

|  |  |
| --- | --- |
|  | УНИВЕРЗИТЕТ У НИШУ  ЕЛЕКТРОНСКИ ФАКУЛТЕТ |

**КЛАСИФИКАЦИЈА ЦРТЕЖА УЗ ПОМОЋ НЕУРОНСКЕ МРЕЖЕ**

Дипломски рад

Студијски програм: Електротехника и рачунарство Модул: Рачунарство и инфроматика

|  |  |
| --- | --- |
| Студент: | Ментор: |
| Немања Станковић, 17956 | Проф. др Леонид Стоименов |

Ниш, 2024. година

Универзитет у Нишу

Електронски факултет

#### КЛАСИФИКАЦИЈА ЦРТЕЖА УЗ ПОМОЋ НЕУРОНСКЕ МРЕЖЕ

#### CLASSIFICATION OF DRAWINGS USING A NEURAL NETWORK

Дипломски рад

Студијски програм: Електротехника и рачунарство

Модул:

Рачунарство и информатика

Студент:

Немања Станковић, 17956

Ментор:

Проф. др Леонид Стоименов

Задатак: *Имплементација система који погађа шта је корисник нацртао помоћу неуронске мреже, односно класификација цртежа у категорије за које је систем научен да препозна цртеже*

Датум пријаве рада: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Датум предаје рада: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Датум одбране: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Комисија

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Председник

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члан

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члан

КЛАСИФИКАЦИЈА ЦРТЕЖА УЗ ПОМОЋ НЕУРОНСКЕ МРЕЖЕ

**САЖЕТАК**

Циљ је саставити систем који ће на основу цртежа да научи алгоритме и шаблоне по којима се категорије цртежа међусобно разликују и да успешно и сам препозна шта се налази на цртежу који пре није никада видео. Систем се реализује помоћу неуронске мреже која као улаз узима слике из базе података и на основу њих учи наведене карактеристике за препознавање. Омогућава кориснику да и сам размотри како систем функционише тако што му пружа поље за цртање и у реалном времену му приказује своју претпоставку о томе шта је на слици. Даје му могућност да провери тачност резултата над тестираним подацима. И да и сам врши процес тренирања притиском на дугме.

**Кључне речи**: Неурон, неуронска мрежа, дубоко учење, цртање, препознавање цртежа

CLASSIFICATION OF DRAWINGS USING A NEURAL NETWORK

#### ABSTRACT

The goal is to put together a system that will learn algorithms and templates, on the basis of drawings by which the categories of drawings differ from each other and to successfully recognize new, never seen before, drawings. The system is implemented using a neural network that takes images from the database as input and use them to learn specified characteristics for recognition. It allows the user to see for himself how the system works by providing him with a drawing field and showing neural network’s guesses about what is in the picture. It gives him the possibility to check the accuracy of the results on the tested data. And that he can activate training process himself on a click of a button.

**Keywords**: neuron, neural network, deep learning, drawing, drawing recognition

**САДРЖАЈ**

[1. УВОД 6](#_Toc182517012)

[2. Примена неуронске мреже за одређивање функционалне зависности 7](#_Toc182517013)

[2.1.Одређивање линеарне зависности помоћу неуронске мреже 7](#_Toc182517014)

[2.2 Одређивање нелинеарне зависности у неуронској мрежи 10](#_Toc182517015)

[3. Примери примене неуронске мреже за препознавање цртежа 21](#_Toc182517016)

[3.1.Quick Draw 21](#_Toc182517017)

[3.2 AutoDraw 22](#_Toc182517018)

[4. Имплементација решења 24](#_Toc182517019)

[4.1. Припрема података за тренирање и тестирање 26](#_Toc182517020)

[4.2. Кориснички интерфејс 28](#_Toc182517021)

[4.3 Имплементација неуронске мреже 30](#_Toc182517022)

[4.4 Приказ грешеке при учењу 34](#_Toc182517023)

[5. Тестирање утицаја различитих параметара неуронске мреже на њене перформансе 37](#_Toc182517024)

[5.1 Утицај понављања броја учења на прецизност и перформансе система 37](#_Toc182517025)

[5.2 Утицај степена учења на прецизност и перформансе система 37](#_Toc182517026)

[5.3 Прецизност у зависности од количине података 39](#_Toc182517027)

[5.4 Прецизност у зависности од броја чворова једног скривеног слоја 40](#_Toc182517028)

[5.5 Прецизност у зависности од броја скривених слојева 41](#_Toc182517029)

[6. Закључак 43](#_Toc182517030)

[LITERATURA 44](#_Toc182517031)

# УВОД

Вештачка интелигенција је данас једна од најважнијих и најпопуларнијих области рачунарства. Осим истраживања на универзитетима широм света, вештачка интелигеницја се данас примењује или се очекује примена у различитим доменима и областима рада за решавање различитих проблема. Ипак, и даље постоје неистражене опције и могућности које могу да доведу до великих помака у овој области.

Основа овог рада је у експерименту који спроводи Google, који се односи на препознавање цртежа који су руком цртани. Идеја је да се имплементира апликација за препознавање цртежа инспирисана гугловом игром quick draw. Гугл је обезбедио јавни, велики скуп података који се може користити за ове намене, који ће се користити за тренирање и тестирање апликације.

У дипломском раду најпре ће бити описани основни појмови који се односе на неуронске мреже, деловима неуронске мреже и њихова улога. Приказаће се најпре функционисање ове мреже на једноставном примеру са линеарном зависношћу као и функционисање са сложенијим типовима зависности. Дипломски рад се бави и темама које се односе на одређивање свих карактеристика неуронске мреже као што су тежине потега, *bias* и како одредити колико треба променити ове вредности да би неуронску мрежу прилагодили конкретном задатку који је пред нама. Анализираће се и грешка при одређивању решења и то како она утиче на промену наведених карактеристика.

Практични део рада се односи на понављање гугловог експеримента. Приказано је како је и сам гугл решио проблем препознавања слике и од чега се састоји њихов модел података. Биће описана и имплементација решења у *javascript*-у над мањим скупом података из базе. Тренирање овог система ће бити извршено за десет објекта и треба омогућити кориснику да даље тренира неуронску мрежу да би добио већу прецизност односно да на притисак дугмета тестира шта је коначна претпоставка те мреже. Као улаз у неуронску мрежу користи се вектор пиксела а као излаз вектор са десет елемената где сваки одговара шанси да је на слици баш тај објекат. Биће описана имплементација класе за рад са матрицама која има кључну улогу у раду неуронске мреже као и класа неуронске мреже помоћу којих се врши тестирање и тренирање. Апликација има једноставан интерфејс који треба да покаже основне функционалности, да омогући кориснику да и сам нацрта нешто и да у реалном времену прикаже у којој мери апликација тачно одређује резултат, као и да ако не буде задовољан да може да понови процес учења.

Након имплементације извршена су тестирања током којих ће да се провери функционисање апликације при мењању неких од карактеристика неуронске мреже. Шта ће се у систему променити ако процес учења покренемо више пута и како ће се то одразити на саму прецизност, како ће промена структуре неуронске мреже (као што је промена броја слојева и чворова) да утиче на перформансе као и да ли повећање броја података нужно значи бољи рад система или је то беспотребно повећање утрошеног времена на рад апликације. Приказано је са којим се проблемима неуронска мрежа суочава при учењу односно како људска непредвидивост утиче на процес учења, а

касније на процес погађања.

# Примена неуронске мреже за одређивање функционалне зависности

Неуронска мрежа је систем вештачке интелигенције који се састоји од низа међусобно повезаних процесора чија је улога да изврше израчунавање над унешеним подацима.

Њена улога је да на основу познатих улазних и излазних података уочи правилности при закључивању и у складу са њима подеси своје параметре. Најпре ћемо на простом примеру где важи линеарна зависност уочити како неуронска мрежа функционише и који су њени саставни делови. Након тога на примеру нелинеарне зависности ћемо испратити како треба проширити систем и прилагодити га сложенијем проблему.

## 2.1.Одређивање линеарне зависности помоћу неуронске мреже

Перцептрон је алгоритам за надгледано учење бинарних класификатора. Ако постоји вектор који се састоји од три елемента *x1*, *x2*, и *x3* он на основу математичког израчунавања над вредностима *x1*, *x2* и *x3* и одређује излаз *y*. Користи се за линеарну зависност.



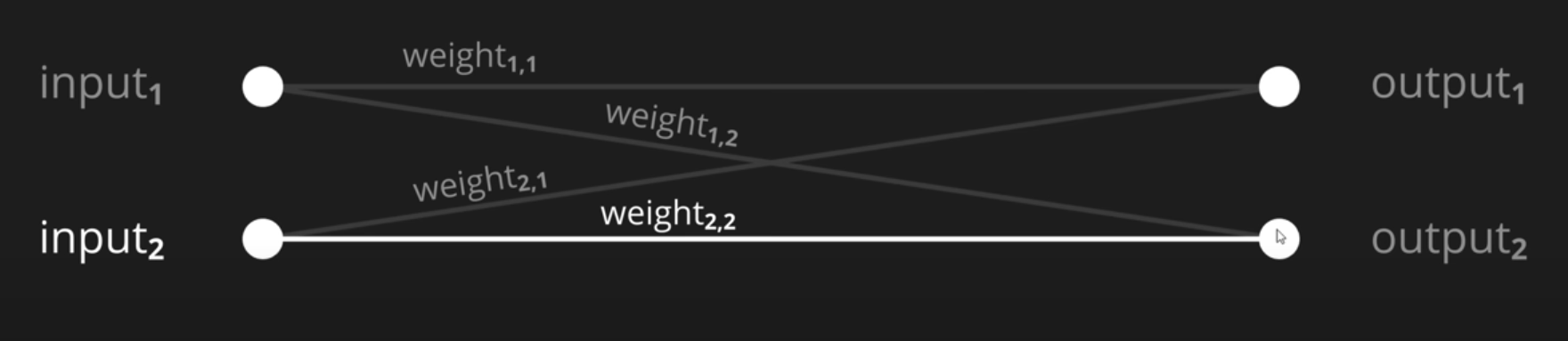
Слика 2.1 Биљка у средини је пример неотровне врсте, а биљке са њене леве односно десне стране примери отровне биљке.

Један прост пример како перцептрон одређује излаз *у* може да буде представљен употребом биљке која на себи има трнове и тачке жуте боје. Међутим ако је познато да постоји још врста биљака са сличним особинама, а за које се испоставило да су отровне поставља се питање како разликовати једне од других (Слика 2.1). Овде се да би пример био што простији узима зависност где већи трнови или веће тачке од неке референтне вредности значе да је она отровна. Свака од биљака може да буде представљена на графу где *y* оса представља дужину трна, а *x* оса величину мрља. Попуњавањем графа за горе наведени пример добијамо линеарну зависност могућности да биљка буде отровна од величине трња и жутих мрља. Након што распоредимо биљке према овим карактеристикама добијамо две површине на графу које могу међусобно да се одвоје једном правом. За реализацију овог решења користићемо неуронску мрежу са два улазна и два излазна чвора. Улази описују величине трња и жутих тачака, а излази ће бити означени као output1 и output2. Уколико је output1 > output2 знаћемо да је у питању неотрован плод, у супротном реч је о отровном. Знамо да оба улаза утичу на излаз али не и у којој мери. Та зависност се описује тежином потега између улаза *i*  и излаза *j* у ознаци *wi,j*. На неуронској мрежи је да у процесу учења уочи ове зависности и на основу њих подеси те тежине између излазних и улазних чворова. Добијамо зависност:

output1 = input1\*weight1,1+input2\*weight2,1

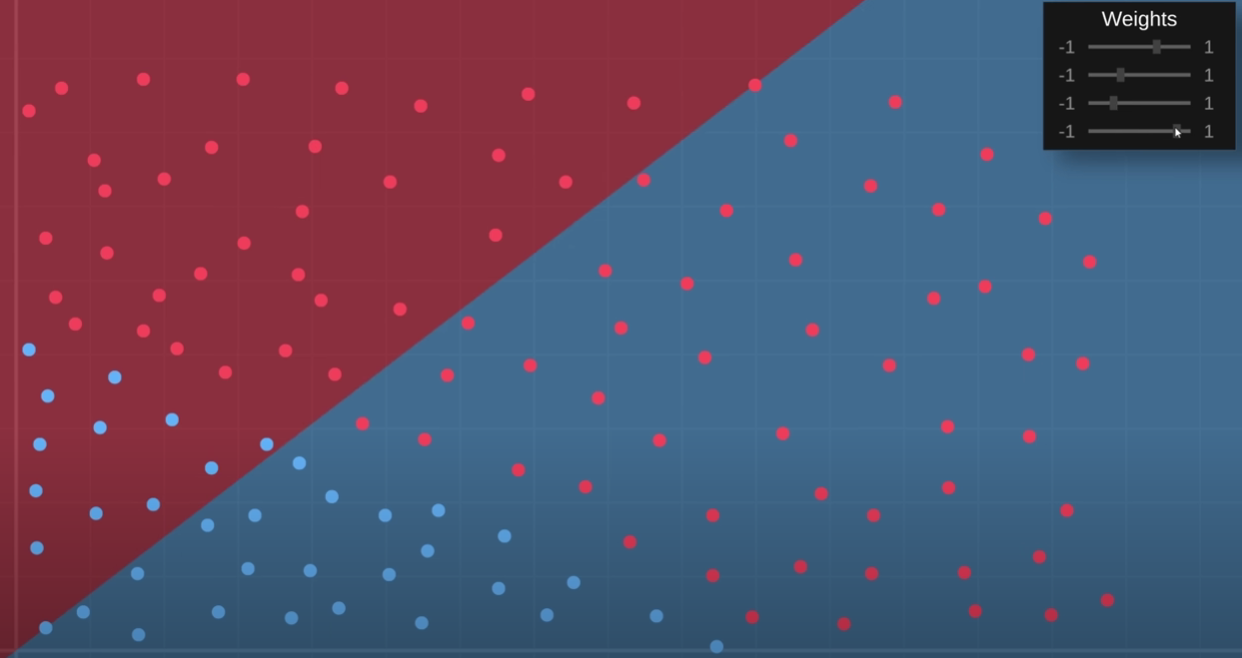
output2 = input1\*weight1,2+input2\*weight2,2

Проста неуронска мрежа која је описана овом зависношћу налази се на слици 2.2



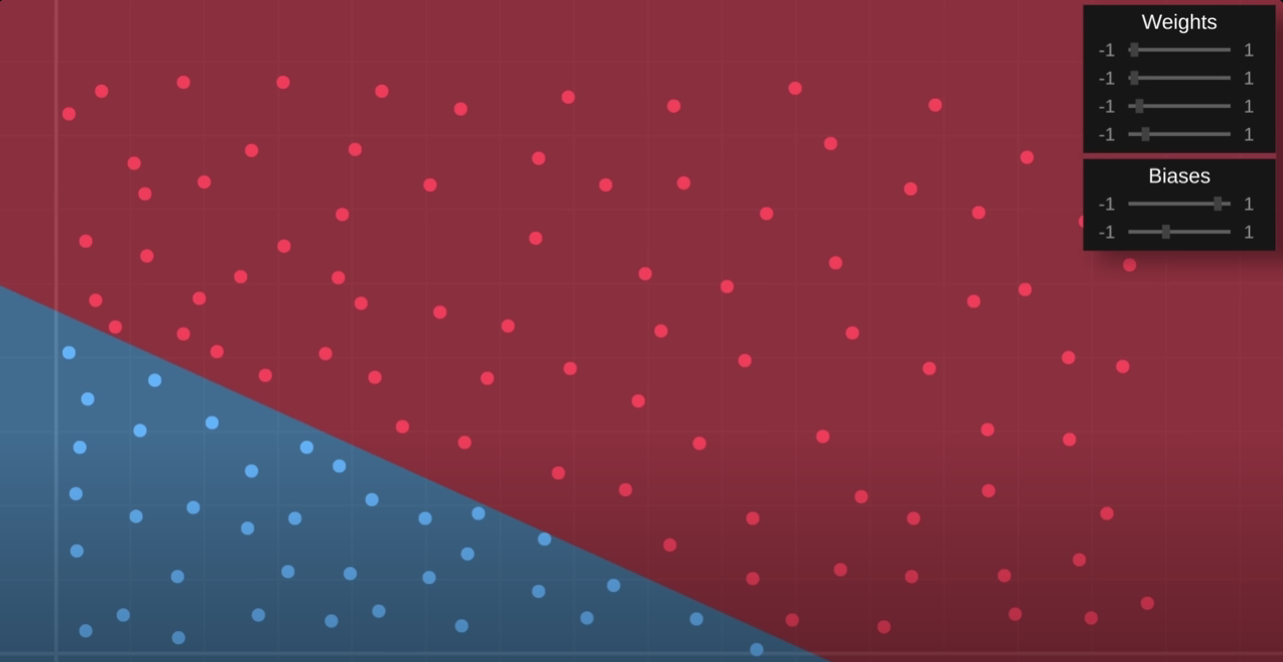
Слика 2.2 Одређивање зависности између улаза и излаза

Променом тежина у процесу учења добијамо ефекат где се линија која одваја отровне од неотровних биљака (односно један излаз од другог) ротира око координатног почетка што за наш пример нема реалне примене. Одавде се добијају две површине (црвена и плава) које представљају случај отровних односно неотровних биљака респективно. Са слике се јасно види да врсте гљива које су означене тачком одређене боје нису правилно распоређене на површину те исте боје. (Слика 2.3). Оно што нама треба је да линија не пролази кроз координатни почетак већ да се пресек помери вертикално навише. [1]



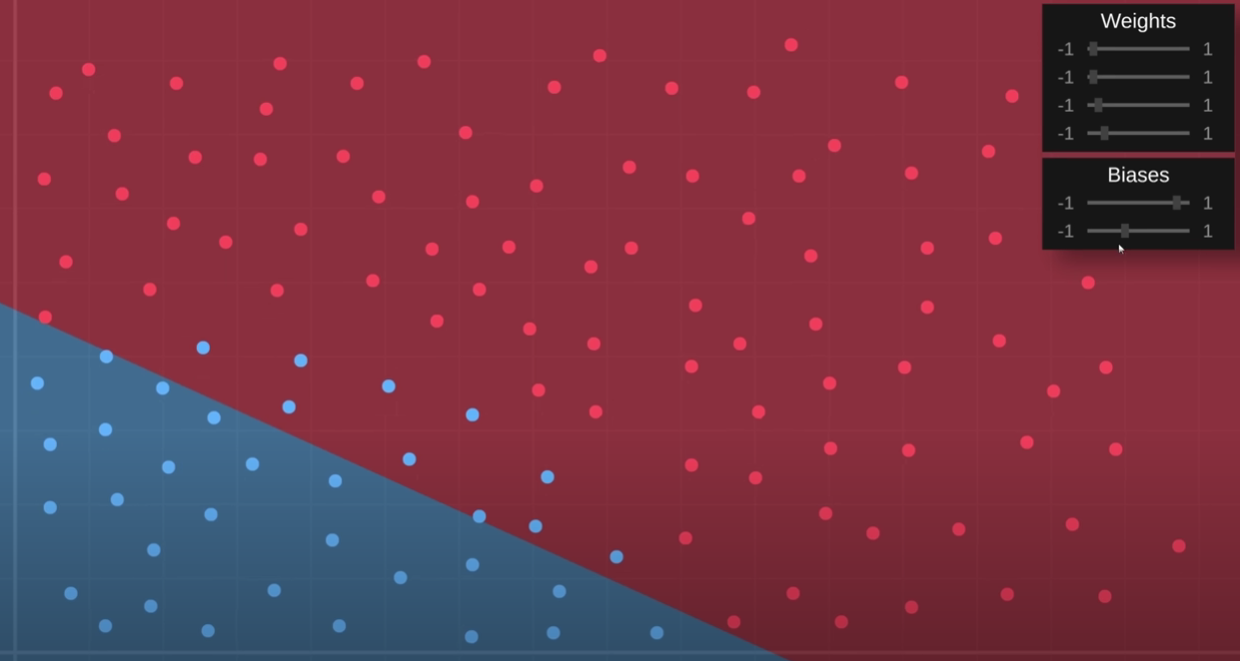
Слика 2.3 Пример класификације без *bias*-а.

Ову промену нам омогућује додавање *bias*-а. *Bias* представља вредност која се додаје на збир производа из претходне једначине. Даљим учењем се добија линија са слике (Слика 2.4) што одговара нашем примеру.



Слика 2.4 Линеарна зависност отровности биљке у зависности од велиличине бодљи и жутих тачака са употребом *bias-*а.

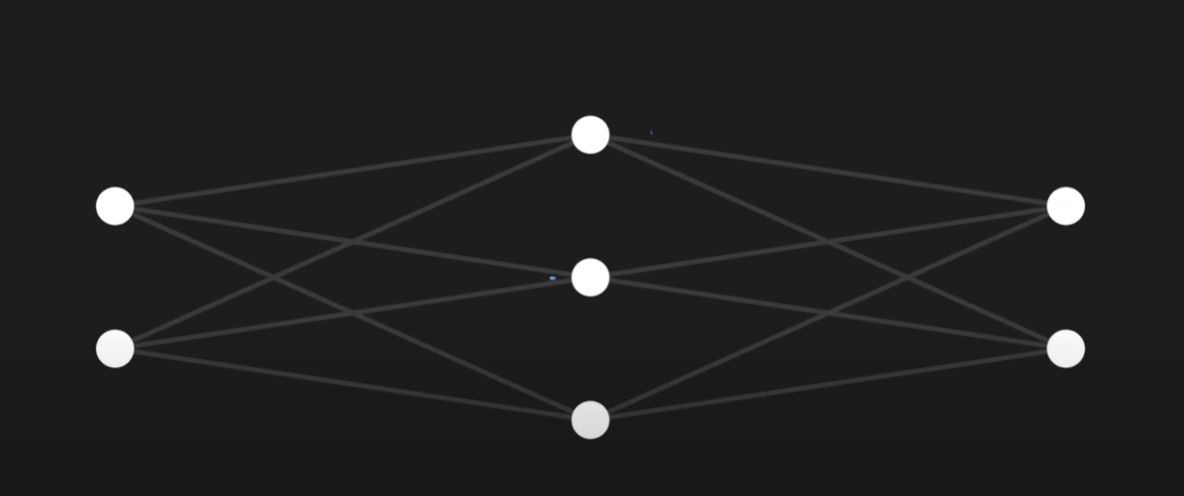
До решења је лако доћи и ручно када је зависност линеарна и за прост пример као што је овај али ако добијемо да зависност изгледа као на слици (Слика 2.5), где сада граница између два случаја није права линија, потребно је извршити даљу модификацију решења.



Слика 2.5 Класификација код које не важи линеарна зависност

За такав сложенији случај неопходно је додати нови слој са чворовима између улазног и излазног. Тај слој се назива скривени слој јер су његови резултати и улазни параметри невидљиви споља. Уместо потега између чворова улазног и излазног слоја сада имамо потеге између свих чворова улазног и скривеног слоја и свих чворова скривеног и излазног слоја.

Свака неуронска мрежа садржи све потеге између свих слојева па се подешавањем вредности тежине на нула постиже ефекат да су неурони међусобно раздвојени. Ово омогућава да се зависност динамички постави током процеса учења и ако није била позната кад је систем креиран



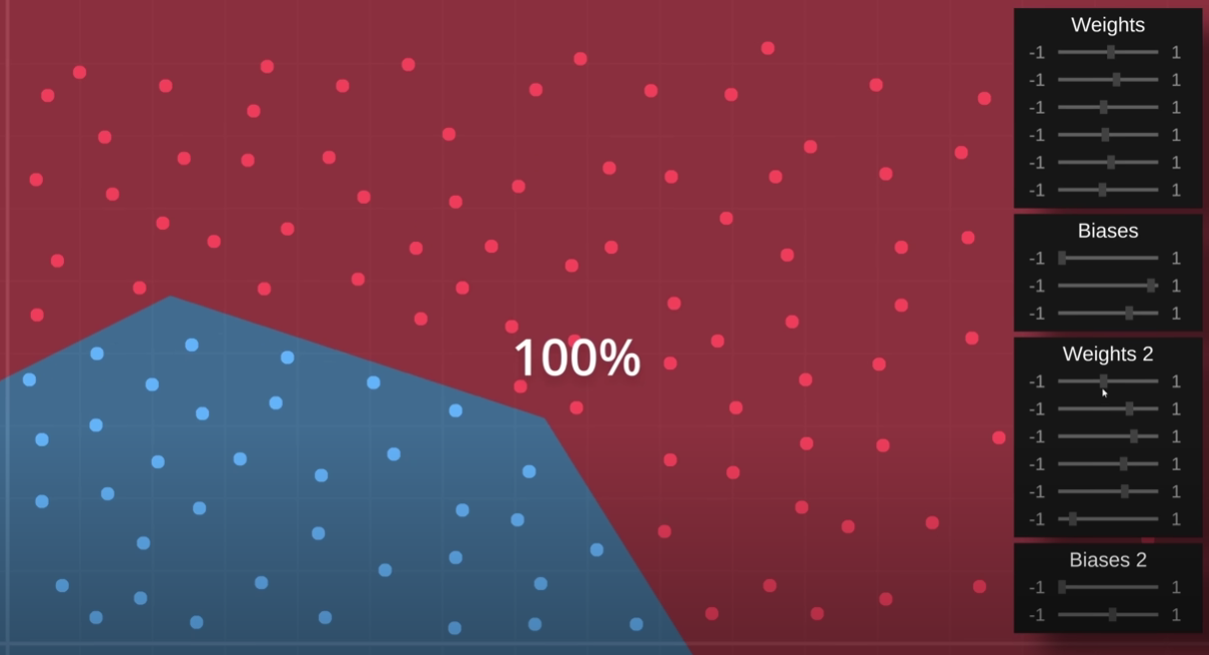
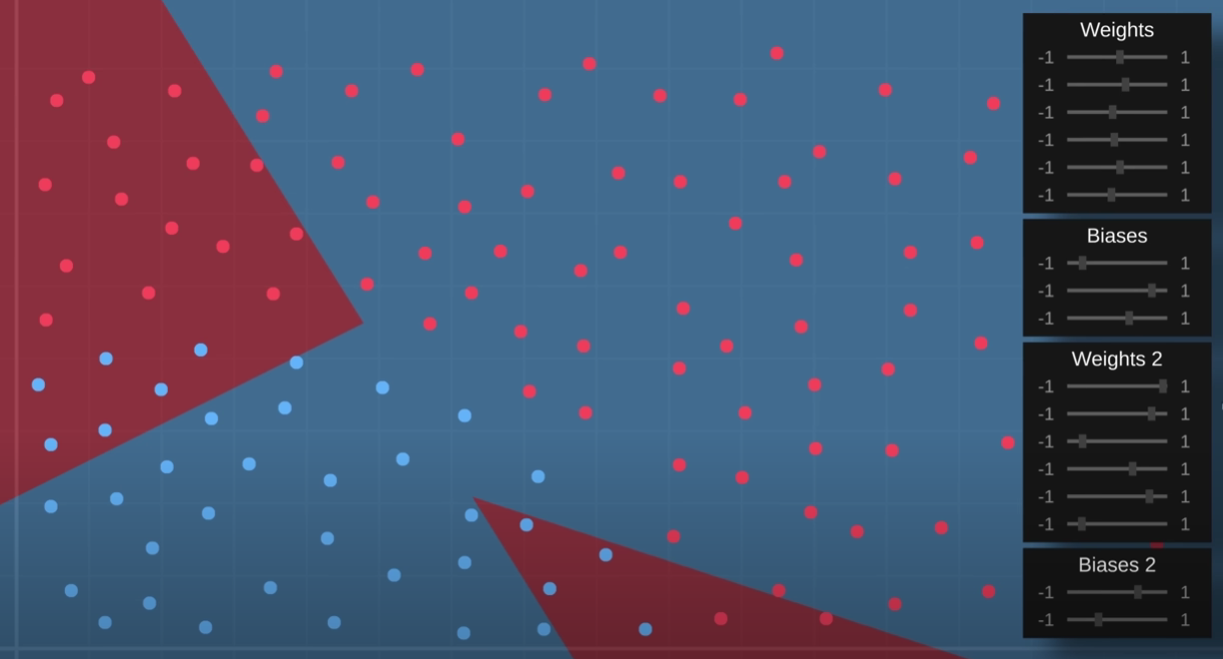
Слика 2.6 Нова шема неуронске мреже са скривеним слојем

Слика 2.6 управо представља мрежу проширену средњим скривеним слојем са три чвора.

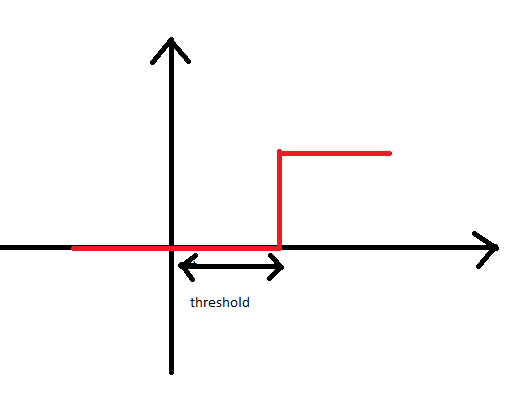
## 2.2 Одређивање нелинеарне зависности у неуронској мрежи

Поново се рачуна збир производа одговарајућег улаза и потега који од њега води само сада прво за скривени па за излазни слој. Међутим, ово није довољно да мрежа дође до очекиваног резултата са слике (Слика 2.7). Потребно је додати логику којом ће чворови у скривеном слоју да пропуштају сигнал тек када он пређе одређену вредност. Та вредност представља доњи праг активационе функције, а активациона функција је функција која претвара излаз у неки мањи опсег врдности који је погоднији за одређивање када треба пропустити сигнал. Ово омогућава посебан случај коришћења где један од улаза може да буде интензивнији од осталих али да и он сам буде довољан за активацију неурона и пролазак сигнала ако може да достигне тај доњи праг за активацију. Код примера за класификацју цртежа ово се јасно види при цртању сунца. Један велики круг је довољан да систем претпостави да је у питању Сунце.

Употребом просте активационе функције која враћа 0 када је улазна вредност негативна, а 1 када је позитивна добијамо праву којом је могуће поделити отровне и неотрновне биљке као на слици (слика 2.7). На левој слици је приказано како ручна промена неких од параметара система може да изазове промену површина отровних и неотровних биљака, а на слици десно валидно решење нашег проблема.

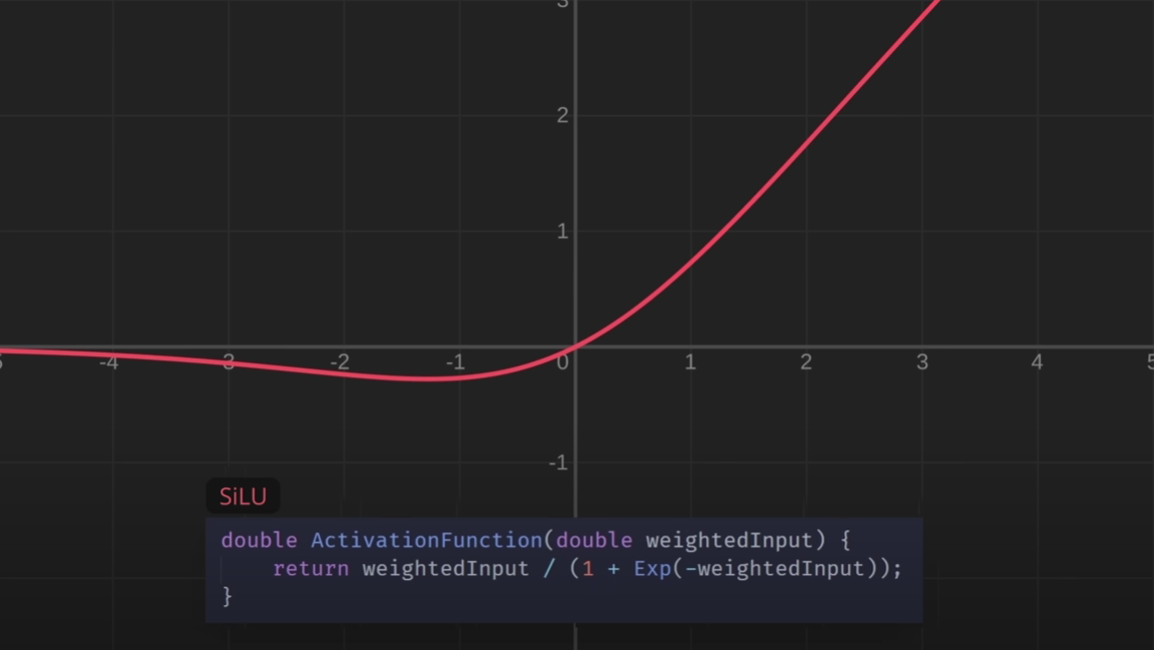


Слика 2.7 Пример рада система са скривеним слојем и применом просте активационе функције



Слика 2.8 Изглед просте активационе функције

Ивице линије која дели две површине на слици (Слика 2.7) су преоштре. То је последица коришћења просте активационе функције чији граф зависности излазног сигнала од улазног је дат на слици 2.8. Она враћа нулу ако је улаз мањи од предефинисане вредности *threshold*, а враћа јединицу ако је већи.Ово се лако решава применом другачије активационе функције. Један од примера такве функције је Сигмоидова функција. Осим ње постоји још пар примера активационих функција које се често користе.



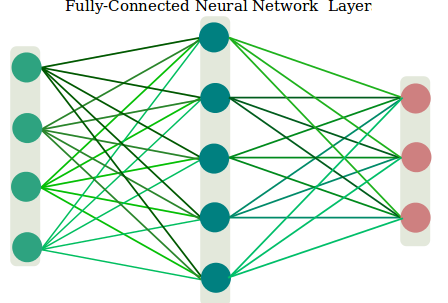


Слика 2.9 Пример сложенијих активационих функција

На слици 2.9 дат је пример сложенијих активационих функција и имплементација функције за одређивање тачака на њој. SiLU даје излаз и за негативну вредност улаза тако да и негативан улаз утиче на излаз, док код ReLU важи да за негативне вредности улаза на излазу иде нула тј. рачуна се као да тај улаз не утиче на излаз. За позитивне вредности само их прослеђује.

SiLU (Sigmoid linear unit).

ReLU (Rectified linear unit).



Слика 2.10 Потпуно повезана науронска мрежа са једним скривеним слојем са пет чворова, улазним слојем са четири чвора и излазним слојем са три чвора

Када су сви потези постављени на вредности између -1 и 1 даље је на неуронској мрежи да подеси те вредности тако да на излазу да очекивани резултат тј. да број грешака сведе на минимум. Још једна особина о којој треба водити рачуна је опсег вредности које улазни подаци могу да имају. Због огромне разлике која може да се јави између улаза неопходно је извршити нормализацију тих вредности. У супротном може доћи до грешке при подешавању тежина потега. На пр. при одређивању цене стана улазни параметри могу да буду број соба, број купатила и квадратура стана где квадратура може да буде и троцифрени број, а број соба и купатила да буде 1 до 3 или 4. Ово се решава нормализацијом тј. свођењем улазних вредности на неки мањи опсег бројева на пр. -1 до 1. На овај начин максимална разлика између вредности може да буде 2 а не стотину или хиљаду.

Ако тежине сваког од потега означимо као w*ij*где је *i* редни број улаза а *j* редни број скривеног чвора за потпуно повезану неуронску мрежу са два скривена чвора и два улаза добијемо тежине *w11*, *w12*, *w21*, *w22*. Матричном анотацијом ово можемо да представимо као

Множењем матрице вектором добијамо што је збир производа о којем је било речи.

Одавде следи да добијамо . Ако се вратимо на ситуацију где сви улази *I* могу да буду 0 што ће увек да врати 0 за излаз, потребно је додати и *bias*

Над добијеном вредношћу се примењује Сигмоидова активациона функција.

Једначина Сигмоидове криве је

Разлог за коришћење Сигмоидове функције је што за било који улаз враћа излаз у опсегу . Веће вредности улаза ће бити ближе јединици а мање ближе нули. Корисно је и при одређивању вероватноће у процентима што се остварује простим множењем са 100.

За случај када се користи вишеслојна неуронска мрежа израчунвање остаје исто само се мењају чланови једначине. Користе се потези између скривеног и излазног слоја који се сада множе са вектором излаза из скривеног слоја уместо са улазним вектором. На крају се додаје *bias* који је везан уз скривени слој и није исти као онај који се користи између улазног и скривеног слоја.

**2.3 Одређивање грешке неуронске мреже**

Након што смо установили начин одређивања излаза из неуронске мреже следећи корак је да видимо како ће она на основу добијеног излаза да одреди какве промене над тежинама треба да изврши.

Први корак у извођењу закључка је надгледано учење. У том процесу се неуронској мрежи или перцептрону дају улазни параметри и очекивано решење. Циљ је да се изврши израчунавање над тренутним вредностима тежина потега, улазних параметара и *bias*-а и да се добијено решење упореди са оним прослеђеним (очекиваним). Управо на овај начин се рачуна грешка на излазу.

*error* = *answer* – *guess*

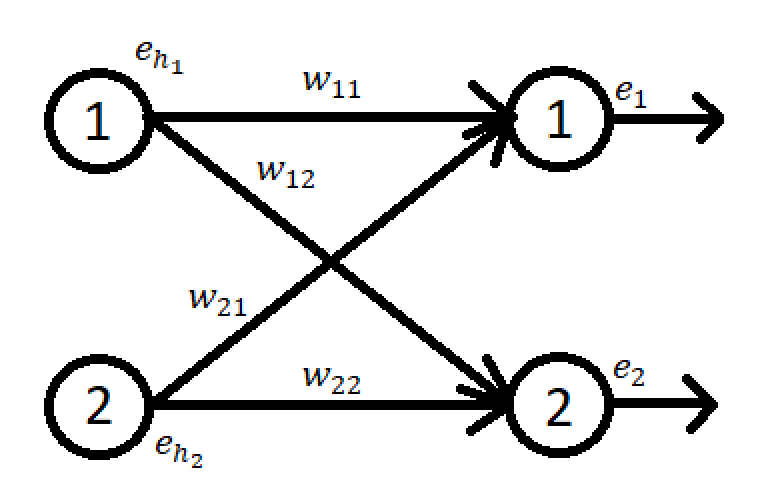
Променом вредности тежина и *bias-*а треба што ближе доћи до тог очекиваног резултата тј. смањити грешку. Свака од тежина потега представља утицај улаза који је тим потегом повезан на излазну вредност. На пример, уколико је вредност потега из једног улаза дупло већа од вредности потега из неког другог улаза значи да ће промена првог дупло више утицати на промену излаза. Пракса је да се ове вредности поставе на произвољне вредности ако није унапред познат њихов однос и да се кроз процес надгледаног учења оне мењају у циљу добијања што веће прецизности целог система. Код простијих решења и оних за које се зна овај однос, тежине могу и ручно да се подесе.

Поставља се питање како променити тежину дубље на мрежи, односно између улазног и скривеног слоја или у случају где има више скривених слојева како одредити вредност грешке за сваки од њих. Како би неуронска мрежа знала какве промене треба да изврши између та два слоја потребно је извести нову вредност грешке на основу оне на излазном слоју и тежина чворова. Одређивање грешке на унутрашњим слојевима мреже на основу спољашњих се зове пропагација у назад или Бекпропагација (*Back propagation*). Она се на чворовима скривеног слоја одређује као:

Ово значи да смо *e* добили као грешку на излазу чвора који је са претходним слојем повезан потезима и . Нас занима који је утицај потега на грешку јер је он повезан са првим од два чвора скривеног слоја за који одређујемо грешку .

За случај када постоји више излаза и одатле више вредности грешака  вредност грешака у скривеним чворовима се одређује као:

Дати пример нуди решење када имамо два скривена и два излазна чвора

.

Слика 2.11 Одређивање грешке средњег слоја пропагациом у назад.

Представљено као производ матрице и вектора:

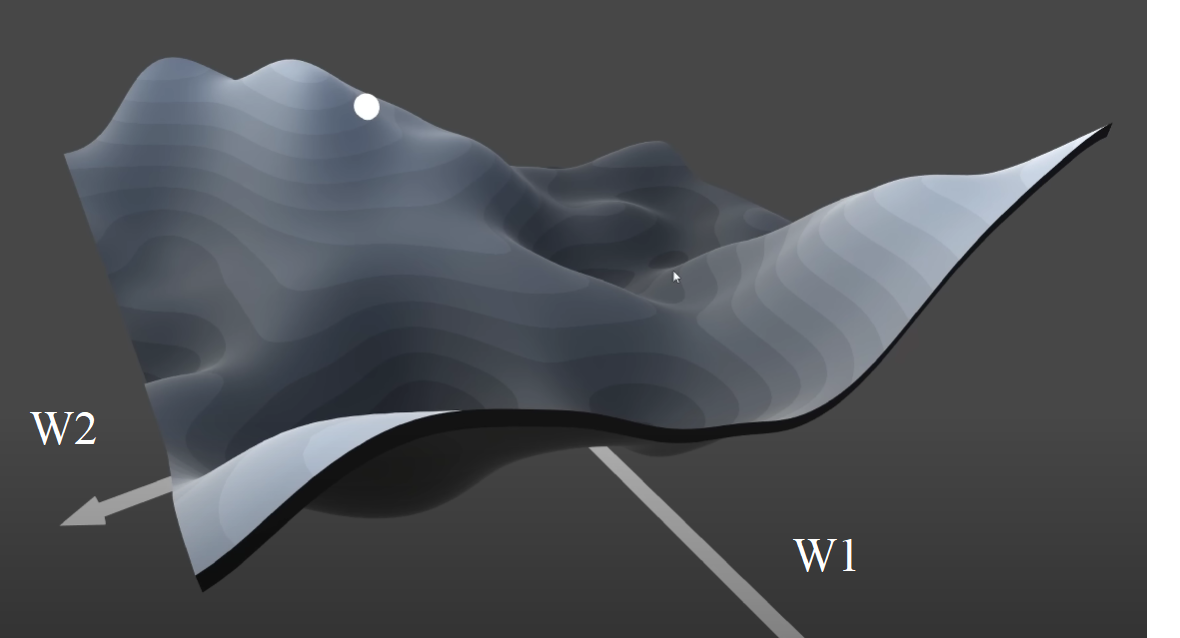
Уклањањем делилаца код свих сабирака добијају се доње једначине. Пропорције којима тежине утичу на резултат остају исте а множење матрица постаје знатно једноставније. Делилац је служио да изврши нормализацију вредности па се његовим избацивањем резултат није променио.

Што се своди на множење матрице:

Потребно је да се на основу грешке и брзине учења одреди промена нагиба и биас-а.

Одређивање тежина потега је компликован процес и сама провера сваке од комбинација у циљу налажења оне праве може да захтева превише корака за оптималан рад система.

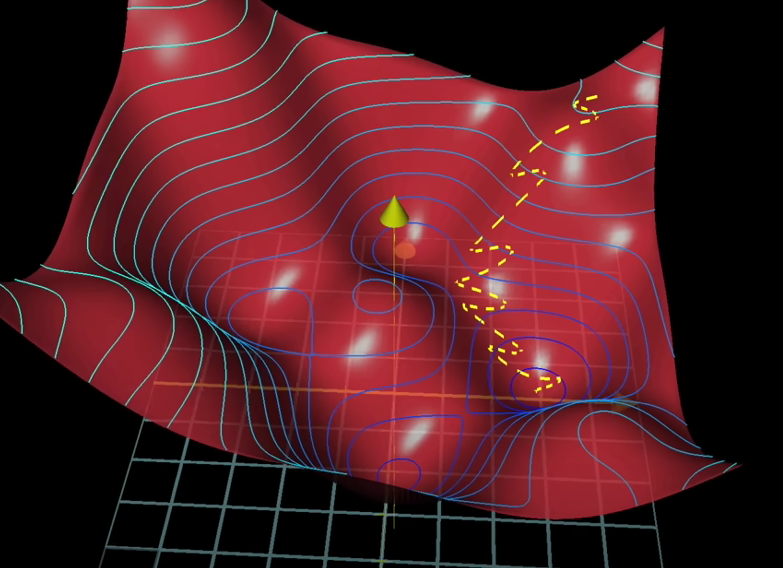
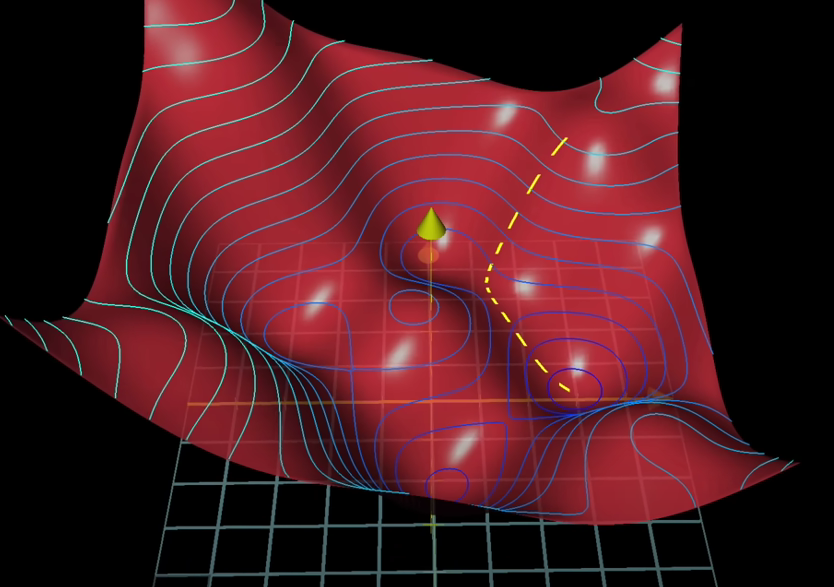
Писац Тарик Рашид је описао проблем при проналажењу правих тежина чворова као проблем планинара изгубљеног у мраку са лампом. Његов циљ је да пронађе пут до подножја планине. Валидан метод је да користи лампу да би у свом окружењу нашао пут који иде наниже и то онај најстрмији јер ће на тај начин најбрже да дође до подножја. Стрмина брда по којој се планинар креће одговара градијенту који ћемо користити при израчунавању нових тежина потега. Веће вредности означавају стрмије равни што значи да брже доводе до минимума[2]. Треба имати у виду да је у питању вишедимензиони простор тј. простор са онолико димензија колико има потега између два слоја па тај минимум не зависи само од кретања у једном правцу лево или десно као у случају дводимензионалног простора на слици (Слика 2.14). Ако пак за приказ проблема користимо тродимензионални простор где су хоризонтале осе (*x* и *z*) тежине два потега (*w1* и *w2*), а вертикална оса *y* представља грешку на излазу (*error = answer – guess*) добијамо ситуацију као у примеру са планинаром.



Слика 2.12 Утицај промена тежина потега на грешку

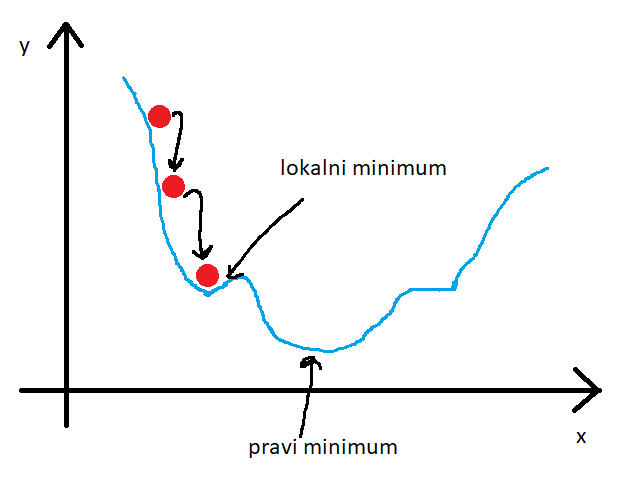
За представљање градијента ћемо да користимо вектор са онолико вредности колико има димензија наш простор што, као што је већ речено, зависи од броја потега између два слоја. На основу тежине сваког од елемената овог вектора се одређује који има највише утицаја на одређивање пута до подножја. А на основу знака се одређује у ком смеру треба да се креће планинар тј. да ли тежина треба да се смањи или повећа.

Често се учење а са њим и одређивање тежина потега између чворова врши тако што се подаци за учење мешају и деле у блокове. Блок или *batch* се састоји од одређеног броја података. Уместо да се за сваки од података одређује промена тежина и *bias*-а, овде се саберу све промене за сваки од податка у блоку и одреди се једна вредност промене *bias-*а и тежина и онда примени на систем. Тако да се на пут до минимума долази инкрементално у више корака (Слика 2.13).[3]



Слика 2.13 Долазак до резултата учењем над свим подацима и над подацима подељених у групе

Нови проблем настаје ако на тој истој планини има више долина од којих сваки пут води навише што одговара локалном минимуму неке сложене функције. И ако ово подсећа на најнижу тачку рељефа она то није. Да би се избегло да планинар стално запиње у ту исту долину треба узети различите почетне тачкe, у случају неуронске мреже то значи узимање различитих почетних вредности тежина.



Слика 2.14 Локални и прави минимум

Ситуација у којој се из неке произвољне тачке долази до локалног минимума приказана је на слици 2.14.

Разликујемо више метода за одређивање грешке при израчунавању резултата. Свака од њих креће од разлике добијене и очекиване вредности где се даље уз помоћ знака одређује шта даље са тежинама потега и bias-а. У зависности од методе израчунавања разликујемо следеће случајеве:

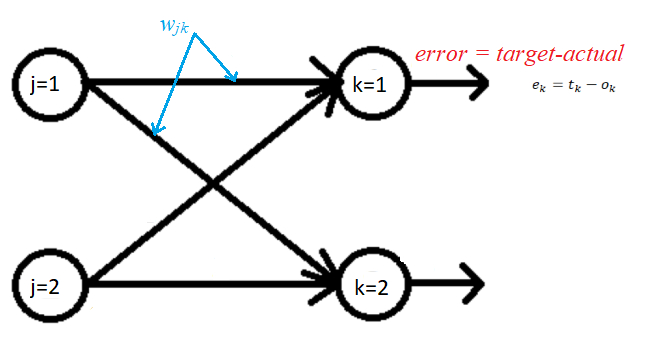
Ако кренемо од тога да се грешка при раду система одређује као проста разлика циљане и добијене вредности може доћи до тога да сума свих грешака буде нула или приближно нули јер се оне међусобно поништавају тј. Сума свих негативних вредности грешке је приближна суми свих позитивних вредности грешке. Укупна грешка ће да буде много мања од оне праве. Одавде се стварају услови за другу варијанту израчунавања грешке.

Други метод укључује сабирање апсолутне вредности разлика претпоставки и добијених вредности. Нови проблем се јавља јер ће сваки од минимума да има облик латиничног слова “V”. Ово је последица управо тога што се вредности грешке међусобно не поништавају. У том случају како се резултат приближава минимуму може да се деси да због величине корака прескочимо минимум и наша претпоставка о минимуму наставља да кружи око њега.

Трећа и најбоља варијанта је да се користи квадрат ове разлике где се не јавља проблем “прескакања” минимума јер је вредност градијента све мања и мања што се више приближавамо најнижој тачки. На овај начин се квадрирањем децималног броја добија још мањи број што ће да изазове да минимум буде шири и лакши за откривање.

**2.4 Одређивање промене тежине потега**

Оно што је сада наш циљ је да одредимо како се мења грешка при промени тежине чворова што одговара и нагибу функције. Ово је уједно и почетна тачка за одређивање промене тежина потега.



Слика 2.15 Пропагација у назад

Одавде за n чворова добијамо да је то сума грешака за сваки од *n* излаза, или:

За чвор k важи да на његов резултат утичу потези тј. Потези који су са њим повезани одавде следи:

Како је очекивани резултат ово је константна вредност која не зависи од тежине потега, док је претпоставка која директно зависи од ње.

Из правила ланца следи да извод може да се разломи као производ две везе за које знамо да важе. Како знамо да променом тежине се мења и излаз неуронске мреже и да се са променом излаза мења и грешка и то на исти начин (повећава или смањује) добијамо однос :

Рачунањем извода квадратне функције из прве једначине добијамо

се добија применом сигмоидове функције над сумама производа тежина чворова и њихових излаза где је излаз из претходног слоја.

Извод сигмоидове функције је:

Што нас доводи до једначине

Задњи облик функције се добија јер и задњем члану који је у загради треба да се одреди извод у односу на и опет из правила ланчања се добија да је то

Брисањем двојке из првог члана можемо да поједноставимо систем. То смемо да урадимо јер нам је битнији правац нагиба функције грешке а овај коефицијент не утиче на њега.

- разлика очекиваног и добијеног резултата (*error*)

- сигнал у последњем слоју пре употребе активационе функције

– излаз из чвора *j* претходног скривеног слоја

У случају промене тежина у дубљлим слојевима користи се иста веза с тим да сада важи да је први члан грешка која је добијена на излазу скривеног слоја.

Сигмоидова функција се сада примењује над сумом производа тежина између улазног и скривеног слоја и одговарајућих улаза у скривени слој.

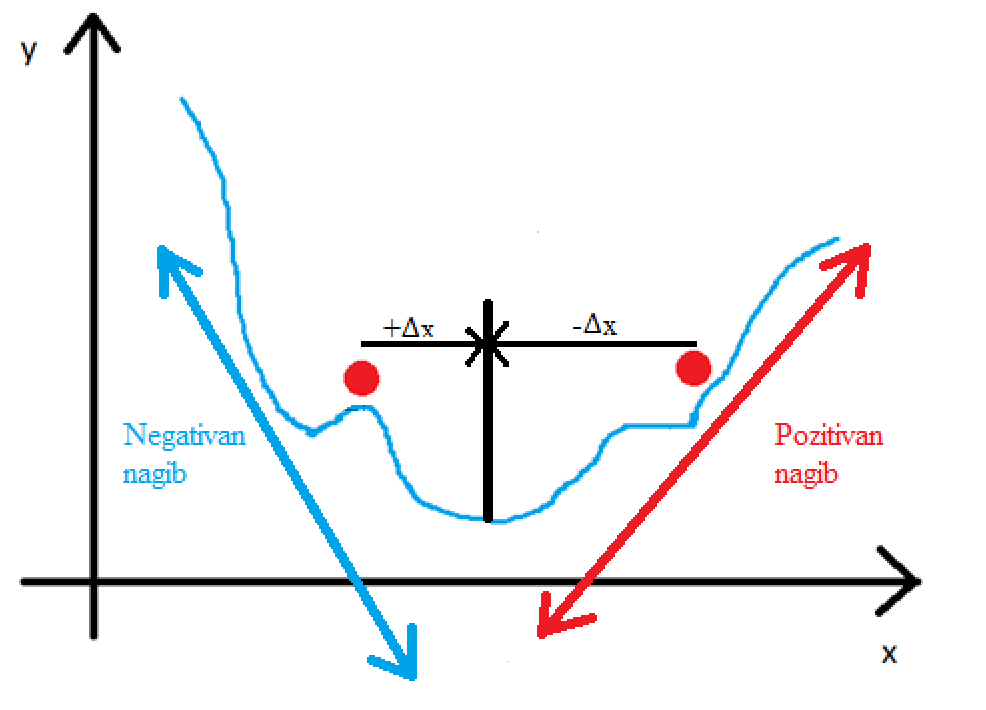
Задњи члан је излаз из првог слоја што је улазни сигнал.

Уз то се додаје и степен учења у ознаци α да би се спречило да се најбоља вредност прескочи због превеликог корака при промени тежина.

Слика 2.16 Промена грешке на сваком од излаза из неуронске мреже током три циклуса учења

Слика 2.16 описује како се мења грешка на излазу за сваки од ентитета током више пута поновљеног процеса учења. У случају неких од вредности дешава се да грешка пређе из негативне у позитивну вредност и обрнуто. Чак се за неке од случајева грешка повећава током процеса учења. Оно што се може закључити је да на послетку поновљено учење позитивно утиче на смањење грешке. И ако се дешавало да се грешке појединих објеката повећавају, сам збир грешака је све мањи и мањи.

Нова тежина се добија као разлика старе вредности и нагиба функције грешке. Алфа је степен учења. Нагиб функције грешке се узима са негативном вредношћу јер при позитивном нагибу хоћемо да повећамо тежине а при негативном да их смањимо што је приказано на слици 2.15.



Слика 2.17 Нагиб функције грешке

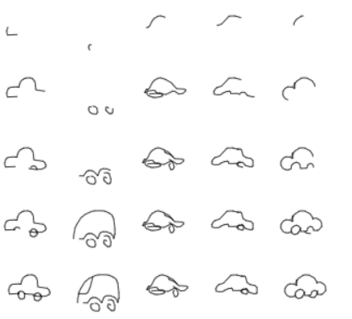
# Примери примене неуронске мреже за препознавање цртежа

Као што је већ поменуто идеја за реализацију овог решења долази од апликације *quick draw* [4] чија је база и коришћена при имплементацији. У наставку ће бити описане основне карактеристике ове апликације као и по нешто о сличним решењима.

## 3.1.Quick Draw

Најпознатије решење овог проблема стиже из гугла у виду онлајн игре *quickdraw*. База коју користи броји преко 50 милиона цртежа подељених у 345 категорија. Из дана у дан играњем игре корисници додају нове цртеже и побољшавају функционисање система. Корисник добије шест предмета које треба да нацрта док неуронска мрежа покуша да погоди шта је на слици. За то време алат на глас погађа шта мисли да корисник црта све док не погоди или не истекне време. У оба случаја се нови цртеж додаје у базу, чак и ако није довршен.

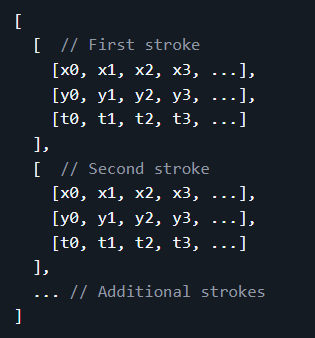
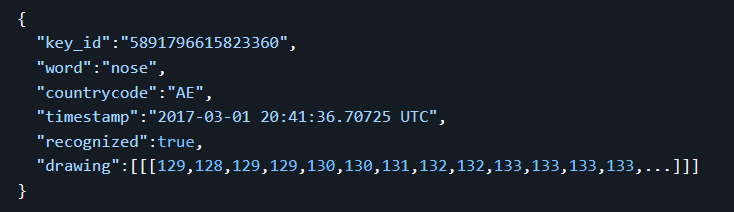
Док корисник црта чувају се информације о редоследу линија које се додају на површину за цртање и смер цртања. Ово је од помоћи јер при цртању неких објеката већина људи се држи неког редоследа па ова информација може да буде довољна да се разликују чак и објекти који су међусобно слични али се њихови делови не цртају у истом редоследу. На пример чак и најмлађи цртају каросерију аута пре точкова, зидове куће пре крова, велику ужарену лопту пре сунчевих зрака. Делови од којих неуронска мрежа највише учи су карактеристике објеката. Уколико корисник добије задатак да нацрта мачку. Свако ће прво да се сети шиљатих ушију, округле главе и бркова. Већи уметници међу корисницима ће да нацртају цело тело и због таквих специфичнијих ситуација је важно да неуронска мрежа има што више цртежа са којим ће да пореди оно што је корисник нацртао.



Слика 3.1. Начин на који су корисници цртали ауто у пет корака

Слика 3.1 илуструје како је пет различитих корисника цртало аутомобил од првог до задњег корака (одозго наниже).

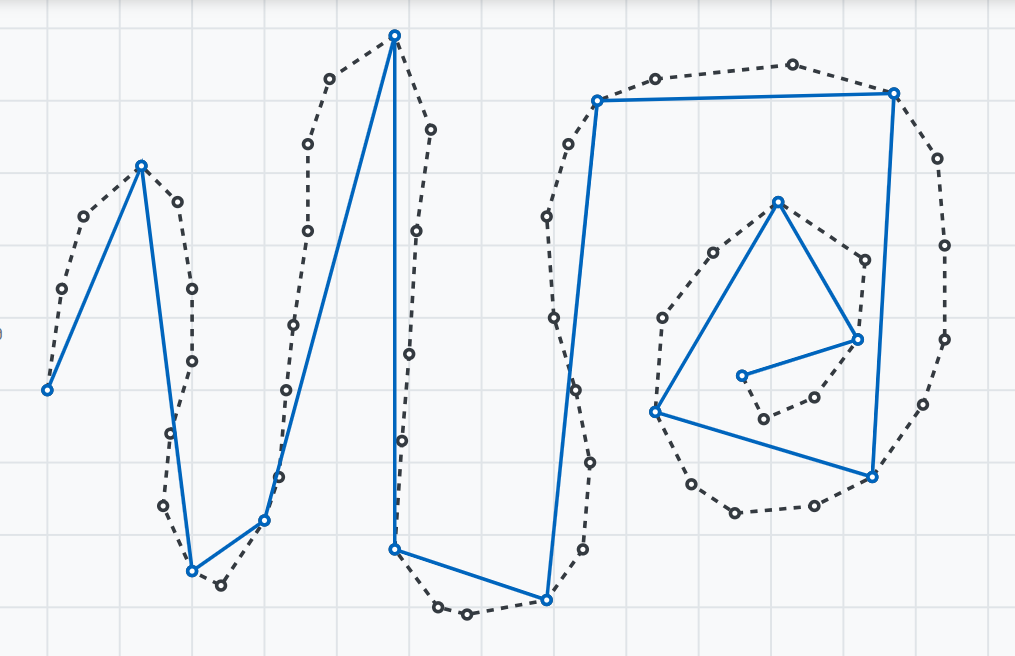
За сваки од цртежа се чувају идентификатор, реч о којој се ради као и ознака државе где је нацртан, време цртања и да ли је препознат. Атрибут *drawing* чува низ објеката где сваки од објеката представља један потез приликом цртања (од тренутка притиска тастера на мишу до отпуштања). Сваки од објеката чува низ x и y координата као и временских тренутака када је тај пиксел нацртан.(Слика 3.2)



Слика 3.2 Информације о цртежу и њихов формат у оквиру гуглове базе за *quick draw*[5]

Оптимизација је извшена следећим редом:

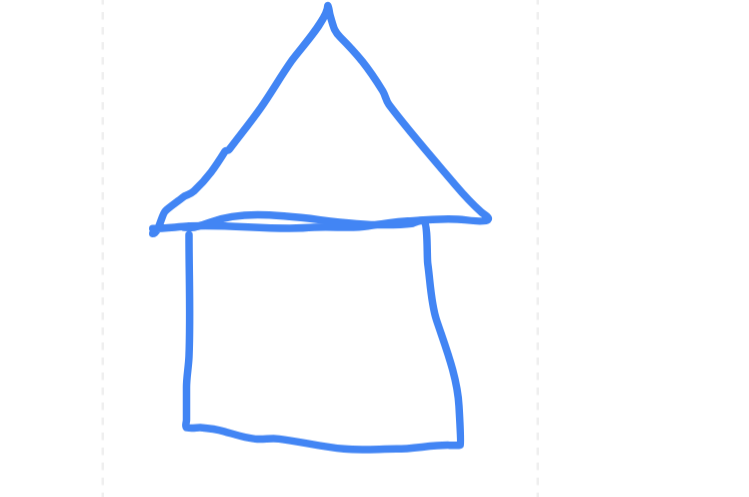
* смањењем слике на формат 255x255.
* цртеж је залепљен за горњу леву ивицу да би се смањио број нула.
* Скалиран је цртеж тако да је максимална вредност 255.
* Одвојени су потези размаком од једног пиксела ради прегледности.
* Извршено је упроштавање потега коришћењем Рамер-Даглас-Пекер алгоритма. (Његова улога је да смањи број линија које описују цртеже односно да замени криве правама. На слици (Слика 3.3) је пример како се овим алгоритмом смањује број ивица са 54 на 14. [6])



Слика 3.3 Примена Рамер-Даглас-Пекер алгоритма

## 3.2 AutoDraw

*AutoDraw* je aplikacija bazirana na *quickdraw* апликацији. Основа је иста. Корисник црта а неуронска мрежа погађа шта је на слици. Али ту није крај. Уместо да заустави цртање и погоди у чему је реч, она даје визуалну презентацију шта мисли да је на слици. Ту почиње део који издваја једну апликацију од друге. Сада корисник може да кликне на неки од понуђених цртежа и његова скица се претвара у лепши цртеж тог истог објекта на истом месту.



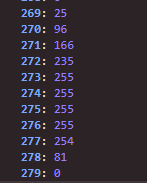


Слика 3.4 Пример цртежа у *autodraw*-у и понуђена решења

На слици 3.4 дат је пример где је на радну површину нацртана кућа коју је неуронска мрежа препознала и понудила сличне објекте. Кликом на било који од понуђених резултата нацртани објекат постаје тај други објекат. На овај начин могуће је и комбиновати више објеката у један цртеж односно даље вршити доцртавање, бојење итд.

# Имплементација решења

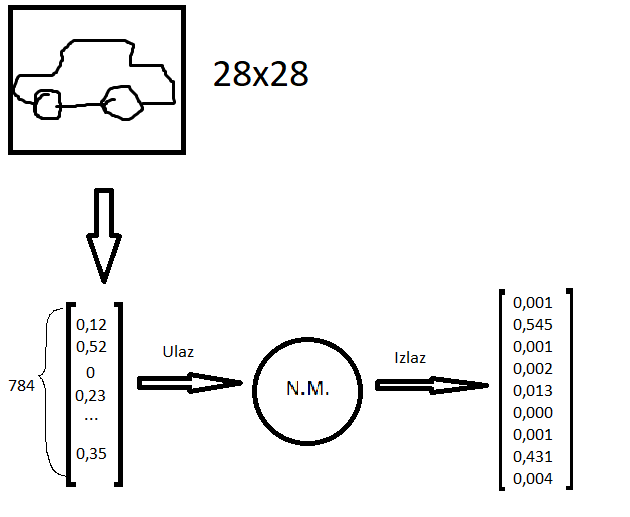
За пример где неуронска мрежа треба да погоди шта је корисник нацртао улазни податак је унешена слика. Процес учења се обавља над сетом података који се преузимају из гуглове базе која садржи преко 300 различитих категорија са по више стотина цртежа. База се непрестано допуњује новим подацима које корисници креирају испробавањем апликације. Сваки нови кориснички цртеж је нови податак у бази. Исти се уписује када апликација погоди шта је на слици па су то често недовршени радови. Ови подаци су јавно доступни за преузимање и коришћени су за учење унутар наше неуронске мреже. Свака од слика је димензија 28x28. То је укупно 784 пиксела за представљање жељеног објекта. Претварањем слике у вектор пиксела добијамо 784 вредности у опсегу 0 до 255. Ове вредности су осветљеност сваког од пиксела и њих треба нормализовати дељењем са 255.



Слика 4.1 Формат цртежа у низу податка за тестирање и тренирање

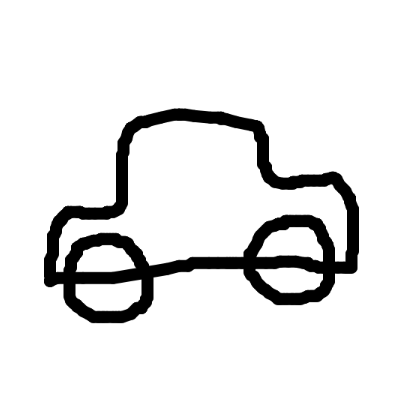
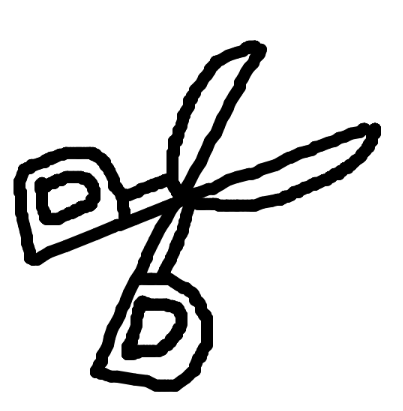
Одатле следи да је улаз у нашу неуронску мрежу вектор са 784 елемената. Излаз је вектор са онолико елемената колико различитих објеката апликација може да препозна. Сваки од елемената излазног вектора је број између 0 и 1, а њихов збир износи 1. Сваки од бројева у вектору одговара шанси да је на слици баш тај објекат.

Слика 4.1 приказује улазни вектор од 784 података који нису још нормализовани. На десној страни је пример дела овог вектора. Вредности ближе 255 су обојени пиксели слике.



Слика 4.2 Модел неуронске мреже са 784 улаза и 10 излаза на примеру цртежа аутомобила.

На слици 4.2 визуелно је скициран систем који за 784 нормализованих вредности, након проласка кроз неуронску мрежу даје излазни вектор. У излазном вектору су дате шансе за сваки од објеката да је на слици. Слика 4.3 приказује пример овог излазног вектора над приказаним цртежима. Зеленом бојом су означене највеће вредности које и одговарају објекту са слике.

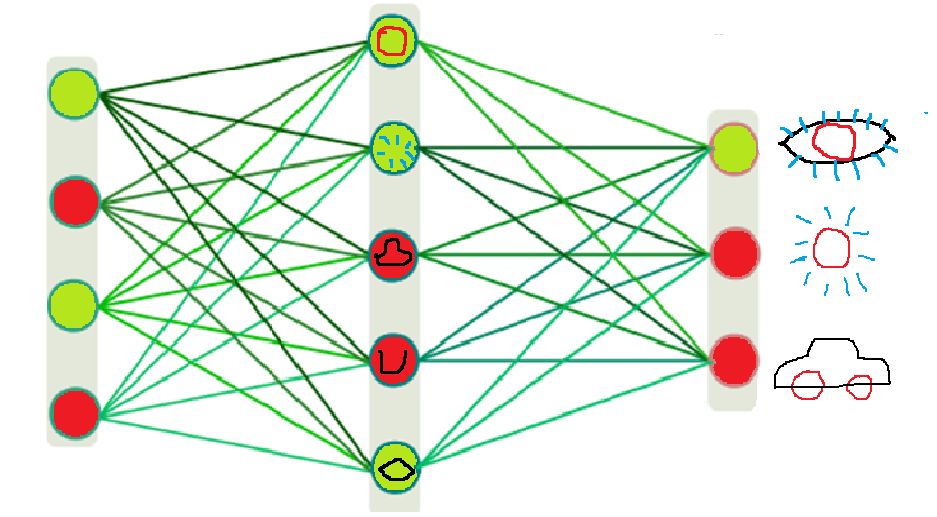
  

|  |  |  |  |
| --- | --- | --- | --- |
| Кактус | 0.00165754055 | 0.79717972010 | 0.34857662831 |
| Ауто | 0.54549012226 | 0.00039636462 | 0.00030386979 |
| Око | 0.00148066523 | 0.01496943274 | 0.02821774797 |
| Гитара | 0.00000125170 | 0.00037642018 | 0.03838793882 |
| Пешчани сат | 0.00000208972 | 0.00047258538 | 0.00038194838 |
| Нож | 0.01316615507 | 0.01801860322 | 0.02334431628 |
| Шољица | 0.00092147138 | 0.04052457433 | 0.02404068975 |
| Маказе | 0.00126367166 | 0.09205986848 | 0.51627735452 |
| Сунце | 0.43178950319 | 0.03351913961 | 0.01215440644 |
| Мач | 0.00422752920 | 0.00248329130 | 0.00831509969 |
|  |  |  |  |

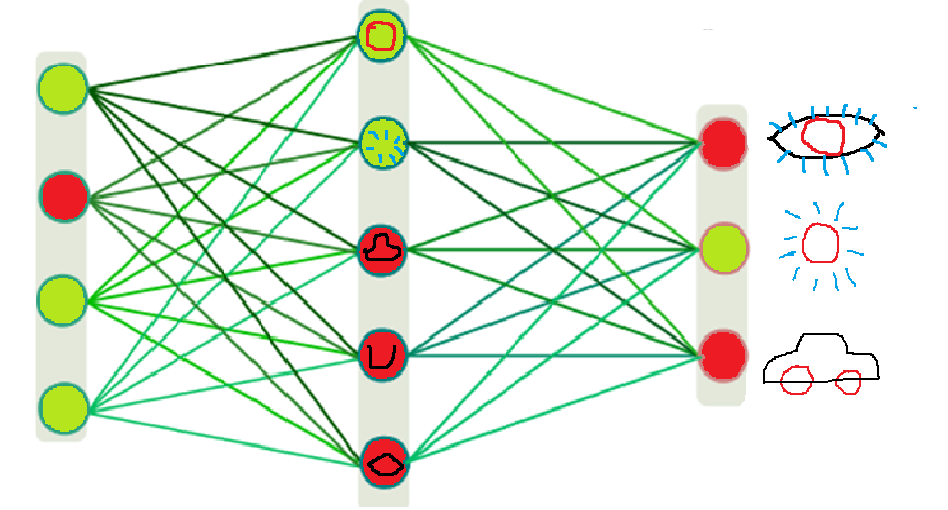
Слика 4.3 Пример излазних вектора за неколико објеката

Позадина иза свега овога креће од претпоставке да сваки од чворова скривеног слоја чува неки од шаблона при цртању објекта. Повећањем броја скривених слојева ти шаблони се даље деле на простије компоненте које цртеж може да садржи. У зависоности од улаза тј. пиксела од којих се састоји одређује се да ли је на слици неки од тих шаблона. У зависности од тога се активирају чворови скривеног слоја који садрже те шаблоне. Даље се врши активација чворова излазног слоја у зависности од тога који објекти садрже дате шаблоне. Онај који има највећу вредност је најтачнија претпоставка.

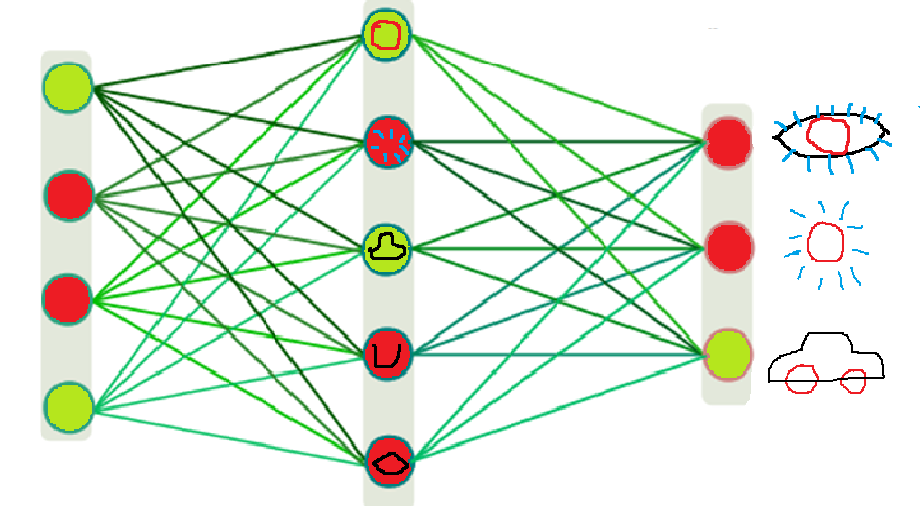
Слике 4.4, 4.5 и 4.6 представљају пример како неуронска мрежа “размишља” при одређивању шта је на слици. За различите улазне сигнале активирају се различити чвореви скривеног слоја који представљају делове коначног цртежа. Одговарајућа конфигурација средњег слоја активира одговарајући излазни чвор.



Слика 4.4 Пример функционисања неуронске мреже за цртеж ока



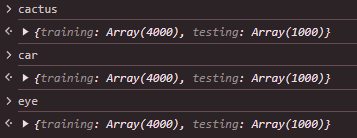
Слика 4.5 Пример функционисања неуронске мреже за цртеж сунца



Слика 4.6 Пример функционисања неуронске мреже за цртеж аута

## 4.1. Припрема података за тренирање и тестирање

Први корак је тренирање неуронске мреже за које ћемо да користимо 800 цртежа за сваку од категорија. За контролисану проверу тачности се користи 200 цртежа. Из оригиналне базе је издвојено 5000 података који су потом подељени у односу 4:1 јер се обично овај однос користи за тренирање и проверу рада (тестирање). За потребе тестирања додаје се још један атрибут који представља циљни вектор тј. Вектор који добијамо у идеалном случају ако систем ради. Он је исте величине као и излазни вектор. Испуњен је нулама сем на позицији која представља ентитет који је на слици која је улаз у неуронску мрежу. Разлика између очекиване вредности из овог вектора и вредности добијене на излазу из мреже представља грешку о којој је раније било речи. Да би учење било ефикасно не тренира се мрежа кетегорију по категорију већ се категорије мешају.



Слика 4.7 Подела података у односу 4:1 пре мешања

Слика 4.7 приказује податке за сваки од објеката одвојене у два низа за тренирање и тестирање у односу 4 : 1.

Имплементација неуронске мреже садржи конструктор са три аргумента који одговарају броју улазних, скривених и излазних чворова мреже. За потребе нашег проблема искоришћено је 784 улазних (укупан број пиксела слике коју корисник црта), а број излазаних чворова је 10. Број чворова скривеног нивоа произвољан, а ми ћемо користити мрежу са 64 чвора овог нивоа. Повећање броја чворова ће да повећа број израчунавања и тачност система а утицаће негативно на перформансе.

Учитавање података врши се на самом почетку програма. Функцијом *loadBytes(file:string)*. То су бинарне датотеке које су скраћене верзије оних из гуглове базе. Та база садржи на хиљаде цртежа за сваки од објеката и било би утрошено много времена на њихову обраду. Са повећањем броја података за обраду повећава се прецизност апликације али и време обраде. Ово решење врши обраду по пет хиљада података за сваку од категорија. Како су подаци у гугловој бази сачувани као *numpy* фајлови који су погодни за коришћење код програмирања у пајтону неопходно је издвојити податке у посебан бинарни фајл брисањем заглавља које ове датотеке имају. Коришћењем друге апликације издвојено је заглавље фајла и на њега надовезано 784 бајта за пет хиљада података.

int total = 10000;

String[] inputFiles = {"cactus.npy", "car.npy", "eye.npy", "guitar.npy", "hourglass.npy", "knife.npy", "mug.npy", "scissors.npy", "sun.npy", "sword.npy"};

String[] outputFiles = {"data1000/cactus1000.bin","data1000/car1000.bin","data1000/eye1000.bin", "data1000/guitar1000.bin", "data1000/hourglass1000.bin", "data1000/knife1000.bin", "data1000/mug1000.bin", "data1000/scissors1000.bin", "data1000/sun1000.bin", "data1000/sword1000.bin"};

int numObjects = inputFiles.length;

for(int fileI=0; fileI<numObjects; fileI++)

{

byte[] data = loadBytes(inputFiles[fileI]);

byte[] outdata = new byte[total\*784];

int outindex = 0;

for (int n = 0; n < total; n++) {

int start = 80 + n\*784; // 80 - header npy fajla

for (int i = 0; i < 784; i++)

{

int index = i + start;

byte val = data[index];

outdata[outindex] = val;

outindex++;

}

}

saveBytes(outputFiles[fileI], outdata);

}

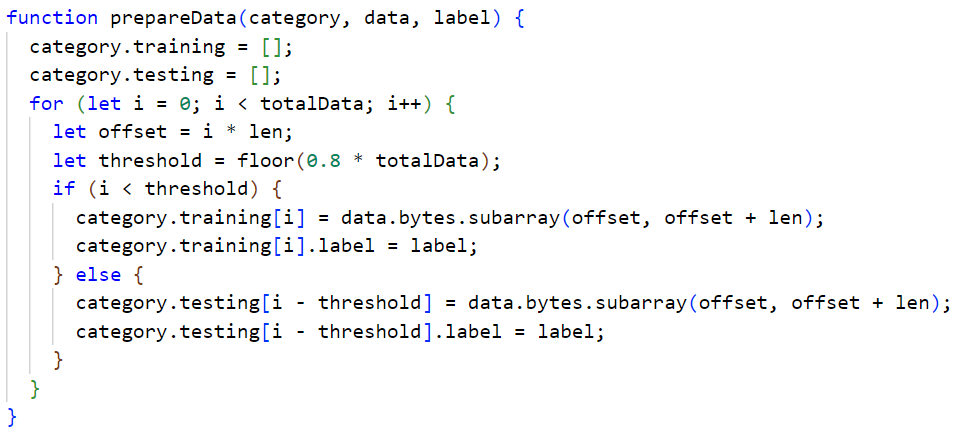
Слика 4.8 Јава апликација за издвајање 10000 података за сваки од ентитета који ће бити разматрани

Припрема података обухвата њихову поделу у односу 80:20 и сврставање у посебне променљиве. Подаци за тестирање и тренирање сваке од категорија се спајају у посебан низ за тренирање односно низ за тестирање. Ово није довољан услов за успешно тренирање. Непоходно је да се категорије промешају и тек онда над промешаним низом се изврши тренирање. Разлог за то је да би се спречило да неуронска мрежа учи распоред којим добија излазе уместо да се сконцентрише на то која комбинација елемената улазних вектора даје који резултат. Уколико након излаза који одговара првом објекту следи исти излаз и тако више стотине пута, уследиће претпоставка да се и у реалном примеру најчешће понавља резултат више стотина пута пре него што пређе на другу вредност.

Функцијом *prepareData(category, data, label)* врши се подела података у односу 4:1. Преноси се променљива *category* што је уједно и излаз из функције. То је објекат који се састоји из два низа и то један за чување тест података, а други за податке за тренирање. Коришћењем особине *javascript*-а можемо да сваком од података у низу доделимо и лабелу на основу које ћемо препознати о ком објекту је реч. У аргументу функције *data* смештени су сви подаци једне категорије цртежа у виду бајтова. Издвајају се редом по 784 бајтова и додељују делу објекта за тренирање, а затим за тестирање.

*totalData* – глобална константа са бројем цртежа по категорији.

*len –* број бајтова по цртежу.



Слика 4.9 Функција за поделу података за тестирање и тренирање

## 4.2. Кориснички интерфејс

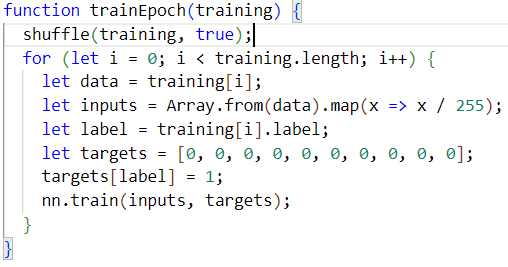
Функција за цртање *draw()*, садржи стандардан код с тим што би било идеално повести рачуна на дебљину линија које се исцртавају. Идеја је да она буде приближна оној која се користи на анализираним цртежима. У оквиру исте функције се позива функција *somethingIsDrawn()* чиме се обавештава неуронска мрежа да треба да проба да погоди шта је на слици. У овој конкретној имплементацији то је функција која се позива притиском на одговарајуће дугме и сваки пут кад корисник нешто нацрта да би претпоставка била доступна после сваке повучене линије.

Функција *somethingIsDrawn()* преузима слику из одговарајућег прозора за цртање и смањује јој величину на 784 пиксела (слика величине 28x28). Затим слику учитава у један низ који сада представља редом сваки од пиксела са слике као РГБ компоненту и бајт који представља осветљење пиксела. За даљу обраду у овом случају је довољно издвојити само светлину јер су цртежи црно-бели. Позива се функција за тестирање *predict(inputs)* над објектом неуронске мреже. То је функција која враћа вектор од десет елемената. Свака позиција овог вектора представља један од ентитета за који је неуронска мрежа тренирана да погоди. Вредности унутар сваке позиције представљају шансу да је на слици тај ентитет. Коришћењем помоћног низа *crtaj* ове шансе се чувају у нове објекте заједно са именом класе за коју је та позиција задужења. Скраћује се вредност на два места иза децималног зареза. Даљом обрадом омогућен је приказ процентуалне шансе за сваки од објеката да је баш он на слици у реалном времену. Рачунањем збира свких вредности и дељењем сваке од вредности понаособ са тим збиром добије се проценат за сваки од ентитета. То је нормализација која није обухваћена у оквиру функције у неуронској мрежи већ је овде накнадно обављена.



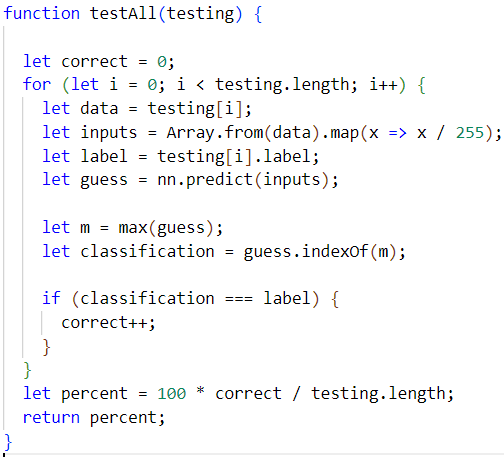
Слика 4.10 Функција која проверава шта је на слици сваки пут кад се изврши промена на табли за цртање

Дугме за тренирање обухвата један *epoch* (једну епоху) тј. jедан пролаз кроз процес тренирања. Ово дугме врши позив функције која пролази кроз низ података за тренирање и сваки од података дели са 255 да би се добила вредност између 0 и 1, тј. она врши непоходну нормализацију улазних података. Ту се и поставља циљани излаз кроз вектор који на свим позицијама има 0 осим код очекиваног излаза. Очекивани излаз се поставља на одговарајућу вредност за сваки од објеката и смешта се у променљиву *targets*. Даље се позива функција за тренирање над објектом неуронске мреже.



Слика 4.11 Функција за припрему података за тренирање

Функција за тестирање пролази кроз низ за тестирање и врши исте припреме као и функција за тренирање с тим да сада позива другу функцију над објектом неуронске мреже која треба да одреди и врати за који објекат је највећа шанса да се налази на слици. И овде се за сваки од цртежа памти и лабела у којој пише индекс очекиваног резултата. Ако се индекс поклапа са лабелом онда је добијен очекивани резултат. Додатно ова функција одређује тачност процене система изражену у процентима. То ради упоређивањем броја тачних резултата и укупне величине низа.

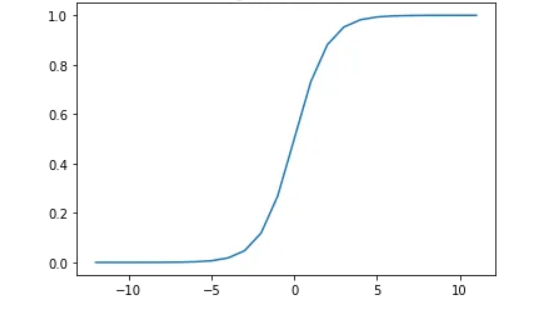


Слика 4.12 Припрема података за тестирање и сам процес тестирања

## 4.3 Имплементација неуронске мреже

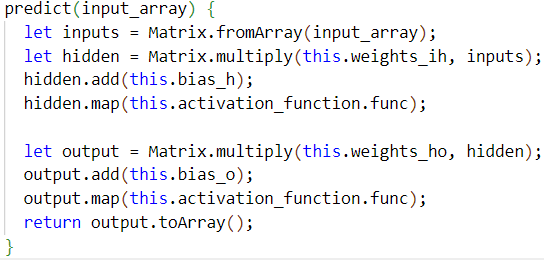
Конструктор за креирање неуронске мреже мора да добије број улазних, скривених и излазних чворова. Неуронска мрежа која је овде коришћена за решавање проблема садржи само један скривени слој. Прва провера је да ли је као улазни чвор прослеђена сама неуронска мрежа. У том случају се у новокреираној неуронској мрежи за сваки од атрибута копирају вредности које има та друга неуронска мрежа. У супротном се атрибути који одговарају броју улазних, скривених и излазних чворова постављају на прослеђене вредности. Сем тога чувају се и матрице са тежинама потега како између улазних и скривених чворова тако и између скривених и излазних. Матрице се попуњавају позивом функције из класе *Matrix* чија је улога да обиђе матрицу и постави елементе на случајне вредности. Разлог за постављање на произвољне вредности је јер није могуће погодити тежине потега и *bias-*а, већ ће оне да се конфигуришу у процесу учења. Додатно се чувају вектори са *bias* вредностима. Њихова величина је једнака броју скривених односно излазних чворова. И ови вектори се постављају на произвољне вредности.

Функција чија је улога одређивање излаза из скривеног и излазног слоја. респективно је функција *predict(input\_array)*. Преузима вектор улазних вредности и претвара га одговарајућом матричном функцијом у матрицу. Овај корак је неопходан да би се омогућило множење са другом матрицом. У овом случају то је множење са матрицом са тежинама потега између улазног и скривеног слоја чворова. Након множења следи додавање *bias*-a који је вектор исте величине као и број излазних чворова из скривеног слоја (због чега је и операција сабирања над њима могућа). Последњи корак је извршење Сигмоидове функције над сваким од резултата да би се излаз нормализовао на број између 0 и 1 о чему је раније било речи. Индентичан процес се извршава и над излазним слојем, а користе се одговарајући вектор за *bias* као и матрица тежина за потеге између скривеног и излазног слоја, а уместо улазног вектора се користи нови вектор излаза из скривеног слоја који је добијен претходно наведеним израчунавањем.



Слика 4.13 Сигмоидова крива

Имплементације Сигмоидове функције и њеног извода су имплементиране посебно у коду како би се спречило дуплирање кода.



Слика 4.14 Функција неуронске мреже за предвиђање резултата

Функција за постављање степена учења је такође имплементирана али као функција са фиксном прослеђеном вредношћу од 0.1. Повећање ове вредности може да проузрокује брже одређивање тежина потега и вредности *bias*-а, али превисока вредност овог атрибута класе може да изазове да се систем “заглави” тј. доведе у стање где не може да одреди вредности које се очекују већ да око њих кружи тј. да применом функције да већу вредност од очекиване, а да поновном применом врати мању од очекиване и тако у круг.

Функција за тренирање *train(input\_array, target\_array)* почиње исто као функција за тестирање. И њен циљ је одређивање излазног вектора. Међутим сада добијену вредност треба упоредити са очекиваном вредношћу. То је прослеђен вектор са свим нулама сем на позицији ентитета који је на слици.

Најпре се улаз који је вектор претвори у матрицу димензије 784x1. Као таква она може да се множи са другом матрицом димензије 64x784 у којој су вредности тежина између улазног и скривеног слоја. Кад се на дати резултат дода *bias* и изврши активациона функција над добијеном вредношћу израчунали смо вредност излаза из скривеног слоја.

Сада множењем тежина потега између скривеног и излазног слоја, додавањем *bias*-a на ову вредност и применом активационе функције над њом врши се рачунање излаза из неуронске мреже.

Претварањем циљаног излаза у матрицу димензије 10x1 омогућена је примена функције одузимања излаза од циљане вредности одаке се добија излазна грешка.

Одавде иде примена формуле

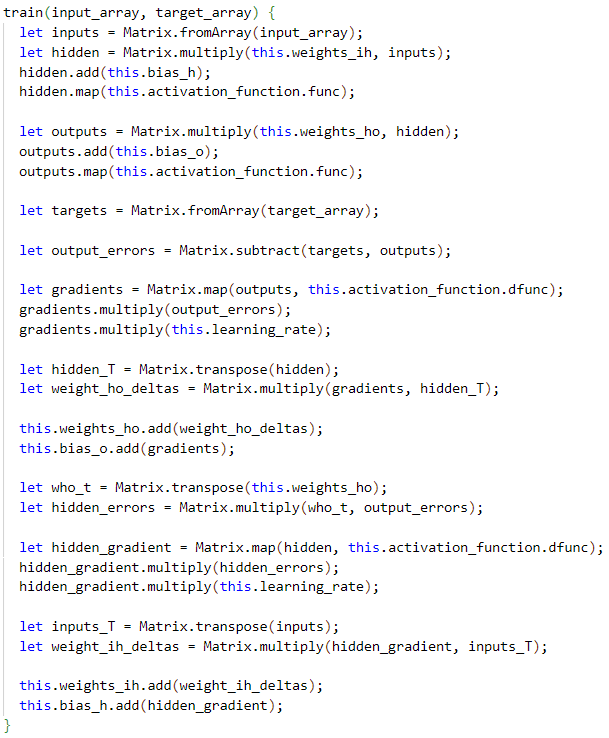
У помоћну променљиву *gradients* се чува производ

Врши се транспоновање излаза скривеног слоја што је задњи део функције за одређивање промене тежина потега.

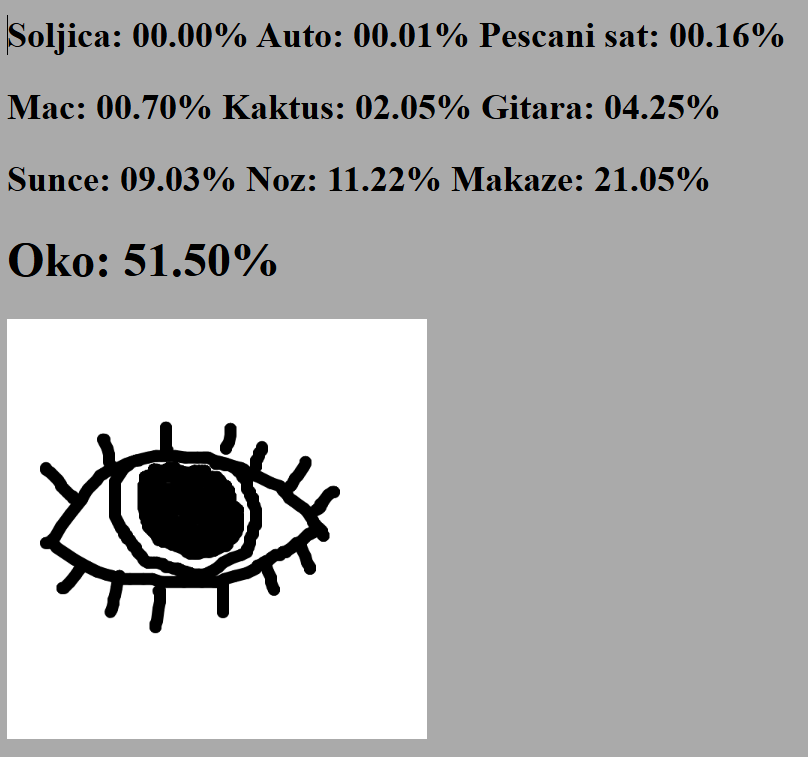
Нове тежине се добијају додавањем промене тежина на старе вредности тежина. Нова вредност за *bias* се добија додавањем градијента на стару вредност.

Следећа се рачуна грешка на излазу скривеног слоја. Њу добијамо из производа тежина између скривеног и излазног слоја и грешке на излазу.

Одавде се понављају кораци из излазног слоја чиме се добија градијент скривеног слоја и нове вредности тежина потега и *bias*-а.[7]



Слика 4.15 Функција за тренирање система

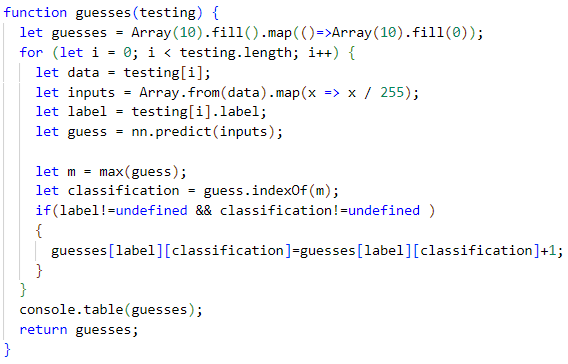


Слика 4.16 Пример рада апликације пре и после процеса учења

Слика 4.16 представља рад апликације пре и после тренирања. Пре тренирања су шансе произвољно распоређене за сваки од објеката. Због постављања тежина на произвољне вредности систем може да да неку претпоставку али не и нужно тачну. Након тренирања и постављања тежина потега систем са сигурношћу може да претпостави шта је на слици.

## 4.4 Приказ грешеке при учењу

Да би се обавила анализа података и дошло до закључка који ентитети изазивају највећу забуну код неуронске мреже додата је функција која осим што проверава колико пута је погођено о ком ентитету је реч тј. Колико пута је претпоставка одговарала тачном резултату, записује и колико пута је тај неки ентитет замењен са којим од осталих ентитета. Да би се ово постигло додата је функција која након извршења враћа матрицу. За приказ је коришћена функција *console.table()* која омогућава приказ матрице у облику табеле у конзоли. Функција је слична функцији за тестирање. Додатно се креира матрица величине 10x10 јер је то број ентитета са којима неуронска мрежа ради. Над сваким од елемената низа који је параметар који се преноси при позиву функције врши се исти део кода. Чита се елемент низа који је и сам низ пиксела цртежа који анализирамо. Он се даље преноси као параметар функције неуронске мреже која на основу тог нормализованог низа пиксела одређује који предмет корисник црта. И ако је у питању низ могуће је додати му додатне атрибуте и третирати га као објекат што је и учињено кад је раније кроз атрибут *label* пренесен редни број ентитета који је на слици. Врши се повећање вредности у матрици на позицији *guesses[label][classification]*. На овај начин на дијагонали матрице добијамо број тачних погађања. Редни број елемената дијагонале одговара редном броју ентитета.Редни број реда представља ентитет о коме је реч а свака од колона ентитет о којем неуронска мрежа мисли да се ради. (Слика 6.2)



Слика 6.1 Функција за цртање табеле са тачним и нетачним претпоставкама приликом тестирања

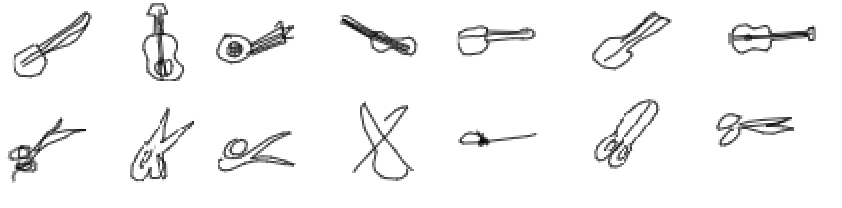
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Кактус | Ауто | Око | Гитара | Пешчаник | Нож | Шоља | Маказе | Сунце | Мач |
| Кактус | 11 | 744 | 8 | 3 | 0 | 13 | 192 | 6 | 0 | 23 |
| Ауто | 22 | 755 | 3 | 4 | 1 | 5 | 162 | 14 | 0 | 34 |
| Око | 13 | 675 | 6 | 8 | 0 | 9 | 267 | 12 | 0 | 10 |
| Гитара | 5 | 676 | 12 | 13 | 0 | 29 | 237 | 13 | 0 | 15 |
| Пешчаник | 22 | 559 | 14 | 2 | 0 | 90 | 295 | 11 | 0 | 7 |
| Нож | 16 | 747 | 4 | 10 | 1 | 28 | 171 | 12 | 0 | 11 |
| Шоља | 79 | 687 | 2 | 3 | 0 | 9 | 205 | 4 | 0 | 11 |
| Маказе | 23 | 569 | 7 | 14 | 0 | 40 | 328 | 17 | 0 | 2 |
| Сунце | 31 | 675 | 6 | 18 | 1 | 12 | 220 | 11 | 0 | 26 |
| Мач | 10 | 746 | 1 | 4 | 0 | 16 | 200 | 16 | 0 | 7 |

Слика 6.2 Однос тачних и нетачних претпоставки пре учења

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Кактус | Ауто | Око | Гитара | Пешчаник | Нож | Шоља | Маказе | Сунце | Мач |
| Кактус | 799 | 13 | 20 | 44 | 5 | 6 | 26 | 20 | 39 | 28 |
| Ауто | 11 | 886 | 38 | 19 | 1 | 3 | 10 | 11 | 19 | 2 |
| Око | 27 | 51 | 779 | 38 | 1 | 11 | 23 | 18 | 42 | 10 |
| Гитара | 10 | 8 | 14 | 890 | 3 | 13 | 3 | 14 | 1 | 44 |
| Пешчаник | 13 | 3 | 8 | 20 | 888 | 4 | 12 | 31 | 7 | 14 |
| Нож | 25 | 17 | 13 | 118 | 3 | 608 | 15 | 17 | 12 | 172 |
| Шоља | 43 | 19 | 27 | 23 | 10 | 1 | 845 | 9 | 19 | 4 |
| Маказе | 62 | 15 | 21 | 237 | 10 | 18 | 6 | 544 | 19 | 68 |
| Сунце | 64 | 14 | 64 | 13 | 0 | 6 | 26 | 14 | 780 | 19 |
| Мач | 15 | 10 | 6 | 113 | 4 | 103 | 7 | 20 | 4 | 718 |

Слика 6.3 Однос тачних и нетачних претпоставки после учења

Из табеле добијене за тестирање које је обављено након једног *epoch*-а учења (Слика 6.3) се види да је највећи број грешака између цртежа мача и ножа. Што је очекивано јер и људско око може да направи исту грешку. Нож је помешан и са гитаром. Неочекивани резултат је број случајева када су маказе замењене гитаром што се десило у 237 случаја што је скоро четвртина свих тестираних цртежа. Овакав резултат доводи до тога да је од хиљаду тестираних слика маказа само 544 погођено. Слика 6.4 приказује неке од примера цртежа гитара и маказа који доста личе једни на друге. Ово је мали проценат у систему који за учење користи десетине или стотине хиљада података, а за систем са мањим бројем података могу да изазову већи број грешака.



Слика 6.4 Примери цртежа гитаре и маказа који нису репрезентативни

Приликом тренирања може да дође до проблема превеликог или премалог опремања неуронске мреже (*overfitting* и *underfitting*).

Премало опремљени систем је последица кратког времена тренирање односно рада са малим бројем података. Као последица тога систем не може да установи потебне шаблоне за погађање шта је на цртежу. Може и да се јави ако тражимо линеарну зависност онде где је зависност описана кривом, па не постоји могуће решење проблема. Променом модела, повећањем броја података и времена извршења може да се реши проблем недовољне опремљености система.

Превише опремљен систем се јавља када је модел превише прецизан. Модел је научио шаблоне који су коришћени у подацима за учење а којих можда неће бити у примерима цртежа које корисник црта. Може да се каже да је систем превише истрениран да погађа цртеже са којима је тренирао, а недовољно за нове цртеже. Најбољи начин да се препозна овај проблем је да се изврши тестирање над тест подацима које систем до сада није видео и над делом података на којима је систем трениран.

Извршено је тестирање над подацима који су коришћени за тренирање тако што су ови подаци промешани и издвојено је првих хиљаду података. Након тестирања над познатим подацима извршено је и тестирање над непознатим. [8] Добијени резултати су приказани на слици 6.5, како су резултати у оба случаја слични значи да је тестирање успешно спроведено.



Слика 6.5 Прецизност пре(лево) и после (десно) тестирања над тест подацима (горе) и делом података за тренирање (доле)

# Тестирање утицаја различитих параметара неуронске мреже на њене перформансе

Након имплементације неуронске мреже потребно је низом тестирања одредити неке њених параметара као што су број података, стопа учења, броја скривених слојева и броја чворова сваког од слојева. Ове вредности зависе од случаја коришћења и одређују се експериментално.

## 5.1 Утицај понављања броја учења на прецизност и перформансе система

Процес учења је временски скуп процес чак и на напреднијим уређајима. Понављањем процеса учења за десет ентитета са по десет хиљада података дошло се до закључка да први циклус учења има највећи утицај на прецизност система. Са сваким следећим понављањем процеса учења резултати су били занемарљиво бољи, а утрошено време је било приближно једнако првом покретању процеса учења. Пре учења тачност система је 7-13% што је очекиван резултат за систем од десет ентитета где по статистици сваки од њих има шансу око 10% да буде на тестираној слици. Прво тренирње је вратило резултат од приближно 80% што је задовољавајуће за овакав систем. После другог тренирања резултат је незнатно бољи. Може да се деси да буде и гори. Понављање процеса учења више пута и ако над истим подацима може да изазове боље или горе резултате јер на овај начин пружамо више шанси систему да дође до глобалног минимума о коме је било речи раније. Резултат тестирања је графички приказан на слици 5.1.

Слика 5.1 Зависност тачности неуронске мреже од броја понављања процеса учења

## 5.2 Утицај степена учења на прецизност и перформансе система

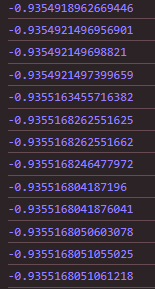
Идеални степен учења се одређује експериментално и зависи од бројних фактора као што су величина и сложеност система и количина података. Превисока или прениска вредност степена учења може да нашкоди раду система. Он представља степен којим ће тежине потега да се мењају при учењу шаблона за препознавање објеката. Циљ учења је да неуронска мрежа пронађе тачку где је разлика између очекиване и добијене вредности минимална. Та тачка је познатија као глобални минимум функције губитка.

Прениска вредност степена учења може да изазове да неуронска мрежа пронађе ту тачку али у већем броју понављања него код неке идеалне вредности. У пракси ово значи да систем неће моћи да уочи образце за препознавање ентитета и неће да даје прецизне резултате.

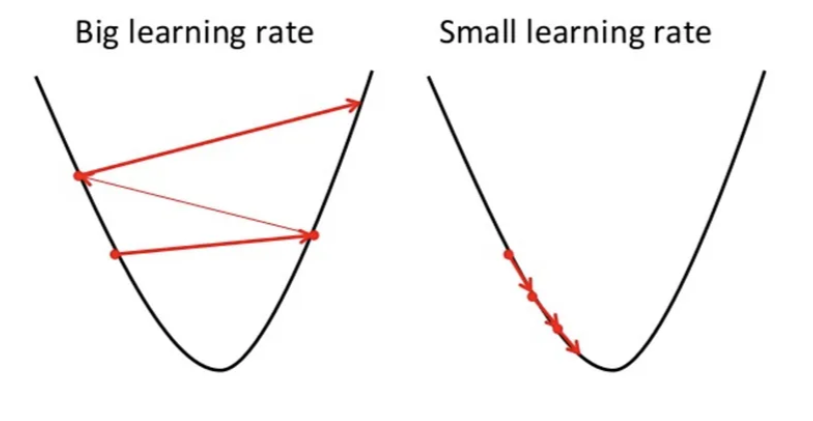
Превисока вредност степена учења може да изазове да неуронска мрежа потпуно промаши тај глобални минимум и да у потпуности изађе из његовог окружења и тиме изазове неочекивано понашање апликације. [9]

Идеално се поставља на вредност 0.1 па се одатле мења и упоређују се резултати.

Степен учења не утиче на време извршења.



Слика 5.2 Промена тежине потега између првог скривеног чвора и првог излазног чвора током неколико итерација



Слика 5.3 Достизање минимума за пример са већом и мањом стопом учења [9]

Слика 5.3 Приказује како превелика стопа учења може да обиђе минимум и никад га не достигне. У ситуацијијама када је стопа учења прениска јасно је да ће вредност тежина потега да се приближи минимуму ал остаје питање да ли ће бити довољно итерација да га достигне.

Слика 5.4 графички представља како се прецизност система мења у зависности од стопе учења која се користи. При стопи учења од 0.5 у једном тренутку прецизност опада што одговара обилажењу минимума. За вредност од 0.01 прецизност се континуално повећава али не у довољној мери као на пример за 0.1 где смо и добили најбољу прецизност после сваког круга учења.

Слика 5.4 Зависност прецизности неуронске мреже од степена учења

## Прецизност у зависности од количине података

Премала количина података може да доведе до тога да неуронски систем нема довољно информација за учење образаца приликом цртања слика. Током првог учења прецизност расте са порастом броја података. Разлика између пет и десет хиљада података је занемарљиво мала, док је време извршења директно зависно од количине података што у овом случају изазива да се време извршења дуплира. У нашем систему се врши обрада десет категорија са по пет хиљада података.

Слика 5.5 Прецизност неуронске мреже у зависности од количине података

Слика 5.5 приказује како количина података утиче на прецизност ситема кроз три циклуса учења. За 100 података систем је недовољно прецизан. За више од 500 података прецизност прелази седамдесет посто што није лош резултат. Најбољи случај се добија за 5000 и 10000 података где је за 10000 података разлика минимална.

Слика 5.6 Зависност времена тренирања од количине података

Слика 5.6 показује однос времена извршења и количине података. Добија се резултат где се повећањем количине података сразмерно повећава и време извршења.

## 5.4 Прецизност у зависности од броја чворова једног скривеног слоја

Прецизност расте са повећањем броја чворова. У једном тренутку напредак постаје занемарљив, а разлика у времену извршења значајно расте. Код нашег система се користи 64 чвора. Са слике 5.7 се јасно види да коришћење мање од десет чвора скривеног слоја не даје довољно добар резултат.

Слика 5.7 Прецизност у односу на број чворова скривеног слоја

Слика 5.8 представља време тестирања у зависности од броја чворова. Већи број чворова даје дуже време извршења. Тестирањем и упоређивањем ове вредности дошло се до закључка да је најоптималније решење коришћење 32 или више чвора. За препознавање слика користићемо 64 чвора. Прецизност се повећа за три посто а време извршења за педесетак секунди. Да ли је вредно жртвовати време зарад ове прецизности зависи од одлуке корисника.

Слика 5.8 Време учења у зависности од броја чворова једног скривеног слоја

## 5.5 Прецизност у зависности од броја скривених слојева

Додатна оптимизација решења може да се оствари повећањем броја скривених слојева. У појединим ситуацијама на овај начин може само непотребно да се успори рад система чак неколико пута а да резултати не буду повољни. Уз то сада треба водити рачуна о броју чворова у сваком од скривених слојева. Једна од опција која се користи као неписано правило је да број чворова буде мањи од броја улазних а већи од броја излазних чворова. Том логиком је обављено и следеће тестирање. Идеално треба одредити број чворова по формули 2/3 броја излазних чворова и на то додати број излазних чворова. Ова опција је одбачена јер иако је дала добро решење код једног скривеног слоја временски је била скупа, а побољшање прецизности недовољно. Да би проверили како ће коришћење више скривених слојева да утиче на рад неуронске мреже вршимо поређење са резултатима која је дала мрежа са једним скривеним слојем од 64 чвора. Први циклус учења на овој мрежи је дао прецизност нешто вишу од 80, а трајао је нешто мање од три минута. Прво тестирање је обављено над истим бројем скривених чворова који су сада распоређени у два скривена слоја од по 32 чвора. Време извршења је скраћено за педесетак секунди а прецизност система је опала на 67%. За потребе следећег тестирања број чворова је дуплиран у сваком од скривених слојева и исход је приближно исти а време извршења дуплирано. Најуспешнији је био тест двослојне архитектуре са 64 и 128 чвора који је достигао прецизност од 69% за 150 секунди. Даљим повећањем броја скривених слојева на три извршено је тестирање где су у сва три скривена слоја распоређена по 32 односно 64 чвора. У првом случају враћена је прецизност од 33% након 74 секунде што је много лошији резултат што временски тако и по прецизности од архитектуре са једним скривеним слојем. Резултат другог тестирања је довео до прецизности од 41% након првог круга учења, док је на другом чак смањио прецизност на 37%. Оба циклуса учења су трајали преко два и по минута и дошли смо до закључка да је најоптималнији случај био коришћење једнослојне архитектуре.

Промена прецизности у зависности од броја и распореда чворова дата је на слици 5.9

Слика 5.9 Прецизност у односу на различити број чворова и слојева

Слика 5.10 представља време извршења за сваку од тестираних комбинација броја чворова и слојева.

Слика 5.10 Време извршења за различити број чворова и нивоа у секундама

# Закључак

Неуронске мреже последњих година добијају све више на значају. Разлог за то је што је њихова примена у свакодневном животу све заступљенија. Њиховим коришћењем омогућено је да рачунар може да извршава све више функција за које је пре био задужен човек. Начин на који она то чини коришћењем великог броја простих израчунавања је описан у овом тексту. Приказани су проблеми који се јављају при њеном коришћењу и како их избећи, како се неуронска мрежа понаша за случај линеарне и нелинеарне зависности.

У раду је приказан начин на који слику претворити у податак разумљив неуронској мрежи, као и како рачунар уочава утицај делова те слике на коначни резултат. Описана су друга решења која су послужила као инспирација за овај рад и како она решавају проблеме са којима се неуронска мрежа суочава.

Описана је имплементација решења и сви неопходни делови да би он функционисао. Приказано је како систем “размишља” у потрази за резултатом и приказан је тај процес помоћу вишедимензионе равни. Наведени су фактори који утичу на време извршења и тачност система и како одабрати праве вредности. Научили смо да оно што важи за један случај коришћења не мора нужно да важи и за неке друге.

# LITERATURA

1. " How to Create a Neural Network (and Train it to Identify Doodles)" YouTube, uploaded by Sebastian Lague, 12. 8. 2022., <https://www.youtube.com/watch?v=hfMk-kjRv4c> . [Accessed: 31-Okt-2024].
2. T Rashid, Make Your Own Neural Network.
3. " What is backpropagation really doing? | Chapter 3, Deep learning" YouTube, uploaded by 3Blue1Brown, 3. 11. 2017., <https://www.youtube.com/watch?v=Ilg3gGewQ5U&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=3> .[Accessed: 31-Okt-2024].
4. quickdraw\_dataset, googlecloud <https://console.cloud.google.com/storage/browser/quickdraw_dataset/full/numpy_bitmap;tab=objects?pli=1&prefix=&forceOnObjectsSortingFiltering=false>
5. "The Quick, Draw! Dataset" GitHub uploaded by mrayinteractive, <https://github.com/googlecreativelab/quickdraw-dataset> [Accessed: 31-Okt-2024].
6. "Douglas-Peucker algorithm" Cartography playground, 28.7.2024., <https://cartography-playground.gitlab.io/playgrounds/douglas-peucker-algorithm/> [Accessed: 07-Nov-2024].
7. "10: Neural Networks - The Nature of Code" YouTube (playlist), uploaded by The Coding Train, <https://www.youtube.com/playlist?list=PLRqwX-V7Uu6aCibgK1PTWWu9by6XFdCfh> . [Accessed: 31-Okt-2024].
8. "Underfitting & Overfitting - Explained" YouTube, uploaded by NStatum, 12. 8. 2022., <https://www.youtube.com/watch?v=o3DztvnfAJg> . [Accessed: 31-Okt-2024].
9. "The Learning Rate: A Hyperparameter That Matters" Medium.com, written by Mohit Mishra, 28.5.2023., <https://mohitmishra786687.medium.com/the-learning-rate-a-hyperparameter-that-matters-b2f3b68324ab>. [Accessed: 31-Okt-2024]