<<Քվանտ>> վարժարան

Հետազոտական աշխատանք

Թեմա՝ Նեյրոնային Ցանցեր

Դասարան՝ 9-4

Առարկա՝ Ինֆորմատիկա

Աշակերտներ՝ Հովհաննիսյան Վահագն, Գևորգյան Գագիկ

Գիտական ղեկավար՝ Ռուզաննա Արշակյան

Բովանդակություն

[Նեյրոնային Ցանց 4](#_Toc102119122)

[Մի փոքր պատմություն 4](#_Toc102119123)

[Նեյրոն 5](#_Toc102119124)

[Սինապս 5](#_Toc102119125)

[Նեյրոնային Ցանցը՝ որպես ամբողջություն 6](#_Toc102119126)

[Նեյրոնային Ցանցերի Օգտագործումը 6](#_Toc102119127)

[Դասակարգում 6](#_Toc102119128)

[Կանխատեսում 6](#_Toc102119129)

[Ճանաչում 6](#_Toc102119130)

[Ինչպես են աշխատում Նեյրոնային Ցանցերը 7](#_Toc102119131)

[Շերտերը 7](#_Toc102119132)

[Ակտիվացման Ֆունկցիա 7](#_Toc102119133)

[Ինչ է ակտիվացման ֆունկցիան 7](#_Toc102119134)

[Ակտիվացման ֆունկցիայի տեսակները 7](#_Toc102119135)

[Գծային ֆունկցիա 7](#_Toc102119136)

[Սիգմոիդ 8](#_Toc102119137)

[Ռելու 8](#_Toc102119138)

[Հիպերբոլիկ Տանգենս (tanh) 9](#_Toc102119139)

[Կորստի ֆունկցիա 9](#_Toc102119140)

[Ինչ է կորստի ֆունկցիան 9](#_Toc102119141)

[Կորստի Ֆունկցիայի Տեսակները 10](#_Toc102119142)

[Միջին Քառակուսային Սխալ (MSE) 10](#_Toc102119143)

[Cross-entropy 10](#_Toc102119144)

[Exponential 10](#_Toc102119145)

[Օպտիմիզացման համակարգ 11](#_Toc102119146)

[Գրադիենտի իջեցում 11](#_Toc102119147)

[Ստոխաստիկ Գրադիենտի իջեցում 12](#_Toc102119148)

[Խմբային Գրադիենտի իջեցում 12](#_Toc102119149)

[AdaGrad 12](#_Toc102119150)

[AdaM 12](#_Toc102119151)

[Նեյրոնային Ցանցի ձևերը 13](#_Toc102119152)

[Q-Learning 13](#_Toc102119153)

[Q-աղյուսակ 13](#_Toc102119154)

[Learning rate 14](#_Toc102119155)

[Գամմա 14](#_Toc102119156)

[Մարկովի հատկությամբ որոշման ընթացք 15](#_Toc102119157)

[Deep Q-learning 16](#_Toc102119158)

[Experience Replay 17](#_Toc102119159)

[Փաթեթային նեյրոնային ցանց 18](#_Toc102119160)

[Ներածություն 18](#_Toc102119161)

[Ներմուծում և արտածում 18](#_Toc102119162)

[Կառուցվածք 19](#_Toc102119163)

[Փաթեթային շերտ 19](#_Toc102119164)

[Պուլինգի շերտ 20](#_Toc102119165)

[Ամբողջությամբ կապակցված շերտ 21](#_Toc102119166)

[NEAT 22](#_Toc102119167)

[Գենետիական ալգորիթմ 22](#_Toc102119168)

[Գենետիկական գործողություններ 22](#_Toc102119169)

[Կրոսինգովեր 22](#_Toc102119170)

[Մուտացիա 23](#_Toc102119171)

[Ֆիթնես ֆունկցիա 23](#_Toc102119172)

[Պրոյեկտներ 24](#_Toc102119173)

[Շան և կատվի տարբերակում 24](#_Toc102119174)

[Նեյրոնային ցանց 24](#_Toc102119175)

[Արդյունքներ 27](#_Toc102119176)

[Ձեռագիր թվերի ճանաչում 28](#_Toc102119177)

[Նեյրոնային ցանցը 28](#_Toc102119178)

[Արդյունքներ 31](#_Toc102119179)

[Google Dino 32](#_Toc102119180)

[Նեյրոնային ցանց 32](#_Toc102119181)

[Արդյունքներ 33](#_Toc102119182)

[Flappy Bird Խաղի անցում 34](#_Toc102119183)

[Նեյրոնային ցանց 34](#_Toc102119184)

[Արդյունքներ 35](#_Toc102119185)

[Ինքնակառավարվող Մեքենա 36](#_Toc102119186)

[Նեյրոնային ցանց 37](#_Toc102119187)

[Արդյունքներ 41](#_Toc102119188)

[Օգտագործված Գրականություն 42](#_Toc102119189)

# 

# Նեյրոնային Ցանց

## Մի փոքր պատմություն

«Նեյրոնային ցանց» տերմինը հայտնվել է 20-րդ դարի կեսերին։ Առաջին աշխատանքները, որոնցում ստացվել են այս ուղղությամբ հիմնական արդյունքները, իրականացվել են ՄակՔալոքի և Փիթսի կողմից։ 1943 թվականին նրանք մշակեցին նեյրոնային ցանցի համակարգչային մոդել՝ հիմնված մաթեմատիկական ալգորիթմների և ուղեղի գործունեության տեսության վրա։ Նրանք առաջ քաշեցին այն ենթադրությունը, որ նեյրոնները կարելի է դիտարկել որպես սարքեր, որոնք գործում են երկուական թվերով։ Ինչպես իրենց կենսաբանական նախատիպը, ՄակՔալոք-Փիթսի նեյրոնները կարողացան սովորել՝ կարգավորելով սինապսային հաղորդունակությունը նկարագրող պարամետրերը:

1949 թվականին կանադացի հոգեբան Հեբը մտքեր է արտահայտել ուղեղում նեյրոնների կապի բնույթի և դրանց փոխազդեցության մասին։ Նա առաջինն էր, ով առաջարկեց, որ ուսուցումը հիմնականում բաղկացած է սինապսային կապերի ուժի փոփոխություններից: 1954 թվականին Մասաչուսեթսի տեխնոլոգիական ինստիտուտում, օգտագործելով համակարգիչներ, Ֆարլին և Քլարկը մշակեցին Հեբի  ցանցի իմիտացիա:

1957 թվականին Ռոզենբլատը մշակել է ուղեղի կողմից տեղեկատվության ընկալման մաթեմատիկական և համակարգչային մոդելներ՝ հիմնված երկշերտ նեյրոնային ցանցի վրա։ Մարզումների ժամանակ այս ցանցն օգտագործում էր գումարման և հանման թվաբանական գործողությունները։

Նեյրոնային ցանցերի ուսումնասիրության նկատմամբ հետաքրքրությունը մարեց 1969 թվականին Մինսկու և Փեյպերտի մեքենայական ուսուցման վերաբերյալ աշխատության հրապարակումից հետո: Նրանք հայտնաբերել են հիմնական հաշվողական խնդիրները, որոնք առաջանում են արհեստական ​​նեյրոնային ցանցերի համակարգչային ներդրման ժամանակ։ Առաջին խնդիրն այն էր, որ միաշերտ նեյրոնային ցանցերը չէին կարող կատարել «XOR» ֆունկցիան։

Այնուամենայնիվ, 2006 թվականին առաջարկվել են մեկ կամ մի քանի շերտերով նեյրոնային ցանցերի մի քանի  այսպես կոչված Deep Q-Learning ալգորիթմներ:

Ինչպես շատ այլ դեպքերում, այնպես էլ բարձր բարդության խնդիրները պահանջում են դրանց լուծման ոչ թե մեկ, այլ մի քանի մեթոդների կիրառում: Նեյրոնային ցանցերը բացառություն չեն: Նրանք օգտագործվում են, օրինակ, կառավարման համակարգերի պարամետրերը կարգավորելու համար։ Ընդհանուր առմամբ, կասկած չկա արհեստական ​​ինտելեկտի մեթոդների հետագա ինտեգրման մեջ իրենց մեջ և խնդիրների լուծման այլ մեթոդների հետ։

## Նեյրոն

Նեյրոնը հաշվողական միավոր է, որը տեղեկատվություն է ստանում, պարզ հաշվարկներ է կատարում դրա վրա և փոխանցում: Դրանք բաժանվում են երեք հիմնական տեսակի՝ մուտքային, թաքնված և ելքային: Այն դեպքում, երբ նեյրոնային ցանցը բաղկացած է մեծ թվով նեյրոններից, ներմուծվում է շերտ տերմինը։ Համապատասխանաբար, կա մուտքային շերտ, որը ստանում է տեղեկատվություն, թաքնված շերտեր, որոնք մշակում են այն, և ելքային շերտ, որը ցուցադրում է արդյունքը: Նեյրոններից յուրաքանչյուրն ունի 2 հիմնական պարամետր՝ մուտքային տվյալներ (input data) և ելքային տվյալներ (output data)։ Մուտքային նեյրոնի դեպքում՝ input=output։ Մնացած մասում մուտքագրման դաշտը պարունակում է նախորդ շերտի բոլոր նեյրոնների ընդհանուր տեղեկատվությունը, որից հետո այն նորմալացվում է՝ օգտագործելով ակտիվացման ֆունկցիան։

Նեյրոնը կատարում է պարզագույն գործողություն՝

Որտեղ wi – սինապսի կշիռը, xi – նեյրոնի արժեքը, b – կոնստանտ, որը նույնպես փոփոխվում է նեյրոնային ցանցի սովորելու ընթացքում։

## Սինապս

Սինապսը կապ է երկու նեյրոնների միջև։ Սինապսներն ունեն 1 պարամետր՝ քաշ։ Նրա շնորհիվ մուտքային տեղեկատվությունը փոխվում է, երբ այն փոխանցվում է մի նեյրոնից մյուսը։ Ենթադրենք կան 3 նեյրոններ, որոնք տեղեկատվություն են փոխանցում հաջորդին։ Այնուհետև մենք ունենք 3 կշիռ, որոնք համապատասխանում են սինապսներից յուրաքանչյուրին: Ավելի մեծ քաշ ունեցող սինապսի համար այդ տեղեկատվությունը գերիշխող կլինի հաջորդ նեյրոնում: Նեյրոնային ցանցի կշիռների հավաքածուն կամ քաշի մատրիցը ամբողջ համակարգի ուղեղն է: Այդ կշիռների շնորհիվ է, որ մուտքային տեղեկատվությունը մշակվում և վերածվում է արդյունքի։

## Նեյրոնային Ցանցը՝ որպես ամբողջություն

Նեյրոնային ցանցը նեյրոնների հաջորդականություն է, որոնք միացված են սինապսներով: Նեյրոնային ցանցի կառուցվածքը եկել է անմիջապես կենսաբանությունից: Այս կառուցվածքի շնորհիվ մեքենան ձեռք է բերում տարբեր տեղեկություններ վերլուծելու ունակություն։ Նեյրոնային ցանցերը նաև ունակ են ոչ միայն վերլուծել մուտքային տեղեկատվությունը, այլև վերարտադրել այն իրենց հիշողությունից։ Այլ կերպ ասած, նեյրոնային ցանցը մարդու ուղեղի մեքենայական մեկնաբանությունն է, որը պարունակում է միլիոնավոր նեյրոններ, որոնք տեղեկատվություն են փոխանցում էլեկտրական իմպուլսների տեսքով։

# Նեյրոնային Ցանցերի Օգտագործումը

Նեյրոնային ցանցերը օգտագործվում են բարդ խնդիրներ լուծելու համար, որոնք պահանջում են վերլուծական հաշվարկներ, որոնք նման են մարդու ուղեղին: Նեյրոնային ցանցերի ամենատարածված կիրառություններն են.

Դասակարգում

Տվյալների բաշխում ըստ պարամետրերի: Օրինակ, մարդկանց մի խումբ տրվում է որպես մուտքագրում, և դուք պետք է որոշեք, թե նրանցից ում տալ վարկ: Այս աշխատանքը կարող է կատարվել նեյրոնային ցանցի միջոցով՝ վերլուծելով այնպիսի տեղեկությունները, ինչպիսիք են՝ տարիքը, վճարունակությունը, վարկային պատմությունը և այլն։

## Կանխատեսում

Հաջորդ քայլը կանխատեսելու ունակություն: Օրինակ՝ բաժնետոմսերի աճը կամ անկումը` ելնելով ֆոնդային շուկայում տիրող իրավիճակից:

## Ճանաչում

Ներկայումս ամենատարածված նեյրոնային ցանցն է: Օգտագործվում է, օրինակ, Google-ում, երբ որոնում եք լուսանկար կամ հեռախոսի տեսախցիկներում, երբ այն հայտնաբերում է ձեր դեմքի դիրքը և ընդգծում այն ​​և ավելին։

# Ինչպես են աշխատում Նեյրոնային Ցանցերը

Ինչպես նշվեց վերևում, նեյրոնային ցանցերը պարունակում են շերտեր։ Դրանք են՝ 1 մուտքային շերտ, n թաքնված շերտ (n>=0) և 1 ելքային շերտ։ Ներկայացնենք դրանք։

## Շերտերը

Մուտքային շերտը նեյրոնային ցանցի առաջին շերտն է։ Այն պարունակում է ֆիքսված թվով նեյրոններ (կախված մուտքի չափից)։ Այն վերցնում է տեղեկատվությունը միջավայրից և փոխանցում է թաքնված շերտեր, որտեղ տեղեկատվությունը մշակվում է և փոխանցվում է ելքային շերտ, որտեղից էլ, կիրառելով ակտիվացման ֆունկցիան, ստանում ենք պատասխանը, որն այնուհետև ստուգում ենք, և սխալանքը հաշվում ենք կորստի ֆունկցիայի միջոցով։ Այնուհետև, օգտագործելով օպտիմիզացման ֆունկցիաներից որևէ մեկը, անցնում ենք նեյրոնային ցանցի վրայով՝ սկսած վերջից (backpropogation), և փոփոխում ենք սինապսների կշիռներն այնպես, որ ստացվի ավելի լավ կանխատեսում։ Այդ պրոցեսը մի քանի հազար անգամ կատարելուց հետո արդեն նեյրոնային ցանցի սինապսների կշիռները կլինեն օպտիմալ՝ այդ առաջադրանքը կատարելու համար։

## Ակտիվացման Ֆունկցիա

### Ինչ է ակտիվացման ֆունկցիան

Ակտիվացման ֆունկցիան արժեքները նորմալացնելու միջոց է: Այսինքն, եթե մուտքում ունենք մեծ թիվ, այն անցնելով ակտիվացման ֆունկցիայի միջով, կստանանք ելք ձեզ անհրաժեշտ միջակայքում։ Նեյրոնային ցանց Ակտիվացման բազմաթիվ ֆունկցիաներ կան, ուստի մենք կդիտարկենք ամենահիմնականները՝ Գծային, Սիգմոիդ, Ռելու, Հիպերբոլիկ Տանգենս (tanh): Նրանց հիմնական տարբերությունը արժեքների միջակայքն է:

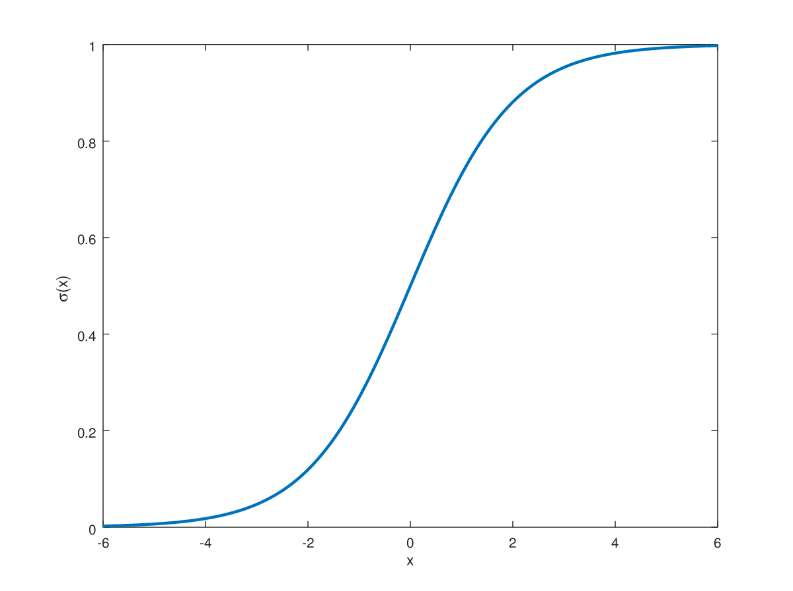
### Ակտիվացման ֆունկցիայի տեսակները

#### Գծային ֆունկցիա

Այս ֆունկցիան գրեթե երբեք չի օգտագործվում, բացառությամբ այն դեպքերի, երբ դուք պետք է փորձարկեք նեյրոնային ցանցը կամ փոխանցեք արժեք առանց փոփոխությունների, քանի որ այն գծային է և ոչ մի փոփոխության չի ենթարկում արժեքը:

#### Սիգմոիդ

Սա ակտիվացման ամենատարածված ֆունկցիան է, դրա արժեքային միջակայքը [0,1] է: Այն ցույց է տալիս ցանցում առկա օրինակների մեծ մասը և երբեմն կոչվում է լոգիստիկ գործառույթ: Համապատասխանաբար, եթե ձեր դեպքում կան բացասական արժեքներ (օրինակ, բաժնետոմսերը կարող են ոչ միայն բարձրանալ, այլև իջնել), ապա ձեզ անհրաժեշտ է ֆունկցիա, որը բացասական արժեքներ նույնպես գրավում է:



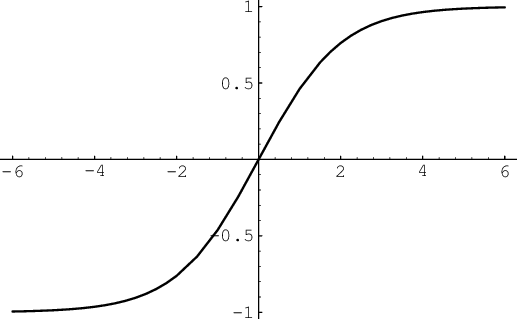
#### Ռելու

Այն նույնպես հաճախ է օգտագործվում, հիմնականում այն դեպքում, երբ կարևորվում է ներմուծվող արժեքի դրական արժեքը։ Օրինակ՝ նկարների մշակման համար կարևոր է պիքսելի գույնը, որը որոշվում է 0-ից 255 ներառյալ արժեքներով։



#### Հիպերբոլիկ Տանգենս (tanh)

Այս ակտիվացման ֆունկցիան ընկած է [-1, 1] միջակայքի վրա։ Այն կիրառվում է այն դեպքում, երբ կարևորվում է ստացվող արժեքի դրական կամ բացասական լինելը, սակայն քիչ է կարևորվում դրա բացարձակ արժեքը։



## Կորստի ֆունկցիա

### Ինչ է կորստի ֆունկցիան

Կորստի ֆունկցիան(Loss function) օգտագործվում է իրական և ստացված պատասխանների միջև եղած սխալը հաշվարկելու համար: Մեր հիմնական նպատակն է նվազագույնի հասցնել այս սխալը: Այսպիսով, կորստի ֆունկցիան արդյունավետորեն մոտեցնում է նեյրոնային ցանցի մարզումն իր նպատակին:

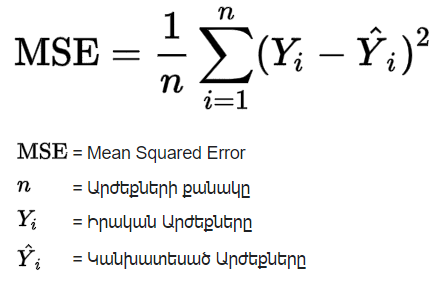
Կորստի ֆունկցիան չափում է, թե որքան լավ է նեյրոնային ցանցն աշխատում տվյալ ուսումնական հավաքածուի և ակնկալվող պատասխանների նկատմամբ: Այն կարող է նաև կախված լինել կշիռներից:

Կորստի ֆունկցիան միաչափ է և վեկտոր չէ, քանի որ այն գնահատում է, թե որքան լավ է գործում նեյրոնային ցանցը որպես ամբողջություն:

### Կորստի Ֆունկցիայի Տեսակները

Որոշ հայտնի կորստի ֆունկցիաներ են.

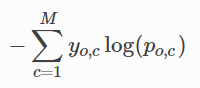
Միջին Քառակուսային Սխալ (MSE)



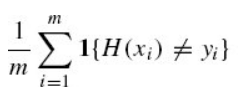
Կորստի ամենապարզ ֆունկցիան է և առավել հաճախ օգտագործվողը։ Կարող է օգտագործվել բոլոր տիպերի նեյրոնային ցանցերի դեպքում, եթե դրանք չեն պահանջում սպեցիֆիկ կորստի ֆունկցիա։

#### Cross-entropy

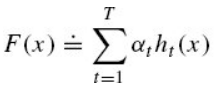
Cross-entropy կորստի ֆունկցիան կամ log loss-ը ֆունկցիա է, որի արժեքների տիրույթն ընկած է [0, ∞) միջակայքում։ Այն օգտագործվում է հիմնականում կլասիֆիկացիայի խնդիրներում։ Ալն վերցնում է ճիշտ պատասխանի հավանականությունը, որը նախապես կանխատեսել է նեյրոնային ցանցը, և հաշվում է դրա ln-ն։



Exponential (AdaBoost)

,

Որտեղ՝ H(x)=sign(F(x)), իսկ

։

Այն օգտագործվում է հիմնականում AdaBoost ալգորիթմի մեջ։

## Օպտիմիզացման համակարգ

Իսկ ինչպե՞ս կարող ենք կշիռները փոփոխել՝ այդ պահին սատցվող ինֆորմացիայից։ Կարող ենք փոխել այնպես, որ այդ պահին կիրառվող թեստային տվյալների համար loss-ը տա 0, սակայն այդ դեպքում կվնասվի արդյունքն այլ թեստային տվյալներում։ Այդ պատճառով մեզ պետք է կոնկրետ ալգորիթմ, որը կփոփոխի այդ արժեքները՝ առանց վնասելու մնացածը։

Խնդիրը կարելի է վերածել մաթեմատիկականի։ Կորստի ֆունկցիան կարող ենք ածանցել ամեն քաշի համար, այնուհետև հաշվել գրադիենտը։ Հիմա մեզ պետք է ընդամենը գտնել տվյալ գրադիենտի գլոբալ մաքսիմումը։ Դա կարելի է անել՝ օգտագործելով հետևյալ ալգորիթմներից մեկը։

### Գրադիենտի իջեցում

Գրադիենտի իջեցումը ամենահիմնական, բայց առավել օգտագործվող օպտիմիզացման ալգորիթմն է: Այն մեծապես օգտագործվում է գծային ռեգրեսիայի և դասակարգման ալգորիթմներում: Նեյրոնային ցանցերում հետ տարածումը (backpropogation) նույնպես օգտագործում է գրադիենտ ծագման ալգորիթմ:

Հետ տարածման միջոցով կորուստը տեղափոխվում է մի շերտից մյուսը, և կշիռները փոփոխվում են՝ կախված կորուստներից, որպեսզի կորուստը նվազագույնի հասցվի: Կշիռները փոփոխվում են հետևյալ բանաձևով՝

,

որտեղ α – learning rate

### Ստոխաստիկ Գրադիենտի իջեցում

Դա Gradient Descent-ի տարբերակ է: Այն փորձում է ավելի հաճախ թարմացնել մոդելի պարամետրերը: Այս դեպքում մոդելի պարամետրերը փոխվում են յուրաքանչյուր ուսուցման օրինակի վրա կորստի հաշվարկից հետո: Այսպիսով, եթե տվյալների բազան պարունակում է 1000 տող, SGD-ն կթարմացնի մոդելի պարամետրերը 1000 անգամ տվյալների բազայի մեկ ցիկլում, այլ ոչ թե մեկ անգամ, ինչպես Gradient Descent-ում:

, որտեղ {x(i), y(i)} ուսուցման օրինակներն են:

Քանի որ մոդելի պարամետրերը հաճախ թարմացվում են, պարամետրերն ունեն տարբեր ինտենսիվությամբ կորստի ֆունկցիաների մեծ շեղումներ և տատանումներ:

### Խմբային Գրադիենտի իջեցում

Այն լավագույն օպտիմիզացման ալգորիթմն է գրադիենտի իջեցման ալգորիթմների մեջ։ Այն բաժանում է ուսուցման օրինակները խմբերի, այնուհետև ամեն խմբի համար փոփոխում է ցանցի կշիռները։ Դա բարելավում է Ստոխաստիկ և ստանդարտ գրադիենտի իջեցման համեմատ:

, որտեղ {B(i)} ուսուցման օրինակների խումբ է:

AdaGrad (Adaptive Gradient)

Բացատրված բոլոր օպտիմիզատորների թերություններից մեկն այն է, որ ուսուցման արագությունը հաստատուն է բոլոր պարամետրերի և յուրաքանչյուր ցիկլի համար: Այս օպտիմիզատորը փոխում է ուսուցման արագությունը: Այն փոխում է ուսուցման արագությունը յուրաքանչյուր պարամետրի և յուրաքանչյուր քայլի համար: Այդ պատճառով այն կոչվում է հարմարվող գրադիենտ։

### AdaM

AdaM (Adaptive Moment Estimation կամ հարմարվող իմպուլսի մոտավորեցում) աշխատում է առաջին և երկրորդ կարգի իմպուլսներով։ Ադամի հիմքում ընկած միտքն այն է, որ մենք չենք ուզում այդքան արագ գնալ այն պատճառով, որ կարող ենք անցնել նվազագույնի վրայով, մենք ուզում ենք մի փոքր նվազեցնել արագությունը՝ զգույշ և մանրակրկիտ որոնման համար:

Վերջին երկու օպտիմիզացման համակարգերի բանաձևերը պարունակում են բարձրագույն մաթեմատիկայի վախենալու բաներ։ Այդ պատճառով դրանք ներառված չեն։

# Նեյրոնային Ցանցի ձևերը

## Q-Learning

Q-learning-ը ուսուցման ալգորիթմ է, որը ձգտում է գտնել լավագույն գործողությունը, որը պետք է ձեռնարկվի՝ հաշվի առնելով ներկա վիճակը:

Այն կիրառվում է հիմնականում այն դեպքում, երբ ունենք ինչ-որ վիճակ (state), որը, փոխանցելով նեյրոնային ցանցին, ակնկալում ենք ստանալ այդ պահին օպտիմալ գործողությունը (գործողություն)։ Օրինակ՝ երբ ունենք լաբիրինթոս, որը ցանկանում ենք անցնել, կարող ենք թույլ տալ նեյրոնային ցանցին հետազոտել միջավայրը և հասկանալ, թե որն է օպտիմալ գործողությունը տվյալ վիճակում։

Q-learning-ի Q-ն նշանակում է Quality (որակ)։ Այս դեպքում այն ցույց է տալիս տվյալ վիճակում տվյալ գործողության արդյունավետությունը։

### Q-աղյուսակ

Q-learning իրագործելուց մենք ստեղծում ենք այն Q-աղյուսակ կամ մատրիցա, որը հետևում է [state, գործողություն] ձևին։ Սկզբում մենք վերագրում ենք մեր արժեքներին զրո: Այնուհետև մենք թարմացնում և պահում ենք մեր Q-արժեքները այդ պահին տեղի ունեցող դրվագից հետո: Այս Q-աղյուսակը դառնում է տեղեկատու աղյուսակ մեր նեյրոնային ցանցի համար՝ այն Q-արժեքի հիման վրա լավագույն գործողությունն է ընտրում:

Գործակալը փոխազդում է շրջակա միջավայրի հետ 2 եղանակներից 1-ով: Դրանցից մեկում օգտագործվում է Q-աղյուսակը որպես հղում։ Գործակալը դիտում է բոլոր հնարավոր գործողությունները տվյալ վիճակի համար: Գործակալն այնուհետև ընտրում է գործողությունը՝ հիմնվելով այդ գործողությունների առավելագույն արժեքի վրա: Սա հայտնի է որպես շահագործում, քանի որ որոշում կայացնելու համար մենք օգտագործում ենք մեզ հասանելի տեղեկատվությունը:

Գործողությունների երկրորդ եղանակը պատահականորեն գործելն է: Սա կոչվում է ուսումնասիրություն: Ապագա առավելագույն պարգևի հիման վրա գործողություններ ընտրելու փոխարեն մենք պատահականորեն գործողություն ենք ընտրում: Պատահականորեն գործելը կարևոր է, քանի որ այն թույլ է տալիս գործակալին ուսումնասիրել և հայտնաբերել նոր վիճակներ, որոնք չեն կարող ընտրվել շահագործման գործընթացում:

Կարելի է հավասարակշռել ուսումնասիրություն/շահագործումը՝ օգտագործելով epsilon (ε) և սահմանելով այն արժեքը, թե որքան հաճախ եք ցանկանում ուսումնասիրել/շահագործել:

Թարմացումները տեղի են ունենում յուրաքանչյուր քայլից կամ գործողությունից հետո և ավարտվում են դրվագի ավարտից հետո: Դրվագի ավարտ այս դեպքում նշանակում է գործակալի կողմից ինչ-որ վերջնական կետի հասնել: Ավարտի վիճակը, օրինակ, կարող է լինել այնպիսի բան, ինչպիսին է ինչ-որ խաղի ավարտին հասնելը, ցանկալի նպատակին հասնելը և այլն:

Գործակալը շատ բան չի սովորի մեկ դրվագից հետո, բայց, ի վերջո, բավականաչափ ուսումնասիրությամբ (քայլեր և դրվագներ) այն կսովորի օպտիմալ Q-արժեքները կամ Q-աստղը ():

Ահա 3 հիմնական քայլերը.

1. Գործակալը սկսում է վիճակից (s1) կատարում է գործողություն (a1) և ստանում պարգև (r1)
2. Գործակալն ընտրում է գործողությունը՝ հղում անելով Q-աղյուսակին՝ ամենաբարձր արժեքով (առավելագույնը) ԿԱՄ պատահականորեն (էպսիլոն, ε)
3. Գործակալը թարմացնում է Q-արժեքները Բելլմանի հավասարումով՝

Եկել է ժամանակը խոսելու այն փոփոխականների մասին, որոնք դեռ անծանոթ են այս հավասարման մեջ։

### Learning rate

Learning rate կամ ուսուցման արագությունը, որը հաճախ կոչվում է ալֆա կամ α, կարող է սահմանվել որպես այն, թե որքանով եք ընդունում նոր արժեքը ընդդեմ հին արժեքի: Վերևում մենք վերցնում ենք նոր և հին արժեքների տարբերությունը և այնուհետև այդ արժեքը բազմապատկում ենք սովորելու արագությամբ: Այնուհետև այս արժեքը ավելացվում է մեր նախորդ Q արժեքին:

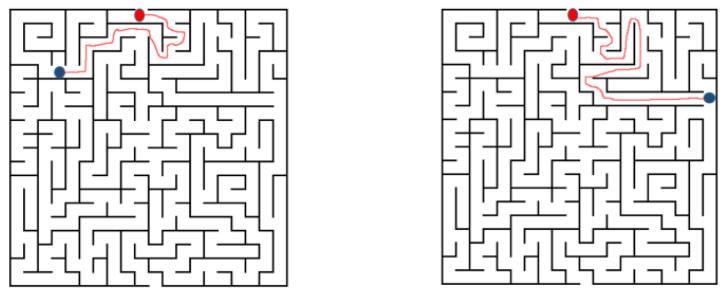
### Գամմա

Գամման կամ γ-ը կոչվում է զեղչի գործոն: Այն օգտագործվում է սկզբնական և ապագա պարգևները հավասարակշռելու համար: Վերևի մեր թարմացման բանաձևից կարող եք տեսնել, որ մենք զեղչը կիրառում ենք ապագա պարգևի համար: Սովորաբար այս արժեքը կարող է տատանվել 0,8-ից 0,99-ի սահմաններում:

### Մարկովի հատկությամբ որոշման ընթացք

Կարևոր կետ, որը պետք է հաշվի առնել. միջավայրի ներսում գտնվող յուրաքանչյուր վիճակ իր նախկին վիճակի հետևանք է, որն իր հերթին իր նախկին վիճակի հետևանք է: Այնուամենայնիվ, այս ամբողջ տեղեկատվության պահպանումը, նույնիսկ կարճ դրվագներով միջավայրերի համար, կդառնա անիրագործելի:

Այս խնդիրը լուծելու համար մենք ենթադրում ենք, որ յուրաքանչյուր վիճակ հետևում է Մարկովյան հատկությանը, այսինքն՝ յուրաքանչյուր վիճակ կախված է բացառապես նախորդ վիճակից և այդ վիճակից ներկայիս վիճակից անցումից: Տե՛ս ստորև բերված լաբիրինթոսը՝ ավելի լավ հասկանալու համար, թե ինչպես է դա աշխատում.



Այսպիսով, կա 2 սցենար՝ 2 տարբեր ելակետերով, և գործակալն անցնում է տարբեր ուղիներ, սակայն հասնում է նույն նախավերջին վիճակին: Այստեղ կարևոր չէ, թե ինչ ճանապարհով է անցնում գործակալը կարմիր վիճակին հասնելու համար: Լաբիրինթոսից դուրս գալու և վերջին վիճակին հասնելու համար հաջորդ քայլը վերև գնալն է: Ակնհայտ է, որ մեզ անհրաժեշտ էր միայն կարմիր վիճակի մասին տեղեկատվությունը` պարզելու հաջորդ լավագույն գործողությունը, որը հենց այն է, ինչ ենթադրում է Մարկովի հատկությունը:

## Deep Q-learning

Իսկ ո՞րտեղ է նեյրոնային ցանցը, կհարցնեք դուք։ Ներկայացնում ենք խորը Q-learning-ը։

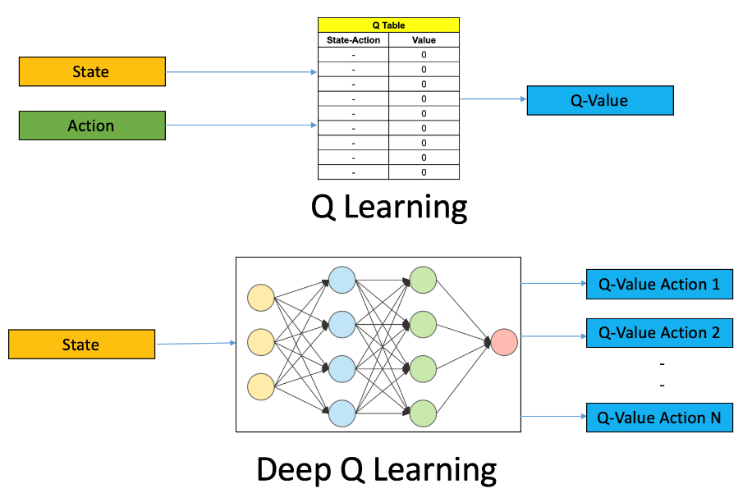
Պատկերացրեք մի միջավայր, որտեղ կա 10000 վիճակ և 1000 գործողություն յուրաքանչյուր վիճակում: Սա կստեղծեր 10 միլիոն արժեքների աղյուսակ: Գործողություններն արագ դուրս կգան վերահսկողությունից:

Միանգամայն պարզ է, որ մենք չենք կարող կիրառել Q-learning մեթոդը, որովհետև այն երկու խնդիր է առաջացնում.

1. Այդ աղյուսակը պահելու և թարմացնելու համար պահանջվող հիշողության քանակը կավելանա, քանի որ վիճակների թիվը մեծանում է,
2. Յուրաքանչյուր վիճակ ուսումնասիրելու համար պահանջվող Q-աղյուսակը ստեղծելու համար պահանջվող ժամանակը անիրատեսական կլինի։

DeepMind կազմակերպության հեղինակների մտքով անցավ Q-արժեքները կանխատեսել՝ կիրառելով նեյրոնային ցանց։ Այդ գաղափարի շնորհիվ էր, որ Google-ը հետագայում գնեց այն 500մլն դոլարով։

Deep Q-Learning-ի և Q-Learning-ի միջև հիմնական տարբերությունը Q-աղյուսակի իրականացումն է: Deep Q-Learning-ը փոխարինում է սովորական Q-աղյուսակը նեյրոնային ցանցով: Վիճակ- գործողություն զույգերին Q-արժեք վերագրելու փոխարեն (այդպես է վարվում Q-աղյուսակը) նեյրոնային ցանցը կազմում է (գործողություն, Q-արժեք) զույգեր տվյալ վիճակի համար:



### Experience Replay

Deep Q-learning-ի դեպքում մենք հաճախ օգտագործում ենք մի տեխնիկա, որը կոչվում է experience replay: Մենք պահում ենք գործակալի յուրաքանչյուր ժամանակային քայլի փորձը տվյալների հավաքածուում, որը կոչվում է կրկնվող հիշողություն:

t ժամանակում գործակալի փորձը կանվանենք et: Այսպիսով՝

,

Որտեղ՝

st – վիճակ t պահին

at – գործողություն t պահին

rt – մրցանակ t պահին

Ամեն քայլին մենք e-երի հավաքածուից կվերցնենք մի քանի պատահական օրինակ օրինակ, և այդ օրինակներով առաջնորդվելով՝ է՛լ ավելի կօպտիմիզացնենք մեր նեյրոնային ցանցի կշիռները և օգտակար գործողության փնտրումը։ Օրինակների պատահական լինելը փաստում է, որ դրանց մեջ կապ չկա, ինչն օգտակար է մեզ համար, որովհետև մենք կիրառում ենք Մարկովի հատկությունը։

## 

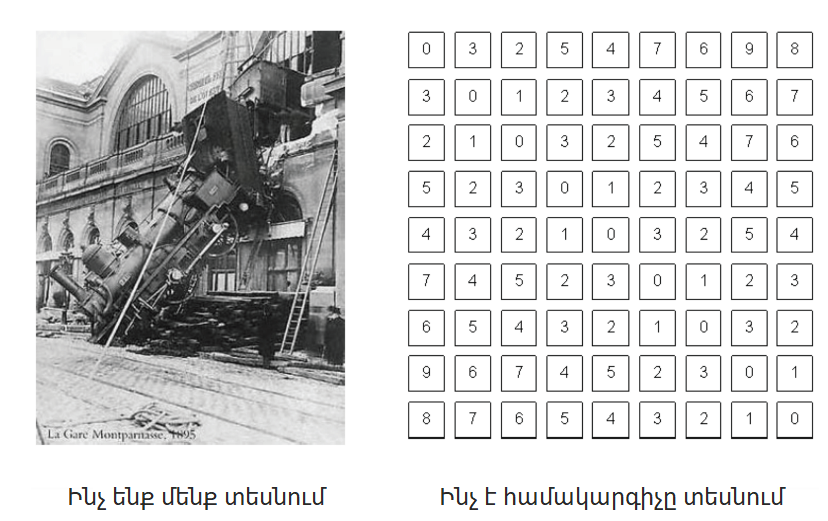
## Փաթեթային նեյրոնային ցանց

### Ներածություն

Փաթեթային նեյրոնային ցանցը (Convolutional Neural Network, CNN) հանդիսանում է կենսաբանության և մաթեմատիկայի խառնուրդ՝ ինֆորմատիկայի լրացումով։ Դրանք ցանցեր են, որոնք համարվում են համակարգչային տեսլակալի (computer vision) ամենաազդեցիկ նորարարություններից մեկը։ Առաջին անգամ նեյրոնային ցանցերը համամասնային ուշադրություն գրավեցին 2012թ, երբ նրանց շնորհիվ կարողացան նվազեցնել սխալների դասակարգման ռեկորդը 26%-ից մինչև 15%, ինչը այդ ժամանակ մեծ առաջընթաց էր։

### Ներմուծում և արտածում

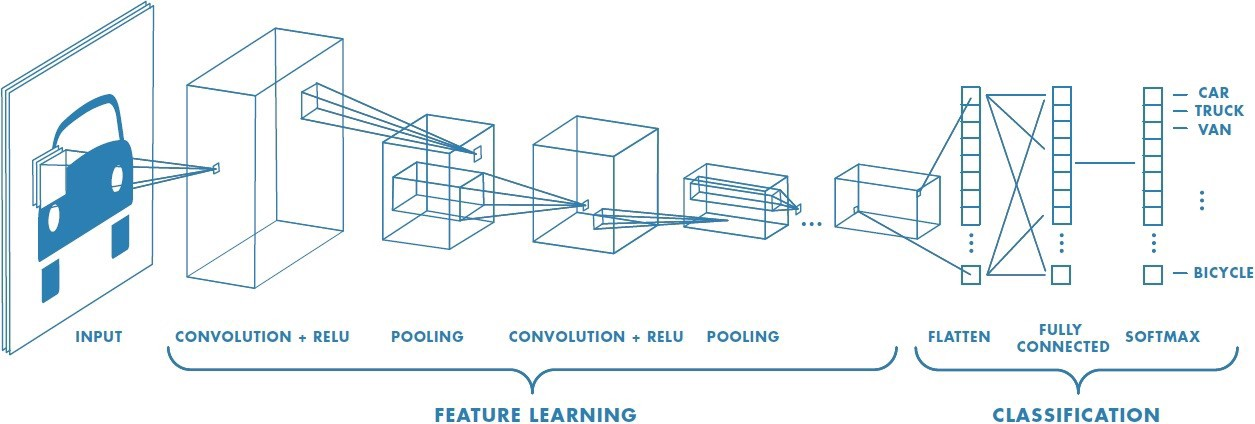
Երբ համակարգիչը տեսնում է պատկերը, այն տեսնում է փիքսելների զանգված։ Անկախ պատկերի ձևաչափից և չափսից, զանգվածի չափսը կարող է լինել, օրինակ, 32х32х3։



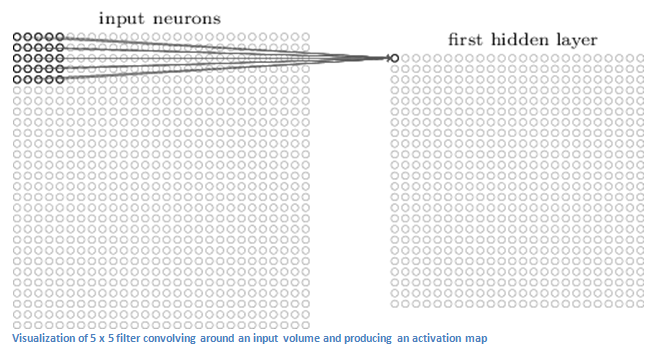
Որպեսզի ավելի հասկանալի լինի, եկեք պատկերացնենք որ ունենք JPG ձևաչափով պատկեր, և նրա չափը կազմում է 480х480х3։ Այդ թվերից յուրաքանչյուրին համապատասխանում է 0 մինչև 255 արժեքներ, որոնք նկարագրում են այդ կետում փիքսելների խտությունը։ Այդ թվերը անզեն աչքի համար անիմաստ են, սակայն հանդիսանում են միակ մուտքագրվող տվյալները, որոնք հասանելի են համակարգչին։ Գաղափարը կայանում է նրանում որ դուք տալիս եք համակարգչին այդ մատրիցը իսկ նա արտածում է թվեր, որոնք նկարագրում են պատկերի այս կամ այն դասի մեջ գտնվելու հավանականությունը։

### Կառուցվածք

Ի՞նչ է իսկապես անում CNN-ը։ Վերցնում է պատկերը, անցկացնում փաթեթային, ոչ գծային շերտերի շարքով, համախմբման և ամբողջությամբ միացված շերտերի միջով և գեներացնում է ելքը։ Ինչպես արդեն ասվել է նախկինում որպես ելք կարող է լինել դասը կամ այն դասերի

հավանականությունը, որոնք ամենից լավն են նկարագրում պատկերը։ Դժվարը հասկանալն է, թե ինչ է անում շերտերից յուրաքանչյուրը։

### Փաթեթային շերտ

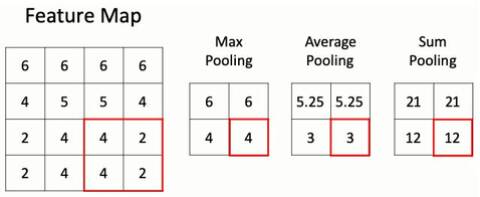
CNN-ում առաջին շերտը միշտ փաթեթայինն է։ Ինչպես նախկինում ասվել էր, ներմուծված պատկերը մատրից է 32 х 32 х 3 փիքսելային նշանակություններով։ Ավելի հեշտ կլինի հասկանալ, թե ինչ է փաթեթային շերտը, եթե ներկայացնենք այն լապտերի տեսքով, որը լուսավորում է պատկերի վերևի ձախ հատվածը։ Ենթադրենք, լույսը, որը արձակում է լապտերը, լուսավորում է 5 х 5 մակերեսով տարածքը ամբողջությամբ։ Իսկ հիմա եկեք պատկերացնենք, որ լապտերը շարժվում է մուտքագրվող պատկերի բոլոր սահմաններով։ Համակարգչային ուսման տերմիններում այդ լապտերը կոչվում է **ֆիլտր** (երբեմն նեյրոն կամ միջուկ), իսկ այն հատվածները, որոնք այն լուսավորում են, կոչվում են ընկալունակ բացատներ։ Այսինքն, մեր **ֆիլտր**ը մատրիցն է։ Նշենք, որ **ֆիլտր**ի խորությունը պետք է լինի այնքան, որքան մուտքագրվող պատկերինն է, և այդ **ֆիլտր** 5 х 5 х 3 չափի է։ Հիմա եկեք որպես օրինակ վերցնենք դիրք, որում գտնվում է **ֆիլտր**ը։ Թող դա լինի վերևի ձախ հատվածը։ Քանի որ ֆիլտրը կատարում է փաթեթավորում՝ տեղաշարժվում է մուտքագրվող պատկերի վրայով, այն բազմապատկում է ֆիլտրի արժեքը պատկերի սկզբնական արժեքների փիքսելներով։ Բոլոր այդ բազմապատկումները գումարվում են և վերջում ստացվում է մեկ թիվ։

Ներմուծվող պատկերի յուրաքանչյուր յուրահատուկ դիրք թիվ է արտահանում։ ֆիլտրի բոլոր դիրքերով անցնելուց հետո ստացվում է 28 х 28 х 1 մատրից, որին անվանում են ակտիվացիայի ֆունկցիա կամ նախանշանների քարտեզ։ 28 х 28 մատրիցը ստացվում է, որովհետև կան 784 տարատեսակ դիրքեր, որոնք կարող են անցկացնել 5 х 5 ֆիլտրով 32 х 32 պատկերներ։ Այդ 784 թվերը վերածվում են 28 х 28 մատրիցի։

### Պուլինգի շերտ

Պուլինգին շերտի հիմնական նպատակն է նվազեցնել մուտքային տենզորի չափը, ինչը նվազեցնում է հետագա հաշվարկների քանակը: Բացի այդ չափսերի նման փոքրացումը թույլ է տալիս ցանցին ավելի դիմացկուն լինել պատկերի առարկաների փոքր տեղաշարժերին:

Մուտքային պատկերի չափը նվազեցնելու շատ եղանակներ կան։ Օրինակ՝ առավելագույն արժեքները (max-pooling), միջինացնված արժեքները (average pooling), գումարը (sum-pooling), կամ նույնիսկ նվազագույնը: Ամենից հաճախ, Փաթեթային նեյրոնային ցանցերում օգտագործվում է max-pooling-ը։



### Ամբողջությամբ կապակցված շերտ

Այս շերտը միանում է ցանցի վերջին: Այն վերցնում է մուտքագրվող տվյալները և արտահանում դեպի N տարածական վեկտոր, որտեղ N-ը տարբերակների թիվն է, որոնցից ծրագիրը ընտրում է անհրաժեշտը։ Օրինակ եթե դուք ցանկանում եք ունենալ ծրագիր որը ընդունակ է թվանշաններ ճանաչել, N-ի նշանակությունը կլինի 10, քանի որ թվերի քանակը 10 է։ Յուրաքանչյուր թիվ այդ N տարածական վեկտորում իրենից ներկայացնում է հստակ տարբերակի հավանականություն։

Եղանակը, որի օգնությամբ աշխատում է ամբողջությամբ կապակցված շերտը նախկին շերտի ելքին դիմումն է և այն հատկությունների որոշումը, որոնք ավելի շատ են կապված որոշակի տարբերակի հետ։ Օրինակ, եթե ծրագիրը նախագուշակում է, որ պատկերի վրա շուն է, հատկությունների քարտերի մոտ, որոնք արտացոլում են բարձր մակարդակ ունեցող բնութագրերը, ինչպիսիք են թաթերը կամ 4 ոտքերը, պետք է լինեն բարձր նշանակություններ։ Այնպես, ինչպես ծրագիրը հասկանում է, որ պատկերի վրա թռչուն է, նրա մոտ հատկությունների քարտերում կլինեն բարձր նշանակություններ, որոնք ներկայացված են բարձր մակարդակի բնութագրերով, ինչպիսիք են թևերը կամ կտուցը։ Ամբողջությամբ կապակցված շերտը հաշվի է առնում այն, որ բարձր մակարդակի ֆունկցիաները ամուր կապակցված են որոշակի տարբերակի հետ և ունեն որոշակի զանգված։ Այսպիսով, երբ դուք հաշվարկում եք զանգվածների հարաբերությունները նախկին շերտի հետ, ապա ստանում ենք տարբեր տարբերակների համար ճշգրիտ հավանականություններ։

## NEAT

### Գենետիական ալգորիթմ

Գենետիկական ալգորիթմն օգտագործվում է նեյրոնային ցանցերը մարզելու և բարելավելու համար:

Գենետիկական ալգորիթմը օպտիմիզացման տեխնիկա է, որը կրկնում է բնական ընտրությունը և գենետիկան:

Գենետիկական ալգորիթմի իրականացման հիմնական քայլերը.

1․ ստեղծել n օբյեկտների նախնական պոպուլյացիան՝ պատահական նեյրոնային ցանցերով

2․ թույլ տալ բոլոր օբյեկտներին աշխատել միաժամանակ՝ օգտագործելով իրենց նեյրոնային ցանցերը

3․ յուրաքանչյուր օբյեկտի համար մենք հաշվարկում ենք դրա ֆիթնես ֆունկցիան

4․ բոլոր օբյեկտների մահից հետո մենք գնահատում ենք ներկայիս սերունդը՝ գենետիկ օպերատորների միջոցով նորը ստեղծելու համար: Լավագույնների գեների հիման վրա ստեղծում ենք նոր սերունդ։

5․ վերադառնում ենք երկրորդ քայլին

### Գենետիկական գործողություններ

#### Կրոսինգովեր

Կրոսինգովերը գործընթաց է, որի ընթացքում  նախկին սերնդի լավագույն առանձնյակների գեները օգտագործելով ստեղծում ենք նոր սերունդ։ Կրոսինգովերի ժամանակ 2 առանձնյակների գեների պատահականորեն ընտրած մասերը օգտագործելով ստանում ենք նոր առանձնյակ։

Այն լինում է 3 տեսակի․

Միակետային - ընտրվում է 1 կետ և այդ կետից աջ գտնվող բիթերը տեղերով փոխվում են

Երկկետային - ընտրվում են 2 կետ և այդ կետերի միջև գտնվող բիթերը տեղերով փոխվում են

Բազմակետային - ամեն բիթը ընտրվում է պատահականորեն

#### Մուտացիա

Մուտացիան գենոտիպում (բիթային հաջորդականությունում) պատահականորեն տեղի ունեցող փոքր փոփոխություններ են։

Մուտացիան կարող է կատարել 2 գործողություն․

1․ գոյություն ունեցող նեյրոնների միջև նոր կապ է առաջանում

2․ գոյություն ունեցող կապի վրա ստեղծվում է նոր նեյրոն

#### Ֆիթնես ֆունկցիա

Քանի որ մենք ցանկանում ենք, որ նեյրոնային ցանցը զարգանա լավագույն տարրերից, մենք պետք է սահմանենք ֆիթնես ֆունկցիան:

Ընդհանուր առմամբ, ֆիթնես ֆունկցիան չափում է օբյեկտի որակը: Եթե յուրաքանչյուր օբյեկտի համար ունենք որակի չափանիշ, մենք կարող ենք ընտրել ամենապիտանի օբյեկտները և օգտագործել դրանք հաջորդ սերունդը վերստեղծելու համար: Օրինակ՝ մեր պրոյեկտներից 2-ում՝ Google dino-ում և Flappy bird-ում,  ֆիթնեսի ֆունկցիան օբյեկտի անցած հեռավորությունն է։

# Պրոյեկտներ

Այսքանը տեսական մասից, եկեք անցնենք փորձնականին։

Կստեղծենք 5 տարբեր նեյրոնային ցանց՝ տարբեր ձևերով, տարբեր առաջադրանքներ կատարող։ Դրանցից երկուսը կլինեն փաթեթային նեյրոնային ցանցեր՝ նկարները տարբերակելու համար, երկուսը՝ NEAT տիպի՝ տարբեր խաղեր անցնելու համար, և վերջինը՝ Deep Q-learning տեխնիկայով՝ ինքնակառավարվող մեքենա։

Պրոյեկտների կոդը գրված է python ծրագրավորման լեզվի տարբեր գրադարաններով։ Այդ կերպ մենք կարողանանք ցույց տալ տարբեր գրադարանների առավելությունները և թերությունները։

Բոլոր պրոյեկտնորի կոդերին կարող եք ծանոթանալ՝ անցնելով <https://drive.google.com/drive/folders/1Irh6dtlEGyeylVZUhJt6itbamsx8jlGk?usp=sharing> հղմամբ։

## Շան և կատվի տարբերակում

### Նեյրոնային ցանց

Փաթեթային նեյրոնային ցանցերի աշխատանքը ցուցադրելու համար գրենք նեյրոնային ցանց, որը կհասկանա նկարում պատկերված է շու՞ն, թե՞ կատու։

Այս անգամ մենք կօգտագործենք tensorflow գրադարանը, քանի որ այն ունի մեծ dataset, որը մենք կարող ենք օգտագործել նեյրոնային ցացին մարզելու համար։

Կցում ենք բոլոր անհարժեշտ գրադարանները և քաշում ենք dataset-ը․

import numpy as np

import tensorflow as tf

import tensorflow\_datasets as tfds

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout

import matplotlib.pyplot as plt

from google.colab import files

train, \_ = tfds.load('cats\_vs\_dogs', split=['train[:100%]'], with\_info=True, as\_supervised=True)

Որպեսզի երկար չմարզենք մեր նեյրոնային ցացնը, կարեղ ենք վերցնել մեկ այլ նեյրոնային ցանցի շերտերի մի մասը։ Այդ նեյրոնային ցանցը կոչվում է MobileNetV2, և այն կարող է տարբերել ավելի քան 1000 պատկեր։

base\_layers = tf.keras.applications.MobileNetV2(input\_shape=(SIZE, SIZE, 3), include\_top=False)

base\_layers.trainable = False

Քանի որ այդ նեյրոնային ցանցը աշխատում է 224x224 չափսերի նկարների հետ, մեր dataset-ում

նկարների չափսերը փոխում ենք, դարձնելով հասանելի նեյրոնային ցացնի համար։

SIZE = 224

def resize\_image(img, label):

  img = tf.cast(img, tf.float32)

  img = tf.image.resize(img, (SIZE, SIZE))

  img = img / 255.0

  return img, label

train\_resized = train[0].map(resize\_image)

train\_batches = train\_resized.shuffle(1000).batch(16)

Գրում ենք մեր նեյրոնային ցանցը․

model = tf.keras.Sequential([

                             base\_layers,

                             GlobalAveragePooling2D(),

                             Dropout(0.2),

                             Dense(1)

])

model.compile(optimizer='adam', loss=tf.keras.losses.BinaryCrossentropy(from\_logits=True), metrics=['accuracy'])

Այնուհետև մարզում այն․

model.fit(train\_batches, epochs=1)

Եվ վերջապես տպում ենք արդյունքները։

files.upload()

img = load\_img('.jpg')

img\_array = img\_to\_array(img)

img\_resized, \_ = resize\_image(img\_array, \_)

img\_expended = np.expand\_dims(img\_resized, axis=0)

prediction = model.predict(img\_expended)[0][0]

pred\_label = 'Dog' if prediction < 0.5 else 'Cat'

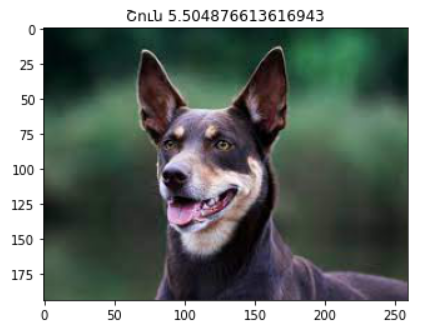
plt.figure()

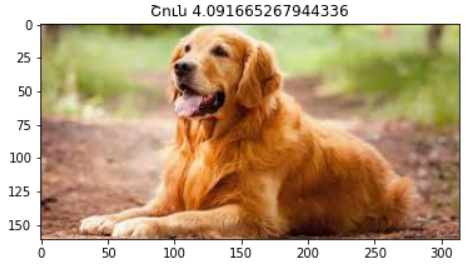
plt.imshow(img)

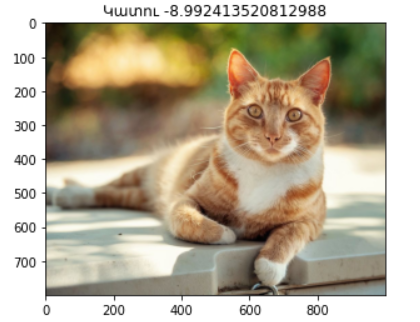
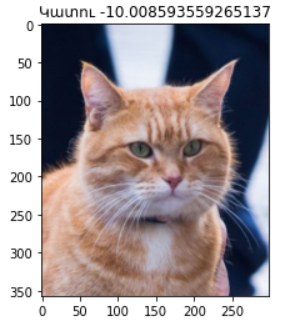
plt.title(f'{pred\_label} {prediction}')

Այն աշխատում է հետևյալ կերպ - ինչքան արտածած թիվը մեծ է 0․5-ից, այնքան նկարում շուն լինու հավանականությունը ավելի մեծ է և ինչան փոքր է 0․5-ից,այնքան ավելի մեծ է հավանականությունը որ նկարոում կատու է։

### Արդյունքներ

Այժմ այս նեյրոնային ցանցի միջոցով ստացված որոշ արդյունքներ.





## Ձեռագիր թվերի ճանաչում

Գրենք նեյրոնային ցանց, որը կկարողանա տարբերակել ձեռագիր թվերը։

Կգրենք այն փաթեթային նեյրոնային ցանցի շնորհիվ, որովհետև պետք է կարողանանք նկարներ վերծանել և գտնել օրինաչափություններ դրանց մեջ։ Կօգտագործենք MNIST տվյալների խումբը։ MNIST-ը տվյալների խումբ է, որը պարունակում է ձեռագիր թվերի մոտ 70000 նկար, և յուրաքանչյուրի համար՝ իրենց արժեքները։

Կկիրառենք torch գրադարանը։

### Նեյրոնային ցանցը

Կցում ենք բոլոր անհրաժեշտ գրադարանները և քաշում ենք dataset-ը․

import torch

import random

import numpy as np

import torchvision.datasets

import matplotlib.pyplot as plt

MNIST\_train = torchvision.datasets.MNIST('./', download=True, train=True)

MNIST\_test = torchvision.datasets.MNIST('./', download=True, train=False)

X\_train = MNIST\_train.train\_data

y\_train = MNIST\_train.train\_labels

X\_test = MNIST\_test.test\_data

y\_test = MNIST\_test.test\_labels

Հաջորդ երկու տողում տվյալները դարձնում ենք միաչափ (torch-ում բոլոր տվյալները տրվում են երկչափ)։

X\_train = X\_train.unsqueeze(1).float()

X\_test = X\_test.unsqueeze(1).float()

Ստեղծում ենք նեյրոնային ցանցի մոդելը։

class model(torch.nn.Module):

def \_\_init\_\_(self):

super(model, self).\_\_init\_\_()

self.conv1 = torch.nn.Conv2d(in\_channels=1, out\_channels=6, kernel\_size=5, padding=2)

self.act1 = torch.nn.Tanh()

self.pool1 = torch.nn.AvgPool2d(kernel\_size=2, stride=2)

self.conv2 = torch.nn.Conv2d(in\_channels=6, out\_channels=16, kernel\_size=5, padding=0)

self.act2 = torch.nn.Tanh()

self.pool2 = torch.nn.AvgPool2d(kernel\_size=2, stride=2)

self.fc1 = torch.nn.Linear(5 \* 5 \* 16, 120)

self.act3 = torch.nn.Tanh()

self.fc2 = torch.nn.Linear(120, 84)

self.act4 = torch.nn.Tanh()

self.fc3 = torch.nn.Linear(84, 10)

(conv – առաջին՝ փաթեթային շերտ, pool – երկրորդ՝ պուլինգի շերտ, fc (fully connected) -ամբողջությամբ կապակցված շերտեր, act – ակտիվացման ֆունկցիա):

Գրենք ֆունկցիա, որի շնորհիվ կկարողանանք անցկացնել մեր տվյալները այդ շերտերով։

def forward(self, x):

x = self.pool2(self.act2(self.conv2(self.pool1(self.act1(self.conv1(x))))))

x = x.view(x.size(0), x.size(1) \* x.size(2) \* x.size(3))

x = self.fc3(self.act4(self.fc2(self.act3(self.fc1(x)))))

return x

Ստեղծենք model կլասսի brain օբյեկտ։

brain = model()

Կօգտագործենք cross entropy կորստի ֆունկցիան, որովհետև սա կլասիֆիկացիայի խնդիր է։

loss = torch.nn.CrossEntropyLoss()

Կկիրառենք նաև AdaM օպտիմիզացման ֆունկցիան։

optimizer = torch.optim.Adam(brain.parameters(), lr=1.0e-3)

Խմբավորենք տվյալները, որպեսզի ավելի արդյունավետ լինի ուսուցումը։

batch\_size = 100

Աշխատացնենք ուսուցման ալգորիթմը 10 անգամ

for epoch in range(10):

order = np.random.permutation(len(X\_train)) # պատահականորեն խառնում ենք տվյալները

for start\_index in range(0, len(X\_train), batch\_size):

optimizer.zero\_grad() # update ենք անում օպտիմայզերը

batch\_indexes = order[start\_index:start\_index + batch\_size] # ստանում ենք խմբերը

X\_batch = X\_train[batch\_indexes]

y\_batch = y\_train[batch\_indexes]

preds = brain.forward(X\_batch) # աշխատացնենք նեյրոնային ցանցը ստացված խմբի համար

loss\_value = loss(preds, y\_batch) # ստանում ենք սխալանքը

loss\_value.backward()

optimizer.step() # աշխատացնում ենք օպտիմայզերը

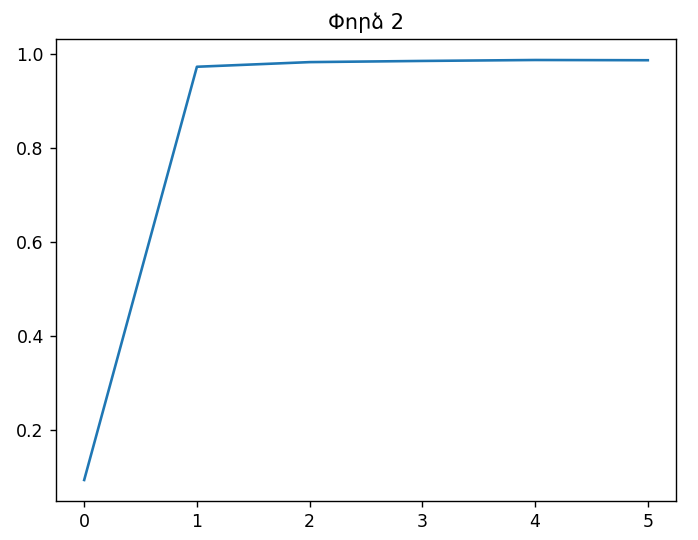
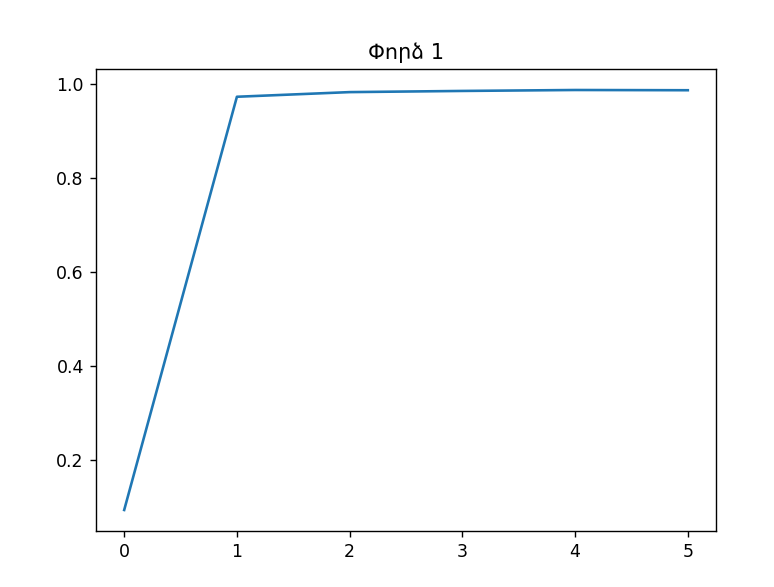
test\_preds = brain.forward(X\_test) # աշխատացնենք ցանցը ստացված խմբի համար

accuracy = (test\_preds.argmax(dim=1) == y\_test).float().mean() # ստանում ենք ճշգրտությունը

print(accuracy)

### Արդյունքներ

|  |  |  |
| --- | --- | --- |
| Generation | Փորձ 1  Ճշգրտություն | Փորձ 2  Ճշգրտություն |
| 0 | 0.0932 | 0.1251 |
| 1 | 0.9736 | 0.9724 |
| 2 | 0.9835 | 0.9856 |
| 3 | 0.9860 | 0.9851 |



## Google Dino

### Նեյրոնային ցանց

Մինչև նեյրոնային ցանցին անցնելը անհրաժեշտ է գրել խաղը, որի կոդին ավելի մանրամասն կարող եք ծանոթանալ <https://drive.google.com/drive/folders/1Irh6dtlEGyeylVZUhJt6itbamsx8jlGk?usp=sharing> հղումով: Խաղը գրելուց հետո կարող ենք անցնել նեյրոնային ցանցին։ Նեյոնային ցանցին մենք տալու ենք 4 արժեք, որոնք նշանակելու են՝ dino-ի դիրքի կորդինատը y առանցքով , խաղի արագությունը, նրա հեռավորությունը ամենամոտ կակտուսից և այդ կակտուսի լայնությունը։ Այդ 4 արժեքների հիման վրա նա պետք է որոշում կայացնի՝ ցատկի՞, թե՞ ոչ։ Ամեն անցած կակտուսի համար մենք նեյրոնային ցանցին տալու ենք 5 միավոր, իսկ կակտուսի հետ բախվելու դեպքում -10 միավոր։ Սակայն որպեսզի dino-ն անըդհատ չցատկի և նեյրոէվոլուցիան ավելի արագ ընդանա ամեն ցատկի համար մենք հանելու ենք 1 միավոր։ Սրանք նեյրոնային ցանցի հիմանկան գաղափարներն էին։ Այժմ անցնենք դրանց իրականացմանը։

Մենք օգտագործելու ենք NEAT-Python գրադարանը։ Նոյրոնային ցանցի համար պետք է գրենք առաջին սերնդի և նեյրոնային ցանցի էվոլուցիայի պարամետրերը։ Այդ բոլորը մենք պահելու ենք config-feedforward ֆայլում և այժմ քննարկենք այդ ֆայլի կարևորագույն մասերը:

Ի սկզբանե գրում ենք հիմնական տվյալները՝

[NEAT]

fitness\_criterion = max (ում ենք համարում ամենալավ առանձնայկը, այս դեպքում ամենաշատ հեռավորություն անցած առանձնյակին)

fitness\_threshold = 10000 (ինչ արժեքի հասնելուց հետո ծրագիրը պետք է անջատվի)

pop\_size = 50 (ամեն սերնդում առանձնյակների քանակը)

reset\_on\_extinction = False

Այնուհետև տալիս ենք ակտիվացման ֆունկցիայի պարամետրերը՝

[DefaultGenome]

activation\_default = tanh (սկզբանական ակտիվացման ֆունկցիան)

activation\_mutate\_rate = 0.0 (ակտիվացման ֆունկցիայի փոփոխության հավանականությունը առանձյակի մոտ)

activation\_options = tanh (ակտիվացման ֆունկցիայի տարբերակները)

Նեյրոնային ցանցի պարամետրերը

num\_hidden = 0 (թաքնված նեյրոնների քանակը)

num\_inputs = 4 (մուտքային նեյրոնների քանակը)

num\_outputs = 1 (ելքային նեյրոնների քանակը)

Ֆայլի մնացած արժեքները հիմանկանում չեն փոփոխվում, քանի որ նեյրոնային ցանցերը լավ են աշխատում այդ արժեքների դեպքում։

Այժմ մենք config փոփոխականին փոխանցում ենք այդ ֆայլի արժեքները

config = neat.config.Config(neat.DefaultGenome, neat.DefaultReproduction, neat.DefaultSpeciesSet,

neat.DefaultStagnation, config\_path)

ստեղծում ենք նոր սերունդ

p = neat.Population(config)

և ակտիվացնում ենք խաղը բոլոր առանձնյակների համար

p.run(run\_game, 1000)

### Արդյունքներ

|  |  |  |
| --- | --- | --- |
| Generation | Փորձ 1  Score | Փորձ 2  Score |
| 1 | 314 | 8028 |
| 2 | 586 | 2824 |
| 3 | 717 | 17463 |
| 4 | 223210 | 27333 |
| 5 | 197532 | 9237 |

## Flappy Bird Խաղի անցում

Flappy Bird խաղին կարող եք ծանոթանալ <https://flappybird.io/> հղումով։

Կանոնները շատ պարզ են՝ պետք է space կոճակը ճիշտ պահին սեղմելով թռչունին անցկացնել ձողերի միջև բաց թողնված անցքով։

### Նեյրոնային ցանց

Խաղը ստեղծելուց հետո կարող ենք անցնել նեյրոնային ցանցի ստեղծմանը։ Կկիրառենք NEAT մեթոդը, որովհետև դրա շնորհիվ կկարողանանք մի քանի առանձնյակ միանգամից փորձել։

Խաղի կոդին կարող եք ծանոթանալ <https://drive.google.com/drive/folders/1Irh6dtlEGyeylVZUhJt6itbamsx8jlGk?usp=sharing> հղմամբ։

Իսկ մենք միանգամից կանցնենք նեյրոնային ցանցի ստեղծմանը։

Կկիրառենք նույն տրամաբանությունը, ինչ Google Dino-ում։

Իրականում, ամբողջ նեյրոնային ցանցի պարամետրերը, բացառությամբ num\_inputs-ի, համընկնում են Google Dino-ի պարամետրերի հետ։ Փոխվել է, իհարկե, մրցանակ տալու տրամաբանությունը։ Այս անգամ, մենք գործակալին մրցանակ կտանք կենդանի մնալու համար (+0.1 ամեն frame-ի համար), ձողի միջով անցնելու համար (+1) և ձողին բախվելու համար (-1): Այս դեպքում, թռչունը կհասկանա, որ պետք է չմահանալ և անցնել ձողերի միջով։

Ինչ վերաբերվում է ներմուծվող և արտածվող տվյալներին, մենք գործակալին կտանք 3 տեղեկություն՝ թռչունի y կոորդինատը, դրա և ձողի վերին մասի տարբերությունը, դրա և ձողի ներքևի մասի տարբերությունը։ Կստանանք մեկ տվյալ՝ թռչե՞լ, թե՞ ոչ։ Տվյալն անցնելու է tanh ֆունկցիայի միջով, այսպիսով ստանալու ենք [-1, 1] միջակայքում գտնվող թիվ։ Եթե այն մեծ է 0-ից, ապա այն կնշանակի թռչել, այլապես՝ ոչ։

Կաշխատացնենք 50 առանձնյակ ամեն քայլին, որ է՛լ ավելի արագացնենք օպտիմալ առանձնյակի մուտացիան։

Կոդը շատ նման է Google Dino-ի կոդին։ Դրան կարող եք ծանոթանալ <https://drive.google.com/drive/folders/1Irh6dtlEGyeylVZUhJt6itbamsx8jlGk?usp=sharing> հղումով։

### Արդյունքներ

|  |  |  |
| --- | --- | --- |
| Generation | Փորձ 1  Score | Փորձ 2  Score |
| 1 | 29 | 46 |
| 2 | 2 | 46 |
| 3 | 32 | 59 |
| 4 | 26 | 103 |
| 5 | 55 | 41 |
| 6 | 167 | 68 |

## Ինքնակառավարվող Մեքենա

Եկեք ստեղծենք մեքենա, որը կկարողանա ինքնակառավարվել։

Կստեղծենք քարտեզ, որտեղ կարող ենք գծել ճանապարհ, որով մեր մեքենան կգնա։ Քարտեզի կոդը չենք մանրամասնի։ Միանգամից անցնենք պլանին։

Մեքենան կլինի պարզագույն ուղղանկյուն, որին միացված է երեք սենսոր։ Դրանցից յուրաքանչյուրը կստանա տեղեկություն այդ միջակայքում ավազի առկայության մասին։ Այդ տեղեկությունները հետագայում պետք կգան՝ նեյրոնային ցանցին փոխանցելու համար։

Քարտեզը կգծենք ինքներս, որպեսզի կարողանանք տարբեր ձևի քարտեզներ գծել։ Ավազը կգծենք մկնիկի ձախ ստեղնը սեղմած մկնիկը շարժելով։ Մեքենան պետք է խուսափի ավազից։

Անցնենք նեյրոնային ցանցի ստեղծմանը։

Կստեղծենք այն Deep Q-learning տեխնիկայով։ Կտանք բացասական մրցանակ, երբ մեքենան բախվի խոչընդոտին։ Նաև կավելացնենք կետեր, որոնց մեքենան պետք է հասնի։ Մոտենալւ համար կտանք դրական մրցանակ։ Կտանք նաև մոդուլով փոքր բացասական մրցանակ թեքվելու համար։ Այսպես մեքենան ավելի ուղիղ կգնա։

Նեյրոնային ցանցին կտանք 5 տվյալ՝ 3 սենսորների տվյալները, մեքենայի ուղղության և նպատակակետով ու մեքենայւվ անցնող ուղղի կազմած անկյունը, և այդ նույն արժեքը՝ բացասական նշանով։

Կկիրառենք Experience Replay տեխնիկան՝ 100000 չափի հիշողությամբ։ Կվերցնենք 100 պատահական տվյալ այդտեղից ամեն քայլին։

Կկիրառենք torch գրադարանը։

Անցնենք նեյրոնային ցանցի ռեալիզացմանը։

### Նեյրոնային ցանց

# Կցում ենք բոլոր անհրաժեշտ գրադարանները

import random

import os

import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

from torch.autograd import Variable

# Ստեղծում ենք նեյրոնային ցանցի մոդելը

class Network(nn.Module):

def \_\_init\_\_(self, input\_size, nb\_action):

super(Network, self).\_\_init\_\_()

self.input\_size = input\_size

self.nb\_action = nb\_action

self.fc1 = nn.Linear(input\_size, 30) # մուտքային շերտ

self.fc2 = nn.Linear(30, 50) # 30 երկարությամբ թաքնված շերտ

self.fc3 = nn.Linear(50, nb\_action) # 50 երկարությամբ թաքնված շերտ

# Ստեղծում ենք ֆունկցիա, որը կանցկացնի տվյալները նեյրոնային ցնացի միջով և կտպի Q-արժեքները

def forward(self, state):

x = F.softmax(self.fc1(state)) # առաջին ակտիվացման ֆունկցիան՝ softmax

y = F.relu(self.fc2(x)) # երկրորդ ակտիվացման ֆունկցիան՝ ReLU

q\_values = self.fc3(y) # Ստանում ենք Q-արժեքները

return q\_values

# Ռեալիզացնում ենք Experience Replay-ը

class ReplayMemory(object):

def \_\_init\_\_(self, capacity):

self.capacity = capacity # Հիշողության չափը

self.memory = [] # Հիշողությունը

# Ստեղծում ենք ֆունկցիա, որը կգցի տեղեկություն հիշողության մեջ

def push(self, event):

self.memory.append(event) # գցումն ենք հիշողության մեջ

if len(self.memory) > self.capacity: # եթե չափից երկար է

del self.memory[0] # ջնջում ենք առաջին էլեմենտը

# Ֆունկցիա, որը կվերցնի պատահական խումբ հիշողությունից

def sample(self, batch\_size):

samples = zip(\*random.sample(self.memory, batch\_size)) # պատահականորեն խառնում ենք

return map(lambda x: Variable(torch.cat(x, 0)), samples) # կիրառում ենք հատուկ torch-ի ֆունկցիաներից մեկը

# Ռեալիզացնում ենք Deep Q-learning-ը

class Dqn:

def \_\_init\_\_(self, input\_size, nb\_action, gamma):

self.gamma = gamma # Գամմա պարամետրը

self.reward\_window = [] # Ստացած վերժին մի քանի մրցանակներ

self.model = Network(input\_size, nb\_action) # Հայտարարում ենք մեր նեյրոնային ցանցը

self.memory = ReplayMemory(100000) # Հայտարարում ենք մեր հիշողությունը

self.optimizer = optim.Adam(self.model.parameters(), lr=0.001) # օգտագործելու ենք AdaM օպտիմայզերը

self.last\_state = torch.Tensor(input\_size).unsqueeze(0) # պահում ենք վերջին վիճակը,

self.last\_action = 0 # վերջին գործողությունը

self.last\_reward = 0 # և վերջին մրցանակը

# Ֆունկցիա, որը կաշխատեցնի նեյրոնային ցանցը և կորոշի գործողությունը

def select\_action(self, state):

probs = F.softmax(self.model(Variable(state, volatile=True)) \* 100) # օգտագործում ենք Softmax-ը

action = probs.multinomial(num\_samples=1) # torch գրադարանի հատուկ ֆունկցիայի շնորհիվ ստանում ենք արժեքը

return action.data[0, 0] # վերադարձնում ենք կանխատեսված լավագույնը

# Ֆունկցիա, որը հետ կգնա նեյրոնային ցանցով և կթարմացնի կշիռները

def learn(self, batch\_state, batch\_next\_state, batch\_reward, batch\_action):

outputs = self.model(batch\_state).gather(1, batch\_action.unsqueeze(1)).squeeze(1)

next\_outputs = self.model(batch\_next\_state).detach().max(1)[0]

target = self.gamma \* next\_outputs + batch\_reward

td\_loss = F.smooth\_l1\_loss(outputs, target)

self.optimizer.zero\_grad()

td\_loss.backward(retain\_graph=True)

self.optimizer.step()

# Ֆունկցիա, որը ստանում է անցած գործողության մրցանակը, օգտագործում է և տալիս տվյալներ դրա մասին

def update(self, reward, new\_signal):

new\_state = torch.Tensor(new\_signal).float().unsqueeze(0)

self.memory.push(

(self.last\_state, new\_state, torch.LongTensor([int(self.last\_action)]), torch.Tensor([self.last\_reward])))

action = self.select\_action(new\_state)

if len(self.memory.memory) > 100:

batch\_state, batch\_next\_state, batch\_action, batch\_reward = self.memory.sample(100)

self.learn(batch\_state, batch\_next\_state, batch\_reward, batch\_action)

self.last\_action = action

self.last\_state = new\_state

self.last\_reward = reward

self.reward\_window.append(reward)

if len(self.reward\_window) > 1000:

del self.reward\_window[0]

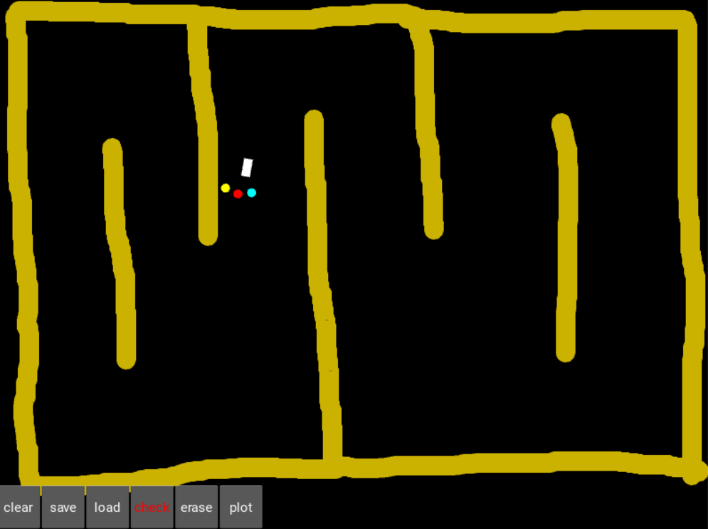
return action

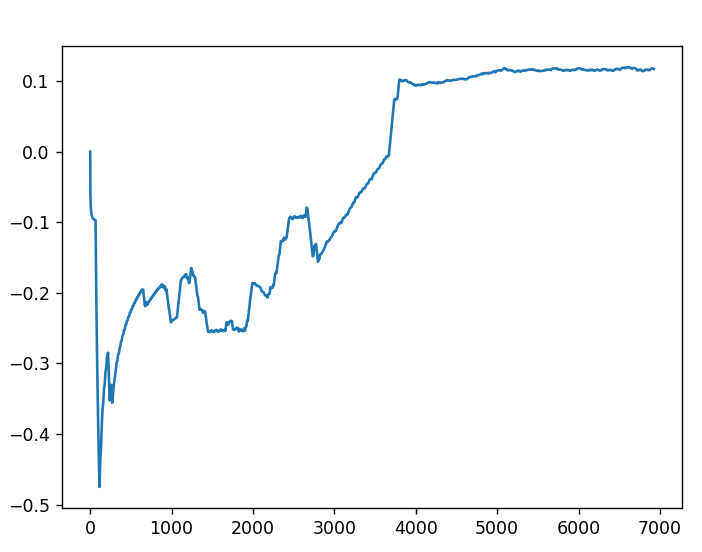
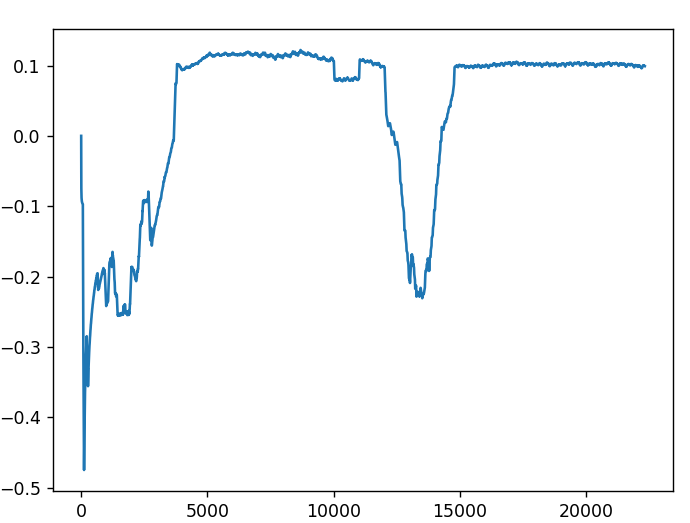
# Ստանում ենք վերջին 1000 արժեքների միջինը

def score(self):

return sum(self.reward\_window) / (len(self.reward\_window) + 1.)

### Արդյունքներ

Մեքենան կարողանում է ճանապարհ գտնել միջին բարդության քարտեզներում։



# Օգտագործված Գրականություն

<https://docs.google.com/document/d/1tuqZVDI7hLM5jHx8se2AB26G0qIXZ3WgKPw293D7IcI/edit?usp=sharing>

<https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0>

<http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.7692>

<https://playground.tensorflow.org/#activation=relu&batchSize=10&dataset=spiral&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=8,6,4&seed=0.51386&showTestData=false&discretize=false&percTrainData=80&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>

<http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf>

<https://cs.nju.edu.cn/wujx/paper/CNN.pdf>

<https://arxiv.org/pdf/1609.04112.pdf>

<https://arxiv.org/pdf/1502.01852.pdf>

<http://ais.uni-bonn.de/papers/icann2010_maxpool.pdf>

<https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html>

<https://peterroelants.github.io/posts/cross-entropy-softmax/>

<https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/>

<https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw/1097#1097>

<https://www.cs.ryerson.ca/~aharley/vis/conv/flat.html>

<http://yann.lecun.com/exdb/mnist/>

<https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56>

<https://en.wikipedia.org/wiki/Reinforcement_learning>

<http://www.incompleteideas.net/book/RLbook2018trimmed.pdf>

<https://www.quora.com/How-does-Q-learning-work-1>

<https://deeplizard.com/learn/video/Bcuj2fTH4_4>

<https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/>

<https://www.udemy.com/course/artificial-intelligence-az>

<https://deeplizard.com/learn/video/Bcuj2fTH4_4>

<https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>

<https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html#:~:text=The%20process%20of%20minimizing%20(or,problems%20by%20minimizing%20the%20function>

<http://yann.lecun.com/exdb/mnist/>