ԼԵԶՎԻ ԿԻՐԱՌՄԱՄԲ

ՈՒՍՈՒՄՆԱԿԱՆ ՁԵՌՆԱՐԿ

Համակարգչային ձևավորումը՝ Կ. Չալաբյանի
Կազմի ձևավորումը՝ Ա. Պատվականյանի
Հրատ. խմբագրումը՝ Լ. Ավետիսյանի

Հեղինակները հաստատում են, որ ծանոթ են «ԵՊՀ գրահրատարակչական քաղաքականությանը», և գրքում առկա փաստերը, դիրքորոշումները, կարծիքները շարադրված են հեղինակային իրավունքի և էթիկայի միջազգայնորեն ընդունված սկզբունքների պահպանմամբ:

Ստորագրված է տպագրության՝ 13.06.2025:
Չափսը՝ 60x84 $^{1}/_{16}$: Տպ. մամուլը՝ 10:
Տպաքանակը՝ 100:

ԵՊՀ հրատարակչություն
ք. Երևան, 0025, Ալեք Մանուկյան 1
www.publishing.ysu.am