МАТЕМАТИЧКА ГИМНАЗИЈА

МАТЕМАТИЧКИ МОДЕЛИ У МАШИНСКОМ УЧЕЊУ

- матурски рад из математике -

Аутор :
Теодора Божићевић

Ментори :
Бојана Матић
Јелена Хаџи-Пурић

Садржај

1 Осврт на историју изучавања вештачке интелигенције

1.1 Дефиниција вештачке интелигенције

Познавање историјског развоја вештачке интелигенције је важно како би се разумело где се она тренутно налази и какве могућности има у будућности. Прво, потребно је дефинисати шта је вештачка интелигенција.

Првобитна дефиниција, коју је 1955. године поставио Џон Макарти, гласи : „Наука и инжењерство прављења интелигентних машина.“. Данас би се то рекло мало другачије.

Вештачка интелигенција (енг. *Artificial Intelligence* - AI) је широка грана рачунарских наука која се бави стварањем машина које могу да уче, доносе одлуке и обављају задатке на људском нивоу. Напредне АИ машине могу да се унапређују и уче саме, независно од човека. Чак и обична АИ машина може сама да се носи са сложеним задацима за које би иначе био потребан човек, али може захтевати помоћ програмера како би учила из својих грешака.

1.2 Начин рада и типови вештачке интелигенције

На најосновнијем нивоу, вештачка интелигенција ради тако што узима податке и користи итеративни систем обраде и различите алгоритме за учење из образаца пронађених у подацима, а затим даје одређену реакцију. Напредна АИ може и да мери сопствени учинак и побољшава га. АИ системи користе модел склоности да би правили предвиђања на основу података које обрађују, а затим користе та предвиђања да би одговорили на различите радње.

Различити типови вештачке интелигенције покрећу различите АИ алгоритме, помоћу којих реагују и уче на другачије начине. Постоје четири главна типа вештачке интелигенције, сваки одређен тиме колику количину података може да складишти и како те податке користи, а то су :

- реактивна (енг. *reactive*),
- са ограниченим памћењем (енг. *limited memory*),
- теорија ума (енг. *theory of mind*),
- самосвесна (енг. *self-awareness*).

Мора се напоменути да од ова четири установљена типа, последња два у пракси и даље не постоје, али научници и инжењери раде на томе да се појаве у будућности.

Реактивна вештачка интелигенција је најосновнији ниво АИ-а, који функционише као „реактивни“ систем. Ове машине не могу да чувају податке у својој меморији, већ могу да реагују само на податке који се налазе пред њима. Оне не могу да уче, нити да се унапређују и увек на исти улаз одговарају истим излазом. Примери овог типа АИ-а су системи за играње видео игара, системи за препоруке и спам филтери на веб-сајтовима.

Вештачка интелигенција са ограниченим памћењем је тип АИ-а који поседује могућност привременог чувања улазних података и на основу њих

одлучује шта ради следеће. Овакве машине раде по принципу узимања улазних података, прављења предвиђања излазних података и у складу са њима, реаговања на одређени начин. Примери овог типа АИ-а су аутономна возила и роботи.

Теорија ума је, за сада теоретски, ниво вештачке интелигенције замишљен тако да машине морају бити испрограмиране и истриниране да разумеју како мисли и осећања утичу на људе. Оне треба да буду способне да узимају у обзир необјективне податке и тако имају бољу комуникацију са човеком и извршавају комплексније задатке.

Самосвесна вештачка интелигенција је такође теоретски ниво АИ-а, где машина постаје свесна себе и свог места у друштву. Она поседује људски ниво свести и способна је да размишља и сама доноси одлуке.

1.3 Историја вештачке интелигенције

Идеја вештачке интелигенције сеже још у античко доба, када су филозофи размишљали о животу, смрти и могућности да машине делују самостално. У том периоду настали су први „аутомати“, механички уређаји који су могли да се крећу без људске интервенције, као што је механичка голубица Платоновог пријатеља. Леонардо да Винчи крајем 15. века конструисао је један од најпознатијих аутомата, показујући дугу историју људске маште у стварању „живих“ машина.

У 20. веку концепт АИ-а постаје конкретнији. Чапек је 1921. године у драми „Универзални роботи Росума“ први пут употребио реч „робот“, док је 1929. јапански професор Нишимура направио првог јапанског робота Gakutensoku. Књига „Гигантски мозгови или машине које мисле“ из 1949. године поредила је рачунаре са људским мозгом и поставила темеље будућим истраживањима вештачке интелигенције.

Петдесете године 20. века обележавају рођење модерне вештачке интелигенције. Алан Тјуринг је предложио Тјурингов тест као меру интелигенције машине, постављајући основу за размишљање о томе да ли машина може да „мисли“. Истовремено, развијани су рани програми који уче сами, попут Артура Самуела и његовог програма за игру даме. Сам термин „вештачка интелигенција“ уведен је 1955. године на радионици у Дартмут-у, чиме је ова област формално дефинисана и постављена на научну и истраживачку карту.

Од касних 1950-их до краја 1970-их АИ је брзо напредовао, али је пролазио и кроз изазове. Створени су програмски језици попут ЛИСП-а, експертски системи, први чатбот ELIZA и индустријски роботи као Unimate. Артур Самуел уводи термин „машинско учење“. Ивахненкова „Метода групног обрађивања података“ поставила је темеље дубоком учењу. Крајем 1970-их аутономни возачки уређај Stanford Cart демонстрирао је практичну примену АИ-а, док је оснивање AAAI 1979. године означило институционализацију ове научне области.

1.4 Појмови вештачке интелигенције, машинског учења и предиктивне анализе

Када се говори о вештачкој интелигенцији, користи се велики број термина и скраћеница који се неретко међусобно мешају. Зато ће укратко бити објашњени

најважнији појмови који ће касније бити коришћени.

Вештачка интелигенција (AI): као што је већ речено, АИ је веома широка област рачунарских наука која има за циљ да створи машине које могу имитирати људски мозак.

Машинско учење (енг. *Machine Learning* - ML) је подобласт вештачке интелигенције која се бави развојем алгоритама и модела за учење користећи статистику и анализу података, односно тренира машине за одређени задатак. Системи МЛ-а често могу да уче из сопственог искуства или историјских података, без експлицитног програмирања.

Предиктивна анализа (енг. *Predictive analytics*) је још ужи појам од машинског учења, она је алат у оквиру анализе података који користи АИ мотор заједно са историјским подацима да би предвидео будуће резултате и трендове.

Из наведеног се може закључити да савремена вештачка интелигенција почива на математички дефинисаним моделима и алгоритмима. Машинско учење представља једну од најзначајнијих области у том оквиру, јер се заснива на примени математичких принципа у обради података. У наредном поглављу биће размотрени математички модели који стоје у основи система МЛ-а.

2 Математички модел машинског учења

2.1 Машинско учење - појам и основне идеје

Развој вештачке интелигенције довео је до потребе за системима који не само да извршавају унапред дефинисане инструкције, већ су способни да уче из података и унапређују своје перформансе. У том контексту, развија се машинско учење, дисциплина која се бави извођењем алгоритама на основу података, без изричитог програмирања. Оно је погодно за проблеме које човек не може лако да дефинише, иако неке од њих врло лако решава, и у којима је прихватљива повремена грешка.

Један пример таквог проблема је препознавање лица на фотографијама. Људи су врло добри у решавању овог проблема, чак је и необично назвати га проблемом. Ипак, његово дефинисање уопште није једноставно. Први покушај би био неки опис, попут тога да је лице нешто што се састоји од носа, очију, уста, чела, јагодица и обрва. Ово води даљем питању дефинисања тих појмова, дефинисања њихових релативних позиција, итд. Упркос труду, одустаје се од решавања проблема на овај начин. С друге стране, као и човек, методе машинског учења се врло лако носе са оваквим проблемом.

Још неки од проблема на које је машинско учење успешно примењено су препознавање различитих објеката на фотографијама и видеу, препознавање тумора на медицинским снимцима, аутономна вожња аутомобила, аутономно летење, играње игара на табли попут шаха, предвиђање развоја болести код пацијената, анализа друштвених мрежа, препознавање говора итд. У многим од ових примена, машинско учење је већ превазишло ниво ефикасности људских експерата.

Све набројане примене представљају примере важних практичних проблема, али иза свих стоји и озбиљна теорија. Основни принцип машинског учења заснива се на томе да се на основу доступних података формира модел који може да предвиди одговарајући излаз за дати улаз. Уместо да се решење проблема дефинише кроз скуп унапред задатих правила, модел се гради тако што „учи“ из примера и уочава обрасце у подацима. Током процеса учења, тај модел се постепено прилагођава како би што боље описао зависност између улазних и излазних података.

Крајњи циљ машинског учења јесте да модел, након обуке, буде способан да доноси исправне одлуке и за нове, до тада невиђене податке. Управо ова способност прилагођавања и генерализације чини машинско учење изузетно моћним алатом у решавању сложених проблема из различитих области.

2.2 Математички модел

Као што је већ наведено, у основи машинског учења налази се модел који се формира на основу података и служи за доношење одлука или предвиђање резултата. Да би се боље разумела његова улога, неопходно је формално дефинисати појам модела у оквиру машинског учења.

Модел у машинском учењу представља апстрактни опис проблема који омогућава да се однос између улазних и излазних података представи помоћу

математичких функција. Да би се такав модел могао конструисати, неопходно је располагати подацима на основу којих се врши учење. У реалним условима, ти подаци нису унапред познати у потпуности, већ се посматрају као коначан узорак из већег скупа могућих података. Због тога се процес машинског учења посматра у оквиру статистичког приступа, који омогућава да се на основу доступног скупа података донесу закључци о општијим правилима.

Ради једноставнијег разумевања процеса учења, разматра се поједностављена ситуација. Замислити сусрет са новом врстом воћа, чије карактеристике и укус нису унапред познати. Циљ је да се, на основу посматраних особина, предвиди да ли је плод добар или не. Прво је потребно издвојити те карактеристике, на пример, то могу бити боја и мекоћа плода. На располагању је скуп примера, где су за сваки посматрани плод познате његове особине и исход. Потребно је формирати правило које и за нове примере доноси одговарајући закључак, односно предвиђа исход.

Да би се овакав, или сличан, проблем прецизније анализирао, уводи се формални статистички оквир машинског учења.

Улаз алгоритма за учење : Алгоритам има приступ :

- **Домену** : Произвољан скуп $\mathcal{X}$, који представља све објекте које посматрамо. На пример, у горе наведеном проблему, ово би био скуп свих воћки. Често се елементи овог скупа представљају помоћу вектора карактеристика, као што су боја и мекоћа плода. Ови објекти се називају *инстанцама*, а скуп $\mathcal{X}$ *простор инстанци*.
- **Скупу ознака** : Скуп могућих ознака $\mathcal{Y}$, који у најједноставнијем случају има два елемента. У примеру са воћем нека $\mathcal{Y}$ буде {0,1}, где 1 представља добар, а 0 лош плод.
- **„Тренинг“ подацима** : Скуп података за учење $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ назива се *тренинг скуп* и представља подскуп скупа $\mathcal{X} \times \mathcal{Y}$. Ови подаци представљају једини извор информација који алгоритам има на располагању.

Излаз алгоритма за учење : Задатак алгоритма је да конструише функцију $f : \mathcal{X} \to \mathcal{Y}$. Ова функција се још назива и *хипотеза*, или *класификатор*, а скуп свих могућих функција које модел може да има назива се *простор хипотеза*. Она служи за предвиђање ознаке нових, до тада невиђених објеката. Резултат алгоритма над скупом података S означава се као $A(S)$.

Модел генерисања података : Над скупом $\mathcal{X}$ дефинисана је вероватносна расподела $\mathcal{D}$, према којој се генеришу инстанце : $x \sim \mathcal{D}$, за све $x \in \mathcal{X}$. Не сматра се да алгоритам зна нешто о овој расподели. Ознаке инстанци се одређују помоћу непознате функције f, за коју важи $y_i = f(x_i)$, за све i. Управо је циљ учења да се ова функција што боље апроксимира. На овај начин се генеришу (x_i, y_i) парови скупа S.

Мера успешности : *Грешка модела* дефинише се као вероватноћа да модел погрешно предвиди ознаку случајно изабраног објекта, односно да се предвиђена вредност разликује од стварне.

Формално, за дати подскуп домена, $A \subseteq \mathcal{X}$, вероватносна расподела $\mathcal{D}$ му придружује број $\mathcal{D}(A)$, који представља вероватноћу да случајно изабрана инстанца припада скупу A. У многим случајевима, скуп A посматра се као догађај

и изражава се помоћу функције $\pi : \mathcal{X} \to \{0, 1\}$, где је $A = \{x \in \mathcal{X} \mid \pi(x) = 1\}$. Тада се користи запис $P_{x\sim\mathcal{D}}[\pi(x)]$ којим се означава $\mathcal{D}(A)$.

На основу овога, грешка хипотезе $h : \mathcal{X} \to \mathcal{Y}$, која се мери у односу на расподелу $\mathcal{D}$ и тачну функцију означавања f, се дефинише на следећи начин :

$$L_{\mathcal{D},f}(h) \stackrel{\text{def}}{=} P_{x\sim\mathcal{D}}\big[h(x) \neq f(x)\big] \stackrel{\text{def}}{=} \mathcal{D}\big(\{x \mid h(x) \neq f(x)\}\big).$$

То значи да је грешка функције h вероватноћа да се насумично изабере пример за који важи $h(x) \neq f(x)$, односно да модел даје погрешно предвиђање. Ова величина се назива *генерализациона грешка* или *стварни ризик модела*, јер описује његову успешност на подацима који нису коришћени током учења.

Још једном се напомиње да алгоритам за учење нема директан приступ расподели $\mathcal{D}$, нити функцији f. У примеру са воћем, то значи да нема унапред знања о томе како су воћке распоређене нити како да се одреди њихов квалитет. Једини начин на који алгоритам може да учи јесте кроз посматрање тренинг скупа.

На основу овако дефинисаног модела, могу се разматрати различити типови и приступи машинском учењу, који ће бити описани у наставку.

2.3 Основни типови машинског учења

Проблеми на које се машинско учење примењује су веома разнородни, како по својој природи, тако и по методама које се користе за њихово решавање. Због тога се методе машинског учења могу класификовати на различите начине, али се најчешће деле према природи проблема учења.

Обично се издвајају три основна типа :

- надгледано учење (енг. *supervised learning*),
- ненадгледано учење (енг. *unsupervised learning*),
- учење поткрепљивањем (енг. *reinforcement learning*).

Надгледано учење представља најзаступљенији и најзначајнији вид машинског учења. Основна карактеристика овог приступа јесте да су за сваки пример у подацима познати и улаз и одговарајући излаз који модел треба да научи. Другим речима, подаци су представљени уређеним паровима улазних и излазних вредности, а задатак модела је да на основу примера уочи зависност међу њима и касније предвиди излаз за нове податке. Назив „надгледано“ потиче од аналогије са класичним процесом учења, у коме наставник ученику задаје задатке, а затим му даје и тачне одговоре. Тако, модел током обуке добија информацију о томе какав је исправан резултат, што му омогућава да постепено смањује грешку и побољшава своја предвиђања. Примери овог типа учења су препознавање лица на сликама, машинско превођење, класификација текстова и предвиђање цена на берзи.

Ненадгледано учење одликује одсуство информације о томе шта је потребно научити. За разлику од надгледаног учења, примери не садрже тачне излазне вредности које би моделу указивале на исправан резултат, већ само улазне. Задатак модела је да самостално уочи структуру, правилности или међусобне односе у подацима. Један од најпознатијих проблема ненадгледаног учења је *кластеровање*,

односно груписање сличних података. На пример, могуће је груписати сличне текстове и фотографије или финансијске податке чије се вредности понашају на сличан начин. Такође, ова метода се користи за смањење димензионалности података, то јест за њихово једноставније представљање уз задржавање најважнијих информација. Оваква предобрада често побољшава перформансе метода надгледаног учења које се касније примењују на тим подацима.

Учење поткрепљењем представља приступ у коме модел учи кроз интеракцију са окружењем. Постоји агент који посматра тренутно стање окружења и на основу њега предузима одређене акције. Након сваке акције, агент добија повратну информацију у виду награде или казне, а циљ је да временом научи стратегију која доводи до што веће укупне награде. Моделу није унапред познато која је акција исправна у одређеном тренутку, већ он постепено учи кроз искуство, испробавајући различите поступке и анализирајући њихове последице. Овај тип учења је посебно значајан код проблема у којима је потребно доносити низ узастопних одлука, при чему последице једне акције утичу на будућа стања система. Један од најпознатијих примера учења поткрепљивањем је аутономна вожња. Систем посматра околину, укључујући друга возила, пешаке и саобраћајне знакове, и помоћу добијених информација доноси одлуке о кретању и брзини. Успешне одлуке доносе награду, док опасне ситуације резултују казном. Овај приступ се користи и у роботици и игрању стратешких игара.

Методе машинског учења се могу поделити и према начину на који модел долази до података током процеса учења. *Пасивно учење* заснива се на подацима који су доступни моделу, а *активно учење* омогућава систему да самостално бира додатне податке који му могу помоћи у процесу учења. На пример, код филтера за нежељену електронску пошту, пасивно учење користи већ означене поруке како би научило да препозна нежељени садржај. Међутим, активно учење би могло самостално да тражи додатне примере порука ради бољег разумевања карактеристика нежељене поште.

Иако различити типови машинског учења користе различите приступе и податке, за све њих заједнички је проблем процене успешности модела. Пошто стварна грешка модела над свим могућим подацима најчешће није позната, у пракси се користе процене засноване на доступном скупу података.

2.4 Емпиријски ризик

Један од основних проблема машинског учења јесте одређивање квалитета модела. Након процеса учења, потребно је проценити колико успешно модел доноси одлуке и какве резултате даје при раду са новим подацима. Због тога се у машинском учењу уводе различите мере грешке и ризика, које оцењују успешност модела и омогућавају избор модела који даје најбоље резултате.

Као што је већ наведено, стварни ризик модела дефинише се у односу на вероватносну расподелу $\mathcal{D}$ и тачну функцију означавања f. Иако он представља основну меру успешности модела, у пракси га није могуће директно израчунати, јер $\mathcal{D}$ и f нису познате. Због тога се успешност модела процењује помоћу коначног скупа података, доступног током процеса учења.

На основу тренинг скупа S, дефинише се **емпиријски ризик** хипотезе $h : \mathcal{X} \to \mathcal{Y}$, који представља просечну грешку модела над примерима из скупа за

учење :

$$L_S(h) \stackrel{\text{def}}{=} \frac{1}{m}\sum_{i=1}^{m} \mathrm{I}\big[h(x_i) \neq y_i\big],$$

где је

$$\mathrm{I}\big[h(x_i) \neq y_i\big] = \begin{cases} 1, & \text{ако модел погрешно класификује пример } (x_i,\, y_i) \\ 0, & \text{иначе} \end{cases}.$$

Емпиријски ризик представља апроксимацију стварног ризика, јер се израчунава само на основу коначног скупа доступних података. Што је тренинг скуп већи и репрезентативнији, то емпиријски ризик боље описује стварну успешност модела.

На основу овога, задатак машинског учења постаје проналажење хипотезе која минимизује емпиријски ризик над скупом за учење. Овај приступ назива се **принцип минимизације емпиријског ризика** (енг. *Empirical Risk Minimization* - ERM). Формално записано :

$$h_S \in \arg\min_{h\in\mathcal{H}} L_S(h),$$

где $\mathcal{H}$ представља простор свих могућих хипотеза, а h_S означава хипотезу која има најмањи емпиријски ризик над тренинг скупом S.

Другим речима, бира се модел који прави најмањи број грешака над примерима из скупа за учење. Дакле, за сваку хипотезу израчунава се вредност емпиријског ризика над тренинг скупом, а затим се бира она која има најмању вредност. На тај начин алгоритам покушава да пронађе модел који најбоље описује доступне податке. У пракси, овај процес се спроводи постепеним прилагођавањем параметара модела током процеса учења.

Међутим, ERM не гарантује нужно добру успешност модела над новим подацима. Уколико је модел превише сложен, може доћи до **преприлагођавања модела** (енг. *overfitting*), при чему модел веома добро описује примере из скупа за учење, али не даје добре резултате над новим подацима. Систем „научи напамет“ тренинг скуп, уместо да научи опште законитости.

На пример, посматрају се тачке унутар круга полупречника R. Нека се унутрашњи концентрични круг полупречника r означи као класа са ознаком 1, а прстен између r и R као класа са ознаком 0, при чему је $\frac{R}{2} < r < R$. Тада је функција означавања :

$$f(x) = \begin{cases} 1, & \|x\| \leq r, \\ 0, & r < \|x\| \leq R \end{cases},$$

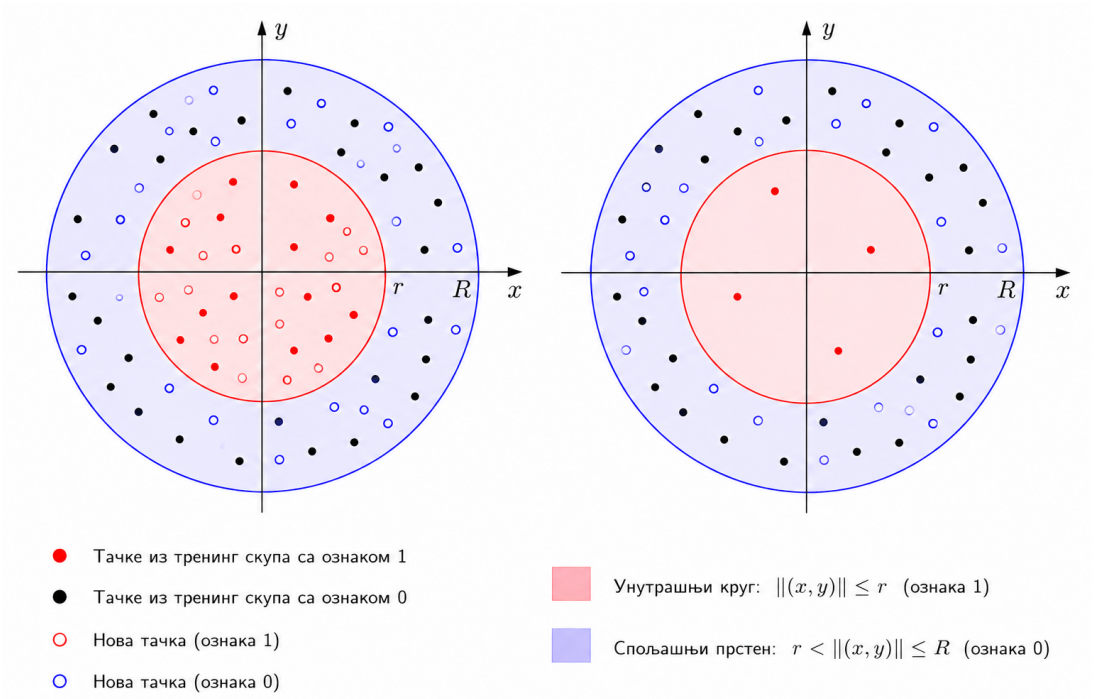
где је $\|x\|$ норма вектора x, то јест растојање тачке x од центра кругова. Претпоставља се да су тачке равномерно распоређене унутар круга, односно да имају униформну расподелу $\mathcal{D}$.

Разматра се хипотеза :

$$h_S(x) = \begin{cases} y_i, & \text{ако постоји } (x_i, y_i) \in S, \text{ такав да је } x = x_i \\ 0, & \text{иначе.} \end{cases}$$

Очигледно је да важи $L_S(h_S) = 0$ па ова хипотеза може бити изабрана ERM алгоритмом, јер ниједна хипотеза не може имати емпиријски ризик мањи од нуле. Са друге стране, за скоро све инстанце које се не поклапају ни са једном тачком из тренинг скупа, хипотеза предвиђа ознаку 0. Због тога модел греши над скоро свим тачкама унутрашњег круга, које нису из скупа за учење, јер функција означавања враћа 1 за такве тачке :

$$L_{\mathcal{D}}(h_S) = \frac{\pi r^2}{\pi R^2} = \left(\frac{r}{R}\right)^2 > \frac{1}{4}.$$



Слика 1 : Лево је приказан добар модел, а десно преприлагођени модел из примера

Дакле, добили смо хипотезу чија је успешност над тренинг скупом одлична, док је њена успешност над стварном расподелом података лоша. Управо је то преприлагођавање.

Након овога, природно се поставља питање под којим условима модел може успешно да ради и над новим подацима, а не само над примерима из тренинг скупа. Способност модела да на основу научених примера исправно доноси одлуке и за до тада невиђене инстанце назива се *генерализација*.

Да би минимизација емпиријског ризика довела до добре генерализације, није довољно само да модел има мали емпиријски ризик. Важну улогу има и сложеност простора хипотеза $\mathcal{H}$, односно број различитих модела које алгоритам може изабрати. Уколико је простор хипотеза превише богат, модел може постати сувише

прилагођен тренинг скупу и научити случајне особине података, уместо стварних правилности. Са друге стране, ако је простор хипотеза сувише ограничен, модел можда неће бити довољно изражајан да опише посматрани проблем.

Ограничавањем алгоритма да бира хипотезу само из скупа $\mathcal{H}$, уводи се одређена пристрасност ка посебној класи хипотеза. Оваква ограничења називају **индуктивна пристрасност** (енг. *inductive bias*).

Теорија машинског учења показује да се вероватноћа преприлагођавања смањује са повећањем броја тренинг примера, али расте са сложеношћу простора хипотеза. За коначне класе хипотеза добија се процена облика :

$$m \geq \frac{\log |H| + \log \frac{1}{\delta}}{\varepsilon},$$

где је m број тренинг примера, ε дозвољена грешка и δ вероватноћа неуспеха алгоритма. Ова формула показује да је за сложеније просторе хипотеза потребно више примера како би модел успешно доносио одлуке и за нове податке. Такође, количина потребних података зависи логаритамски од величине простора хипотеза, што значи да умерено повећање сложености модела не доводи нужно до великог повећања потребног броја примера.

Због тога је у машинском учењу неопходно пронаћи равнотежу, односно компромис између сложености модела и његове способности генерализације (енг. *complexity-generalization tradeoff*).

Ипак, у реалним применама машинског учења често се јављају проблеми чија сложеност превазилази могућности једноставних модела и ограничених простора хипотеза. Такви проблеми биће разматрани у наставку.

2.5 Дубоко учење

Дубоко учење (енг. *deep learning*) представља област машинског учења која се бави развојем модела способних да аутоматски издвајају и уче сложене структуре података. Овакви модели, током процеса учења, сами формирају хијерархију особина, од једноставних ка све сложенијим. Сваки слој обрађује информације добијене из претходног слоја и прослеђује резултат наредном. На пример, приликом препознавања лица на фотографији, први слојеви могу препознавати ивице и контуре, наредни делове лица попут очију и уста, док виши слојеви препознају целокупно лице.

Основу дубоког учења чине вештачке неуронске мреже, математички модели инспирисани организацијом неурона у људском мозгу. Оне се састоје од великог броја међусобно повезаних јединица које обрађују и преносе информације кроз мрежу. Неуронске мреже данас представљају основу већине савремених система вештачке интелигенције. У наредном поглављу биће размотрени основни принципи њиховог рада.

3 Неуронске мреже

3.1 Неурон

3.1.1 Природни неурон

Развој неуронских мрежа настао је као покушај да се математички опише начин на који људски мозак обрађује информације. Иако су савремене неуронске мреже поједностављене, њихова основна идеја инспирисана је организацијом и радом природних неурона.

Нервни систем човека састоји се од великог броја међусобно повезаних неурона. **Неурон** је основна јединица за обраду и пренос информација у нервном систему. Свака нервна ћелија се састоји од дендрита, тела ћелије и аксона. Дендрити примају сигнале од других неурона, тело ћелије обрађује примљене информације, док аксон преноси сигнал ка другим неуронима. Везе између неурона називају се синапсе.

Природни неурон истовремено прима велики број сигнала. Неки од њих делују подстицајно, а неки инхибиторно. Уколико укупни сигнал пређе одређени праг, долази до активације неурона.

На основу оваквог начина функционисања природних неурона развијени су поједностављени математички модели, вештачки неурони, који граде неуронске мреже.

3.1.2 Вештачки неурон

Када се каже поједностављени модели, мисли се на то да се не посматра прецизан тренутак појаве сваког појединачног импулса, већ се активност неурона описује његовом просечном учесталошћу активације у одређеном временском интервалу. На тај начин сложен процес преноса сигнала своди се на једноставнији математички приказ, што омогућава ефикасније моделовање и анализу великих неуронских мрежа.

Овако заснован модел вештачког неурона се математички описује на следећи начин. Посматра се неурон i, који прима сигнале од других неурона из скупа J, $j \neq i$. Ниво активности сваког неурона означава се ненегативним реалним бројем S_j, који представља учесталост активације у одређеном временском интервалу. Вредност $S_j = 0$ одговара стању мировања неурона.

Укупни улазни сигнал који неурон i прима од других неурона означава се са x_i и представља суму свих улазних сигнала помножених одговарајућим тежинама :

$$x_i = \sum_{j \in J} w_{ij} S_j.$$

Вредност w_{ij} означава јачину синаптичке везе између неурона j и i. Позитивне вредности имају подстицајно, а негативне инхибиторно дејство.

Активност неурона i добија се применом нелинеарне активационе функције h

на укупни улазни сигнал :

$$S_i = h(x_i).$$

Активациона функција описује начин на који неурон реагује на примљене сигнале и одређује његов коначни излаз. Њена нелинеарност има кључну улогу у способности неуронских мрежа да описују сложене зависности у подацима. У вештачким неуронским мрежама користе се различите активационе функције, а у наставку ће бити размотрене неке од најједноставнијих.

3.1.3 Сигмоидални модели неурона

Сигмоидалне функције су једна од најзначајнијих класа активационих функција, јер добро описују постепен прелаз неурона из неактивног у активно стање. Њихова вредност расте са повећањем улазног сигнала, али остаје ограничена, што представља поједностављен модел понашања биолошких неурона. Природно је претпоставити да важи :

$$\lim_{x\to-\infty} h(x) = 0$$

$$h'(x) \geq 0$$

$$\lim_{x\to+\infty} h(x) = 1$$

Један од најпознатијих примера овакве функције је :

$$h(x) = \frac{1}{2}\big(1 + \tanh[\gamma(x-\theta)]\big),$$

где је $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, параметар θ праг активације, а параметар γ одређује стрмину функције. У овом контексту, праг активације представља вредност улазног сигнала око које долази до најбрже промене активности неурона.

Поред приказа у коме активност неурона припада интервалу $[0, 1]$, често се користи и симетризован приказ активности, у коме вредности припадају интервалу $[-1, 1]$. Тада вредност -1 представља неактивно стање неурона, док вредност 1 одговара максималној активности. У том случају важи :

$$\lim_{x\to-\infty} h(x) = -1$$

$$h'(x) \geq 0$$

$$\lim_{x\to+\infty} h(x) = 1$$

Један од примера овакве активационе функције је :

$$g(x) = \tanh[\gamma(x-\theta)].$$

Овакав приказ није у потпуности биолошки реалистичан, јер уводи негативне вредности активности неурона. Ипак, он значајно поједностављује математичку и рачунарску анализу неуронских мрежа, због чега се често користи у теоријским моделима.

Један од таквих модела вештачког неурона је **Мекалок–Питсов неурон** (енг. *McCulloch–Pitts neuron*). Предложен је 1943. године. Његова популарност заснива

се на једноставности и сличности са бинарном логиком која се користи у рачунарству. Код овог модела активациона функција има два могућа излаза, то јест неурон може имати два стања, активно и неактивно :

$$g(x) = \begin{cases} 1, & x \geq \theta \\ -1, & x < \theta \end{cases}.$$

Мекалок–Питсов неурон представља основу каснијег развоја перцептрона и савремених неуронских мрежа.

Повезивањем већег броја вештачких неурона настају неуронске мреже, способне за обраду сложених информација и решавање различитих задатака машинског учења. Везе између неурона одређују структуру и понашање мрежа.

3.2 Архитектура неуронских мрежа

Начин на који су неурони међусобно повезани, као и смер преноса информација кроз мрежу, одређују њену архитектуру. У зависности од архитектуре, неуронске мреже могу имати различита својства, могућности и области примене.

Неуронске мреже се могу класификовати према различитим критеријумима, као што су број слојева, врста веза између неурона, начин обучавања или врста података које обрађују. У наставку, посебна пажња биће посвећена подели према смеру простирања информација, на рекурентне, односно повратне (енг. *feedback*) и неповратне (енг. *feedforward*) неуронске мреже.

3.2.1 Рекурентне неуронске мреже

Рекурентне мреже представљају моделе код којих постоје повратне везе између неурона, тако да излаз једног од њих може поново утицати на активност других у мрежи. Због тога се овакве мреже посматрају као динамички системи чије се стање мења током времена.

Активност неурона зависи од сигнала које прима од других неурона. Посматрањем дискретних временских корака t, стање мреже се постепено мења током времена. Активност неурона i у наредном тренутку може се описати рекурентном везом :

$$S_i(t+1) = g\left(\sum_{j \in J} w_{ij} S_j(t)\right), \quad t = 0, 1, 2, \ldots$$

где $S_j(t)$ означава активност неурона j у тренутку t, док је g активациона функција неурона.

Полазећи од почетног стања мреже $S(0)$, које садржи активности свих неурона у почетном тренутку,

$$S(0) = \begin{bmatrix} S_1(0) \\ S_2(0) \\ \vdots \\ S_N(0) \end{bmatrix},$$

динамика система током времена генерише низ узастопних стања $S(t)$, која представљају одговор мреже на почетни стимулус.

Један од најпознатијих и најједноставнијих примера рекурентне архитектуре јесте **Хопфилдова мрежа**. Она се састоји од N Мекалок–Питсових неурона који су међусобно повезани двосмерним синапсама. Везе између неурона су симетричне и они нису у вези сами са собом, то јест за тежине важи $w_{ij} = w_{ji}$ и $w_{ii} = 0$.

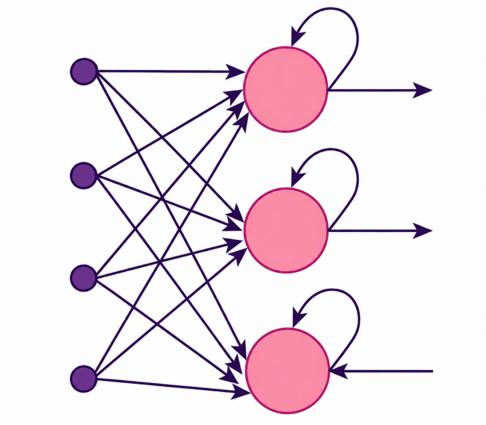
Активност неурона у овом моделу описује се на следећи начин :

$$S_i(t+1) = \operatorname{sgn}\left(\sum_{\substack{j=1 \\ j\neq i}}^{N} w_{ij} S_j(t)\right).$$

Функција $\operatorname{sgn}(x)$ одређује стање неурона на основу знака укупног улазног сигнала. Уколико је тај сигнал позитиван, неурон прелази у активно стање, док при негативној вредности остаје неактиван.

У Хопфилдовој мрежи чувају се одређени обрасци, односно карактеристични распореди активности неурона које мрежа треба да запамти. Сваки образац показује који су неурони активни, а који неактивни. На основу тих образаца мрежа подешава тежине веза између неурона, тако да касније може да препозна и обнови запамћену информацију чак и ако је почетни улаз непотпун или садржи грешке. Због тога оваква мрежа функционише као асоцијативна меморија, јер је способна да на основу дела информације реконструише цео запамћени образац.

Хопфилдове мреже имају примену у задацима обраде слика, препознавања образаца, говора и рукописа, као и у различитим проблемима који укључују временске и секвенцијалне податке.



Слика 2: Шематски приказ рекурентне неуронске мреже

3.2.2 Неповратне неуронске мреже

Неповратне неуронске мреже представљају архитектуре код којих су неурони организовани у слојеве. Обрада информација одвија се постепено, проласком сигнала кроз мрежу, при чему се информације простиру само у једном смеру, од улазног ка излазном слоју, без постојања повратних веза.

Основну структуру оваквих мрежа чине улазни, скривени и излазни слој.

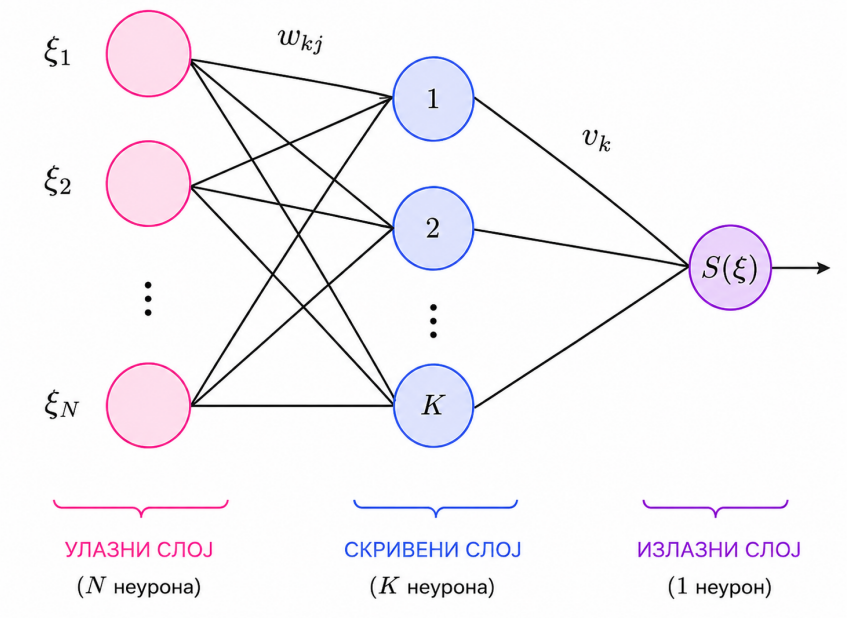
Улазни слој прима спољашње податке, на пример вредности мерења или пикселе слике. Излазни слој даје коначан резултат рада мреже, док се између њих налазе скривени слојеви, чији неурони врше обраду и трансформацију информација. Неурони скривених слојева не комуницирају директно са спољашњим окружењем, већ примају сигнале од претходног слоја и прослеђују их наредном.

Активност неурона i у слоју k се представља изразом :

$$S_i^{(k)} = g\left(\sum_j w_{ij}^{(k)} S_j^{(k-1)}\right),$$

где је $S_j^{(k-1)}$ активност неурона j у претходном слоју, $w_{ij}^{(k)}$ тежина везе између неурона j из претходног слоја, и неурона i у слоју k, а g активациона функција. За разлику од рекурентних мрежа, код неповратних мрежа не постоји механизам памћења претходних стања, информације се обрађују усмерено, слој по слој и не враћају се назад. Због тога се овакве мреже најчешће користе у задацима код којих није потребна временска зависност података.

Као пример неповратне неуронске мреже може се посматрати мрежа која се састоји од улазног слоја, једног скривеног слоја и излазног слоја који чини један излазни неурон (*слика 3*).



Слика 3: Пример неповратне неуронске мреже

Улазни слој прима вектор улазних података :

$$\xi = (\xi_1, \xi_2, \ldots, \xi_N),$$

при чему свака компонента вектора представља један улазни сигнал мреже.

Сваки неурон скривеног слоја формира своју активност на основу тежинског збира улазних сигнала и одговарајуће активационе функције. Активности

скривених неурона се означавају са σ_k :

$$\sigma_k = g\left(\sum_{j=1}^{N} w_{kj}\,\xi_j\right).$$

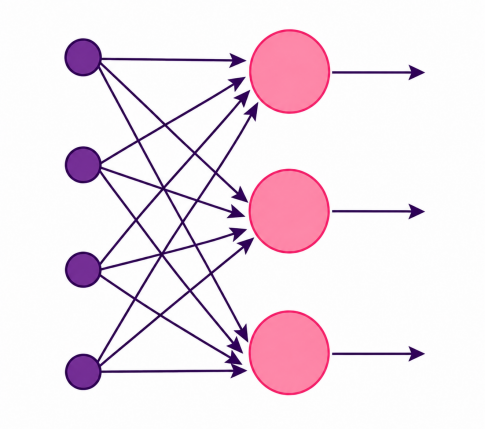
Излазни неурон затим комбинује активности свих неурона скривеног слоја, при чему се свака од њих множи одговарајућом тежином ка излазном неурону, v_k. Коначан излаз мреже гласи :

$$S(\xi) = g\left(\sum_{k=1}^{K} v_k\,\sigma_k\right) = g\left(\sum_{k=1}^{K} v_k\, g\left(\sum_{j=1}^{N} w_{kj}\,\xi_j\right)\right),$$

при чему је g активациона функција. Ради једноставности, претпоставља се да сви неурони скривеног и излазног слоја користе исту активациону функцију. Наравно, ово ограничење није неопходно, па различити слојеви, па чак и појединачни неурони, могу имати различите активационе функције.

Неповратне неуронске мреже се користе у задацима класификације. Модел додељује улазне податке једној од унапред дефинисаних класа. На пример, мрежа може одредити да ли се на слици налази мачка или пас, да ли је електронска порука жељена или не, или које слово је написано руком. У том случају излаз мреже не представља произвољну вредност, већ ознаку одговарајуће категорије.

Уз довољан број неурона и одговарајуће тежине, овакве мреже могу апроксимирати и веома сложене нелинеарне зависности. Захваљујући томе оне имају широку примену у проблемима препознавања образаца, обраде слика и говора, медицинској дијагностици итд.



Слика 4: Шематски приказ неповратне неуронске мреже

Најједноставнији пример неповратне неуронске мреже је *перцептрон*. Он се користи за решавање проблема бинарне класификације, односно за раздвајање података у две различите категорије. Захваљујући једноставној структури и значајној улози у развоју машинског учења, перцептрон представља основу многих сложенијих модела неуронских мрежа. Наредни одељак посвећен је управо њему.

3.3 Перцептрон

3.3.1 Појам и структура перцептрона

Перцептрон представља један од првих модела вештачких неуронских мрежа. Развио га је Френк Розенблат крајем педесетих година 20. века, са циљем да моделује једноставан процес доношења одлука на основу улазних података. Заснива се на идеји Мекалок–Питсовог неурона, али уводи могућност обучавања, односно прилагођавања тежина током рада система.

Био је један од првих покушаја да се процес учења формализује помоћу математичког модела. У време његовог настанка сматрало се да би повезивањем већег броја оваквих јединица било могуће конструисати системе способне за препознавање образаца и доношење једноставних одлука, слично начину на који функционише људски мозак. Због тога је перцептрон имао велики утицај на даљи развој неуронских мрежа и машинског учења.

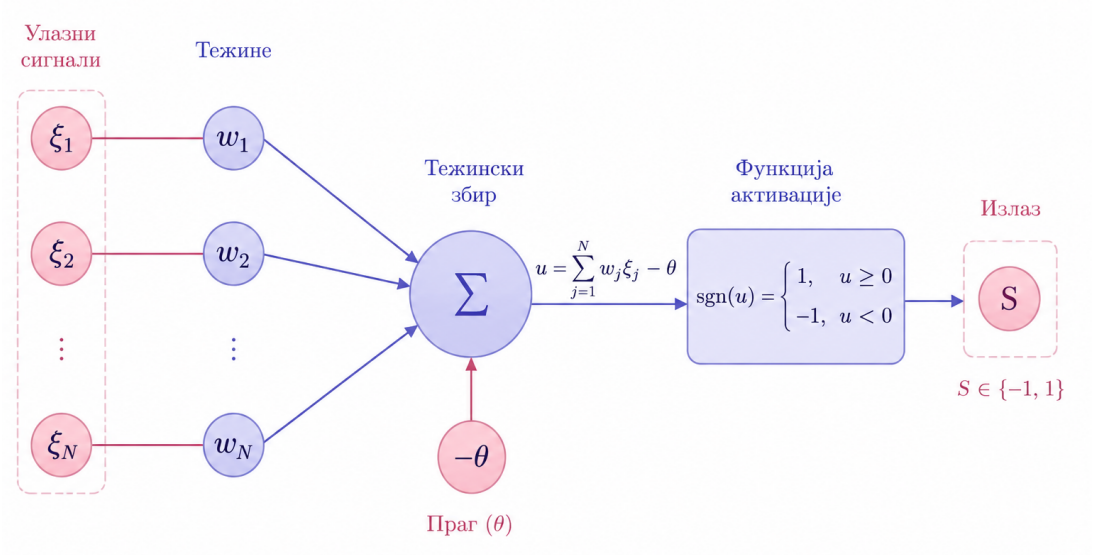
Основни задатак перцептрона је бинарна класификација, односно разврставање података у две категорије. На основу улазних сигнала и одговарајућих тежина, перцептрон доноси одлуку о томе којој класи посматрани податак припада.

У свом најједноставнијем облику, перцептрон се састоји од једног вештачког неурона који прима улазне сигнале, обрађује их и даје излазну вредност. Комбиновањем већег броја перцептрона настају сложеније структуре, које могу решавати комплексније проблеме.

Он је линеарни класификатор одређен параметром θ, који означава праг активације :

$$S(\xi) = \operatorname{sgn}\left(\sum_{j=1}^{N} w_j \xi_j - \theta\right),$$

где је ξ_j компонента улазног вектора ξ, а w_j њој придружена тежина. На основу овога јасно је да $S(\xi) \in \{-1, 1\}$, то јест да перцептрон дели улазне податке на позитивну и негативну класу, у зависности од знака горе наведеног израза.



Слика 5 : Начин рада перцептрона

Линеарни класификатор представља модел који улазне податке раздваја помоћу линеарне границе одлучивања. У зависности од димензије простора, та граница може бити права, раван или, у општем случају, хиперраван. Перцептрон на основу вредности тежина и прага активације одређује са које стране те границе се улазни податак налази, односно којој класи припада. Због тога је рад перцептрона уско повезан са појмом линеарне раздвојивости.

3.3.2 Линеарна раздвојивост

Да би перцептрон успешно класификовао улазне податке, потребно је да између различитих класа постоји јасна граница раздвајања (енг. *decision boundary*). Скуп података назива се линеарно раздвојивим уколико је могуће раздвојити његове класе једном линеарном границом одлучивања.

У најједноставнијем случају, када се подаци представљају у равни, граница раздвајања има облик праве. У тродимензионалном простору, граница је раван, док се у просторима већих димензија користи појам хиперравни.

Нека је $\xi = (\xi_1, \xi_2, ..., \xi_N)$ улазни вектор и $w = (w_1, w_2, ..., w_N)$ вектор тежина перцептрона. Израз

$$w \cdot \xi - \theta = 0$$

описује границу раздвајања. Вектор w одређује положај и оријентацију границе, док параметар θ представља праг активације. Са различитих страна ове границе налазе се подаци који припадају различитим класама.

Пример *Права као граница раздвајања*

Посматра се скуп тачака у равни. Нека тачке $A(1,1), B(2,1), C(1,2)$ припадају позитивној класи, а тачке $D(4,4), E(5,4), F(4,5)$ негативној. Потребно је одредити границу раздвајања између ових класа.

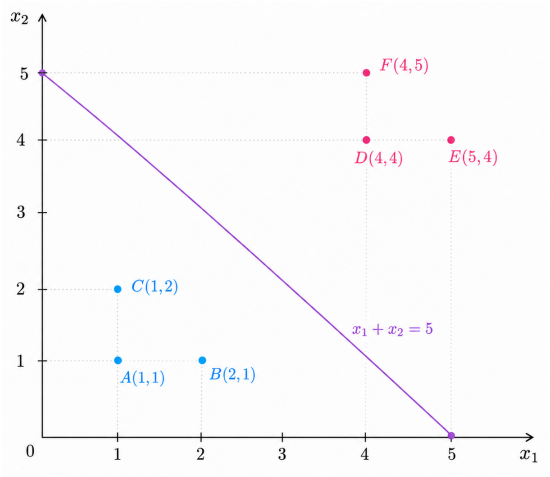
Пошто су улазни подаци задати тачкама у равни, сваки улаз описан је са две координате. Због тога и улазни вектор ξ, као и вектор тежина w, имају по две компоненте. У том случају, граница раздвајања је описана једначином :

$$w_1 x_1 + w_2 x_2 - \theta = 0.$$

Тада се за дате тачке добијају следећи изрази :

$$\begin{aligned}
A(1,1): &\quad w_1 + w_2 - \theta > 0 \\
B(2,1): &\quad 2w_1 + w_2 - \theta > 0 \\
C(1,2): &\quad w_1 + 2w_2 - \theta > 0 \\
D(4,4): &\quad 4w_1 + 4w_2 - \theta < 0 \\
E(5,4): &\quad 5w_1 + 4w_2 - \theta < 0 \\
F(4,5): &\quad 4w_1 + 5w_2 - \theta < 0
\end{aligned}$$

Једно од решења система је $(w_1, w_2, \theta) = (-1, -1, -5)$. Тада је граница раздвајања $x_1 + x_2 = 5$. То што решење није јединствено значи да постоји више различитих граница раздвајања које успешно класификују исте податке. Различите вредности тежина и прага могу довести до различитих правих, при чему свака од њих исправно раздваја посматране класе.

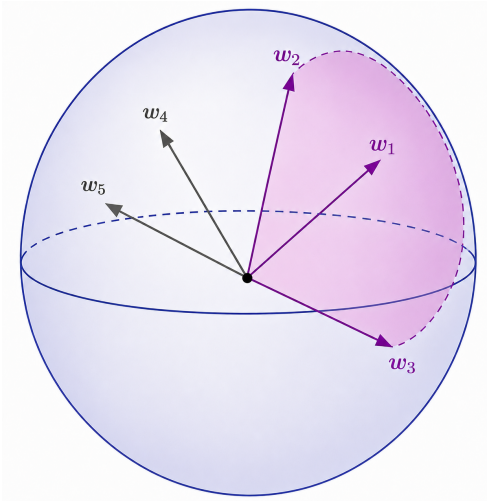


Слика 6 : Геометријска интерпретација примера

У тродимензионалном простору границу раздвајања више не представља права, већ раван. Она раздваја податке на два дела простора, при чему се тачке различитих класа налазе са разних страна равни.

У општем случају, када подаци имају више од три димензије, граница раздвајања назива се *хиперраван*. Иако такве просторе није могуће директно геометријски приказати, математички принцип остаје исти: хиперраван дели простор на два дела и омогућава класификацију података.

Уколико се проблем линеарне раздвојивости посматра из другог угла, уместо самих података могу се анализирати могући избори вектора тежина w. У том случају, сваки вектор тежина представља једну тачку у простору параметара. Ако се посматрају само вектори исте дужине, односно $|w| = const$, тада се сви могући вектори тежина налазе на сфери, односно, у општем случају, на хиперсфери. Одређени региони те хиперсфере одговарају изборима тежина који успешно раздвајају посматране класе података. На тај начин, проблем линеарне раздвојивости може се геометријски интерпретирати као тражење одговарајућег региона у простору тежина (енг. *learning in version space*).



Слика 7 : Геометријска интерпретација простора тежина и сфере у њему. Осенчени регион одговара векторима који успешно раздвајају посматране класе.

Међутим, поставља се питање шта се дешава уколико подаци нису линеарно раздвојиви, односно ако не постоји једна права, раван или хипер раван која може раздвојити различите класе. Један од најпознатијих примера таквог проблема је *проблем ексклузивне дисјункције* или XOR проблем. Посматрају се четири тачке:

$$(0,0),(0,1),(1,0),(1,1).$$

Нека тачке (0,0) и (1,1) припадају једној класи, а тачке (0,1) и (1,0) другој. Геометријски, ове тачке су распоређене тако да није могуће повући једну праву која би раздвојила класе без погрешне класификације бар једне тачке. Због тога једнослојни перцептрон није способан да успешно реши XOR проблем, што представља једно од његових основних ограничења. Историјски, ово је било огромно откриће, јер је покренуло развој сложенијих, вишеслојних неуронских мрежа.

Упркос наведеним ограничењима, једнослојни перцептрон представља основу развоја савремених неуронских мрежа и има значајну улогу у теорији машинског учења. Због тога ће у наставку бити размотрен Розенблатов модел једнослојног перцептрона и алгоритам његовог обучавања.

3.3.3 Розенблатов алгоритам обучавања перцептрона

Као што је већ наведено, Розенблатов модел перцептрона описује се изразом

$$S(\xi) = \operatorname{sgn}\left(\sum_{i=1}^{N} w_i \xi_i - \theta\right).$$

Ова формула описује начин на који перцептрон доноси одлуку, али најзначајнији део Розенблатовог модела је поступак обучавања, односно начин на који се тежине прилагођавају на основу примера.

Током обучавања, перцептрону се редом задају примери из тренинг скупа. За сваки пример израчунава се излаз модела и упоређује са тачном ознаком класе. Уколико је пример исправно класификован, тежине се не мењају. Ако је класификација погрешна, тежине се коригују тако да се повећа вероватноћа исправне класификације тог примера у наредним корацима. На тај начин перцептрон постепено помера границу раздвајања и прилагођава се подацима.

Формално речено, корекција тежина у Розенблатовом моделу заснива се на разлици између добијеног и очекиваног излаза. Нека је за дати улазни вектор ξ очекивани излаз означен са y, док је стварни излаз перцептрона једнак $S(\xi)$. Ако је $S(\xi) = y$ тежине остају непромењене. Међутим, у супротном, врши се измена тежина према правилу :

$$w_i^{(t+1)} = w_i^{(t)} + \eta y \xi_i,$$

где је η параметар учења, а t редни број итерације током процеса обучавања. На сличан начин се може кориговати и праг активације :

$$\theta^{(t+1)} = \theta^{(t)} - \eta y.$$

Уколико перцептрон погрешно класификује неки пример, вредности тежина се мењају тако да се граница раздвајања помери ка исправној класификацији тог примера. Процес обучавања понавља се над свим примерима из тренинг скупа све док перцептрон не престане да прави грешке или док се не достигне задати број

итерација. Уколико су подаци линеарно раздвојиви, Розенблатов алгоритам ће након коначног броја корака пронаћи одговарајуће вредности тежина.

Међутим, једноставни модели попут перцептрона нису довољни за решавање многих сложених проблема машинског учења. Због тога су развијене дубоке неуронске мреже, које захваљујући већем броју слојева и различитим активационим функцијама могу описивати сложене нелинеарне зависности у подацима.

3.4 Дубоке неуронске мреже и активационе функције

Дубоке неуронске мреже (енг. *deep neural networks*) представљају мреже које садрже већи број скривених слојева између улазног и излазног слоја. Сваки слој обрађује информације добијене из претходног слоја и прослеђује резултат наредном, при чему мрежа постепено учи све сложеније карактеристике података. Захваљујући оваквој хијерархијској обради података, дубоке неуронске мреже данас имају широку примену у препознавању говора и рукописа, машинском превођењу, аутономној вожњи, системима препоруке, генеративној вештачкој интелигенцији итд.

Важну улогу у раду дубоких мрежа имају активационе функције, које одређују начин на који неурони реагују на улазне сигнале. Њихова нелинеарност омогућава мрежама да описују сложене зависности у подацима и успешно решавају проблеме које једноставни линеарни модели не могу описати.

Једна од најпознатијих и данас најчешће коришћених активационих функција јесте **ReLU функција** (енг. *Rectified Linear Unit*). Њена популарност заснива се на једноставности, ефикасности и доброј успешности у обучавању дубоких неуронских мрежа. Дефинише се на следећи начин :

$$g(x) = \max\{0, x\} = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}, \qquad g'(x) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases}.$$

То значи да ReLU за негативне вредности улаза даје излаз 0, док за позитивне вредности излаз постаје једнак самом улазу. Графички посматрано, функција се за негативне вредности поклапа са x-осом, док за позитивне вредности представља праву $y = x$.

Дубоке неуронске мреже могу се посматрати као композиције већег броја функција, јер сваки слој прима излаз претходног слоја као улаз и над њим врши нову трансформацију. Први извод описује колико се нека величина мења, а током обучавања циљ је да се одреди на који начин треба променити тежине како би се смањила грешка модела. Због тога се током обучавања користи ланчано правило извода, према коме је извод сложене функције једнак производу одговарајућих извода. При процесу обучавања, грешка модела се постепено преноси уназад кроз мрежу како би се одредило на који начин треба изменити тежине. Код сигмоидалних функција, за велике позитивне и негативне вредности улаза извод постаје близак нули. Овако мале вредности се међусобно множе и постепено постају занемарљиве. Због тога се тежине у дубљим слојевима веома мало мењају, па мрежа споро учи или практично престаје да се обучава. Ова појава назива се проблем нестајања градијента (енг. *vanishing gradient*).

ReLU функција ублажава овај проблем јер је њен извод за позитивне вредности

једнак 1. Због тога градијенти остају већи током простирања кроз мрежу, па се и тежине ефикасније ажурирају. На тај начин обучавање дубоких неуронских мрежа постаје брже и стабилније.

Ипак, ова функција има и одређене недостатке. Уколико неурон током обучавања често добија негативне вредности улаза, излаз ReLU функције постаје једнак нули, а извод функције такође постаје нула. Због тога се тежине тог неурона више не мењају, па неурон практично престаје да учи. Ова појава позната је као проблем „мртвих“ неурона (енг. *dying ReLU*).

Као једно од решења овог проблема развијена је **Leaky ReLU функција**, дефинисана изразом :

$$g(x) = \begin{cases} ax, & x < 0 \\ x, & x \geq 0 \end{cases}, \qquad g'(x) = \begin{cases} a, & x < 0 \\ 1, & x > 0 \end{cases},$$

$$0 < a < 1.$$

За разлику од класичне ReLU функције, код Leaky ReLU функције негативне вредности улаза не пресликавају се у нулу, већ задржавају малу негативну вредност. На тај начин извод функције остаје различит од нуле и за негативне вредности, што омогућава да неурон настави процес обучавања.

Поред ове функције, развијене су и друге модификације и алтернативе ReLU функцији, са циљем побољшања процеса обучавања дубоких мрежа. Неке од њих су :

- **Softplus функција**

  $$g(x) = \ln(1 + e^x)$$

  Ова функција представља „глатку“ верзију ReLU функције, јер нема нагли прелаз у тачки $x = 0$. За велике позитивне вредности понаша се слично ReLU функцији, док за негативне вредности постепено тежи нули.

- **ELU функција** (енг. *Exponential Linear Unit*)

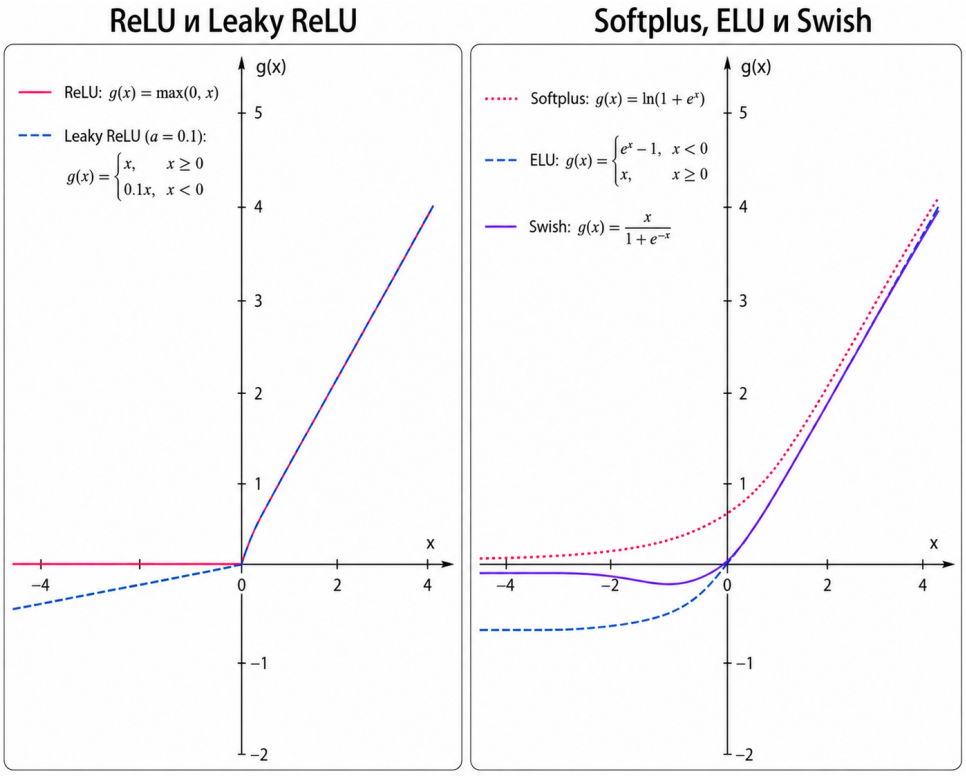
  $$g(x) = \begin{cases} e^x - 1, & x < 0 \\ x, & x \geq 0 \end{cases}$$

  За разлику од ReLU функције, ELU за негативне вредности не даје нулу, већ постепено тежи вредности -1. На тај начин се смањује вероватноћа појаве „мртвих“ неурона и омогућава стабилније обучавање мреже.

- **Swish функција**

  $$g(x) = \frac{x}{1 + e^{-x}}$$

  Ово је глатка и нелинеарна функција која комбинује особине сигмоидалних и ReLU функција. Захваљујући доброј успешности у пракси, примењује се у појединим савременим моделима дубоког учења.



Слика 8 : Графички приказ активационих функција ReLU, Leaky ReLU, Softplus, ELU и Swish

Захваљујући развоју дубоких неуронских мрежа и различитих активационих функција, савремени модели машинског учења постали су способни за решавање веома сложених проблема из различитих области. Иако се њихова основа заснива на математичким моделима и теоријским принципима, неуронске мреже данас имају и значајну практичну примену. У наредном одељку, пар примера ће бити приказано.

3.5 Имплементација неуронских мрежа

Упркос сложеној математичкој основи, савремене неуронске мреже могу се релативно једноставно имплементирати помоћу специјализованих библиотека. Данас постоји велики број алата за развој модела машинског учења, који омогућавају ефикасно креирање, обучавање и тестирање неуронских мрежа.

Једна од најзначајнијих библиотека за развој неуронских мрежа у програмском језику Python је PyTorch. Она омогућава једноставно дефинисање и обучавање модела, као и примену различитих активационих функција и алгоритама учења. Захваљујући једноставној синтакси и великој примени у пракси, PyTorch представља један од најчешће коришћених алата у области дубоког учења. Наредна три једноставна примера то показују.

Пример 1 *Неурон са ReLU функцијом*

```
import torch
import torch.nn as nn
x = torch.tensor([-2.0, -1.0, 0.0, 1.0, 2.0])
relu = nn.ReLU()
y = relu(x)
print(y)
```

На почетку програма увозе се PyTorch библиотека и модул за неуронске мреже, који садржи компоненте за рад са неуронским мрежама. Затим се формира тензор улазних вредности над којима ће бити примењена ReLU активациона функција. Након дефинисања ReLU функције, она се примењује на улазне податке, при чему све негативне вредности постају нула, док позитивне остају непромењене. На крају се исписује добијени резултат.

Излаз :

```
tensor([0., 0., 0., 1., 2.])
```

Пример 2 *Неуронска мрежа са једним скривеним слојем*

```
import torch
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(2, 4),
    nn.ReLU(),
    nn.Linear(4, 1) )
x = torch.tensor([[1.0, 2.0]])
y = model(x)
print(y)
```

Дефинише се једноставна неуронска мрежа са једним скривеним слојем који садржи четири неурона. Слојеви мреже су организовани тако да се излаз једног слоја прослеђује наредном. Мрежа прима два улазна податка, затим у скривеном слоју формира четири нове вредности над којима се примењује ReLU функција, а након тога се добија коначан излаз мреже. На крају се дефинише и улазни тензор који пролази кроз мрежу и исписује се добијени резултат.

Пошто мрежа није обучена, тежине у њој су случајно постављене, па ће и излаз бити нека случајна реална вредност.

Пример 3 *Обучавање неуронске мреже*

```
import torch
import torch.nn as nn
import torch.optim as optim
x = torch.tensor([[0.0],[1.0],[2.0],[3.0]])
y = torch.tensor([[0.0],[2.0],[4.0],[6.0]])
model = nn.Sequential(
    nn.Linear(1, 1))
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
for epoch in range(1000):
    prediction = model(x)
    loss = criterion(prediction, y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
print(model(torch.tensor([[5.0]])))
```

Увози се и модул који омогућава оптимизацију модела током обучавања. Затим се дефинишу улазни и излазни подаци на основу којих ће мрежа учити зависност између вредности. У овом примеру мрежа треба да научи правило $y = 2x$. Након тога формира се једноставна неуронска мрежа са једним улазним и једним излазним неуроном. Затим се дефинише функција грешке, која мери разлику између предвиђених и стварних вредности, као и алгоритам који служи за

корекцију тежина мреже током обучавања. Покреће се процес обучавања, током кога мрежа више пута пролази кроз све податке, формира предвиђања, израчунава грешку и постепено прилагођава своје тежине како би резултати били што прецизнији. На крају се обученој мрежи задаје нова улазна вредност, а програм исписује предвиђени резултат.

Добијена вредност не мора бити тачно 10, јер се тежине мреже одређују приближно током процеса обучавања. Ипак, након довољног броја пролаза кроз податке, резултат ће бити веома близак очекиваној вредности. Зато програм може исписати, на пример :

```
tensor([[9.98]])
```

У последњем примеру посебно се види да обучавање подразумева постепено прилагођавање тежина како би се грешка модела смањила. Управо се тим питањем бави наредно поглавље, у коме ће бити покаазано како модел током обучавања тежи све бољем решењу, односно како долази до његове конвергенције.

4 Конвергенција модела

У претходним поглављима разматрани су математички модели машинског учења, структура неуронских мрежа и процес њиховог обучавања. Међутим, поставља се питање на који начин модел током обучавања постепено долази до све бољих резултата. Да би неуронска мрежа успешно решавала одређени проблем, потребно је да њени параметри буду прилагођени тако да грешка модела буде што мања. Процес постепеног прилагођавања параметара и смањивања грешке назива се конвергенција модела.

4.1 Линеарна регресија

Линеарни предиктори представљају једну од најзначајнијих и најчешће коришћених класа хипотеза у машинском учењу. Њихова популарност заснива се на томе што се ефикасно обучавају, а истовремено се релативно једноставно тумаче. У великом броју практичних проблема линеарни модели успевају да веома добро опишу зависности између података.

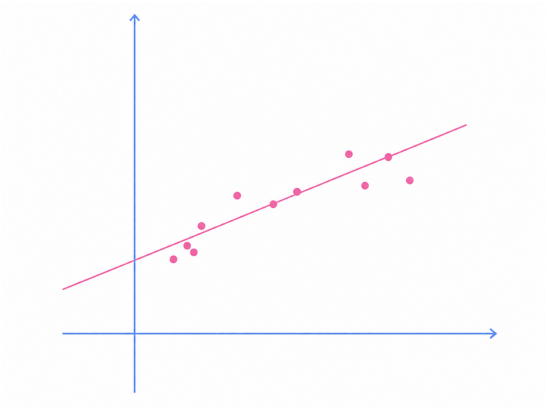
У оквиру ове породице модела издваја се линеарна регресија. Она представља један од основних модела машинског учења и често служи као полазна основа за разумевање сложенијих метода обучавања.

Нека је $x = (x_1, x_2, ..., x_d)$ улазни вектор, $w = (w_1, w_2, ..., w_d)$ вектор тежина и b слободни члан из скупа реалних бројева. Класа линеарних функција представља следећи скуп хипотеза :

$$L_d = \left\{ h_{w,b} \;\middle|\; h_{w,b}(x) = \left(\sum_{i=1}^{d} w_i x_i \right) + b \right\},$$

$$\mathcal{H}_{reg} = L_d.$$

Линеарна регресија (енг. *linear regression*) служи за моделовање зависности између улазних података и реалне излазне вредности. Циљ модела је да пронађе функцију $h : \mathbb{R}^d \to \mathbb{R}$ која што боље апроксимира посматране податке. Код једнодимензионалног случаја ($d = 1$) модел настоји да пронађе праву која најбоље описује распоред тачака у равни (*слика 9*).



Слика 9 : Линеарна регресија за $d = 1$

Приликом обучавања потребно је одредити колико се предвиђене вредности разликују од стварних података. Због тога се уводи **функција грешке**, која представља меру одступања модела од очекиваних резултата. Код линеарне регресије најчешће се користи **средња квадратна грешка** (енг. *Mean Squared Error* — MSE), одређена формулом :

$$L_S(h) = \frac{1}{m}\sum_{i=1}^{m}\big(h(x_i) - y_i\big)^2,$$

где је S тренинг скуп, а m број чланова тог скуп. Квадрирањем разлике између предвиђених и стварних вредности већа одступања добијају већу тежину, па модел тежи проналажењу параметара за које је укупна грешка што мања.

За решавање ERM проблема код линеарне регресије често се користи **метода најмањих квадрата** (енг. *Least Squares*). Основна идеја ове методе јесте проналажење таквих вредности параметара модела за које збир квадрата одступања између предвиђених и стварних вредности достиже минимум. Формално, потребно је пронаћи вредности параметара за које функција грешке има најмању могућу вредност :

$$arg\min_{w} L_S(h_w), \qquad \text{за} \qquad h_w(x) = w \cdot x = \sum_{i=1}^{d} w_i x_i.$$

Минимум функције грешке одређује се израчунавањем извода функције по параметрима модела и изједначавањем добијеног израза са нулом :

$$\frac{2}{m}\sum_{i=1}^{m}\big(h_w(x_i) - y_i\big)x_i = 0.$$

Овај проблем може другачије да се запише :

$$\text{како је } x_i = \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \vdots \\ x_{i_d} \end{bmatrix}, \text{ онда је } {x_i}^T = \begin{bmatrix} x_{i_1} x_{i_2} \dots x_{i_d} \end{bmatrix},$$

$$\text{па је } h_w(x_i) = \sum_{j=1}^{d} w_j x_{i_j} = w_1 x_{i_1} + w_2 x_{i_2} + \dots + w_d x_{i_d} = \begin{bmatrix} x_{i_1} x_{i_2} \dots x_{i_d} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} = {x_i}^T w.$$

Убацивањем овога у полазну једначину добија се :

$$\sum_{i=1}^{m}\left(x_i^T w - y_i\right)x_i = 0,$$

$$\sum_{i=1}^{m} x_i^T w x_i - \sum_{i=1}^{m} y_i x_i = 0,$$

$$\left(\sum_{i=1}^{m} x_i x_i^T\right) w = \sum_{i=1}^{m} y_i x_i.$$

Узмиају се ознаке :

$$A = \sum_{i=1}^{m} x_i x_i^T, \quad b = \sum_{i=1}^{m} y_i x_i.$$

Тада се добија систем линеарних једначина, са d непознатих и d једначина :

$$Aw = b.$$

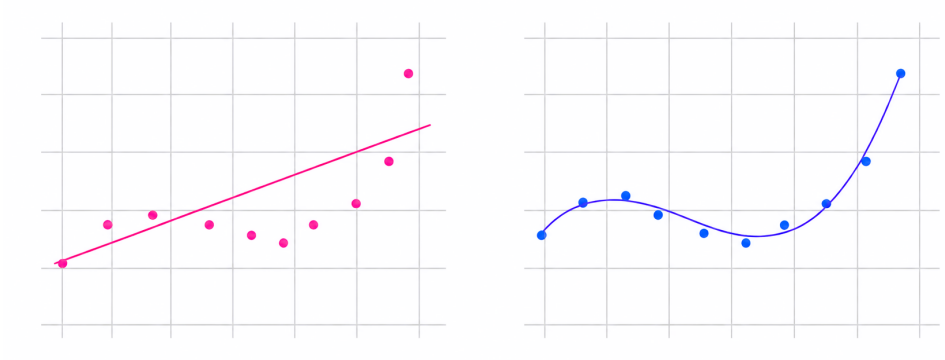
Уколико је A инверзибилна, то јест важи $\det A \neq 0$, решење система је јединствено и може се записати у облику :

$$w = A^{-1}b.$$

Ипак, нису сви проблеми машинског учења линеарни, па се зависности између података често не могу довољно добро описати правом. У неким од таквих ситуација користи се **полиномска регресија**, код које се зависност између улазних и излазних вредности описује полиномом n-тог степена :

$$p(x) = a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0.$$

Вектор $a = (a_0, a_1, ..., a_n)$ представља вектор коефицијената, односно параметара модела. За разлику од линеарне регресије, полиномска регресија омогућава описивање сложенијих зависности и боље прилагођавање распореду података (*слика 10*).



Слика 10 : Лево је приказана линеарна, а десно полиномска регресија

Посматра се класа једнодимензионих полиномских регресионих модела n-тог степена :

$$\mathcal{H}_{pol}^{n} = \{x \mapsto p(x)\}.$$

Иако полиномска регресија не описује линеарну зависност између x и y, она се може свести на линеарну регресију у новом простору карактеристика. Наиме, уместо да се као улаз посматра само вредност x, формира се нови вектор $\psi(x) = (1, x, x^2, \ldots, x^n)$. Тада важи :

$$p(x) = a \cdot \psi(x).$$

Због тога се тражење коефицијената полинома своди на исти тип проблема као код линеарне регресије, то јест потребно је пронаћи параметре који најбоље описују податке. Управо зато се за полиномску регресију може применити метода најмањих квадрата.

Иако метода најмањих квадрата омогућава аналитичко решавање проблема линеарне регресије, код сложенијих модела и великих скупова података такво решење често није практично. Због тога се у машинском учењу широко примењују итеративне методе оптимизације. Једна од најзначајнијих међу њима је градијентни спуст, о коме ће бити више речи у наредном одељку.

4.2 Градијентни спуст

Градијентни спуст представља један од најважнијих алгоритама оптимизације у машинском учењу. Његова основна улога јесте постепено минимизовање функције грешке модела, односно проналажење таквих вредности параметара модела при којима је грешка што мања. За разлику од аналитичких метода, код којих се решење добија директно, градијентни спуст до решења долази постепеним ажурирањем параметара кроз већи број итерација.

Основни појам на коме се заснива овај алгоритам јесте **градијент функције**. Нека је дата диференцијабилна функција $f : \mathbb{R}^d \to \mathbb{R}$. Њен градијент у тачки w означава се са $\nabla f(w)$ и представља вектор парцијалних извода :

$$\nabla f(w) = \left(\frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, \ldots, \frac{\partial f(w)}{\partial w_d} \right).$$

Свака компонента овог вектора описује колико се функција мења у односу на одговарајући параметар. Градијент показује правац најбржег раста функције у посматраној тачки. Због тога се, приликом минимизације функције грешке, параметри модела мењају у супротном смеру од градијента, јер се на тај начин вредност функције постепено смањује.

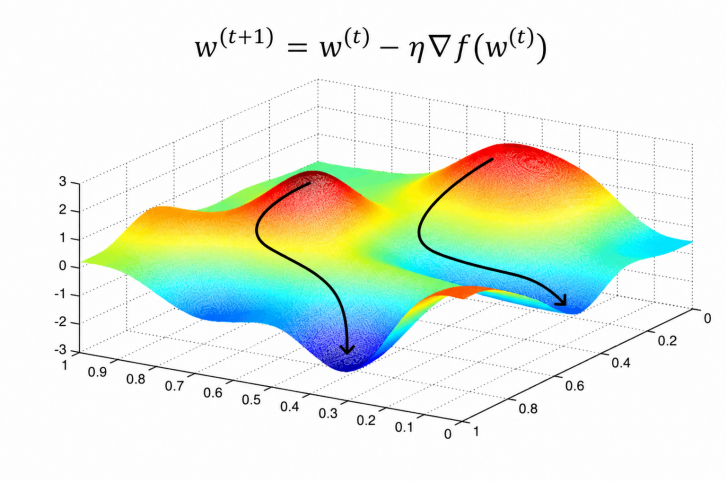
Градијентни спуст (енг. *gradient descent*) је итеративни алгоритам. Он почиње од почетне вредности параметара модела, која се најчешће бира насумично или поставља на нулу. Након тога, у свакој итерацији израчунава се градијент функције грешке у тренутној тачки и параметри се ажурирају према правилу :

$$w^{(t+1)} = w^{(t)} - \eta \nabla f\big(w^{(t)}\big),$$

где $w^{(t)}$ представља вектор параметара у t-тој итерацији, $\nabla f\big(w^{(t)}\big)$ градијент функције у тој тачки, а $\eta > 0$ параметар брзине учења (енг. *learning rate*). Избор вредности параметра η има велики утицај на успешност обучавања модела, јер одређује величину корака алгоритма. Уколико је вредност превелика, алгоритам може прескакати минимум функције грешке и постати нестабилан. Са друге стране, сувише мала вредност доводи до веома спорог приближавања решењу, па обучавање може трајати дуго.

На слици је приказан интуитиван рад алгоритма градијентног спуста при минимизацији функције f за $d = 2$. Површ представља функцију грешке модела, при чему веће висине одговарају већим вредностима грешке. Црне линије и стрелице приказују путању којом се параметри модела крећу током обучавања. У свакој итерацији алгоритам израчунава градијент функције, који показује правац њеног најбржег раста, а затим се креће у супротном смеру како би се вредност функције грешке постепено смањивала. На тај начин параметри модела постепено конвергирају ка минимуму функције. У сложеним моделима функција грешке може имати више локалних минимума, па алгоритам градијентног спуста може

конвергирати ка различитим решењима у зависности од почетних вредности параметара.



Слика 11 : Приказ конвергенције градијентног спуста ка минимуму функције

Градијентни спуст може се објаснити и помоћу Тејлоровог развоја првог реда. Идеја је да се функција у околини тренутне тачке локално апроксимира линеарним изразом, што омогућава процену како се функција мења при малој промени параметара. У општем случају, Тејлоров развој првог реда у околини тачке w има облик :

$$f(u) \approx f(w) + (u - w)f'(w).$$

Међутим, у машинском учењу функција најчешће зависи од већег броја параметара, па се уместо извода користи градијент функције :

$$f(u) \approx f(w) + (u - w)\nabla f(w).$$

Ако се градијент распише по компонентама, добија се :

$$f(u) \approx f(w) + (u_1 - w_1)\frac{\partial f(w)}{\partial w_1} + \cdots + (u_d - w_d)\frac{\partial f(w)}{\partial w_d}.$$

Вредност $f(w)$ представља вредност функције у тренутној тачки, док изрази $u_i - w_i$ описују промену појединачних параметара. Парцијални изводи показују колико је функција осетљива на промену сваког параметра. На тај начин могуће је локално проценити како ће се функција понашати након мале промене параметара.

У пракси, код великих скупова података и сложених модела, израчунавање градијента над целокупним скупом података може бити веома споро и рачунарски захтевно. Због тога се често користи **стохастички градијентни спуст** (енг. *Stochastic Gradient Descent* — SGD), који представља модификацију основног алгоритма градијентног спуста.

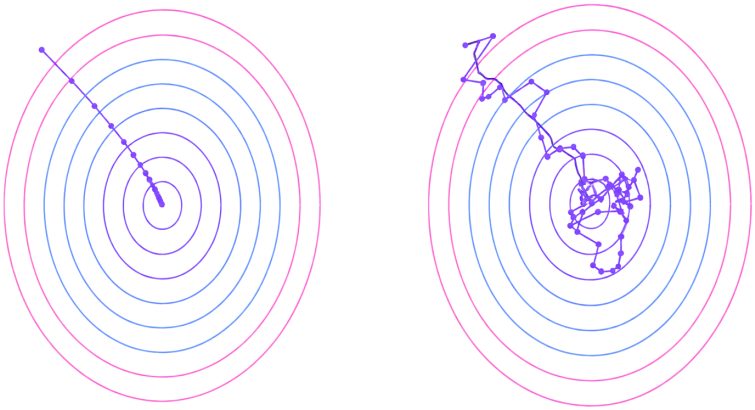
Код класичног градијентног спуста, градијент функције грешке израчунава се на основу целог тренинг скупа пре сваког ажурирања параметара. Насупрот томе, код SGD алгоритма параметри се ажурирају након обраде појединачног примера или мање групе примера из скупа података. На тај начин обучавање постаје знатно брже, посебно код великих неуронских мрежа и великих количина података.

У овом случају, правило ажурирања параметра је :

$$w^{(t+1)} = w^{(t)} - \eta v_t,$$

где v_t представља апроксимацију градијента добијену на основу случајно изабраних примера из тренинг скупа.

Због насумичног избора података, кретање ка минимуму функције није потпуно равномерно, већ садржи одређене осцилације. Ипак, упркос томе, SGD у пракси често брже долази до доброг решења него класични градијентни спуст. Зато је он данас један од основних алгоритама за обучавање неуронских мрежа има широку примену у савременим системима машинског учења.



Слика 12 : Лево је приказан рад алгоритма градијентног спуста, а десно рад алгоритма стохастичког градијентног спуста

Линеарна регресија и алгоритми градијентног спуста представљају основу великог броја савремених метода машинског учења. Иако су ови модели математички релативно једноставни, они илуструју кључне принципе обучавања, оптимизације и прилагођавања параметара на основу података. Разумевање ових принципа омогућава боље схватање начина на који функционишу данашњи модели машинског учења.

5 Закључак

Кроз рад је показано да процес машинског учења није заснован на унапред дефинисаним правилима, већ на способности модела да на основу података постепено проналази зависности и прилагођава своје параметре. Посебно је истакнута улога математичких модела и метода оптимизације у процесу обучавања и побољшавања успешности модела.

Такође је показано да неуронске мреже, иако на први поглед веома сложене, у суштини представљају комбинацију једноставних математичких елемената повезаних у веће системе. Повећањем броја параметара и слојева мрежа добија способност да апроксимира све сложеније функције и решава све теже проблеме.

Један од главних закључака рада јесте да је математика суштински део машинског учења и да разумевање математичких основа омогућава много боље разумевање начина на који функционишу савремени системи вештачке интелигенције. Иако су савремени модели често веома сложени, принципи на којима се заснива њихово обучавање могу се описати јасним математичким идејама.

Развој машинског учења и неуронских мрежа указује на то да ће ови модели у будућности имати још значајнију улогу у различитим областима науке, технологије и свакодневног живота. Због тога, важно је разумети начин њиховог функционисања и како се они могу искористити.

Овом приликом, желим да се захвалим мојим менторима, професорки Јелени Хаџи-Пурић и професорки Бојани Матић, на уложеном труду и тежњи да овај рад буде што бољи. Такође, желим да се захвалим свим професорима који су ми улепшали процес учења математике и подстакли ме да будем истрајна у њеном проучавању.

6 Литература

[1] Shai Shalev-Shwartz, Shai Ben-David, *Understanding Machine Learning*, Cambridge University Press, 2014.

[2] Michael Biehl, *The Shallow and the Deep*, University of Groningen Press, 2023.

[3] *A Brief History of AI*, скрипта :
https://drive.google.com/file/d/1GZM7v60WYy1-wZPvodhjqxpcxOEiZHme/view?usp=sharing