

Vještačka inteligencija

Bilješke sa predavanja

Akademski 2020/2021

Vještačka inteligencija

Bilješke sa predavanja

Akademska 2020/2021

Elektrotehnički fakultet Sarajevo

Ovaj dokument predstavlja radnu skriptu sa bilješkama sa predavanja za predmet Vještačka inteligencija na prvom ciklusu studijskog programa Računarstvo i Informatika Elektrotehničkog fakulteta Sarajevo. Dokument je baziran na prezentacijama i snimcima predavanja vanr. prof. dr Amile Akagić.

Predgovor

NAPOMENA: SLIKE SU PLACEHOLDERI!

Sadržaj

1 Predavanje 1 i 2	1
1.1 Uvod u vještačku inteligenciju	1
1.2 Kratka historija vještačke inteligencije	9
1.3 Prvi tragovi vještačke inteligencije	10
1.4 Aktuelnosti	12
1.5 Futurističke predikcije	15
1.6 Podoblasti vještačke inteligencije	15
2 Predavanje 3	17
2.1 Uticaj vještačke inteligencije na poslove	17
2.2 Klasifikatori	20
2.3 Historijski pregled neuronskih mreža	23
2.4 Primjene konvolucionih neuronskih mreža	28
2.5 Koncept klasifikacije slika	31
3 Predavanje 4	41
3.1 Uvod u neuronske mreže	41
3.2 Biološki i vještački neuron	50
4 Predavanje 5	57
4.1 Aktuelnosti	57
4.2 Motivacija za kreiranje vještačkih neuronskih mreža	58
4.3 Podjela neuronskih mreža	59
4.4 Neuronska mreža	59
4.5 Model neurona	60
4.6 Mreže bez povratnih veza	61
4.7 Mreže sa povratnim vezama (rekurentne mreže)	70
4.8 Funkcije gubitka	72
5 Predavanje 6	75
5.1 Korištenje podataka za proces učenja	75
5.2 Linearni klasifikatori	77
5.3 Funkcije gubitka	86

6 Predavanje 7	95
6.1 Regularizacija	95
6.2 Optimizacija	96
6.3 Metoda nasumičnog pretraživanja	97
6.4 Metoda nasumičnog lokalnog pretraživanja	98
6.5 Metoda gradijentnog spusta	98
6.5.1 Serijski gradijentni spust	101
6.5.2 Stohastički gradijentni spust	101
6.5.3 Mini-serijski gradijentni spust	102
6.6 Optimizacije SGD metode	103
6.6.1 SGD + Momentum	103
6.6.2 Nesterov Momentum	105
6.6.3 AdaGrad	106
6.6.4 RMSProp	107
6.6.5 Adam	108
6.6.6 Poređenje algoritama	108
7 Predavanje 8	109
7.1 Ekspertni sistemi	110
7.2 Historijat ekspertnih sistema	116
7.3 Vrste ekspertnih sistema	116
7.4 Osobine ekspertnih sistema	117
7.5 Baza znanja	117
7.6 Mašina za zaključivanje	118
7.7 Metod zaključivanja unaprijed	120
7.8 Metod zaključivanja unazad	121
8 Predavanje 9	125
8.1 Inženjering znanja	125
8.2 Pristup razvoju vještacke inteligencije	127
8.3 Ontološki inženjering	129
8.4 Znanje	131
8.4.1 Kategorije znanja	132
8.4.2 Predstavljanje znanja	133
Stablo odlučivanja	134
Produkciona pravila	136
Semantičke mreže	136
Okyiri	137
Zaključivanje na osnovu znanja	139
9 Predavanje 10	141
9.1 Algoritmi evolucijskog računarstva	141
9.2 Evolucijsko računarstvo	142
9.3 Genetički algoritmi	147
9.3.1 Genetičke operacije	150

Reprodukција	150
Ukrštavanje	150
Mutacija	151
9.3.2 Proces	151
9.3.3 Parametri genetičkog algoritma	152
9.4 Prednosti i mane	154
10 Predavanje 11	157
10.1 Evolucija	157
10.2 Vještački život	161
10.3 Optimizacijski problemi	161
10.3.1 Optimizacijske metode	162
Konvencionalna rješenja	162
Metaheuristički algoritmi	162
Hibridni algoritmi	163
10.4 Roj	163
10.4.1 Optimizacija roja	163
Optimizacija roja čestica	163
Optimizacija mrvlje kolonije	167
11 Predavanje 12	169
11.1 Agent	169
11.1.1 Koncept racionalnosti	171
11.2 Okruženje agenta	172
11.2.1 Osobine okruženja	173
11.2.2 Vrste okruženja	173
U odnosu na broj agenata	174
U odnosu na stanja	174
U odnosu na prethodne akcije	174
U odnosu na stanje	175
U odnosu na vrijeme	175
U odnosu na poznavanje okoline	175
11.3 Struktura agenta	175
11.4 Program agenta	176
11.4.1 Jednostavni refleksni agenti	176
11.4.2 Model-bazirani refleksni agenti	177
11.4.3 Ciljno-bazirani agenti	179
11.4.4 Korisno-bazirani agenti	179
11.5 Agenti za učenje	180

Predavanje 1 i 2

1.1 Uvod u vještačku inteligenciju

Vještačka - napravljeno ili proizvedeno od strane ljudskih bića (ne javlja se prirodno, kopija nečeg prirodnog)

Inteligencija (lat. *intelligere*) - razabirati, shvaćati, razumijevati

Ne postoji adekvatna definicija kojom možemo reći šta je zapravo inteligencija jer se radi o deskriptivnom pojmu koji se ne može precizno mjeriti.

Po definicijama inteligencije koje se javljaju u rječnicima, **inteligencija je:**

- sposobnost učenja ili razumijevanja ili rješavanja novih situacija
- sposobnost primjene znanja za manipulaciju okruženjem u kojem se nalazimo ili za apstraktno razmišljanje, mjereno objektivnim kriterijima (kao što su testovi)
- mentalna prodornost
- čin razumijevanja

Pored toga, postoji još mnogo definicija inteligencije po raznim literaturama. Neke od njih su:

Osobina uspješnog snalaženja (adaptacije) jedinke u novim životnim situacijama. (*R. Pinter*)

Inteligencija se manifestira u odnosu na neki posebni društveni i kulturni kontekst. (*J. Weizenbaum*)

Sinonimi riječi inteligencija su: mozak, snaga mozga, siva materija, intelekt, intelektualnost, mentalitet, razum, smisao, pamet.

Inteligencija se može mjeriti na razne načine. Danas često mjerimo inteligenciju pomoću **Stanford-Binet IQ Test-a** ili po **Binet-Simon Skali**

Inteligencije. Ove načine mjerenja inteligencije je osmislio psiholog *Alfred Binet* koji je kao cilj imao da razvrsta različite nivoje inteligencije. Pored toga, osmislio je "generalnu napravu za reagovanje i svrstavanje sve djece u određene metalne skupine", odnosno, prvi test inteligencije.



Slika 1.1: Alfred Binet

Pored toga, bitno je spomenuti i **Dunning-Krugerov efekat**. Radi se o tendenciji ljudi ispodprosječne inteligencije koji smatraju da su inteligentniji nego što to jesu u stvarnosti i istodobno podcjenjuju inteligenciju ljudi koji im stoje nasuprot.

Alternativni način definisanja inteligencije daje *Howard Gardner* u svojoj knjizi **Frames of Mind**. On iznosi drugi pogled na inteligenciju gdje kaže da

"Ljudska bića posjeduju složen skup sposobnosti izvan onoga što se mjeri tradicionalnim koeficijentom inteligencije".

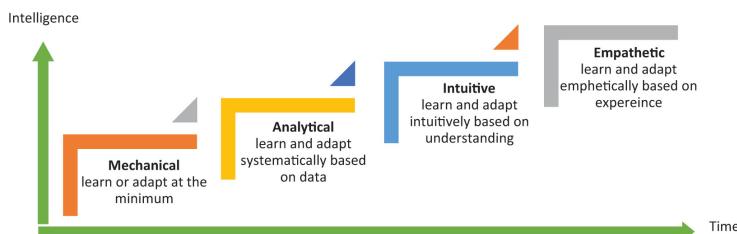
Shodno tome, on definiše 8(9) ključnih inteligencija:

1. **lingvistička** - sposobnost za izražavanje u odnosu na okolnosti u kojima se nalazimo
2. **logičko-matematička** - sposobnost za logičko i apstraktno razmišljanje
3. **prostorna** - sposobnost za snalaženje u prostoru
4. **tjelesno-kinestetička** - sklonost ka bavljenju fizičkim aktivnostima
5. **muzička** - razumijevanje ritma, melodije i zvučnih motiva
6. **interpersonalna** - zapažanja osjećaja kod drugih ljudi, sposobnost da se uspostavi komunikacija s drugima
7. **intrapersonalna** - očituje se u samosvjesnosti, samorazumijevanju, razumijevanju vlastitih potreba i mogućnosti

8. **prirodna** - sposobnost razumijevanja različitih situacija i snalaženja u njima ili na različitim mjestima
9. **(egzistencijalna/duhovna)** - osjetljivost i kapacitet osobe da se bavi dubokim pitanjima o ljudskoj egzistenciji

Naučna literatura razlikuje četiri vrste inteligencije i podijeljene su po vremenu koje je potrebno da se određeni posao izvrši (za mehaničku je potrebno najmanje vremena, a za empatičnu najviše):

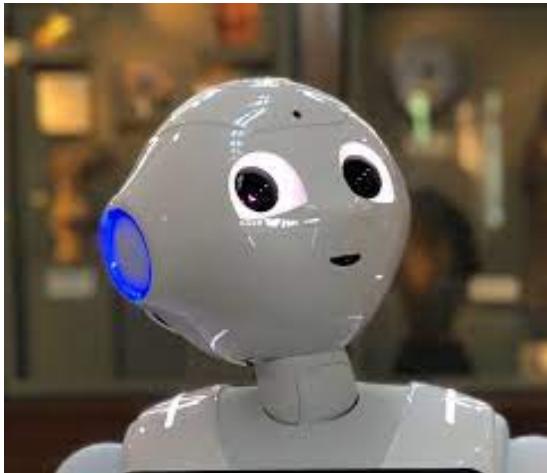
- **mehanička** - mogućnost izvršavanja rutinskih i ponavljajućih zadataka (može se replicirati određenim algoritmima)
- **analitička** - obrada podataka/informacija za rješavanje problema i učenje iz procesa
- **intuitivna** - kreativno razmišljanje i prilagođavanje novim situacijama (radi na bazi stečenih iskustava)
- **empatična** - prepoznavanje i razumijevanje ljudskih emocija, razvoj adekvatnog emocionalnog odgovora i uticanje na emocije drugih



Slika 1.2: Podjela inteligencije

Empatična inteligencija je najteža za postignuti. Industry 4.0 (Četvrti Industrijska revolucija) teži ka ostvarivanju ove inteligencije. Podrazumijeva korištenje raznih modernih tehnologija za usavršavanje industrijskih procesa. Glavni cilj jeste brisanje granica između fizičke, digitalne i biološke sfere.

Primjer visokog nivoa inteligencije je društveni robot Pepper. Cilj ovog robota je da on u uslužnim ustanovama daje odgovore na određena pitanja. Međutim, robot još uvijek nije razvijen do te tačke gdje on može odgovoriti na svako postavljeno pitanje, već nekada odgovara određenim nepovezanim izrazima. Također, on nije u mogućnosti da prepozna emocije druge osobe s kojom razgovara, a pored toga nema ni svoje emocije. Samim time, ovaj robot ne pripada najvišem nivou inteligencije.



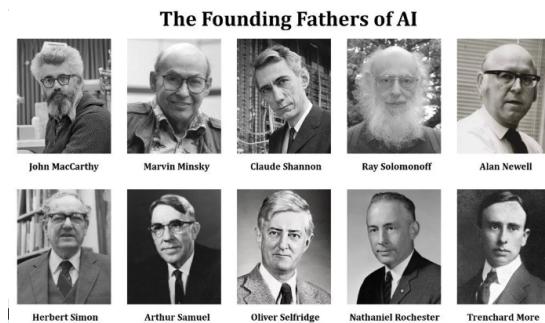
Slika 1.3: Robot Pepper

Svaka navedena vrsta inteligencije može da mijenja određene današnje poslove na različite načine. Jasno je da će se nivoi inteligencije razlikovati u načinu pristupa poslu, te će biti ograničene po količini posla koje mogu zamijeniti.

Uzmimo kao primjer posao psihijatra. Ukoliko bi ovaj posao preuzeila mehanička vještačka inteligencija, posao koji bi ona mogla da izvrši jeste da vodi zapisnik onoga o čemu pacijent govori. Kada bismo pričali o analitičkoj inteligenciji, ona bi uz vođenje zapisnika mogla i da analizira razgovor sa pacijentom. Intuitivna inteligencija bi mogla da poveže dobijene informacije kako bi došla do rješenja i ponudila odgovarajući lijek za pacijenta. Jasno je da bi ova vrsta inteligencija kroz više rada mogla da rješava probleme jasnije i brže (iz činjenice da uči iz stečenih iskustava). Na kraju, ukoliko bi ovaj posao preuzeila empatična inteligencija, ona bi pacijentima nudila empatičnost i komunikaciju kako bi se pacijent osjećao ugodno i shvaćeno.

Šta je vještačka inteligencija?

Smatra se da je sam pojam vještačke inteligencije nastao 1956. godine u New Hampshire-u gdje su se skupili naučnici sa vodećih institucija. Raspravljali su o temeljima i stvaranju ovog novog naučnog polja. Sastank je sazvao *John McCarthy*.



Slika 1.4: Founding Fathers of AI

Ako bismo gledali prvu definiciju vještačke inteligencije, vežemo ju za John McCarthy-a koji je dao sljedeću definiciju 1956. godine na navedenom sastanku:

"Naučna disciplina koja se bavi izgradnjom računarskih sistema čije se ponašanje može tumačiti kao intelligentno."

Jedan od osnovnih ciljeva je, ne samo mogućnost razumijevanja intelligentnih jedinki, nego i imati potrebno razumijevanje za izgradnju jednog takvog entiteta.

Kao ni za inteligenciju, tako ni za vještačku inteligeniciju ne postoji općeprihvaćena definicija. Jedna od definicija vještačke inteligencije koja obuhvata širi kontekst koju je dao *Dan Patterson* 1990. godine:

"Vještačka inteligencija je grana računarskih nauka koja se bavi proučavanjem i oblikovanjem računarskih sistema koji pokazuju neki oblik inteligencije, te takvi sistemi mogu da:

1. uče
2. donose zaključke o svijetu koji ih okružuje,
3. mogu razumijeti prirodni jezik,
4. mogu spoznati i tumačiti složene vizuelne scene i
5. obavljati druge vrste vještina za koje se zahtijeva čovjekova vrsta inteligencije."

Intelligentni sistemi se mogu podijeliti na četiri grupe koje se vežu za određene fenomene, odnosno, mogu se predstaviti kroz dvije dimenzije:

- razmišljati kao čovjek
- razmišljati racionalno

- ponašati se kao čovjek
- ponašati se racionalno

Ukoliko je moguće napraviti ovakav sistem, onda možemo reći i da imamo jedan vještački inteligentan sistem.

Razmišljati kao čovjek

Ocijeniti da li neki program razmišlja ljudski podrazumijeva da poznajemo neke metode koje mogu utvrditi da li čovjek postupa ljudski ili ne.

Postoje tri načina da se ovo utvrdi:

1. **introspekcija** - na ovaj način pokušavamo uhvatiti sopstvene misli kako nastaju
2. **psihološka testiranja** - posmatramo ponašanje osobe u akciji
3. **snimanje mozga** - posmatramo direktne reakcije mozga u akciji

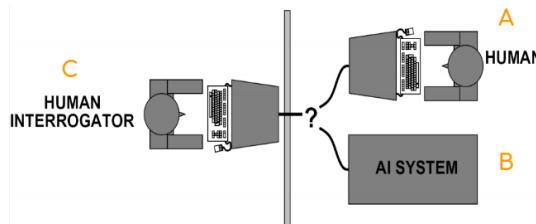
Za ovu grupu vještačke inteligencije vežemo naučno interdisciplinarno polje koje nazivamo **kognitivna nauka** koja pokušava da spoji nekoliko oblasti u svrhu razumijevanja principa rada ljudskog mozga. Sami cilj je spojiti računarske modelle iz polja vještačke inteligencije i eksperimentalne tehnike iz psihologije kako bi se definisale precizne teorije rada ljudskog mozga, te testovi koji mogu testirati razne teorije.

Ponašati se kao čovjek

Unutar ove grupe inteligencije, postavljaju se pitanja **mogu li mašine misliti i mogu li se mašine ponašati intelligentno**. Unutar ove grupe postoji operativni test za intelligentno ponašanje pod nazivom **The Imitation Game**. Pomoću ovog testa možemo da izvršimo analizu nekog intelligentnog ponašanja određenog entiteta. Test se bazira isključivo na analizi pisanog teksta gdje računar treba da nam odgovori na tekstualno postavljeno pitanje.

Test podrazumijeva sistem gdje imamo 3 sudionika:

- čovjek koji treba da ispita da li na pitanja koja postavlja odgovara čovjek ili AI sistem (na slici označen slovom **C**)
- entitet čiju inteligenciju ispitujemo (**B**)
- čovjek koji će odgovarati na pitanja s ciljem da zavara sudionika C (na slici označen sa **A**):



Slika 1.5: The Imitation Game

Prethodno navedeni test predstavlja Turingov test (*Alan Turing, 1950.*).

Turing je previdio da će do 2000. godine napraviti AI sistem za koji će postojati 30% šanse da zavara laika da je riječ o čovjeku u trajanju od 5 minuta (što se na kraju i ostvarilo). Predvidio je sve glavne argumente protiv AI u narednih 50 godina. Zatim, predložio je glavne komponente vještačke inteligencije:

- znanje
- obrazloženje
- razumijevanje jezika
- učenje

Sve ove komponente imamo danas u naprednjim sistemima.

Problem (ograničenje) ovakvog testa je što se ljudi ponekad ponašaju neinteligentno, a inteligentno ponašanje ne mora nužno biti ljudsko. Samim time, smatra se da je test značajniji za filozofiju vještačke inteligencije, nego za njen razvoj.

Da bismo napravili sistem koji može da položi Turingov test, potrebno je da on ima određene osobine. Prva osobina jeste da ima **sposobnost obrade prirodnog jezika** (engl. *natural language processing*) kako bi se uspješno ostvarila komunikacija. Pored toga, potrebno je da ima **mogućnost predstavljanja znanja** (engl. *knowledge representation*) kako bi mogao čuvati ono što zna ili čuje. Treća osobina je da ima engl. **automatizirano zaključivanje** (automated reasoning) kako bi autonomno donosio sopstvene zaključke i odluke. Ove odluke su u spremi sa znanjem koje ima i situacijom u kojoj se nalazi. Posljednja osobina je **mašinsko učenje** (engl. *machine learning*) koja se koristi za prilagođavanje novim okolnostima ili za otkrivanje i ekstrapoliranje šablonu.

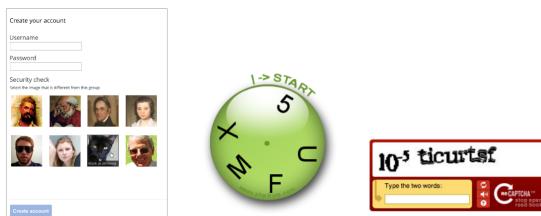
Pored navedenog testa, postoji i **Ukupni Turingov test** koji podrazumijeva uključivanje i video signalata koji omogućava čovjeku da testira perceptualne sposobnosti nekog sistema. Sam sistem mora biti u mogućnosti da prepozna određeni objekat i da izvrši manipulaciju nad njime. U praksi se ovo realizira na način da se napravi mali otvor između zida i da čovjek prosljedi određeni

objekat kroz taj otvor. Na ovaj način, sistem može uzeti objekat te manipulisati sa njim (okretati objekat i slično). Nakon toga, sistem prepoznaće o kojem objektu je riječ i daje odgovor na neko postavljeno pitanje.

Za navedeno prepoznavanje objekta, potrebno je da sistem ima dvije dodatne osobine, odnosno znanje iz oblasti **računarske vizije i robotike**.

Računarska vizija je oblast u računarstvu koja se bavi analizom vizualne scene i podataka, dok je robotika usko vezana za manipulaciju određenim objektima.

Još jedna pojava umjetničke inteligencije s kojom se susrećemo često je **CAPTCHA** (skraćeno od Completely Automated Public Turing test to tell Computers and Humans Apart). U literaturi se predstavlja kao **obrnuti Turingov test**.



Slika 1.6: Captcha

Razmišljati racionalno

Treća grupa vještacke inteligencije se tiče načina razmišljanja, odnosno, kako definisati da li neki entitet razmišlja racionalno ili ne. Temelje koje danas koristimo za ovo je postavio grčki filozof **Aristotel**.

Aristotel je prvi pokušao da definiše razmišljanje, tj. proces koji dovodi do ispravnog zaključka. Također, smatra se i kreatorom podoblasti u logici koja se naziva **silogizam**. Sillogizam (grč. zaključiti, računati, smatrati) je oblik deduktivnog obrazloženja koji se sastoji od velike premise, manje premise i zaključka. Pomoću silogizma predstavljeni su šabloni za formiranje **argumentovanih struktura** koje će dovesti do ispravnih zaključaka kada se predstave ispravne premise.

Na bazi onoga što je definirano logikom, naučnici su već 1965. godine uspješno programirali programe koji mogu rješavati određene probleme koristeći logičke notacije.

Postoje dva ograničavajuća faktora korištenja isključivo logičkih notacija.

Prvi problem je što nije jednostavno predstaviti neformalno znanje korištenjem isključivo logičke notacije. Kao primjer ovog problema imamo

činjenicu da znanje sa kojim radimo ne mora uvijek da bude 100% sigurno. Ukoliko imamo ovaj slučaj, onda ne možemo reći da je znanje koje bismo mogli opisati logičkim notacijama ujedno i formalno znanje.

Drugi problem je što postoji razlika između rješavanja problema "u principu" i rješavanja problema u praksi. Primjer ovog problema je program koji ima više stotina činjenica koje pokušavamo da isprogramiramo. Na ovaj način se vrlo brzo troše računarski resursi. Da bi se ovakav problem riješio, koriste se određene kratice (heuristike) kako bi se nepotreba dokazivanja preskočila (dokazivanja koja nisu u vezi sa problemom kojeg je potrebno riješiti).

Ponašati se racionalno

Četvrta grupa vještačke inteligencije se tiče načina ocjenjivanja ponašanja, i to kako ocijeniti da se neki sistem ponaša racionalno. U vezi sa ovom grupom se vežu određeni pojmovi koji će biti navedeni u nastavku.

Agent (lat. agere - uraditi) se može definisati kao nešto što postupa, čini, glumi i slično. **Računarski agent** se definiše kao koncept koji se bavi izgradnjom ili hardverskih ili softverskih sistema. Moraju imati mogućnost da djeluju autonomno i da se prilagođavaju okruženju u kojem se nalaze, te da slijede svoje ciljeve. Primjer ovakvog agenta je Rover koji se nalazi na Marsu. Navedni agent je udaljen od kontrole čovjeka, te se nalazi u potpuno nepoznatom okruženju gdje mora da djeluje sam. **Racionalni agent** je onaj agent koji djeluje na onaj način da postigne najbolji (očekivani) ishod.

1.2 Kratka historija vještačke inteligencije

Prvi počeci vještačke inteligencije se javljaju neposredno nakon Drugog svjetskog rata (40tih i 50tih godina). McCulloch i Pitts **1943.** godine daju model mozga baziran na logičkim kolima gdje nastaje prvi model vještačkog neurona. Inspiracija za kreiranje ovog modela je sami biološki neuron. Nakon toga, **1950.** godine, Turing objavljuje naučni rad u kojem opisuje i definiše Turingov test. **1951.** godine Minsky i Edmons kreiraju prvu neuronsku mrežu od 40 neurona koristeći vakumske cijevi.

U razdoblju od 1950 - 1970. godine, vještačka inteligencija bilježi prve uspjehe. **1952.** godine, Samuel piše prvi AI program za šah. Newell, Shaw i Simon prave program **Logic Theorist** baziran na dokazu teorema iz knjige "Principia Mathematica" **1956.** godine. Iste godine, održan je sastanak u Dartmouth-u gdje je usvojeno korištenja termina Artificial Intelligence. Dvije godine nakon sastanka, John McCarthy objavljuje programski jezik **LISP** koji se koristi za definisanje određenih pravila koja mogu odgovoriti na određena pitanja. **1959.** godine, napravljen je Gelernter-ov prvi napredni AI program **Geometry Engine**. Zatim, **1962.** godine Rosenblatt dokazuje da perceptron može konvergirati što će imati uticaja na vještačku inteligenciju u budućnosti.

Robinson **1965.** godine objavljuje algoritam za logičko rezonovanje, a nakon toga Minsky i Papert **1969.** godine objavljuju knjigu "Perceptrons" u kojoj se razrađuje matematički model Perceptron-a i dokazuju da postoji ograničenja neuronskih mreža.

Pristupi bazirani na znanju se javljaju u razdoblju 1970 - 1990. godine. Pored toga, od 1980. do 1988. godine se javlja nagla popularnost ekspertnih sistema. Međutim, u zadnjim godinama navedenog razdoblja se javlja pojam "**AI winter**". Ovaj period, odnosno pad ulaganja u oblast vještačke inteligencije, se javlja zbog objavljivanja knjige "Perceptrons" u kojoj je dokazano da postoji ograničenja neuronskih mreža.

Statistički pristupi se koriste u periodu od 1990. do 2012. godine kada je fokus stavljen na neizvjesnost. Također, dolazi do unapređenja različitih oblasti vještačke inteligencije.

U periodu od 2012. godine do danas, imamo nova otkrića i značajan napredak. Rješavaju se različiti problemi, te sistemi dolaze do tačke gdje mogu samostalno da uče. Koriste se veće količine podataka, bolje računarske arhitekture i neuronske mreže.

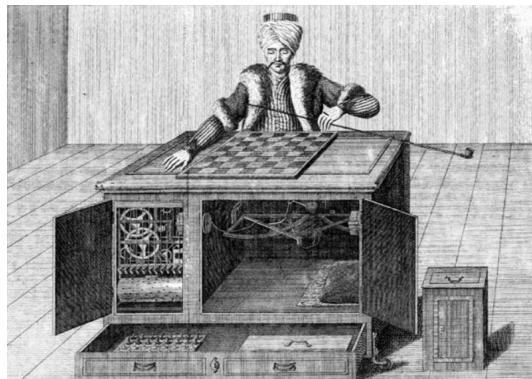
1.3 Prvi tragovi vještačke inteligencije

Prvi pisani trag o reproduciranju misli se javlja u knjizi **Frankenstein** autorice *Mary Shelley* 1918. godine. Knjiga opisuje pokušaj naučnika da stvori vještački život.



Slika 1.7: Frankenstein

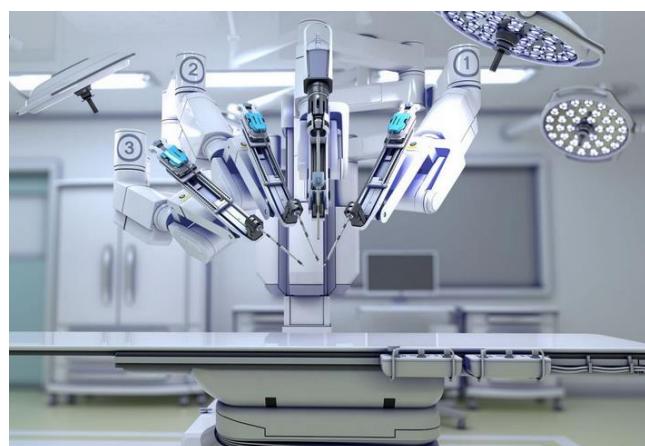
Još jedan pokušaj stvaranja vještačkog života je automaton **The Turk**. Radi se o spravi koja je stvorena sa ciljem da igra šah. Međutim, ipak je bila riječ o jednoj mehaničkoj iluziji koja je omogućavala čovjeku skrivenom ispod mašine da igra šah (ipak je uspio da zavara i pobijedi neke velike šahiste).



Slika 1.8: The Turk

Robot kao riječ se prvi put javlja 1921. godine u predstavi *Rossumovi Univerzální Roboti*. Smatra se da je riječ **robot** češkog porijekla i dolazi od riječi **robota** što znači prisilan rad.

Smatra se da je Japan najveći predstavnik koji najviše ulaže u razvoj robota. Zahvaljujući velikom razvoju u ovoj oblasti, danas imamo robote koji pomažu da doktori izvrše udaljene operacije iz jedne države na neko potpuno udaljeno mjesto na svijetu.



Slika 1.9: Mašina za robotsku hirurgiju

1.4 Aktuelnosti

U nastavku su navedeni primjeri koji su vezani za umjetničku inteligenciju i služe kao dokaz do koje mjere se ona zapravo može razviti i za šta se može koristiti:

GPT-3 bot

Jezički model znanja kreiran na osnovu velike količine podataka razvijen s ciljem prevare čovjeka. Razgovor sa ovim modelom dovodi do osjećaja komunikacije sa pravim čovjekom, a ne robotom.

Identifikacija maloljetnika sa kriminalnim dosjeom

U Buenos Aires-u je razvijen sistem, baziran na procesiranju slika, koji vrši identifikaciju i praćenje maloljetnih osoba koje su osumnjičene za neki zločin. Radi se o jako teškom problemu identifikacije zbog stalnog mijenjanja karakteristika lica kod maloljetnika.



Slika 1.10: Identifikacija maloljetnika

Deepfake

Model koji uči iz velike količine podataka i vezan je za neuronske mreže (engl. *deep learning*). Daje lažni osjećaj ljudima pomoći kojeg može da mijenja stanje u društvu (dovodi do socijalne dileme u kojoj teško možemo odlučiti šta je istina, a šta je laž).

Postoje dva važna koncepta deepfake modela (koji se mogu integrirati u jedan):

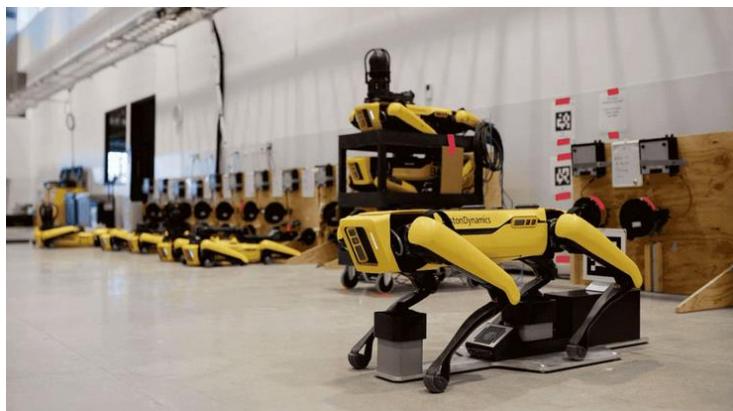
Video deepfake - mijenjanje lica osobe za lice neke druge osobe (do rezultata

se dolazi analizom pokreta i gestura određene osobe)

Audio deepfake - mijenjanje glasa osobe sa glasom neke druge osobe (do rezultata se dolazi procesiranjem zvučnih datoteka i ekstraktovanjem znanja na novi zvučni signal)

Spot Enterprise

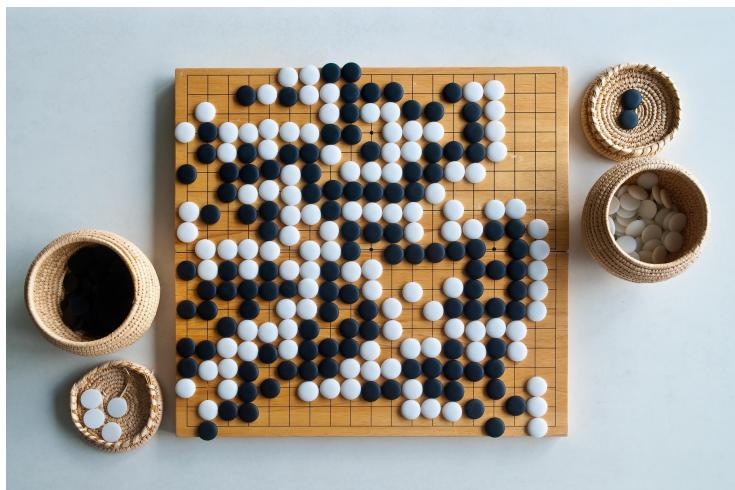
Razvoj tehnologije dovodi do modela robota koji može da zamijeni ljudi u određenim poslovima inspekcije, analize i sličnih poslova pomoću kamere, te da procesira dobijene informacije u stvarnom vremenu. Bitna funkcionalnost ovog robota jeste mijenjanje određenih rizičnih poslova čime se ne ugrožava čovjek. Loša strana razvijanja ovakvih modela jeste što veliki broj ljudi ostaje bez posla. Pored toga, potrebno je za svaki posao postaviti različit model, dok jedan čovjek može da riješi niz problema.



Slika 1.11: Spot Enterprise

AlphaGo

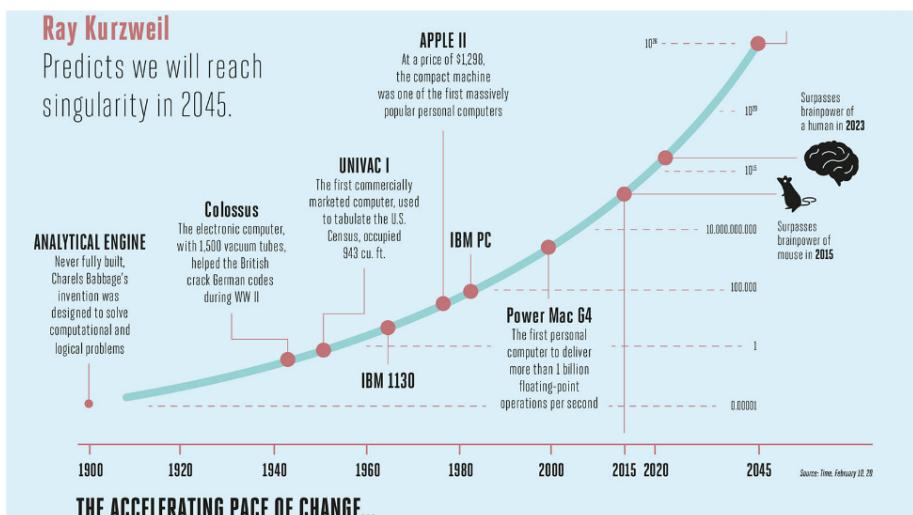
Prvi računarski program koji je porazio čovjeka u igri Go. Smatra se da je program najjači igrač ove igrice u historiji. Model je baziran na pojačanom učenju (engl. *reinforcement learning*), odnosno, kroz pokušaj i grešku nastoji da dostigne najbolji mogući rezultat. Broj mogućih koraka u igrici daleko premašuje broj atoma u posmatranom svemiru čime shvatamo kompleksnost razvoja algoritma ovog programa.



Slika 1.12: Go

1.5 Futurističke predikcije

U nastavku će biti opisana ideja koju je predstavio futurist Ray Kurzweil. Napravio je određena predviđanja u smislu dostizanja računarskih sposobnosti u odnosu na mozak. Na prikazanom grafu, x-osa predstavlja godine, dok je y-osa predviđeni mogući broj operacija koje se mogu izvršiti u jednoj sekundi. Po njegovim pretpostavkama, do 2045. godine ćemo imati sistem koji će moći u jednoj sekundi da izvrši 10^{25} operacija!



Slika 1.13: Futurističke predikcije

1.6 Podoblasti vještačke inteligencije

Ako govorimo o bazičnog podjeli vještačke inteligencije, najzastupljenije podoblasti su

- mašinsko učenje (ML)
 - supervizirano
 - nesupervizirano
 - duboko učenje
- natural language processing (NLP)
- ekspertni sistemi
- vizija

- prepoznavanje objekata na slici
- računarska vizija
- govor
 - govor u tekst
 - tekst u govor
- planiranje
- robotika



Slika 1.14: Podoblasti vještačke inteligencije

Ovo je samo jedna podjela vještačke inteligencije jer njih ima dosta više. Na ovoj podjeli ne postoji fuzzy sistemi i intelligentni agenti.

Predavanje 3

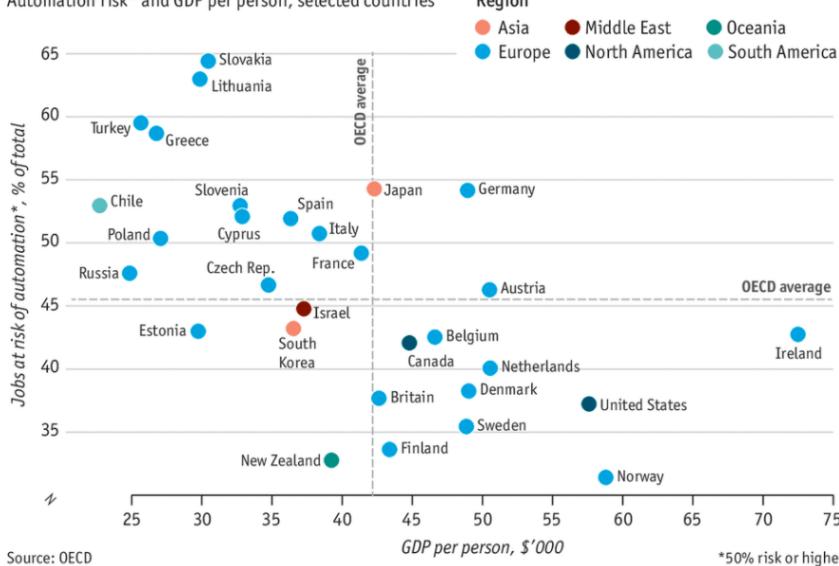
2.1 Uticaj vještačke inteligencije na poslove

Sa navedenim aktuelnostima u oblasti vještačke inteligencije, postavlja se pitanje **da li će vještačka inteligencija ugrožavati radna mjesta**. Radi se o studiji koja je objavljena u magazinu **The Economist**. Ova studija kaže da je skoro polovina poslova ranjiva na proces automatizacije, odnosno na algoritme, agente i robote koji će mijenjati poslove u industriji (bez obzira na to o kojoj industriji je riječ). Pored ovoga, studija navodi da će se upravo zbog ovoga ljudi potaći da se bave zanimljivijim karijerama.

Prikazan je graf koji je nastao na osnovu izvora podataka **Organisation for Economic Co-Operation and Development** (OECD). Graf daje pregled BDP (bruto domaći proizvod) po osobi i riziku u procentima od procesa automatizacije.

Wage against the machine

Automation risk* and GDP per person, selected countries



Source: OECD

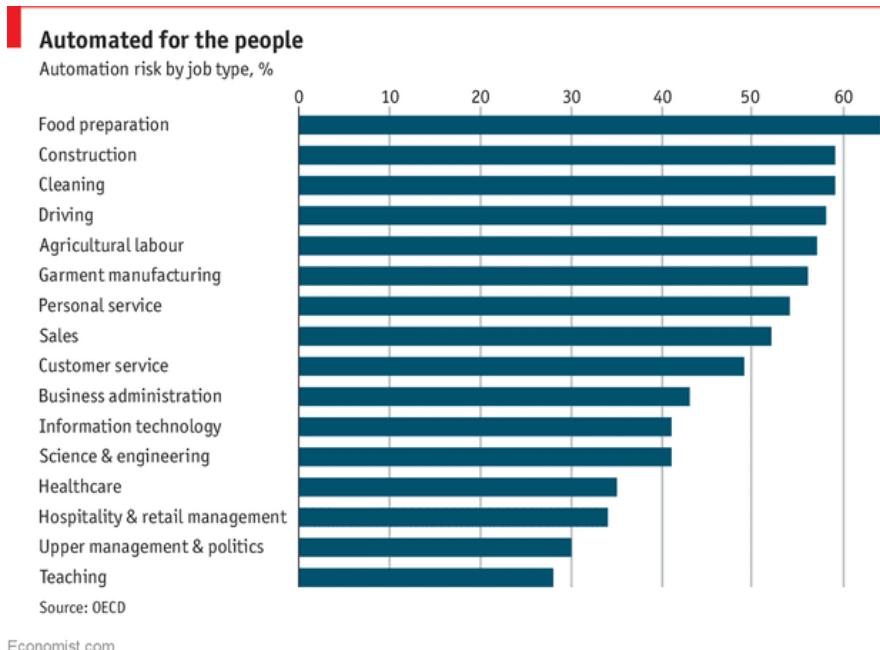
*50% risk or higher

Slika 2.1: Izvor podataka OECD

Uspješne države su ujedno i bogatije, te samim time imaju veći BDP po osobi. Pored toga, poželjno je da ta ista država ima manji rizik od procesa automatizacije. Ukoliko neka država zadovoljava ove dvije navedene stavke, ona ujedno predstavlja i **idealnu državu** po navedenom pregledu podataka. Ove države se nalaze u donjoj desnoj četvrtini datog grafra.

Sami graf je povezan sa vrstama poslova unutar određene države. Naprimjer po ovom izvoru podataka, Njemačka ima puno više poslova koji se mogu automatizirati nego što to ima Švedska. S druge strane, Irska ima jako visok BDP, ali ima i veći broj poslova koji se mogu automatizirati nego posmatrajući neku državu kao što je Norveška.

Na istoj studiji se nalaze druge informacije koje govore o vrstama poslova koji se mogu automatizirati. U 2013. godini, Carl Benedikt Frey i Michael Osborne sa Univerziteta Oxford koristili su algoritam mašinskog učenja kako bi procijenili koliko lagano se može automatizirati 700 različitih poslova u Americi. Uspjeli su zaključiti da bi mašine u potpunosti mogle izvršavati 47% poslova u narednih deset ili dvadeset godina. Na sljedećem grafu su prikazani poslovi sa najvećim rizikom, gdje su na vrhu grafa poslovi pripremanja hrane, dok su na dnu prosvjetni radnici.



Economist.com

Slika 2.2: Rizik od automatizacije

Iste godine, OECD je dala izjavu da ova studija nije toliko tačna koliko je to bilo prezentirano. Pojavili su se oštri komentari i kritike, zbog čega je napravljena revizija ove studije gdje je navedeno da poslovi u Americi nisu pod toliko velikim rizikom. Krajnji zaključak govori da bi mašine mogle izvršavati oko 10% poslova u Americi, a 12% u Velikoj Britaniji. Jasno se vidi razlika između prvih navedenih podataka i ispravljenih iz razloga što se radi o predikciji.

Kada se pričalo o aktuelnostima vještačke inteligencije, spomenuo se GPT-3 model. Radi se o softveru koji može da razumije semantiku jezika. Postavlja se pitanje da li ovaj softver može da zamijeni industriju softver developera.

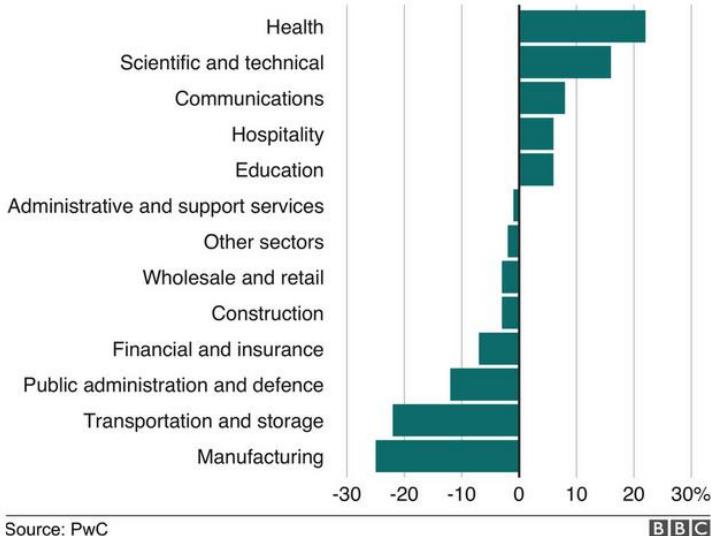
Ovdje je potrebno spomenuti još jednu studiju koju je objavio BBC. Bavi se sličnim pitanjem i daje odgovor koji kaže sljedeće:

"Vjerovatno će četvrta industrijska revolucija favorizirati one s jakim digitalnim vještinama, kao i sposobnostima poput kreativnosti i timskog rada koje je mašinama teže replicirati."

Sama poenta jeste da programeri nemaju posao da samo pišu ili kopiraju kod, već je potrebno da imaju i sposobnost za fizički rad, zbog čega je vjerovatnoća automatiziranja ovog posla jako mala.

Pored toga, ova studija daje jedan optimistični pogled na sami problem i kaže da će vještačka inteligencija kreirati određeni broj poslova. Na sljedećoj slici su prikazani poslovi koji će imati određene beneficije od metoda vještačke inteligencije i suprotni poslovi. Ovdje se radi o još jednoj estimaciji (predikciji) kreiranja poslova u periodu od 2017 do 2037 godine.

How AI could change the job market
Estimated net job creation by industry sector, 2017-2037



Slika 2.3: Vještačka inteligencija na tržištu

2.2 Klasifikatori

Klasifikator je bilo koji algoritam koji ima mogućnost sortiranja podataka u predefinisane klase ili kategorije informacija. Kada imamo jako veliku količinu podataka, potrebno je iste klasificirati u određeni broj klasa. Pored ovog pojma, postoji i pojam **klasterizacija** koja opisuje klastera podataka. Osnovna razlika između ova dva pojma je da kod *klasterizacije* grupišemo podatke u klastera, ali ne znamo unaprijed koji je to broj klastera niti znamo njihove specifikacije. Naprimjer, ako uzmemos sortiranje slika na bijele i crne, kod *klasifikacije* znamo da postoje dvije klase i znamo ih vrlo jednostavno klasificirati. S druge strane, kod klasterizacije ne znamo da postoje dvije klase i ne znamo da se radi o crnoj i o bijeloj klasi.

Klasifikator je baziran na konceptu učenja, odnosno, radi se o algoritmu koji

ima sposobnost da uči iz podataka. Da bi imao ovu mogućnost, moramo označiti podatke u klase odakle dolazi pojam **označenih klasa**. Da bismo imali dobar klasifikator, moramo imati veliku količinu podataka.

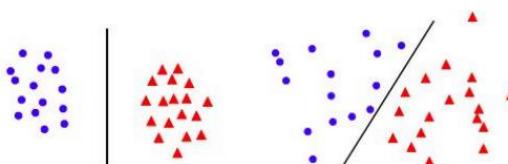
Postoji više vrsta klasifikatora, ali možemo ih podijeliti na dvije osnovne vrste, a to su **binarni** (dvije klase) i **višeklasni** (više od dvije klase).

Najosnovniji primjer pojma klasifikatora je **dizajn filtera** za sortiranje pošte na onu koja je poželjna i onu koja nije (spam pošta). Ovaj primjer ujedno predstavlja i binarnu klasifikaciju. Pored toga, imamo i primjer klasifikatora koji vrši **predikciju** gdje estimiramo nešto što će se desiti u budućnosti. Na osnovu navedenog možemo izvršiti predikciju da li osoba ima neko oboljenje na osnovu predočenih nalaza. Još jedan primjer ove vrste klasifikatora jeste predikcija da li je neka finansijska transakcija lažna ili ne (engl. *fraud detection*). Radi se o jako izazovnom problemu jer u praksi imamo dosta više ispravnih, nego neispravnih transakcija. Kada imamo ovakav debalans podataka, javlja se težak problem definisanja klasifikatora. Međutim, uz pomoć raznih metoda, moguće je napraviti sistem koji i pored ovog problema daje jako korisne informacije.

Generalno možemo reći da su klasifikatori konkretna implementacija prepoznavanja obrazaca (engl. *pattern recognition*) u mnogim oblicima mašinskog učenja.

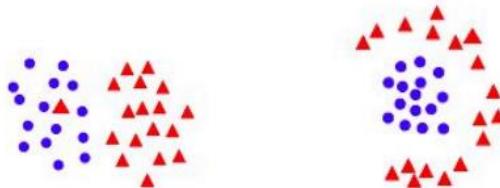
Jasno je da postoje različite vrste klasifikatora. Najjednostavniji oblik je **linearни klasifikator** koji se bavi linearnim kombinacijama različitih karakteristika podataka. Cilj ovakve vrste klasifikatora je da izvrši prepoznavanje određenih šablonu u podacima. Nakon što se oni identifikuju, vršimo linearnu kombinaciju tih karakteristika kako bismo dobili jedan robustan klasifikator.

Posmatrajući sljedeću skupinu podataka, jasno je da postoje dvije klase koje su dobro podijeljenje. Upravo zbog ovoga, lahko je definisati jedan linearni klasifikator. Linearni klasifikator može imati različite oblike u prostoru, gdje je potrebno naći onaj najbolji oblik (najčešće se traži minimizacijom greške).



Slika 2.4: Primjer linearnog klasifikatora

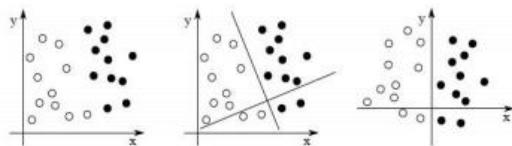
Pored toga, postoje i nešto teža vrsta klasifikatora gdje podaci nisu egzaktno odvojeni - **nelinearni klasifikatori**. Kao primjer ove vrste klasifikatora, uzmimo sljedeći skup podataka koji nije pogodan za rješavanje linearnim klasifikatorima.



Slika 2.5: Primjer nelinearnog klasifikatora

Postoje razne kategorije linearnih klasifikatora. Jedan od načina da izvršimo efikasnu linearnu klasifikaciju podataka je kroz proces transformacije podataka.

Recimo da imamo dva skupa podataka koje je moguće klasificirati pomoću jednostavnog linearog klasifikatora. Međutim, pravila kojim se može definisati taj linearni klasifikator su relativno kompleksna. Kako bismo pojednostavili ta pravila, možemo izvršiti transformaciju podataka, odnosno, translaciju koordinata. Navedeno je prikazano na sljedećem primjeru:



Slika 2.6: Primjer linearog klasifikatora

Na ovom primjeru, y-osa ujedno predstavlja i traženi linearni klasifikator zbog čega je pravilo za razdvajanje ovih podataka dosta efikasno i lagano za implementirati.

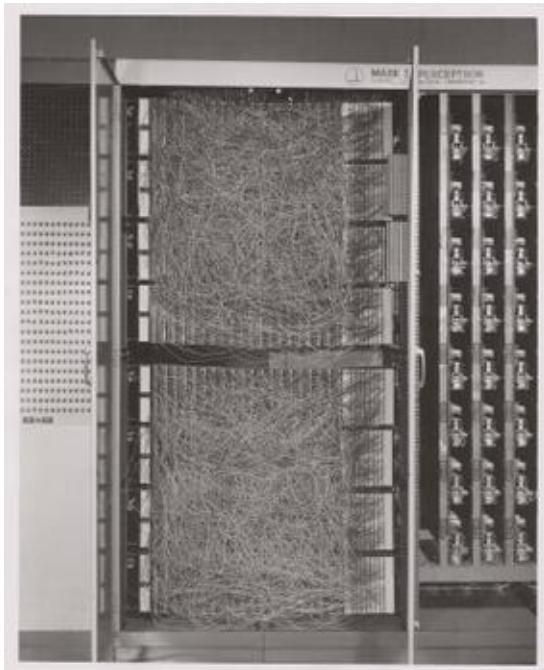
Primjeri klasifikatora iz prakse su navedeni u nastavku:

- perceptron
- naivni bayes
- stablo odlučivanja
- logistička regresija
- K-nearest neighbor

- vještačke neuronske mreže (duboko učenje)
- support vector machine (SVM)
- ...

2.3 Historijski pregled neuronskih mreža

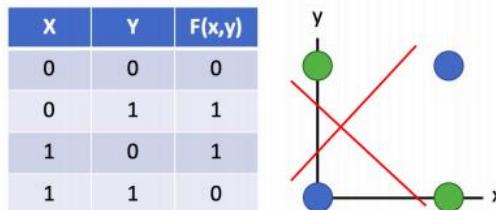
Perceptron je jedan od prvih algoritama koji je imao mogućnost učenja iz podataka. Bio je implementiran 1958. godine u hardveru pomoću **Mark I Perceptron**. Osmislio ga je Frank Rosenblatt nakon Drugog svjetskog rata. Nastao je na bazi analize rada ljudskog mozga. Implementiran je na način da se iskoristi model neurona koji se dalje veže u mrežu neurona kako bi se došlo do određenih rezultata. Iako se to tada nije tako nazivalo, kasnije je uviđeno da je sama implementacija perceptrona linearni klasifikator. Svaki neuron ima određeni koeficijent koji se množi sa određenim parametrima, te se ti rezultati sabiraju i prosljeđuju dalje da se na kraju donose određena odluka. Ove vrijednosti koeficijenata su bile čuvane u potenciometrima, a električni motori su korišteni za njihovo ažuriranje. Analizirali su slike dimenzija 20x20 koristeći kamere koje su koristile fotoćelije kadmium-sulfida.



Slika 2.7: Mark I Perceptron

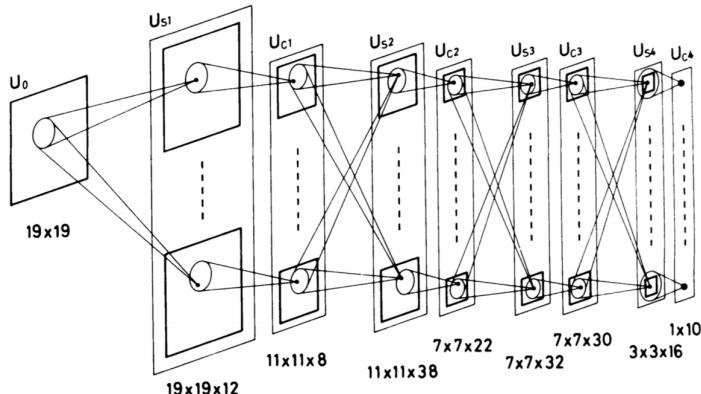
Perceptron je snimao slike, te je operator određivao da li se radi o muškarcu ili ženi. Pored toga, označavao je podatke sa tačno ili netačno.

1969. godine je izšla knjiga sa naslovom **Perceptrons: an introduction to computational geometry** dva autora Marvin Minsky i Seymour Papert. Pokušali su da daju matematički model rada perceptrona, te dali kritiku rada perceptrona u smislu ograničenja. Dokazali su da perceptron nije u stanju da izvrši klasifikaciju XOR kola. Knjiga je izazvala velika povlačenja novca u polju vještacke inteligencije.



Slika 2.8: Klasifikacija XOR kola

Nakon toga, 1980. godine japanski naučnik Kunihiko Fukushima objavljuje jedan naučni rad na temu neuronskih mreža u kojem je predstavljen sistem **Neocognitron**. Ovaj sistem predstavlja organizaciju neurona u jednu kompleksniju strukturu koja je bazirana na hijerarhijskoj organizaciji neurona. Neocognitron je nastao na osnovu istraživanja mozga, a pogotovo vizuelnog korteksa koji se nalazi na potiljku mozga. Oni su zapravo izučavali vizuelni korteks kod mačaka, te su na osnovu ovih istraživanja napravili određene zaključke. Nakon toga su pokušali da prošire perceptron na način da povećaju broj nivoa (slojeva). Kao rezultat navedenih istraživanja nastaje **Neocognitron**. On zapravo predstavlja novi algoritam za prepoznavanje šablonu (engl. *pattern recognition*) baziran na modelu vizuelnog sistema.



Slika 2.9: Neocognitron

Na ulazu u model imamo neku sliku (u ovom slučaju sa dimenzijama 19×19), a zatim imamo neke hijerarhijske nivoe koji su kompeksni na početku i jednostavniji na kraju. Na zadnjem sloju imamo jedan vektor (dimenzija 1×10), što znači da je Neocognitron imao mogućnost da razvrsta podatke u 10 klasa. Klasa kojoj slika pripada zavisi od aktivacije vektora unutar zadnjeg nivoa. U ovom slučaju, postoje dvije mogućnosti. Prva mogućnost je da smo imali raspodjelu različitih vrijednosti vektora, ili da smo imali samo jedan aktivan vektor koji bi pokazivao o kojoj klasi je riječ.

Neocognitron je baziran na modelu vizuelnog sistema i predstavlja prvu višeslojnu neuronsku mrežu. Razlog lošeg ocjenjivanja perceptronu je bila jako mala kompleksnost mreže, zbog čega su kasnije povećali broj slojeva unutar modela. Radi se o samoorganizirajućoj mreži koja je nastala učenjem bez nadzora. Dakle, sistem je sam učio iako nije imao označene podatke.

Osnovne karakteristike modela su da je hijerarhija zasnovana na jednostavnim (**konvolucija**) i složenim (**pooling**) ćelijama.

Na samoj slici se vidi da imamo određene operacije. Prva operacija je predstavljena krugom na prvom sloju i predstavlja konvoluciju. Ova operacija uzima dio slike, te vrši proces konvolucije nad tim datim dijelom. Sami rezultat se upisuje u narednu fazu prezentovanu u drugom sloju. Postoje različiti filteri s kojima se radi konvolucija te se dobijaju razni rezultati. Pored ove, postoji i operacija **pooling** koja treba da smanji količinu podataka koji će se koristiti u sljedećem sloju.

Povećanjem broja slojeva, smanjuje se osnovna informacija od koje smo krenuli, a povećava se broj filtera sve dok ne dođemo do posljednjeg sloja.

Spomenuta mreža je bitna iz razloga što je 2012. godine objavljena slična neuronska mreža (**AlexNet**). Značajno je spomenuti činjenicu da su bile potrebne 32 godine da se napravi funkcionalniji sistem koji ima slične karakteristike kao Neocognitron.

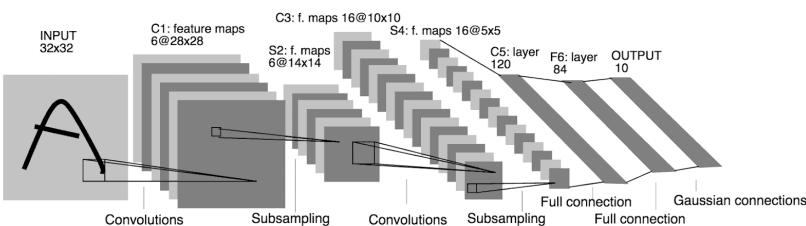
Nakon Neocognitrona, objavljen je novi algoritam **Backprop** 1986. godine. Ovaj algoritam predstavlja osnovu neuronskih mreža. Naime, Neocogniton nije imao mogućnost propagiranja eventualne greške jer nismo imali označene podatke. Međutim, ukoliko imamo proces učenja i označene podatke, možemo znati da li je došlo do greške ili ne. Na ovaj način možemo izračunati kolika je ta greška i propagirati tu informaciju na početak same mreže. Sa ovom informacijom ponovo iterativno prolazimo kroz tu mrežu sve dok ona ne smanji tu grešku.

Upravo iz ovog razloga, naučnici **Rumelhart, Hinton i Williams** su objavili naučni rad u kojem su opisali algoritam **backpropagacije**. Na ovaj način, omogućeno je treniranje višeslojnih perceptronova. Ovaj proces istraživanja je doveo do razvijenih neuronskih vještackih mreža koje uspješno možemo istrenirati nad bilo kojim podacima. Osnova ovog algoritma je računanje gradijenta u neuronskim mrežama.

Nakon backpropagacije, imamo nastupanje **AI winter**.

Dvanaest godina nakon ovog perioda, naučnik **Yann LeCun** je objavio neuronsku mrežu koja je bazirana na operaciji konvolucije koja se iz tog razloga naziva **Konvolucionna neuronska mreža** (ConvNet). LeCun-ova mreža je prva mreža koja je uspješno implementirala backpropagaciju u jednu višeslojnu neuronsku mrežu.

Model mreže je prikazan sljedećom slikom i predstavlja sistem koji je u stvarnosti bio realizovan u japanskoj firmi NEC za prepoznavanje pisanih slova alfabet-a.



Slika 2.10: Arhitektura konvolucionne neuronske mreže

Na samom ulazu se nalazi slika dimenzija 32x32, nakon čega se nalazi niz slojeva pri čemu je karakterističan poslednji sloj gdje se nalazi niz neurona

koje organizujemo u određene strukture i gdje je svaki ulaz povezan sa svakim skrivenim neuronom.

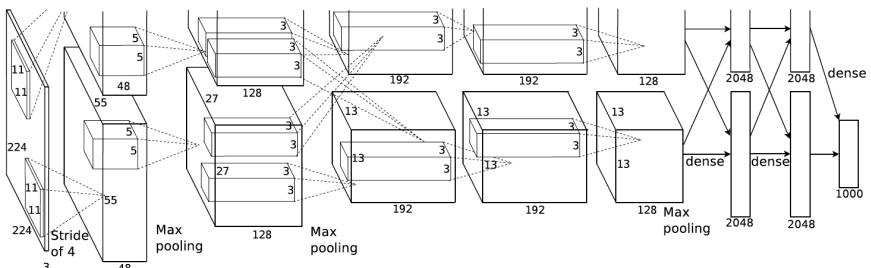
Početkom 2000. godine dolazi era dubokog učenja. Javljuju se različiti naučni radovi koji su najavili dolazak nove oblasti. Osnovna ideja je bilo trenirati sve dublje i dublje mreže (sa više slojeva).

Čitav proces u vezi uspješnosti dubokog učenja je baziran na uspješnosti kreiranja različitih baza podataka. Jedna od takvih baza podataka je **ImageNet** čiju je inicijativu za kreiranje pokrenula naučnica sa Stanford-a. Počelo se sa skupljanjem različitih javnih slika, te je izvršena klasifikacija tih slika u hiljadu objekata (kao što su automobil, miš, šešir i slično). Na početku su imali nekih 1 431 167 slika, dok je danas aktuelno oko 15 000 000 slika. Slike se mogu koristiti za treniranje kompleksnije neuronske mreže, a rezultate možemo koristiti za različite zadatke.

Na izlazu se dobije vjerovatnoća da određena slika pripada jednoj od 1000 klasa. Obično se sortiraju vrijednosti i posmatra se pet najvećih vjerovatnoća. Ukoliko je traženi rezultat jedan od pet dobijenih kategorija, smatra se da je neuronska mreža dala dobre rezultate.

Novi model **AlexNet** se javlja 2012. godine u radu **ImageNet Classification with Deep Convolutional Neural Networks**.

Na ulazu se nalazi slika koja je definisana sa tri kanala, i tu se nalaze različiti filteri. Nakon toga, nalaze se pooling i konvolucijski slojevi. Na kraju se nalaze fully connected slojevi i izlaz iz neuronske mreže dimenzija 1x1000. Ovaj model daje jako dobre rezultate pri razvrstavanju slika u ImageNet bazi podataka. Osnova AlexNet-a su konvolucijske neuronske mreže (**ConvNets**).

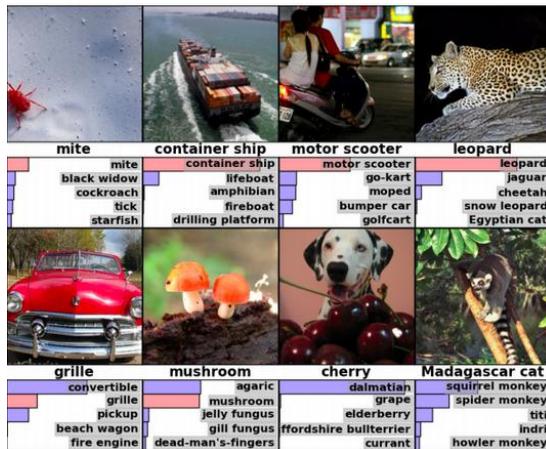


Slika 2.11: AlexNet

2.4 Primjene konvolucionih neuronskih mreža

Image Classification

Kod ove klasifikacije, imamo niz nekih slika za koje model daje rezultate prepoznavanja šta se nalazi na njima. Izlaz ovog modela je jedan vektor dimenzija 1x1000 gdje u svakom elementu imamo određenu procenat koji predstavlja vjerovatnoću da se na određenoj slici nalazi klasa.



Slika 2.12: Klasifikacija slika

Image Retrieval

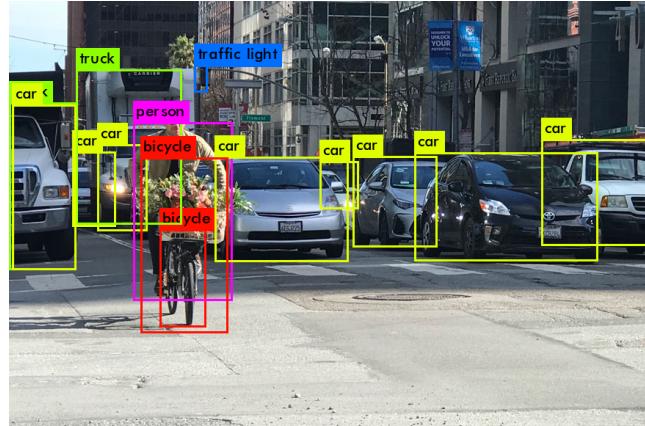
U ovoj primjeni konvolucionih neuronskih mreža, pretražuje se skup podataka sa ciljem da se pronađu slike sličnih karakteristika kao neka proslijedena slika.



Slika 2.13: Pronalaženje slika

Object Detection

Radi se o zadatku računarske vizije koji prepoznaće objekte od interesa i obilježava objekte bounding box-om (kvadrat, pravougaonik koji označava određeni objekat).



Slika 2.14: Detekcija slika

Pored detekcije objekata, pomoći neuronskih mreža možemo izvršiti i dodjeljivanje naslova slikama. Radi se o opisnom tekstu kojeg možemo vezati za sliku. Na prvom mjestu je potrebno izvršiti detekciju objekata, a zatim imamo dodatne algoritme bazirane na rekurentnim neuronskim mrežama koji mogu da generišu određeni tekst. Posmatrajući sljedeću sliku, dobar algoritam bi mogao da ispiše tekst "Osoba vozi biciklo".



Slika 2.15: Titovanje slika

Image Segmentation

Rezultat ovog primjera neuronske mreže jeste da se slika podijeli u određene

klase. Na ovaj način, mogu se prepoznati objekti od interesa i označiti određenom bojom.



Slika 2.16: Segmentacija slike

Pose Recognition (DeepPose)

Zadatak ove klasifikacije jeste da se napravi estimacija poze čovjeka na osnovu neke proslijedene slike, te da nacrti položaj tijela.



Slika 2.17: Estimacija poza

Algoritmi u igricama

Interesantan primjer primjene neuronske mreže jeste za polje igrica. Ovakva neuronska mreža se može koristiti u već spomenutoj igrići **Go** gdje je cilj

neuronske mreže da dâ sljedeći potez u odnosu na prethodne korake koje smo imali.



Slika 2.18: Go

2.5 Koncept klasifikacije slika

Klasifikacija slika za računar predstavlja složen zadatak prvenstveno zbog toga što računar slike vidi u drugačijem obliku jer nema jedinstveni vizuelni sistem. Cilj klasifikacije je da se napravi algoritam koji će uzimati određenu sliku, te na izlazu dati određenu vjerovatnoću estimacije objekta sa slike.

Međutim, računaru je slika na ulazu predstavljena nizom brojeva. Obično imamo određeni broj kanala koji predstavljaju određeni broj boja. Većinom se koristi **RGB prostor boje** predstavljen crvenim, zelenim i plavim kanalom. **Piksel** (engl. *Pixel*) predstavlja osnovnu informaciju na samoj slici i svaki piksel se može kodirati sa 8 bita. To znači da možemo predstaviti 256 boja, odnosno imamo vrijednosti od 0 do 255 (0 - crna, 255 - bijela). Svaki navedeni kanal predstavljamo matricom dimenzija same slike, i na svakoj poziciji unutar te matrice sa nalazi neki broj od 0 do 255 koji predstavlja prisutnost određenog kanala na slici.

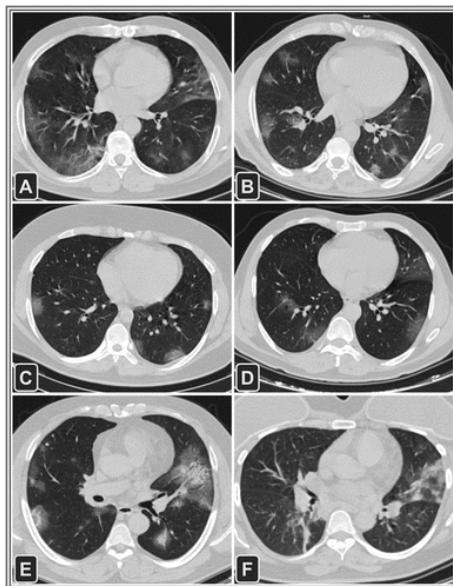
Postoje razni izazovi u klasifikaciji slika. Neki od njih su:

- Varijacija unutar jedne klase
- Varijacije među klasama/kategorijama
- Okluzija (zaklonjeni objekti)
- Različito osvjetljenje
- Različite pozadine
- Različite pozicije (deformacije)

Dobar klasifikator mora biti robustan na različite promjene koje se mogu desiti među slikama. Sa aspektom označavanja, neuronskoj mreži možemo obezbijediti dovoljno slika da ista može da nauči razne oblike klasifikacije.

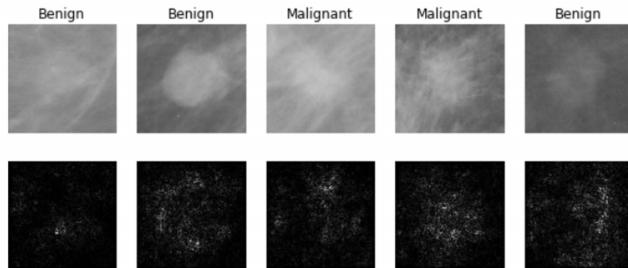
Ukoliko bismo pravili linearni klasifikator isključivo na analizi samih brojeva unutar slike, radilo bi se o puno težem zadatku.

Postoje različite primjene korištenja konvolucionih neuronskih mreža. Jedna od njih je predviđanje upale pluća na osnovu samih slika. Prikazan je izgled pluća pacijenata nastalim CT-om. Stručnjak može lagano prepoznati da li postoji upala pluća ili ne, ali danas smo u mogućnosti i koristiti neuronsku mrežu koja će pomoći stručnjaku da izvrši identifikaciju.



Slika 2.19: Predviđanje upale pluća kod COVID-19 pacijenta

Druga primjena je analiza medicinskih slika koja se koristi pri analizi tumora ili tkiva koji imaju određeni stepen degradacije. Uz dovoljnu količinu slika, možemo koristiti neuronske mreže za označavanje i prepoznavanje ovih dijelova tkiva.



Slika 2.20: Analiza medicinskih slika

Pored toga, možemo koristiti neuronske mreže za klasifikaciju različitih galaksija gdje možemo poslati određeni broj slika i označiti tipične izglede galaksija.



Slika 2.21: Klasifikacija galaksija

Kod klasifikatora slika, cilj je napraviti algoritam koji treba da generiše određeni izlaz, odnosno, da dodijeli neku predefinisanu kategoriju za ulaz. Ovaj algoritam može da radi na različite načine. Iz originalne slike se mogu izvući neke konkretnе informacije. Prvi osnovni algoritam jeste **detekcija ivica** na samoj slici. Primjer ovakvog algoritma je **Canny Edge Detector** ili **Sobelov operator** ili niz drugih operatora. Rezultat detekcije ivica je dat sljedećim primjerom:



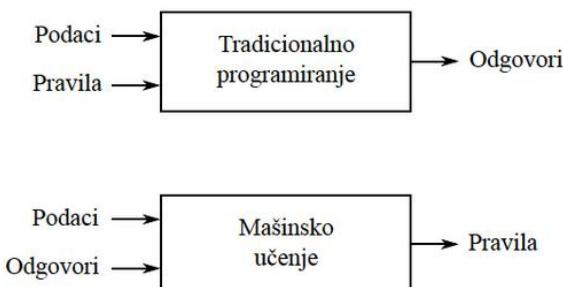
Slika 2.22: Detekcija ivica

Kako bismo mogli koristiti ove informacije za klasifikaciju slike, potrebno je pronaći određene karakteristike na dатој slici. Na osnovу тих карактеристика, правимо неки алгоритам који може да изврши класификацију. Међутим, ствар код овог приступа јесте да постоји могућност изузетака што може спријечити eksplicitno kodiranje карактеристика које користимо за формирање алгоритма.

Rješenje za prethodno navedeni problem daje mašinsko učenje. Radi se o pristupu vođenim podacima koji zahtijeva jako puno podataka (engl. *data-driven approach*). Ideja je da se umjesto ručno kodiranih карактеристика објектa koristi приступ уčenja iz samih података.

Kod tradicionalnog програмирања, на самом улазу имамо податке уз које moramo da poznajemo pravila (neke matematičke relacije i slično). Kao programeri, uzimamo pravila i програмирајмо алгоритам kako бисмо добили одređeni odgovor.

S друге стране, код машинаског учења имамо податке и predefinisane odgovore на улазу. Cilj mašinskog učenja je da se izvrši ekstrakcija znanja, odnosno identifikacija pravila koja će se kasnije koristiti. Na ovaj način, dobijamo određene odluke o novim podacima koje model nije video. Dakle, postoji mogućnost učenja nad подацима који су označени.



Slika 2.23: Razlika programiranja i mašinskog učenja

Uobičajni koraci mašinskog učenja su:

- nađi ili skupi puno podataka i svakom podatku u skupu pridruži labelu
- koristi jednu od metoda mašinskog učenja da istreniraš klasifikator
- napravi evaluaciju klasifikatora nad nekim skupom testnih podataka

Rezultati testova nad podacima mogu imati tačnosti od 1% do 100%. U praksi, dobar algoritam ima tačnost $\geq 99\%$, dok je loš algoritam do nekih 60%. Sa dubokim neuronskim mrežama, možemo da imamo tačnost u rasponu od 80% do 90%. Ako imamo visoku tačnost, metod se smatra uspješnim.

Mašinsko učenje je zapravo nova paradigma programiranja gdje se računar programira preko podataka koje dovodimo na ulaz programa, a ne preko eksplisitnih pravila. Dakle, može se reći da program mijenjamo dovođenjem novih podataka. Ova paradigma je posebno važna za probleme za koje ne znamo kako ih riješiti ili nemamo definisana pravila.

Metode mašinskog učenja se baziraju na dvije osnovne karakteristike, a to je **proces treniranja** i **proces predikcije** (testiranje, evaluacija). Za realizaciju je potrebno definisati neku funkciju koja može da trenira i uči iz podataka i za rezultat daje model. Zatim, potrebno je i definisati funkciju predikcije koja će kao rezultat vratiti odgovarajuću labelu za prosljeđeni podatak. Kod procesa učenja iz podataka, bitno nam je da imamo dobre rezultate na skupu za testiranje. Performanse se vrlo često mijere u odnosu na tačnost i na brzinu izvršavanja. Tačnost se obično određuje u odnosu na **Top 1 tačnost** ili **Top 5 tačnost**, odnosno, da li model može efikasno da pronađe klasu posmatrajući prvi rezultat ili prvih pet rezultata.

Danas postoji više različitih izvora podataka (engl. *datasets*). Jedan od njih je **CIFAR10**. Radi se o jako malim slikama različitih objekata gdje postoji deset klasa u koje pokušavamo razvrstati određenu sliku. Za svaku klasu, postoji 5 000 RGB slika (dimenzija 32x32) za treniranje i po 1 000 RGB slika za testiranje.



Slika 2.24: CIFAR10

Prije ovog dataset-a, pojavio se **MNIST** koji je danas dosta prevaziđen iz razloga što danas imamo dosta dobre rezultate (nema potrebe da se ovaj dataset dalje trenira ili izučava). MNIST ima podatke o ciframa (ima 10 klasa) gdje je svaka cifra predstavljena sa slikom sive nijanse dimenzija 28x28. Ukupno postoji 60 000 slika, gdje 50 000 uzimamo za treniranje, a 10 000 za testiranje. Na samom početku su se koristili linearni klasifikatori koji su imali grešku od 12%. Nakon toga, prelazilo se na razne oblike klasifikatora čime se greška smanjivala.



Slika 2.25: MNIST

Nakon CIFAR10, javlja se i **CIFAR100**. Sličan je prethodnom skupu podataka, osim što sada postoji 100 klasa koje sadrže po 600 slika. Postoji

500 slika za trening i 100 slika za testiranje po klasi. Još jedna razlika od skupa podataka CIFAR10 jeste što su kod CIFAR100 klase grupirane u 20 super klase. Dakle, svaka slika dolazi sa oznakom *fine* (klasa kojoj pripada) i *coarse* (superklasa kojoj pripada).

Još jedan niz podataka je već spomenuti **ImageNet** sa 1000 klasa i oko 14 miliona slika u skupu podataka. Slike su različite veličine, ali se najčešće smanjuju na dimenzije 256x256 u procesu treniranja. Performanse se mijere u odnosu na **top 5 accuracy** gdje algoritam vrši predikciju o 5 klasa za svaku sliku, s tim što jedna od njih mora biti tačna. Pored slika za treniranje i testiranje, kod ovog dataset-a se javlja i pojам slika za **validaciju**. Ova skupina podataka kroz vjerovatnoću pomaže da izaberemo bolje hiperparametre.

MIT Places predstavlja još jedan izvor podataka. Unutar ovog dataset-a, nalazi se 365 klase različitih vrsta scena i oko 8 miliona slika za treniranje. Kao i kod ImageNet-a, slike su različitih dimenzija, ali se najčešće smanjuju na dimenzije 256x256 u procesu treniranja.



Slika 2.26: MIT Places

U nastavku će se predstaviti osnovni klasifikatori. Prvi od njih je **Nearest Neighbor** (najblizi susjed). On se obično sastoji od dvije metode: **treniranje** i **predikcija** (testiranje). Unutar prve faze se pamte podaci i labele koje su asocijirane sa tim podacima. U drugoj fazi se radi predikcija gdje se uzimaju neki novi podaci koje nismo imali u procesu treniranja. Na osnovu najsljčnijih slika koje su korištene u procesu treniranja, donosi se neka predikcija o prosljедenoj slici. Obzirom na to da je klasifikator baziran na sličnostima između dvije slike, postavlja se pitanje koje metrike možemo koristiti unutar samog algoritma.

Prva metrika koja se koristi je **L1** ili **Manhattan distanca**. Ona je jednaka sumi razlika apsolutnih vrijednosti između dva piksela.

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

Već je poznato da se slika predstavlja kao matrica. Dakle, potrebno je uzeti sliku koja je korištena u procesu treniranja i od njene reprezentativne matrice piksela oduzeti matricu prosljedeđene slike predikcije. Zatim, potrebno je sumirati ove vrijednosti da bismo dobili odgovarajući broj. Ovaj broj predstavlja datu distancu između slika. Što je ovaj rezultat manji, to bi teoretski trebalo značiti da su slike sličnije.

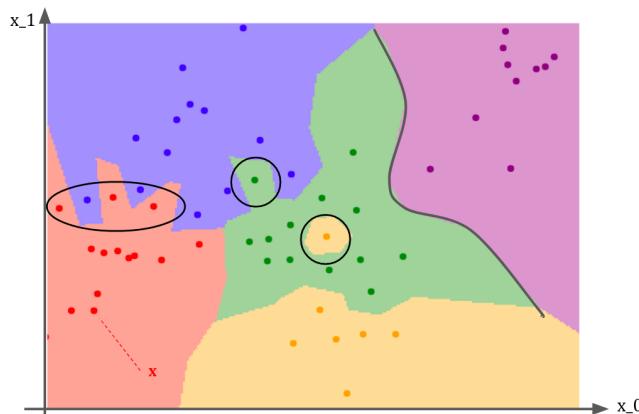
Pored ove metrike, postoji i **L2** ili **Euclidean distanca** koja ima kompleksniji proračun. Ova metrika računa distancu između dva piksela na način da se sumira kvadrat razlika između dva piksela, a zatim se traži korijen te sume.

$$d_1(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

Ako govorimo o implementaciji klasifikatora Nearest Neighbor, radi se o laganoj implementaciji. Funkcija treniranja ima zadatak da izvrši memorizaciju gdje se podaci i labele učitavaju u memoriju. Podaci su predstavljeni matricom gdje svaki red predstavlja pojedinačan podatak, dok su labele predstavljene običnim nizom. Funkcija predikcije uzima neku novu sliku i računa distancu između svih podataka koji se nalaze u skupu za treniranje. Nakon što se pozove funkcija predikcije, traži se najmanja distanca pomoću koje donosimo zaključak kojoj klasi slika pripada.

Postavlja se pitanje koliko je efikasan navedeni algoritam. Za funkciju treniranja, kompleksnost je $O(1)$ jer je potrebno samo zapisati određene podatke u memoriju. Međutim, za proces predikcije je potrebno svaku prosljedeđenu sliku porebiti sa svakom slikom iz skupa treniranja. Dakle, kompleksnost je $O(N)$ gdje je N broj slika iz skupa treniranja. U praksi se obično toleriše da je vrijeme treniranja dugo, ali ne i vrijeme testiranje. Samim time, potrebno je tražiti efikasnije metode koje brže dolaze do rješenja.

Problemi koji nastaju kod ovog klasifikatora se mogu prikazati preko jednog koordinatnog sistema. Prikazane tačke iz skupa za treniranje su raspoređene po ovoj ravni, a boja podloge predstavlja određenu klasu. Kod ove podjele, imamo podatke koji iskaču iz tipičnih predstavnika podataka iz određene klase (**outlier**). Pored toga, imamo i granice između skupova koje nisu dovoljno precizno određene. Postavlja se pitanje može li se efikasnije podijeliti prostor između datih klasa.



Slika 2.27: Nedostaci algoritma Nearest Neighbor

Odgovor daje algoritam **k-Nearset Neighbor** koji je nastao nakon prethodno opisanog algoritma. Osnovna ideja je uzeti k najsličnijih slika i izabrati neku od njih koju ćemo uzeti kao klasu slike koju testiramo. Međutim, prilikom implementacije ovog algoritma se javljuju tačke koje više nisu ispravno klasificirane. Ovaj problem se javlja upravo zbog modifikacije skupova kako granice ne bi bile oštре. Još jedna mana ovog algoritma je što se javljuju nedefinisani bijeli prostori. Ukoliko neka nova tačka tačno spada u ovaj prostor, morali bismo pomoći heuristike da odlučimo u koju klasu ta tačka pripada.



Slika 2.28: Nedostaci algoritma k-Nearest Neighbor

Izvršimo uporedbu dva navedena klasifikatora uz date metrike. Recimo da imamo 4 klase i 40 tačaka nasumično raspoređenih u određenom prostoru. Primijenit ćemo klasifikator Nearest Neighbor. U prvom slučaju uzimamo metriku **L1**, a u drugom uzimamo metriku **L2**.

U prvom slučaju, smjer kretanja granica unutar klasa je orijentisan u smjerovima koji daju oštре linije. S druge strane, ukoliko koristimo **L2**

distancu, granice mogu primati bilo koji oblik što nam daje puno više fleksibilnosti. Smjer kretanja granica je prikazan na sljedeći način:



Slika 2.29: Primjer metrike za distancu

Kao drugi primjer, uzmimo klasifikator k-Nearest Neighbor uz iste distance gdje je $k = 4$. Granice su veoma slično određene kao u prvom slučaju, uz razliku što se sada javljaju i nedefinisani bijeli prostori. Na ovom primjeru se može uočiti velika razlika između korištenja L1 i L2 distance u smislu podjele prostora na same klase.



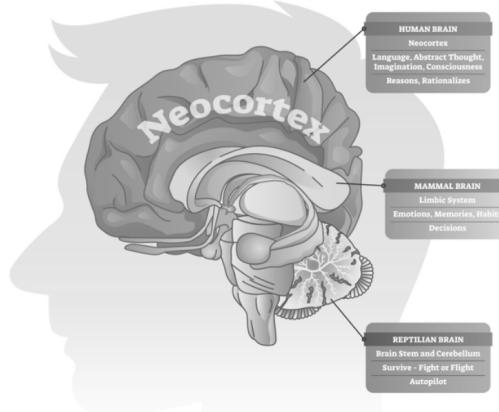
Slika 2.30: Primjer metrike za distancu

Predavanje 4

3.1 Uvod u neuronske mreže

Motivacija za izučavanje vještačkih neuronskih mreža jeste automatiziranje poslova preko računara. Iako se danas mnoge aktivnosti mogu automatizirati, postoje i one za koje je to još uvjek nemoguće. Da bismo napravili sistem koji može samostalno donositi zaključke ili učiti, moramo prvo pronaći motivaciju u prirodi, odnosno, u **mentalnoj aktivnosti**. Mentalna aktivnost se sastoji od elektrohemijskih aktivnosti moždanih ćelija koje se nazivaju **neuroni**. Inspirisani ovom hipotezom, naučnici su pokušali da naprave sisteme koji rade na bazi **vještačke neuronske mreže** koje mogu na isti ili sličan način procesirati informacije i donositi zaključke. Sistem vještačke inteligencije koji uspješno oponaša rad mozga naziva se **inteligentnim sistem**. Pored ovog naziva, u praksi se javljaju i pojmovi konektivizam, paralelno distribuirano procesiranje i neuronsko računanje.

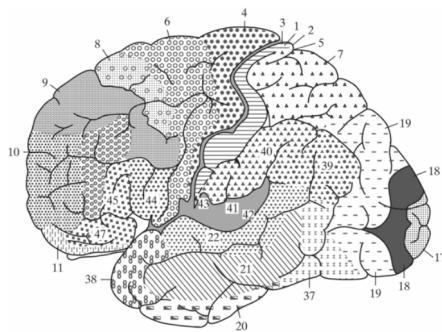
Ljudski mozak je veoma složen središnji dio živčanog sistema čovjeka. Kažemo da se sastoji od četiri osnovna dijela: veliki mozak, srednji mozak, mali mozak i kičmena moždina. Najsloženiji dio predstavlja **neokorteks** (veliki mozak). Neokorteks je dio mozga u kojem možemo da procesiramo jezik, da apstraktno razmišljamo i imamo svijest o nekim pojavama. Drugi dio mozga je **limbički sistem** (srednji mozak) pomoću kojeg spoznajemo emocije, donosimo odluke i čuvamo navike. Treći dio mozga je **moždano stablo** (mali mozak) koji nam omogućava da preživimo u sredini koju manje poznajemo (fight or flight - ili bježi ili se bori). Ovaj dio mozga je zadužen za donošenje odluka na samom licu mesta.



Slika 3.1: Ljudski mozak

Mozak se sadrži od niza podsistema pomoću kojih čovjek obavlja razne aktivnosti. Prvi dio koji je bitan za spomenuti je **vizuelni kortex** koji je povezan sa senzorima (odnosno sa očima) pomoću kojeg dobijamo vizuelne informacije. Pored toga, postoji i **slušni kortex** (uši) koji služi za primanje zvuka. Zatim imamo **motorni kortex** i **predmotorni kortex** koji su zaduženi za sva kretanja koja obavljamo u svakodnevnim aktivnostima.

Figure 4. Cytoarchitectural map of the cerebral cortex. The different areas are identified by the thickness of their layers and types of cells within them. Some of the key sensory areas are as follows. Motor cortex: motor strip, area 4; premotor area, area 6; frontal eye fields, area 8. Somatosensory cortex: areas 3, 1, and 2. Visual cortex: areas 17, 18, and 19. Auditory cortex: areas 41 and 42. (From A. Brodal, 1981, with permission of Oxford University Press.)



Slika 3.2: Podsistemi mozga

Znamo da je mozak sačinjen od velikog broja neurona koji većinsko rade paralelno. Pored ove činjenice, poznato je i da postoji više od 100 vrsta različitih neurona. Svaka od ovih vrsta radi vrlo jednostavnu obradu podataka, a vrijeme koje se potroši na obradu je 2ms. Broj neurona u ljudskom mozgu je 10^{11} , a svaki od njih u prosjeku dobija informacije od 10^3

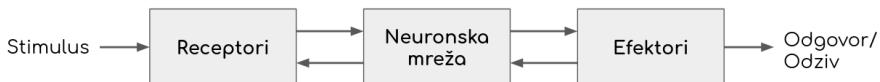
do 10^4 drugih neurona. Obrada samih podataka je tolerantna na greške. Ova činjenica govori da neće doći do pada čitavog sistema ukoliko se desi greška u procesu odlučivanja.

Inspirisani različitim istraživanjima na poljima biologije, psihologije i sličnih nauka, naučnici u polju računarskih tehnologija su došli do nove paradigme da se napravi model **vještačkih neuronskih mreža** (engl. *Artificial Neural Networks*). Model se bazira na funkcionalnostima ljudskog mozga. Samim time, javlja se i nova grana računarskih nauka koja se bavi izučavanjem ove nove paradigme koja se naziva **neuro-računarstvo**.

Postoje dva pristupa razvoju vještačke inteligencije. Prvi pristup razvoju predstavlja **simbolički pristup**. Znanje se predstavlja pomoću simbola, te se tim znanjem manipuliše pomoću algoritamskih pravila. Radi se o vrlo ranom pristupu koji nije ispunio početna očekivanja, iako je imao određene uspjehe u oblasti ekspertnih sistema. Razlog ovoga je što svako znanje ne možemo predstaviti formalnim pravilima jer ne znamo tačno na koji način ljudski mozak radi.

Konektivistički pristup je drugi pristup razvoja vještačke inteligencije. Zasniva se na izgradnji sistema arhitektura slične arhitekturi mozga koji uči **samostalno** na bazi iskustva.

Mozak je izuzetno složen, nelinearan i paralelan računar. Zadaci prepoznavanja ili percepcije se rutinski izvode za otprilike 100-200ms.

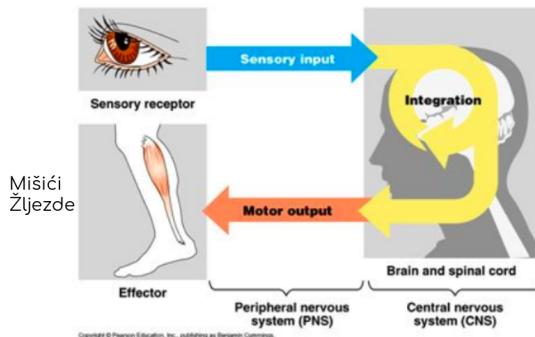


Slika 3.3: Opći model neuronske mreže

Na samom početku modela imamo stimuluse koji dolaze iz okoline. Ove podatke primaju **receptori** koji ih dalje prosleđuju **neuronskoj mreži**. Zatim, neuronska mreža donosi neke odluke i šalje ih **efektorima** koji mogu da daju određenu reakciju (generisanje razgovora, pokret ruke i slično). Te informacije se šalju u okolinu kao **odgovor**. Također, na osnovu određenih informacija iz okoline možemo analizirati efektore drugih ljudi te prosljedivati te informacije neuronskoj mreži. Upravo navedeno predstavlja **povratnu spregu** koja se konstantno odvija.

Jedna osobina mozga koja se posebno izdvaja je **plastičnost mozga**. Ova osobina omogućava nervnom sistemu da se prilagodi svojoj okolini. Iako se smatra da ova osobina postaje slabija kroz odrastanje, istraživanja kažu da je moguće održavati plastičnost mozga konstantnim osvježavanjem znanja.

Na sljedećoj slici je vizuelno prikazano primanje podataka preko receptora. Receptori šalju određene informacije preko senzorskog ulaza do ljudskog mozga. Na ovom mjestu se informacije procesiraju, te se određene odluke šalju preko centralnog nervnog sistema do različitih motornih izlaza (mišići ili žlezde).



Slika 3.4: Prikaz povratne sprege

Vještačka neuronska mreža je mašina koja je dizajnirana da modelira način na koji možak izvršava određene zadatke ili funkcije od interesa. U istraživanju se fokusiramo na neuronske mreže koje izvode **korisne proračune** kroz **proces učenja**.

Algoritam učenja je postupak koji se koristi za izvođenje procesa učenja. Zanimljiva je činjenica je da biološka neuronska mreža *može modificirati vlastitu topologiju* koja je motivirana činjenicom da ljudske moždane ćelije prirodno mogu "umrijeti", te da se mogu formirati nove sinaptičke veze koje ne uključuju ove neurone. Računska snaga neuronske mreže predstavlja masivno distribuiranu i paralelnu strukturu koja ima **sposobnost učenja** (generaliziranja).

Proces učenja se sastoji od dvije faze rada:

- **faza učenja** (treniranja)
- **faza zaključivanja ili obrade podataka**

Učenje je iterativan pristup u kojem nekom sistemu predstavljamo skup ulaznih podataka (uzoraka, iskustva) i informaciju o očekivanom izlazu. Pri ovom procesu je potrebno da postepeno prilagođavamo koeficijente (veze) između neurona.

Epoха je jedna faza ili stadij u kojem se predstavljaju neki ulazni podaci.

Dakle, unutar jedne epohe tačno definišemo koliko uzoraka čemo proslijediti sistemom na osnovu kojih će on dalje učiti. Razlikujemo tri vrste učenja:

1. **pojedinačno učenje** (engl. *online*) - učenje se dešava nakon svakog predstavljenog uzorka
2. **učenje s mimigrupama** (engl. *mini-batches*) - učenje se dešava nakon više predočenih uzoraka
3. **grupno učenje** (engl. *batch*) - učenje se dešava tek nakon svih predočenih uzoraka

Proces učenja omogućava odlučivanje o nekim budućim događajima na osnovu posmatranja određene okoline ili pravljenja opservacija o svijetu. Učenje može biti trivijalan proces (kao što je biranje broja na telefonu) ili može biti složen proces (nove teorije o univerzumu).

Mogućnost učenja je bitna iz mnogo razloga. Jedan od njih je činjenica da je nemoguće predvidjeti sve moguće scenarije u kojima će se program (agent) naći. Pored toga, nemoguće je i predvidjeti sve promjene u vremenu koje će se desiti. Kao primjer možemo navesti program koji treba da predviđa stanje na tržištu dionica. Ovakav program mora imati mogućnost da se prilagodi na nove promjene onda kada one nastanu. Iako je moguće napraviti sistem koji nema mogućnost ažuriranja na neku novu okolinu, on će se većinu vremena smatrati nepotrebnim jer ima ograničenu upotrebu. Dalje, mogućnost učenja je bitna jer je neka znanja nemoguće isprogramirati. Primjer ovakvog problema jeste pisanje programa za prepoznavanje lica. Pravljenje ovakvog algoritma nije bilo uspješno jer su programi imali tačnost od oko 60%. Međutim, dolaskom dubokih neuronskih mreža, posao pisanja ovakvih algoritama se znatno olakšao.

Induktivno učenje je sposobnost pronalaženja generalne funkcije ili pravila na osnovu tačno određenih ulazno/izlaznih parova podataka.

Postoji nekoliko različitih načina učenja u odnosu na **povratnu spregu** (engl. *feedback*) u procesu učenja. Dakle, ukoliko nekom sistemu proslijedimo neku informaciju, on će donijeti određene zaključke. Ako sistem ima mogućnost da ima povratnu spregu o tim zaključcima, on tu informaciju može koristiti kako bi ažurirao svoje znanje.

U odnosu na povratnu spregu koju možemo da dobijemo, proces učenja možemo podijeliti na tri načina učenja:

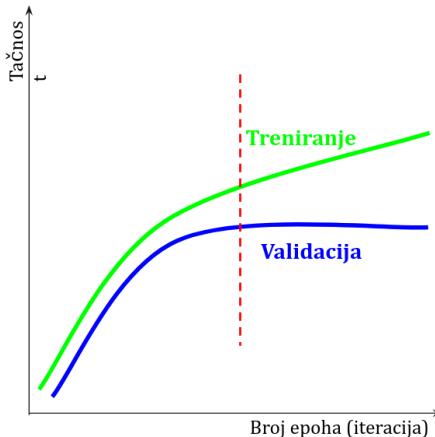
1. **nadzirano** (engl. *supervised*) - u procesu učenja postoje ulazno/izlazni parovi koji pomažu programu/agentu da uči iz podataka
2. **nenadzirano** (engl. *unsupervised*) - ne postoji povratna sprega u procesu učenja (iz podataka se izvlači funkcija koja odgovara samom zadatku)

3. podržano (engl. reinforcement) - učenje na bazi nagrada i kazni

Proces učenja se uvijek izvršava nad nekim **skupom podataka** (engl. *dataset*). Tok procesa se stalno prati kroz **provjeru tačnosti** (engl. *learning accuracy*) nad skupom podataka za treniranje, zatim nad skupom za validaciju i skupom za testiranje. Ako se krećemo u dobrom pravcu (tačnost se povećava), to znači da su arhitektura i hiperparametri dobro određeni.

Model nad kojim vršimo proces učenja mora da ima sposobnost **generalnog zaključivanja** nad skupom za testiranje, odnosno, nad podacima koje mreža nikad nije vidjela. Negativna pojava koja se javlja u procesu učenja je **pretreniranost** (engl. *overfitting*) nad podacima za treniranje. Dešava se da model previše dobro procjenjuje zadatke u procesu treniranja, te iz ovog razloga on nema dobre rezultate nad podacima za testiranje. Dakle, potrebno je pronaći određeni balans između ova dva procesa.

Na grafu je prikazan odnos broja epoha (iteracija) u odnosu na tačnost. Obično se kaže da je broj iteracija povezan sa tačnošću (što je veći broj iteracija, veća je i tačnost). Međutim, kako ne bi došlo do pretreniranosti, proces treniranja je potrebno pratiti i zaustaviti u trenutku kada tačnost počinje stagnirati ili opadati na skupu za validaciju.



Slika 3.5: Odnos broja epoha i tačnosti

Pošto su neuronske mreže pretežno vezane za nadzirano učenje, u nastavku će se opisati zadatak ovog načina učenja.

Neka je poznat skup podataka za treniranje:

$$(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$$

Svaki x_i predstavlja ulazni podatak koji može biti slika, niz brojeva i slično. Za ove podatke imamo labelu y_i koja je generisana nama nepoznatom

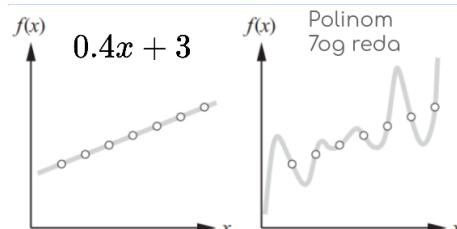
funkcijom $y = f(x)$. Zadatak je naći funkciju h koja vrši aproksimaciju nepoznate funkcije y . Funkcija h se naziva **hipoteza** i traži se u prostoru hipoteza.

Dakle, proces učenja je pretraživanje prostora svih hipoteza H i pronalaženje one hipoteze koja će imati zadovoljavajuće rezultate i nad podacima izvan skupa za treniranje (**testni skup podataka**). Kažemo da hipoteza ima sposobnost generaliziranja ukoliko ispravno izvrši predikciju vrijednosti y nad testnim skupom podataka.

Kao što je već rečeno, zadatak klasifikacije je da se definiše podjela postojećih podataka na određene klase. Drugi zadatak koji se javlja jeste **regresija** pri čemu je izlaz iz ovog algoritma broj (npr. ukoliko provjeravamo temperaturu određenog dana).

Ukoliko imamo neku funkciju hipoteze h koja može konzistentno da opiše sve podatke nekog skupa, onda se ona naziva **konzistentnom hipotezom**. Problem s kojim se susrećemo jeste kako odabratи najbolju hipotezu. Sami izbor ovisi o testnim podacima, odnosno, ukoliko oni leže na funkciji hipoteze, onda se bira ta hipoteza i kaže se da je to idealan slučaj.

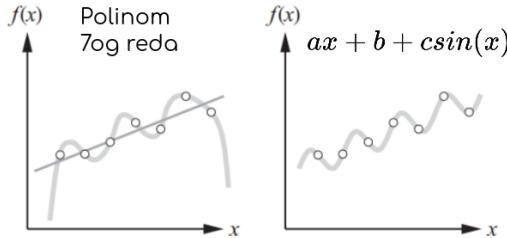
Sljedećim grafom je prikazan niz tačaka i date su funkcije koje ih opisuju. Možemo zaključiti da i jedna i druga funkcija jako dobro opisuju date podatke, te se obe mogu koristiti kao klasifikatori podataka. Međutim, jedna je kompleksnija (polinom sedmog reda) a druga jednostavnija (linearna funkcija). Iz skupa odabranih hipoteza, potrebno je izvršiti ispravan izbor.



Slika 3.6: Primjer traženja hipoteza

Jedno pravilo koje se bavi ovim problemom je pravilo **Ockhamove oštice** (engl. *Ockham's razor*). Ime je dobilo po engleskom filozofu **William of Ockham** iz XIV vijeka. Ovo filozofsко pravilo pokušava da dà odgovor na pitanje kako odabratи između dvije ili više konzistentnih hipoteza. Jedno od mogućih rješenja je da biramo najjednostavnije pravilo/funkciju u odnosu na konzistentnost sa podacima.

Drugi primjer sličnog problema je predstavljen sa sljedećim skupom podataka. Na samom grafu su prikazane i funkcije koje opisuju zadane podatke. Predstavljene su dvije hipoteze, pri čemu je jedna funkcija polinom sedmog reda, dok je druga funkcija modifikovana sinusoida sabrana sa linearном funkcijom.



Slika 3.7: Primjer traženja hipoteza

Postavlja se pitanje koja je funkcija bolja. Potrebno je napraviti *kompromis* između složenosti funkcije i mogućnosti generalizacije. Suštinski problem je što ne znamo pravu funkciju koja bi mogla da opiše dati skup podataka. Samim time, teško je reći da li je proces učenja moguće realizirati. Obzirom na to da izbor hipoteze ovisi i o brzini izračunljivosti funkcije h , dolazimo do zaključka da biramo jednostavnije funkcije kao što to opisuje i Ockhamovo pravilo.

Polinomi većeg reda mogu značajno bolje opisati skup podataka za treniranje. Međutim, bitnije je naći funkciju koja će pokazati zadovoljavajuće rezultate na validacijskom i testnom skupu podataka. **Izbor modela** je proces kojim se bira red polinoma (odnosno **prostor hipoteze**). Zatim se procesom **optimizacije** pronalazi funkcija najbolje hipoteze unutar tog prostora. Da bismo izvršili optimizaciju, moramo definisati i **funkciju gubitka L** (engl. *loss*).

Recimo da imamo neku funkciju $f(x) = y$ koja će nam dati pravu labelu. Prvo je potrebno naći hipotezu kojoj šaljemo iste ulazne podatke, a na izlazu dobijamo procjenu funkcije \hat{y} . Recimo da je tražena vrijednost 137.035999, a vrijednost koju smo dobili pomoću hipoteze iznosi 137.036. Da bismo procijenili da li je ova funkcija dobra ili ne, definišemo **funkciju gubitka $L(x, y, \hat{y})$** .

Imamo nekoliko načina izračunavanja vrijednosti gubitka. Prvi način je **apsolutna vrijednost gubitka (L1)** data funkcijom

$$L_1(y, \hat{y}) = |y - \hat{y}|$$

dok je drugi način putem **kvadratne vrijednosti greške** definiranom

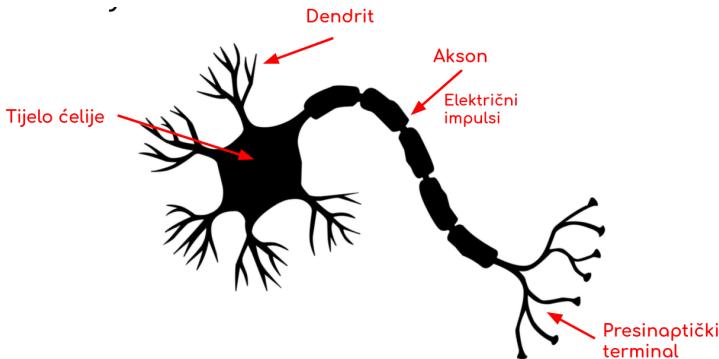
funkcijom

$$L_2(y, \hat{y}) = (y - \hat{y})^2$$

Ovu informaciju o gubitku je moguće dalje propagirati kroz mrežu putem već spomenute backpropagacije.

3.2 Biološki i vještački neuron

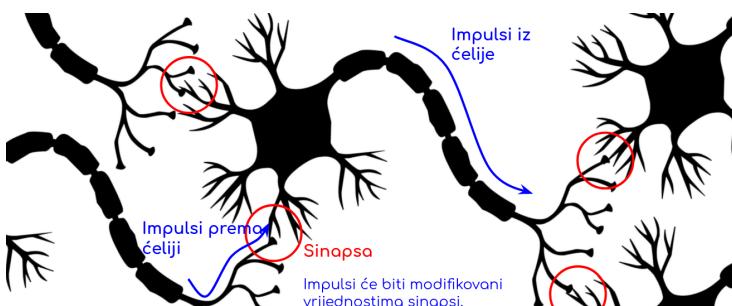
U nastavku će se uvesti osnovne informacije o biološkom i računarskom neuronu. Biološki neuron je ilustrativno predstavljen sljedećim modelom:



Slika 3.8: Biološki neuron

Funkcionalnost ovakvog biološkog neurona je bazirana na različitim istraživanjima. Komponenta svakog neurona je **tijelo ćelije**. Na tijelo ćelije je povezan **akson** kroz koji prolaze elektrohemski impulsi. Tijelo ćelije prihvata informacije od drugih neurona preko **dendrita**, dok se preko **aksona** šalju informacije ka drugim neuronom. Shodno tome, impulsi se mogu slati prema ćeliji i izvan ćelije. Na krajevima aksona su definisani **presinaptički terminali** koji služe za spajanje sa drugim neuronom. Kada se spoji terminal sa dendritom drugog neurona, dobijamo pojavu **sinapse**.

Jedan neuron nema neku značajnu funkciju, već je potrebno imati određenu arhitekturu neurona.

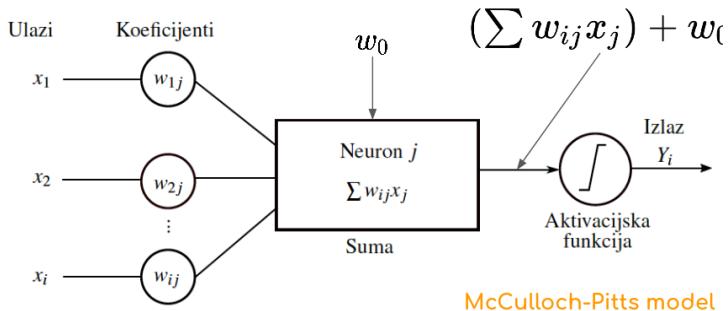


Slika 3.9: Mreža bioloških neurona

Obzirom na to, u realnosti većinom posmatramo niz neurona. Konekcija između dendrita i presinaptičkih terminala omogućava prihvatanje informacija

od drugih neurona. Navedeno ne mora značiti da će doći do slanja impulsa drugom neuronu svaki put kada se ostvari konekcija između njih. Ovom činjenicom se javlja pojava nelinearnog ponašanja čime se definiše **stopa aktiviranja neurona**. Iz ovog razloga je potrebno preći određeni prag da bismo poslali informaciju iz jednog neurona u drugi neuron.

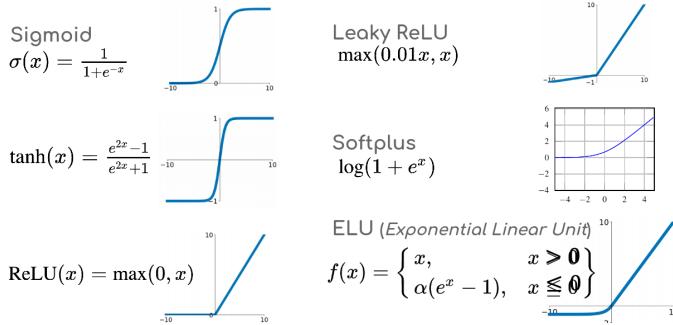
Prvi model biološkog neurona je nastao 1943. godine i naziva se **Threshold Logic Unit** ili **TLU perceptron**. Predstavljen je od strane Warren McCulloch-a i Walter Pitts-a i ujedno predstavlja prvi model biloškog neurona.



Slika 3.10: McCulloch-Pits-ov model neurona

Jedinica koja definiše prag (engl. *Threshold*) zapravo predstavlja aktivacijsku funkciju. McCulloch-Pits-ov model se sastoji od više elemenata. Na samom početku se nalaze ulazi koje čine neki drugi neuroni (x_i). Veze između ovih ulaznih neurona i biološkog neurona j su omogućene preko koeficijenata koji se ažuriraju u samom procesu učenja. Zatim, imamo tijelo neurona koje se implementira kao suma skaliranih vrijednosti ulaza. Na ovu sumu se dodaje odredena vrsta otklona (engl. *bias*) w_0 koja predstavlja neku predefinisanu vrijednost i može biti pozitivna ili negativna vrijednost (vrlo često ima vrijednost 1). Vrijednost na izlazu iz tijela neurona predstavlja **funkciju sume** koja se dalje proslijedi na procesiranje **aktivacijskoj funkciji** (engl. *threshold logic unit*). Aktivacijska funkcija donosi odluku da li šaljemo informaciju do narednog neurona. Ukoliko nećemo vršiti slanje određene informacije, izlaz će biti nula, a u suprotnom slučaju je jedan. U realnosti postoje kompleksnije aktivacijske funkcije koje daju više izlaza. Aktivacijska funkcija je nelinearna funkcija i unosi nelinearnost u računarski model. Dakle, model će generisati određeni izlaz koji prelazi prag definisan u odnosu na odabranu aktivacijsku funkciju.

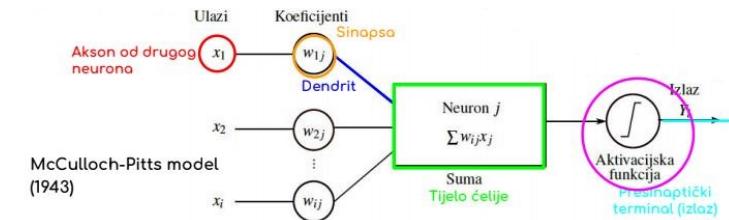
Neke aktivacijske funkcije koje se mogu iskoristiti su date u nastavku:



Slika 3.11: Primjer aktivacijskih funkcija

Izbor funkcije zavisi od vrste problema koji se rješava. Najčešće se koristi funkcija $ReLU(x)$ jer ona vrši najjednostavniju operaciju. Unutar ove funkcije se traži maksimalna vrijednost između nule i izlaza koji dobijemo iz tijela neurona. Ukoliko je pozitivna vrijednost, izvršiti će se proslijedivanje informacija, a ukoliko je rezultat nula, neće se ništa desiti. Pored jednostavnosti, funkcija daje i zadovoljavajuće rezultate.

Preko računarskog modela biološkog neurona je moguće predstaviti biološki neuron sa svim spomenutim konceptima. Ulaz u model će predstavljati aksone od drugog neurona. Zatim, sinapsu možemo predstaviti putem koeficijenata, dok će veza između koeficijenata i tijela ćelije biti prikazana preko dendrita. Tijelo ćelije se predstavlja preko već opisane sume, a rezultat se šalje na presinaptički terminal.



Slika 3.12: Model biološkog preko računarskog modela neurona

U narednoj tabeli, možemo vidjeti kako se neki dijelovi biološkog neurona mapiraju na vještačke neurone. **Soma** ili **ćelija** se obično mapira na čvor (engl. *Node*) i predstavlja osnovnu ćeliju neurona. **Dendriti** koji se koriste za slanje informacija drugim neuronima su zapravo ulazi. Zatim, aksoni predstavljaju izlaze iz neurona. Sinapse predstavljaju konekcije između određenih elemenata i određene su koeficijentima. Koeficijent se određuje pomoću snage i smjera, te se na osnovu toga utvrđuje da li se povećava ili smanjuje funkcija veze. Dalje, posmatrajući jedinicu biološkog neurona,

možemo reći da su oni dosta spori dok su s druge strane vještački neuroni jako brzi. Međutim, nemoguće je imati onoliko broj neurona koliko ih postoji u ljudskom mozgu jer je treniranje takve mreže jako kompleksno.

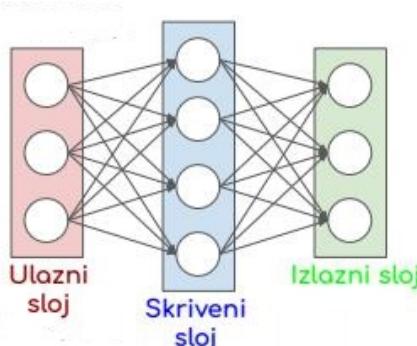
Biološki	Vještački
Soma	Čvor
Dendridi	Ulazi
Akson	Izlaz
Sinapsa	Koeficijent
Sporo	Brzo
Veliki broj neurona (10^9)	Nekoliko neurona (desetina ili stotine hiljada)

Slika 3.13: Tabela poređenja neurona

Kao što je već rečeno, da bismo mogli izvršiti neku korisnu klasifikaciju, potrebno je više neurona. Ti neuroni se obično organizuju u tri vrste slojeva: ulazni, skriveni i izlazni. Ukoliko se radi o kompleksnijim modelima, postojati će više skrivenih slojeva i u tom slučaju se radi o **dubokoj mreži**.

Ulagani sloj sadrži ulazne podatke. Svaki podatak se može prikazati preko određenih atributa. Stoga, ulazni sloj će imati onoliko podataka koliko on ima atributa. Iako se ulazi predstavljaju kao neuroni, oni nemaju mogućnost računanja dok neuroni u skrivenom i izlaznom sloju posjeduju ovu mogućnost.

U sljedećem primjeru, svaki ulaz je povezan sa svakim neuronom u skrivenom sloju, te je svaki neuron iz skrivenog sloja povezan sa izlaznim slojem. Ovaj model predstavlja **unaprijed formiranu neuronsku mrežu sa više slojeva**, te nema povratnu spregu.



Slika 3.14: Slojevi neuronskih mreža

Uzmimo kao primjer binarnu klasifikaciju. U tom slučaju, izlazni sloj ima samo jedan neuron koji može biti ili nula ili jedan. Dakle, za potrebe klasifikacije ovog modela, jedan sloj je sasvim dovoljan. Povećavanjem broja klase povećava se i broj izlaza.

Sličnosti između bioloških i vještačkih neurona su simbolične, te računarski neuron ne predstavlja stvarnu reprezentaciju biološkog neurona. Dakle, u realnosti ne možemo tačno replicirati biološki neuron u računarskim sistemima iz više razloga. Biološki neuroni imaju izuzetno **kompleksnu topologiju** (vezu između neurona). Dendriti u biološkim neuronima mogu izvršavati složene nelinearne proračune, što je teško precizno opisati u računarskom sistemu. Pored toga, pojava sinapse predstavlja **složeni nelinearni dinamički sistem** (nisu pojedinačne težine). S druge strane, računarski neuroni su organizovani u **slojeve regularnog formata** i primarno koriste operacije sabiranja i množenja. Ovi slojevi predstavljaju apstraktne vještačke tvorevine koje trebaju da preslikaju stvarnu strukturu bioloških neurona.

Nakon objave McCulloch-Pitts-ovog modela biološkog neurona, 1949. godine je definisano i **Hebbovo pravilo učenja**. Ovo pravilo je uveo **Donald O. Hebb** i ono glasi:

"Kad je akson neurona A dovoljno blizu da aktivira neuron B i to ponavlja veći broj puta, dolazi do metaboličkih promjena tako da se povećava efikasnost neurona A u aktiviranju neurona B."

Nakon njega, psiholog **Gunther S. Stent** je proširio Hebbovo pravilo 1973. godine:

"Ako jedan neuron ne utiče na drugog, onda sinapsa među njima postaje slabija ili se potpuno eliminiše."

Dakle, pravilo govori da što su dva neurona bliže jedan drugom da imamo veću mogućnost promjene koeficijenata između ta dva neurona. Odnosno, što više ponavljamo neku materiju, neuroni koji definišu znanje se češće mijenjaju. Samim time, ukoliko imamo dva neurona koji ne utiču jedan na drugog, sinapsa će biti slabija.

Rosenblatt 1958. godine uzima ova dva pravila i McCulloch-Pitts-ov model, te definije **pravilo učenja perceptronu** koje glasi:

1. Ciklično se prolazi kroz svih N uzoraka za učenje, jedan po jedan.
2. Izvrši se klasifikacija trenutnog uzorka.
 - (a) Ako je klasifikacija ispravna, ne mijenjaj koeficijente
 - i. ako je N-ti uzastopni uzorak klasificiran ispravno, prekini proces učenja

- ii. inače pređi na sljedeći uzorak
- (b) Ako je klasifikacija neispravna, izvrši korekciju vrijednosti koeficijenta perceptronu prema sljedećoj formuli

$$w_i(k+1) = w_i(k) + \eta(y - \hat{y})x_i$$

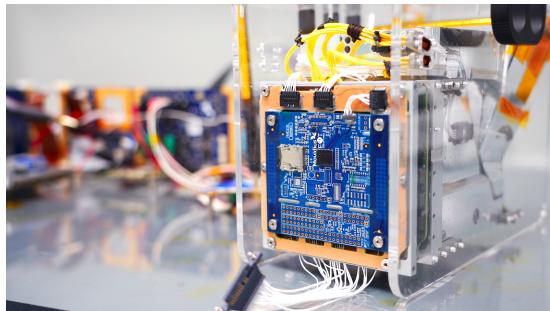
gdje je w_i vrijednost trenutnog koeficijenta, $y - \hat{y}$ razlika između stvarne i predviđene labele

Parametar η se naziva **stopa učenja** i može da poprimi različite vrijednosti. Naprimjer, može biti pozitivan veoma mali broj koji određuje koliko treba ažurirati trenutnu vrijednost koeficijenta. Ako je broj premali, postupak učenja će napredovati veoma sporo. S druge strane, ako je preveliki, postupak može divergirati u neželjenom smjeru.

Predavanje 5

4.1 Aktuelnosti

PhiSat-1 je računar poslan u oktobru 2020. godine u Svetmir. Nalazi se na satelitu i nadgleda površinu planete Zemlje. Ima mogućnost da pravi **multispektralnu analizu** informacija koje dolaze sa optičkog senzora preko **hiperspektralne termalne kamere**. Kamera ima mogućnost da snima u puno više nivoa od optičkih kamera, a pored toga može da snima i različite frekvencije. Na početku su se podaci isključivo snimali i direktno slali na površinu Zemlje. Danas, ovaj model ima mogućnost da snimi i analizira neku sliku, te nakon toga šalje sliku na površinu zemlje. Ukoliko tokom analize nema značajnog sadržaja, slika se jednostavno odbaci. Jedna od primjena je da se snima površina Zemlje zbog požara. U ovom slučaju, često se dešava da je površina prekrivena oblacima, te na taj način dolazimo do zaključka da se slika treba odbaciti. Radi se o prvom čipu koji je odletio u svemir, a isti je izradio **Intel**. Cipovi koje danas imamo na računarima nisu korisni jer bi ih radijacija jako brzo uništila.



Slika 4.1: PhiSat-1

Senzori napreduju dosta brže nego tehnologija koja se koristi za prenos podataka. Stoga, često imamo slike velikih rezolucija čije je slanje jako skupo.

Iz ovog razloga, potrebne su velike količine memorije kako bi se slike spasile na jednom mjestu dok ne dođe vrijeme za njihovo slanje.

4.2 Motivacija za kreiranje vještačkih neuronskih mreža

Motivacija za kreiranje vještačkih neuronskih mreža dolazi od studiranja ljudskog mozga. Mozak računa na sasvim drugačiji način od konvencionalnih digitalnih računara. Imamo neke interesantne karakteristike koje možemo iskoristiti kod digitalnih računara kako bismo izvršili modeliranje bazirano na načinu rada mozga. Postoje prednosti modeliranja neurona u odnosu na digitalne računare i suprotno. Recimo, jedna od negativnih karakteristika neurona je što su neuroni pet-šest redova veličine sporiji od digitalne logike (ms i ns). Međutim, ovaj nedostatak u brzini mozak nadoknađuje ogromnim brojem neurona. Istraživanja kažu da ljudski mozak ima oko 100 milijardi neurona i oko 10^{15} konekcija među njima.

Još jedna prednost ljudskog mozga u odnosu na digitalni računar je činjenica da je mozak izuzetno energetski efikasan. Digitalni računar zahtijeva negdje oko $10^{16} J$ po operaciji u sekundi, dok mozak zahtijeva oko $10^{-6} J$ po operaciji u sekundi. Sve ove činjenice su bile motivacija za izgradnju vještačkih neuronskih mreža. Mozak je veoma kompleksan, nelinearan sistem i možemo reći da distribuirano procesira informacije na paralelan način. U računarstvu se javlja nova aktuelna oblast koja se naziva **In-memory-computing**. Oblast nalazi motivaciju u radu ljudskog mozga i proučava sistem koji ima mogućnost da izvršava određene operacije i istovremeno snima operacije u jednoj procesnoj jedinici. Današnji računari imaju potpuno odvojena dva koncepta, a to su procesiranje i snimanje podataka. Iz ovog razloga vrlo često imamo *usko grlo* koje nam onemogućava brzo dobavljanje velikih količina podataka za procesiranje.

Postoji projekat **Human Connectome** (HCP) koji služi da bi se izgradile mape kompletnih strukturnih i funkcionalnih neuronskih veza *in vivo* unutar i između pojedinaca. HCP predstavlja prvi veliki pokušaj prikupljanja i razmjene podataka opsega i detalja dovoljnih za započinjanje procesa rješavanja temeljnih pitanja o ljudskog vezivnoj anatomiji i razlikama. Projekat je interesantan zbog činjenice da se izučavanje mozga vrlo često izvršava na materiji koja nije živa. Dakle, radi se o vrlo zahtjevnom poslu koji analizira rad mozga uživo.

4.3 Podjela neuronskih mreža

Kao što je već rečeno, neuronske mreže možemo okvirno podijeliti na **biološke neuronske mreže i vještačke neuronske mreže**.

Biološke (ili prirodne) neuronske mreže su biološki organizmi. Primjeri koje nalazimo u praksi su mozak ljudi ili mozak životinja. Radi se o visokosloženim tvorevinama koje imaju mogućnost paralelnog izvršavanja.

Vještačke neuronske mreže su nastale na osnovu motivacije iz proučavanja bioloških neuronskih mreža. Još uvijek se smatra da se radi o dosta primitivnim imitacijama bioloških mreža. Nedostaci se javljaju unutar samih struktura koje zahtijevaju jako puno energije da bi izvršile slične operacije ljudskog mozga. Vještačke neuronske mreže se danas implementiraju na digitalnim računarima opće namjene ili pomoću specijaliziranih kola (analognih, digitalnih ili hibridnih). Danas postoje velike laboratorije koje izučavaju neuronske mreže isključivo pomoću analognih sistema koristeći specijalizirane uređaje.

4.4 Neuronska mreža

Neuronska mreža generalno predstavlja skup jednostavnih komponenti koje nazivamo **vještačkim neuronima**. Strukture u koje povezujemo te komponente nazivamo **mrežom**.

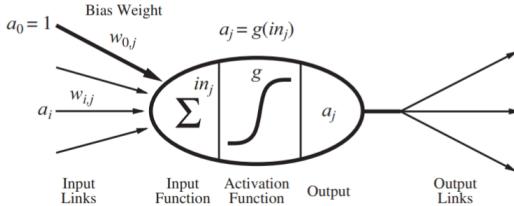
Osobine svake neuronske mreže su utvrđene **topologijom i osobinama pojedinačnih neurona**. Nauka koja se bavi izučavanjem sistema baziranih na osobinama neurona i topologijama neuronskih mreža se naziva **računarska neuronauka** (engl. *computational neuroscience*).

Neuronske mreže imaju mogućnost distribuiranih proračuna. Pored toga, imaju i otpornost na šum kod podataka (**robustnost**). U idealnom slučaju radimo sa podacima koje nemaju šumove jer podaci koji imaju neku vrstu šuma ne predstavljaju dobru reprezentaciju problema kojeg rješavamo. Zbog osobine robustnosti neuronskih mreža, ovakve vrste podataka neće uticati puno na tačnost predikcije neuronske mreže. Najvažnija karakteristika neuronskih mreža jeste sposobnost učenja iz podataka.

Pored neuronskih mreža, postoje i neki drugi sistemi koji imaju mogućnost učenja iz podataka, poput **Bayesove mreže**. Međutim, neuronske mreže postižu dosta bolje rezultate zbog čega su one popularnije.

4.5 Model neurona

U nastavku će biti prikazan drugačiji model neurona, iako su koncepti potpuno isti kao i za McCulloch-Pits-ov model.



Slika 4.2: Model neurona

Imamo osnovne funkcionalnosti predstavljene ranijim modelom. Ulazi se sada nazivaju **aktivacijama**, te se označavaju sa a_i . Svaka aktivacija ima neki koeficijent ili težinu (engl. *weight*) $w_{i,j}$ koji označava snagu i smjer konekcije (sinapsa). Ova vrijednost se često izražava u opsegu $[0, 1]$. Pored toga, koristi se i opseg $[-1, 1]$ gdje koeficijent može da ima određeni smjer što dalje utiče na rad same mreže. Ovaj koeficijent se podešava prilikom konfiguracije neuronske mreže. Otklon (engl. *bias*) je predstavljen koeficijentom a_0 i predstavlja neku konstantnu vrijednost koja može biti različita od neurona do neurona. Otklon može imati efekat smanjenja ili povećanja uticaja ulaza na aktivacijsku funkciju. Prva funkcija unutar tijela neurona je funkcija sume koja sumira ulazne vrijednosti skalirane sa odgovarajućim koeficijentima. Ova funkcija se većinom označava sa

$$in_j = \sum_{i=0}^n w_{i,j} a_i$$

Vrijednost sume se dalje proslijeđuje na **aktivacijsku funkciju** g . Rezultat aktivacijske funkcije predstavlja **izlaz** koji je zadan izrazom

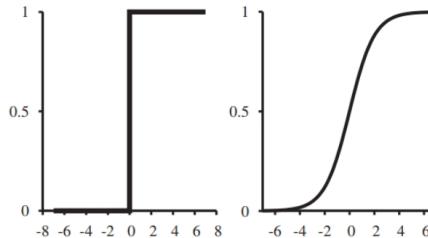
$$a_j = g(\sum_{i=0}^n w_{i,j} a_i)$$

Dakle, izlaz se označava sa a_j i nosi informaciju dalje kroz sami sistem.

Aktivacijska funkcija osigurava da mreža može predstaviti neku nelinearnu funkciju. Prikazane su dvije najčešće korištene funkcije. Prva funkcija predstavlja **strog prav**. U ovom slučaju, ako su ulazne vrijednosti negativne, izlaz će biti 0. U suprotnom slučaju, dobija se izlaz koji ima vrijednost 1. Ova funkcija ne daje zadovoljavajuće rezultate jer nema puno varijacija preko kojih mreža može naučiti. Još jedan problem ove funkcije predstavlja činjenica da izvod nije definisan za $x = 0$.

Drugi primjer predstavlja logističku **sigmoid funkciju** koja ima sasvim drugačije ponašanje i osobine. U ovom konkretnom slučaju, vrijednost

funkcije se definiše u odnosu na sami ulaz, te na izlazu može da primi bilo koju vrijednost koja se nalazi na grafu.



Slika 4.3: Aktivacijske funkcije

Ukoliko imamo strogi prag, onda takav neuron nazivamo **perceptronom**. S druge strane, ako koristimo neku logističku funkciju, onda takav neuron nazivamo **sigmoid perceptronom**.

Nakon što je utvrđen model neurona, potrebno je definisati **topologiju mreže**, odnosno, povezati neurone na odgovarajući način. Postoje dvije osnovne strukture koje se nalaze u praksi koje će se predstaviti u nastavku.

Mreže bez povratnih veza (engl. *feed-forward network*) formiraju topologiju unaprijed. Navedeno znači da se veze formiraju samo u jednom smjeru od jednog neurona ka drugom. Stoga, mreža se može predstaviti direktnim acikličnim grafom jer ne postoji povratna sprega. Ne postoji **interni stanje** što znači da se izlaz formira isključivo u odnosu na trenutne ulaze.

Mreže sa povratnim vezama (engl. *recurrent network*) predstavljaju mreže dosta popularne u posljednje vrijeme. Kod ovih mreža postoji povratna sprega koja može da dovodi izlaze iz jedne neuronske mreže na ulaz te iste mreže ili na neki njen drugi sloj. Ovakva mreža mora imati mogućnost da pamti i ima neko interni stanje. Obično se radi o dinamičkom sistemu koji može biti u stabilnom stanju ili imati određene oscilacije ili biti u stanju haosa. Stanje sistema ovisi o geneiranom izlazu. Rekurentne mreže se još i nazivaju **sistemima sa kratkom memorijom** (*short-term memory*).

4.6 Mreže bez povratnih veza

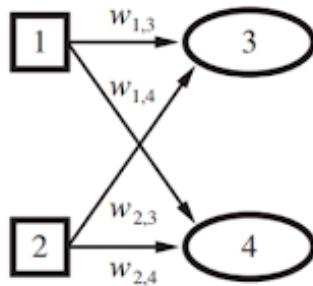
Mreže bez povratnih veza se obično organizuju u **nivoima ili slojevima** (engl. *layers*) na način da jedan neuron može da prihvati ulaze samo ih prethodnog sloja.

Razlikujemo dvije vrste:

1. Naprijed usmjerena jednoslojna mreža (engl. *single layer feed-forward network*)
2. Naprijed usmjerena višeslojna mreža (engl. *multiple layer feed-forward network* ili *Multiple Layer Perceptron (MLP)*)

Naprijed usmjerena jednoslojna mreža

Naprijed usmjerena jednoslojna mreža se naziva još i **perceptron mreža**. Ulazi su direktno povezani na same izlaze što znači da nemamo skrivenog sloja. Arhitektura se definiše kao broj neurona i način njihovog povezivanja. Sljedećim primjerom je prikazan još jedan način predstavljanja neuronske mreže, gdje 1 i 2 predstavljaju ulaze, a 3 i 4 predstavljaju neurone. Između svake konekcije postoje određeni koeficijenti označeni adekvatnim nazivom ($w_{1,3}$ predstavlja vezu između ulaza 1 i neurona 3).



Slika 4.4: Perceptron mreža

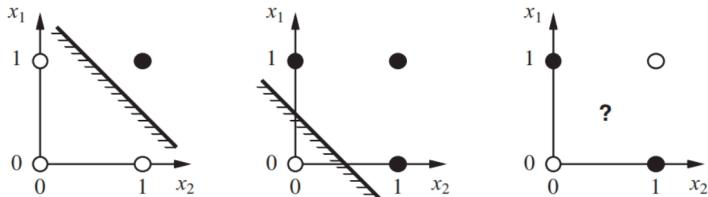
Perceptron mreža sa m izlaza se može posmatrati kao mreža koja ima m odvojenih mreža (m mogućih kombinacija) iz razloga što jedan koeficijent utiče na samo jedan izlaz.

Prikazan je jednostavan primjer iz logičkog dizajna gdje imamo dva ulaza i dva izlaza. Ulazi su označeni sa x_1 i x_2 , a izlazi su označeni sa y_3 i y_4 . Tabelom je prikazana funkcija binarnog sabiranja gdje na izlazu imamo sumu i prenos. Potrebno je odrediti da li pomoću predstavljenog modela neuronske mreže možemo klasificirati funkciju binarnog sabiranja.

x_1	x_2	y_3 (carry)	y_4 (sum)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Slika 4.5: Binarno sabiranje

Izlaze čemo predstaviti na grafu zbog lakšeg posmatranja problema.

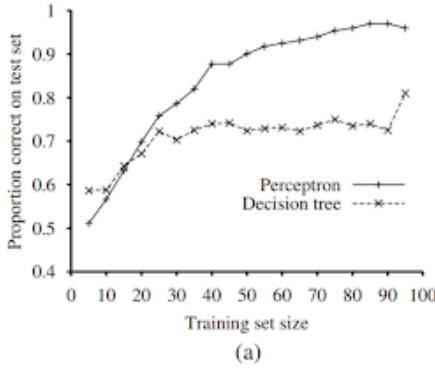


Slika 4.6: Linearna klasifikacija operacija

Prvi graf predstavlja operaciju AND, dok drugi predstavlja operaciju OR. Ove operacije se mogu klasificirati linearnim klasifikatorima koji su označeni ravnim linijama nad datim grafovima. Međutim, za operaciju prikazanu trećim grafom je vizeulno nemoguće pronaći linearnu funkciju koja će razdvojiti ova dva skupa (crne i bijele tačke na grafu).

Ukoliko ponovo pogledamo tabelu binarnog sabiranja na slici 4.5., možemo uočiti da je carry označen sa AND operacijom, a suma odgovara XOR operaciji. Na ovom primjeru uočavamo ograničenja jednoslojnih mreža. Da bismo uspješno mogli klasificirati ovaj problem, potrebno je imati neku nelinearnost što predstavlja složeniji problem. Iako se javlja ovaj nedostatak, to ne znači da su perceptron mreže beskorisne. Ove mreže se mogu koristiti za rješavanje nekih možda čak i težih problema nego što je problem klasificiranja XOR operacije.

Jedan od primjera gdje se ove mreže mogu koristiti je problem izračunavanja **Majority funkcije** koja može da odredi neki izlaz na osnovu vrijednosti većine elemenata. U ovom slučaju, problem ima 11 ulaza gdje je potrebno odrediti izlaz. Izlaz će biti 0 ukoliko većina ulaznih podataka ima neku false vrijednost. U suprotnom slučaju, izlaz će biti 1.

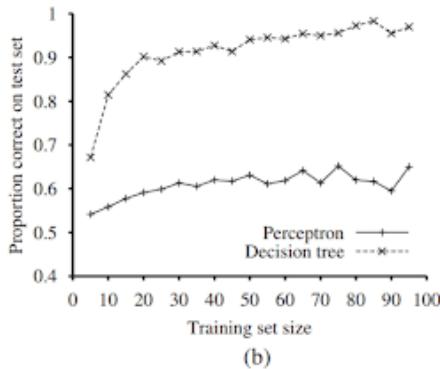


(a)

Slika 4.7: Majority funkcija

Kod ovog problema, na grafu je prikazan broj elemenata nad kojima je vršen trening u odnosu na tačnost samog modela. Dvije linije predstavljene na grafu prikazuju dva moguća rješenja i to uz korištenje **perceptron** i uz korištenje algoritma **stabla odluke** (engl. *decision tree*). Decision tree algoritam predstavlja određeni problem uz pomoć drveta, te se na odnosu određenih parametara donosi neko rješenje. U ovom slučaju, perceptron daje značajno bolje rezultate nego metoda stabla odluke koja je nastala znatno kasnije.

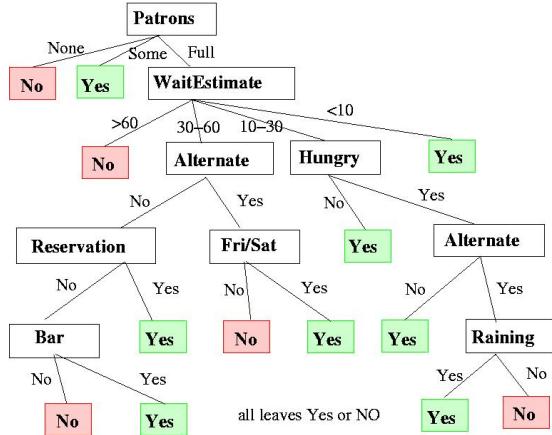
Drugi primjer je problem čekanje na hranu u restoranu. U ovom problemu se javljaju određeni parametri u odnosu na to da li smo gladni, postoji li gužva i slično. Potrebno je donijeti odluku da li se isplati čekati na hranu ili jednostavno otići u neki drugi restoran.



Slika 4.8: Čekanje na hranu u restoranu

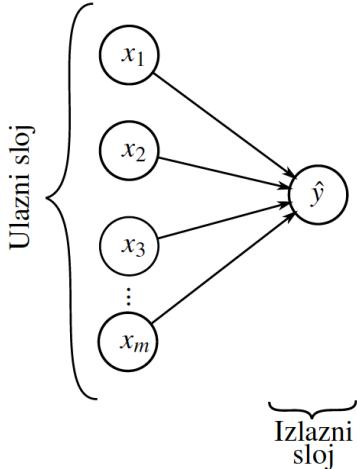
Na ovom primjeru je prikazan algoritam stabla odluke. Na osnovu odgovora na određena pitanja, dolazimo do nekog drugog podpitanja ili dolazimo do finalnog

odgovora.



Slika 4.9: Algoritam stabla odluke

Dakle, naprijed usmjerene jednoslojne mreže predstavljaju jednu od najjednostavnijih mreža. Na sljedećem modelu je prikazan primjer ovakve mreže.

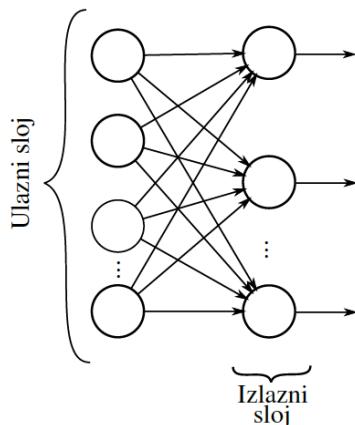


Slika 4.10: Model naprijed usmjerene jednoslojne mreže

Na primjeru imamo nekoliko ulaza predstavljenih sa x_i i oni predstavljaju atribute koje uzimamo za određeni problem. Recimo da trebamo donijeti odluku da li odobriti ili odbaciti zahtjev za izdavanje pozajmnice. U ovom

slučaju, atributi mogu biti dob aplikanta, godišnja plata, bračni status, posjedovanje imovine i slično. Ovakvih parametara možemo imati proizvoljno mnogo. Numeričke vrijednosti ovih atributa su ulazi u samu mrežu. Kao što je već rečeno, neuroni u ulaznom sloju nemaju mogućnost računanja pa je izlazni sloj jedini sloj unutar ovakvih vrsta neuronskih mreža.

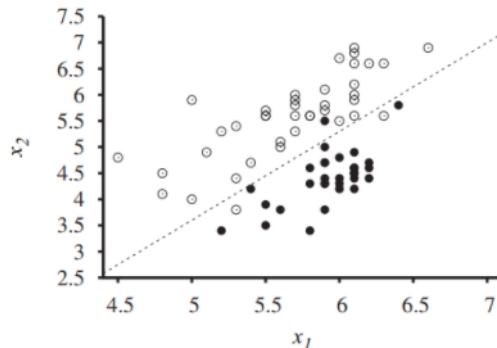
Postoje različite konfiguracije ovakve vrste mreže. Na sljedećem modelu se može vidjeti da za neki broj ulaza možemo imati više različitih izlaza. U primjeru je svaki ulazni sloj je povezan sa svakim neuronom, pa ovaj model nazivamo **potpuno povezanim neuronskim mrežom** (engl. *fully connected neural network*).



Slika 4.11: Potpuno povezana neuronska mreža

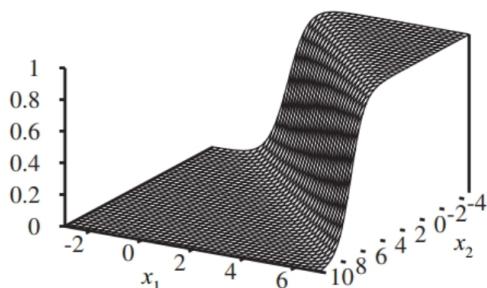
Dakle, kada su neuroni u jednom sloju povezani sa svakim neuronom u sljedećem susjednom sloju, onda za takvu mrežu kažem da je ona potpuno povezana. Pored ovoga, postoje i mreže koje ne moraju biti potpuno povezane i u tom slučaju imamo naizmjenično uključivanje i isključivanje neurona. Ovakve mreže se javljaju kod kompleksnijih konfiguracija.

U nastavku će biti prikazan primjer jednoslojne mreže. Podaci su predstavljeni preko dvije promjenljive x_1 i x_2 koje predstavljaju veličinu tijela i površinu talasa za seizmičke signale. Potrebno je odrediti koje tačke pripadaju klasi zemljotresa (kružići), a koje pripadaju nuklearnoj reakciji (crne tačke). Zadatak klasifikatora je da se nađe funkcija koja će napraviti **granicu** (engl. *decision boundary*) između ova dva skupa. Tražena granica može biti ili linija ili površina (ako imamo više dimenzija).



Slika 4.12: Primjer jednoslojne mreže

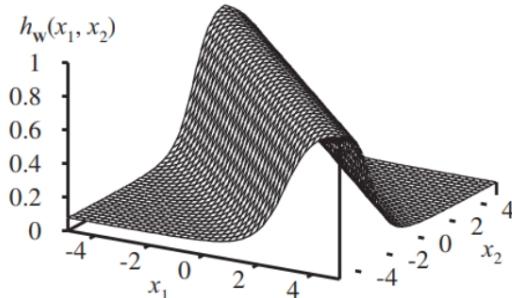
Za rješavanje koristimo neuronsku mrežu. Definišemo aktivacijsku funkciju koja se prikazuje na sljedeći način:



Slika 4.13: Aktivacijska funkcija

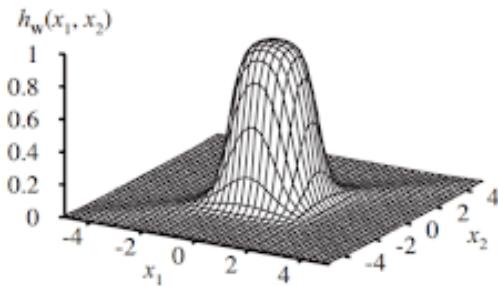
Dakle, imamo dvije dimenzije x_1 i x_2 i izlaz koji se može kretati od 0 do 1. Ovim je zadana funkcija praga i ima oblik sigmoid funkcije. Pored toga, svaki neuron ima funkciju gubitka što znači da za svaki neuron možemo da generišemo različit broj funkcija gubitka.

Kombinovanjem dvije funkcije praga koje su okrenute suprotno jedna drugoj, možemo dobiti funkciju koju još nazivamo **greben** (engl. *ridge*). U našem primjeru, dobijamo sljedeći greben:



Slika 4.14: Funkcija grebena

Dakle, predstavljena je kombinacijom dvije aktivacijske funkcije i koriste se dva neurona. Ako su izlazi iz ta dva neurona suprotne vrijednosti, možemo generisati datu funkciju grebena. Također, korištenjem više neurona (npr. četiri) sa više aktivacijskih funkcija različitih predznaka možemo doći do funkcije **ispupčenja**.



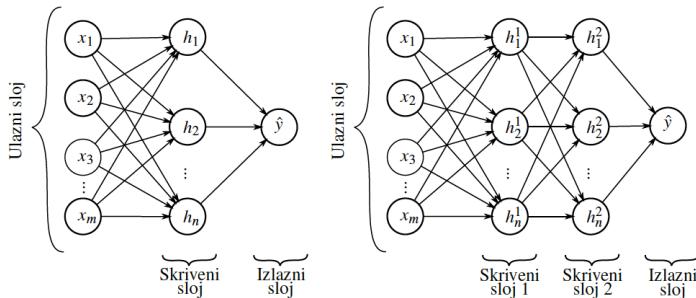
Slika 4.15: Funkcija ispupčenja

Povećavanjem broja neurona u jednom sloju možemo doći do neke proizvoljne funkcije koja će uspješno podijeliti skup podataka na odgovarajuće vrijednosti. Sa jednim skrivenim slojem sa puno neurona moguće je predstaviti bilo koju kontinualnu funkciju, dok je sa više skrivenih slojeva moguće prezentirati i prekide u funkciji.

Naprijed usmjerena višeslojna mreža

Naprijed usmjerene višeslojne mreže su zasnovane na sličnom konceptu. Dakle, imaju određeni broj ulaza određenim atributima i izlazni sloj.

Razlikuju se po tome što imaju i **skriveni sloj** kojih može biti jako puno. Teoretski nema ograničenja, ali ono se javlja u performansama takve mreže u procesu treniranja. Što imamo veći broj slojeva, imati ćemo kompleksnije proračune čime će izračunavanje tih operacija zahtijevati određeno vrijeme i proces će biti skuplj.



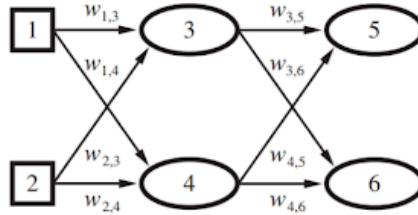
Slika 4.16: Naprijed usmjerena višeslojna mreža

Po broju slojeva, ove mreže možemo podijeliti na **plitke** (engl. *shallow*) i **duboke** (engl. *deep*). Plitke neuralne mreže imaju manji broj skrivenih slojeva, dok duboke imaju veći broj. Svaki skriveni sloj je predstavljen brojem neurona koji se označava sa h_n s tim što različiti skriveni slojevi ne moraju imati isti broj neurona.

McCulloch i Pitts su 1943. godine predvidjeli ograničenja njihovih modela, te su teoretsali o ideji povezivanja više neurona u složenije mreže. Međutim, u to vrijeme niko nije znao kako trenirati takve strukture. Tek 2012. godine dolazimo u vrijeme kada imamo mogućnost da uspješno treniramo mreže koje imaju 8 skrivenih slojeva i ovom godinom se ulazi u eru dubokog učenja.

Moguće je dati naziv nekoj topologiji ili strukturi na osnovu broja neurona koje se nalaze u ulaznom, skrivenom i izlaznom sloju. Recimo da ulazni sloj ima 10 ulaznih atributa, 6 skrivenih neurona u skrivenom sloju i 3 izlazna neurona, onda se takva mreža naziva 10-6-3 mreža. Dakle, općenito za n ulaznih neurona, h_1 neurona u prvom i h_2 neurona u drugom sloju, takvu mrežu nazivamo $n-h_1-h_2$ mrežom.

U nastavku je prikazan primjer naprijed usmjerene višeslojne mreže.



Slika 4.17: Model naprijed usmjerenje višeslojne mreže

Mreža je dosta slična jednoslojnoj mreži, s tim što sada postoje još dva neurona 5 i 6. Pravila, što se tiče koeficijenata i veza, su ista kao i u jednoslojnoj mreži. Javljuju se nove veze od neurona 3 i 4 ka neuronima 5 i 6 koje se dobijaju iz aktivacija neurona 3 i 4. Potrebno je izračunati izlaze neurona 5 i 6.

Izlaz iz neurona 5 dobijamo na način da uzmemo otklon $w_{0,5}$ koji je vezan za neuron 5, te dodamo skaliranu vrijednost aktivacija a_3 i a_4 .

$$a_5 = g(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4)$$

Aktivacije a_3 i a_4 dobijamo na isti način.

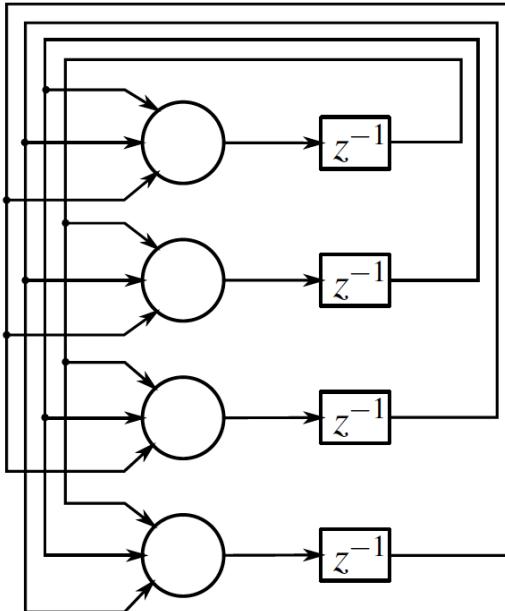
$$a_3 = g(w_{0,3} + w_{1,3}a_1 + w_{2,3}a_2)$$

$$a_4 = g(w_{0,4} + w_{1,4}a_1 + w_{2,4}a_2)$$

Naprijed usmjerna višeslojna mreža nema povratnu spregu što sprječava performanse ove mreže.

4.7 Mreže sa povratnim vezama (rekurentne mreže)

Rekurentne mreže se razlikuju po tome što izlaze možemo ponovo dovesti do samih ulaza na istu mrežu, te na ovaj način ponovo trenirati mrežu sa koeficijentima koji su nađeni u međuvremenu. Imaju barem jednu povratnu spregu. Najjednostavniji model je prikazan na slici:

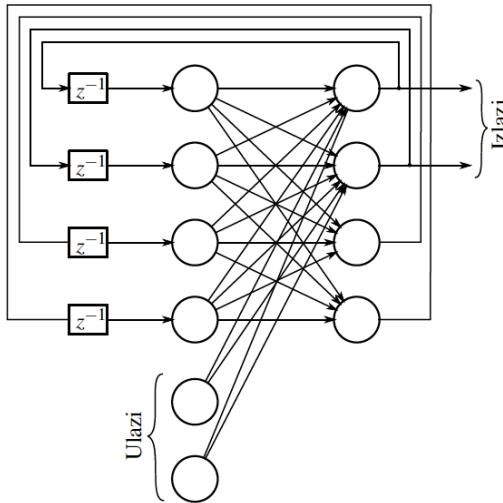


Slika 4.18: Mreža sa povratnim vezama

Na modelu se vidi **kolo zadrške** koje u ovom slučaju pamti samo jedan prethodni element izlaza (da je bilo z^{-2} , pamtila bi se dva prethodna elementa). Kolo zadrške je predstavljeno **z -transformacijom**. U ovom primjeru nema skrivenog sloja, a izlazi se dovode na ulaze ulaznih čvorova. Na primjeru imamo četiri izlaza. Međutim, ne dovodimo sva četiri izlaza na svaki ulaz, već samo ona tri različita izlaza u odnosu na dati čvor.

Postojanje povratne sprege u neuronskim mrežama ima značajan uticaj na proces učenja, kao i na performanse mreže. Korištenje povratne sprege uslovljeno je korištenjem kola zadrške koje se označava sa z^{-1} . Kolo zadrške uvodi jedan trenutak kašnjenja za određeni signal i može uzrokovati nelinearno dinamičko ponašanje mreže.

Sljedećim primjerom je prikazana kompleksnija struktura rekurentne mreže. Ova struktura podrazumijeva jedan skriveni sloj.



Slika 4.19: Mreža sa povratnim vezama

U ovom slučaju, povrat imamo samo kod nekih izlaza i oni se ne dovode na sve ulaze.

4.8 Funkcije gubitka

Funkcija gubitka se definiše kao iznos gubitka korisnosti u odnosu na predviđenu vrijednost funkcije $h(x) = \hat{y}$ kada je tačan odgovor $f(x) = y$. Dakle, treba da dâ neku ocjenu razlike između stvarne vrijednosti funkcije koju tražimo $f(x) = y$ i predikcije koju smo pronašli $h(x) = \hat{y}$. Funkcija predikcije predstavlja odabranu hipotezu iz prostora svih hipoteza. Funkcija gubitka se označava sa $L(x, y, \hat{y})$ ili $L(y, \hat{y})$. Pored toga, vrijedi pravilo da funkcija gubitka mora biti jednaka nuli kada računamo ovu vrijednost u odnosu na dvije iste funkcije, odnosno $L(y, y) = 0$. U suprotnom, matematički model nije ispravno postavljen.

Recimo da tražena vrijednost funkcije $y(x)$ za neku određenu tačku daje vrijednost $y(x) = 137.035999$. Ako nađemo funkciju predikcije $h(x)$ koja za određenu tačku daje vrijednost $h(x) = 137.036$, postavlja se pitanje da li je funkcija predikcije dovoljno dobra ili ne. U problemima optimizacije neuronskih mreža, isključivo vršimo minimizaciju funkcije gubitka.

Postoji više načina izračunavanja funkcije gubitka. Prvi način je **apsolutna**

vrijednost gubitka (L1) data funkcijom

$$L_1(y, \hat{y}) = |y - \hat{y}|$$

dok je drugi način putem **kvadratne vrijednosti greške** definiranom funkcijom

$$L_2(y, \hat{y}) = (y - \hat{y})^2$$

Možemo definisati **generalizirani gubitak** kada su poznati svi ulazno/izlazni primjeri za neki problem, odnosno, potrebno je definisati sve kombinacije ulaza i izlaza. Za veliki broj problema nije moguće definisati ovaj gubitak zbog kompleksnosti.

Neka su ulazno/izlazni primjeri označeni sa ε . Generalizirani gubitak se definiše na sljedeći način:

$$\text{GenLoss}_L(h) = \sum_{(x,y) \in \varepsilon} L(y, h(x)) P(x, y)$$

Dakle, generalizirani gubitak se izračunava u odnosu na nađenu hipotezu. Za svaki pojedinačni element u skupu ε ćemo izračunati gubitak, te isti pomnožiti sa vjerovatnoćom koja će biti poznata jedino ako su nam poznati svi elementi u posmatranom prostoru. Kada ne znamo sve moguće kombinacije, ovu vjerovatnoću ne možemo izračunati.

Kada nađemo generalizirani gubitak, izračunati ćemo najbolju moguću hipotezu u tom prostoru. Ova hipoteza treba da bude minimalna očekivana vrijednost generaliziranog gubitka:

$$h^* = \operatorname{argmin}_{h \in H} \text{GenLoss}_L(h)$$

gdje je H skup hipoteza.

Generalizirani gubitak se računa iz razloga što možemo da napravimo jedan model kojem ćemo poslati različite podatke iz istog skupa. Za ove podatke ćemo dobiti različite funkcije od kojih ćemo odrediti koja je najbolja na opisani način.

Ako nije poznata vjerovatnoća distribucije $P(x, y)$, moguće je definisati samo **empirijski gubitak** na skupu primjera koji označavamo sa E :

$$\text{EmpLoss}_{L,E}(h) = \frac{1}{N} \sum_{(x,y) \in E} L(y, h(x))$$

Najbolja estimirana (procjenjena) hipoteza se računa kao minimalna očekivana vrijednost empirijskog gubitka:

$$\hat{h}^* = \operatorname{argmin}_{h \in H} \text{EmpLoss}_{L,E}(h)$$

Postoje četiri razloga zašto će se pronađena funkcija \hat{h}^* razlikovati od stvarne funkcije f :

1. **Funkcija f je neostvariva**, odnosno, moguće je da se ne nalazi u prostoru hipoteza ili može da se nalazi, ali da postoje druge funkcije koje su efikasnije.
2. **Funkcija f varira** što znači da algoritam učenja može vratiti drugu funkciju za drugi skup primjera, iako su ti primjeri nastali sa istom funkcijom f .
3. **Funkcija f je nedeterministička funkcija** ili je pod uticajem šuma, što znači da vrijednost funkcije može biti različita za isti ulaz x .
4. **Računska kompleksnost** opisuje nerješivost problema ukoliko je prostor hipoteza složen jer u tom slučaju možemo pretraživati samo jedan dio prostora (lokalno pretraživanje).

U našem slučaju, možemo praktično da radimo samo sa empirijskim gubitkom jer se generalizirani gubitak definiše na širi način.

Kada govorimo o učenju iz podataka, može se podijeliti na dva pristupa:

1. **učenje iz malog skupa podataka** - nekoliko desetina do nekoliko hiljada podataka
2. **učenje iz velikog skupa podataka** - milioni podataka

Jasno je da je bolje učiti iz velikog skupa podataka i veže se za **big data** koncept.

Predavanje 6

5.1 Korištenje podataka za proces učenja

Postoje različite vrste podataka koje koristimo za treniranje različitih modela za učenje. Podaci se dijeli na određene skupine iz kojih se dalje uči. Postavlja se pitanje kako napraviti tu podjelu podataka, odnosno, skupa podataka (engl. *dataset*).

Na prethodnim poglavljima su se spominjali osnovni klasifikatori, među kojima je i K-NN klasifikator. Kod ovog klasifikatora imamo dva parametra koji određuju performanse klasifikatora: K i distanca. Ovi parametri se nazivaju **hiperparametrima** i njih biramo prije samog treniranja na određene načine bez učenja iz podataka. Ukoliko se napravi dobar izbor hiperparametara, dobiti ćemo bolje performanse samog modela (tačnost i gubitak). Vrsta i skup hiperparametara koji biramo se direktno veže za problem koji rješavamo. Najbolje rješenje bi bilo da isprobamo sve moguće varijacije i na osnovu tih rezultata odredimo najbolji model. Međutim, ovo ujedno predstavlja i najskuplje rješenje u smislu računarskih resursa i vremena.

Postoje četiri osnovne ideje podjele podataka.

Prva ideja jeste da uzmemmo čitav skup podataka i da nad tim skupom pravimo treniranje. Ako je cilj da postignemo najveću tačnost nad skupom podataka za treniranje, ovo je efikasan način podjele podataka. Međutim, vrlo rijetko pravimo model koji daje veliku tačnost isključivo nad skupom za treniranje. Model će imati jako loše rezultate nad skupom podataka koje nikad nije "vidio".

Druga ideja je podjela podataka na skup za treniranje i testiranje. Kod ovog modela se postavlja pitanje nad kojim skupom ćemo izabrati hiperparametre. Jedno rješenje je da uzmemmo neke proizvoljne parametre, te istreniramo model nad njima. Na kraju nam preostaje samo da testiramo taj model nad skupom za testiranje. Međutim, model mora biti u mogućnosti da generalizira nad "neviđenim podacima". Stoga, ukoliko postignemo jako veliku tačnost nad skupom za treniranje, ne mora značiti da će taj model imati dobre

performanse nad skupom za testiranje. Vrlo često se proces treniranja prekida u trenutku kada model ne postiže tačnost preko 99% kako bi model imao mogućnost generaliziranja nad testnim podacima.

Treća ideja (**hold-out strategija**) jeste da se podaci podijele na tri skupa: treniranje, validacija i treniranje. U ovom slučaju, hiperparametri se biraju na osnovu skupa za validaciju. U praksi se testiranje modela dešava jednom na kraju nakon odabira zadovoljavajućih hiperparametara. Pored toga, većinom se 80% podataka uzima za treniranje i validaciju, dok se preostalih 20% uzima za testiranje. Skup za validaciju je značajno manji od skupa za treniranje i testiranje, te se za ovaj skup podataka uzima 5% do 10% podataka.

Četvrda ideja je **K-fold strategija** ili **unakrsna validacija** (engl. *cross-validation*) koja se koristi za manje skupove podataka. Ova strategija podrazumijeva podjelu podataka na dva podskupa: treniranje i testiranje. Zatim ćemo skup za treniranje podijeliti u određen broj grupa. U prvoj iteraciji, posljednja grupa predstavlja skup za validaciju. U narednoj iteraciji će to biti pretposljednja grupa i tako sve dok ne izvršimo validaciju nad svim manjim podgrupama. Zatim se koristi strategija srednje vrijednosti gdje biramo model koji prikazuje najbolje rezultate nad različitim skupovima podataka, te na osnovu toga odredimo hiperparametre. Ova tehnika je skupa jer je proces treniranja prilično skup kod većih skupova podataka (broj treniranja je jednak broju odabranom broju K).

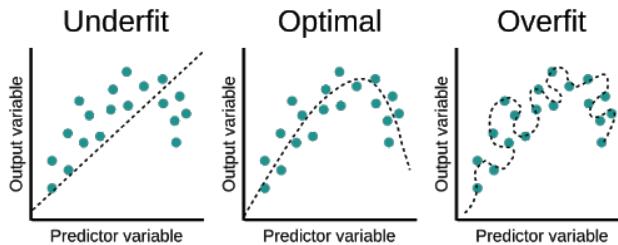
Na sljedećem primjeru je prikazan primjer za K=5 (skup za treniranje dijelimo na pet grupa).

Cijeli skup podataka				
Treniranje				Testiranje
Grupa 1	Grupa 2	Grupa 3	Grupa 4	Grupa 5
Grupa 1	Grupa 2	Grupa 3	Grupa 4	Grupa 5
Grupa 1	Grupa 2	Grupa 3	Grupa 4	Grupa 5
Grupa 1	Grupa 2	Grupa 3	Grupa 4	Grupa 5
Grupa 1	Grupa 2	Grupa 3	Grupa 4	Grupa 5
Grupa 1	Grupa 2	Grupa 3	Grupa 4	Grupa 5

Slika 5.1: K-fold strategija

U toku procesa treniranja, mogu da se dese dva problema: **overfitting** i **underfitting**.

Problemi su prikazani sljedećim primjerom.



Slika 5.2: Problem treniranja modela

Cilj je pronaći funkciju koja najbolje odgovara modelu kretanja podataka prikazanih na grafu. Kod **underfit** modela je predstavljena linearna funkcija koja ne odgovara raspodjeli podataka. Dakle, underfit model rezultira visokim greškama u predviđanju kako za trening tako i za test podatke. Greške se javljaju iz razloga što relacija između podataka nije dovoljno naučena.

S druge strane, kod **overfit** modela imamo nelinearnu funkciju koja značajno bolje opisuju raspodjelu podataka. Međutim, vrlo često ne želimo da nađemo funkciju koja idealno odgovara skupu za treniranje, već da ima mogućnost generalizovanja. To znači da bi funkcija za svaku narednu tačku trebala da ima što manji gubitak. Overfit model daje vrlo nisku grešku predviđanja na podacima za trening, ali vrlo visoku nad podacima testa. Ovaj model je prosto memorizirao podatke i nema mogućnost generaliziranja.

Optimalni model daje dosta manji gubitak u odnosu na prethodna predstavljena dva modela nad skupom za testiranje. Pored toga, ova funkcija će bolje reprezentirati neki novi skup podataka (za treniranje ili za validaciju).

Navedene probleme možemo izbjegići dobrim izborom hiperparametara ili sa prestankom treniranja u određenoj epohi.

5.2 Linearni klasifikatori

Vraćamo se na već spomenute linearne klasifikatore. Linearni klasifikatori spadaju u grupu tzv. **Metoda sa parametarskim pristupom**. Ove metode podrazumijevaju da postoji skup fiksnih parametara (srednja vrijednost i standardna devijacija) koji mogu **modelirati vjerovatnoću** nad nekim skupom podataka. U nastavku su predstavljena tri različita pogleda na linearne klasifikatore: algebarski, vizuelni i geometrijski. Svaki od ovih načina ima svoje karakteristike i koriste se u analizi samih klasifikatora.

Algebarski pogled

Radi pojednostavljenja, podrazumijevati ćemo da imamo slike kao skup podataka. Sljedeća slika podsjeća na sliku iz skupa podataka **CIFAR10** jer ima dimenzije 32x32 i ima RGB kanal boje.



Slika 5.3: RGB slika dimenzija 32x32

Da bismo trenirali neuronsku mrežu (ili klasifikator) sa ovim podacima, sliku predstavljamo kao vektor. Sliku iz tri dimenzije pretvaramo u vektor množenjem bitnih parametara. U ovom slučaju to je širina i visina slike, te broj kanala. Dakle, ova slika će se predstavljati vektorom sa $32 \cdot 32 \cdot 3 = 3072$ dimenzije.

Dalje, potrebno je definisati neki klasifikator koji će imati mogućnost da klasificira sliku u određene klase. Broj klasa je definisan skupom podataka, te ćemo za ovaj primjer uzeti CIFAR10. Dakle, svaka slika će biti povezana sa nekom labelom koja je fiksna i predefinisana.

Klasifikator možemo definisati kao neku funkciju. Ovdje je definišemo kao neku općenitu funkciju sa dva parametra.

$$f(x, W)$$

Parametar x predstavlja ulaze, odnosno **vektor ulaznih značajki** (engl. *input feature vector*) gdje je jedna slika prikazana jednim vektorom. Ukoliko imamo N slika, onda ćemo imati N takvih vektora. Sljedeći parametar je W koji obično dolazi u obliku matrice i on predstavlja **parametre** ili **težine** (engl. *weights*). Dakle, funkcija će predstavljati linearan ili nelinearan oblik kombinacije ova dva parametra kako bismo dobili željeni rezultat. Cilj funkcije jeste da mapira vektor značajki u labele (numeričke vrijednosti). Postavlja se pitanje kako definisati ovaku funkciju.

U nastavku će fokus biti na linearnim klasifikatorima, ali sve isto vrijedi i za nelinearne klasifikatore pri čemu se mijenja funkcija klasifikatora.

Najobičniji linearni klasifikator možemo definisati kao umnožak između matrice W i x uz dodavanje nekog otklona (engl. *bias*) b .

$$f(x, W) = Wx + b$$

Postavljaju se pitanja kako predstaviti ulaze u ovaj sistem, koliko ima ulaza/izlaza, te koja je dimenzionalnost matrice W .

Jedan ulaz (jedna slika) se predstavlja na već opisani način. Dakle, množimo bitne parametre slike kao što su širina, visina i broj kanala. U našem slučaju, imati ćemo vektor od 3072 atributa gdje će svaki atribut biti vezan za vrijednost jednog piksela unutar slike. Izlaz će se direktno vezati za broj klasa. Kako koristimo CIFAR10 skup podataka, imati ćemo 10 izlaza. Dimenzionalnost matrice W je direktno vezana za broj ulaza i izlaza. Za ovaj primjer, matrica ima dimenzije 10×3072 .

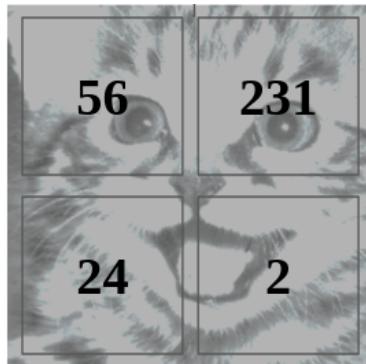
Otklon (bias) opisuje koliko se dobro model treba priklanjati nekom skupu podataka, pa bias može biti veći ili manji. Model sa visokim otklonom neće se previše prikloniti skupu podataka. Sa druge strane, model sa niskim otklonom će se uskladiti (odnosno prikloniti) sa skupom podataka. Na ovaj način definišemo ili bolju ili lošiju generalizaciju modela. Ovaj parametar određujemo apriori i veže se za broj izlaza (ukoliko imamo 10 izlaza, bias će biti vektor od 10 elemenata).

Vrlo često se funkcija linearног klasifikatora predstavlja u pojednostavljenom obliku.

$$f(x, W) = Wx$$

U ovom slučaju, otklon nije nestao već smo ga modifikacijom pridružili matrici W .

U nastavku je prikazan primjer izračunavanja izlaza. Primjer se odnosi na sljedeću sliku:



Slika 5.4: Slika dimenzija 2x2

Prikazana slika ima dimenzije 2x2 (4 piksela ukupno) sa datim vrijednostima. Ova slika predstavlja ulaz. Da bismo izračunali izlaz iz apstraktne funkcije $f(x, W) = Wx + b$, moramo izabrati vrijednosti parametara W i b .

Recimo da sliku predstavljamo kao vektor kolone gdje navodimo sve vrijednosti piksela. Zatim, nasumično izaberemo matricu W sa vrijednostima koji mogu dati dobar rezultat ili ne. Vrijednosti matrice se formiraju nakon procesa učenja, te su ovdje date samo kao ilustrativni primjer. Otklon je vektor vezan za broj klase i isti biramo nasumično (u ovom slučaju, uzeti ćemo da imamo tri klase na izlazu).

Kako je jednostavna slika na ulazu, tako je pojednostavljen i izlaz. Samim time, uzmimo da na izlazu nemamo 10 već 3 klase (recimo da imamo klase mačka, pas i brod). Navedeno znači da će se za matricu W dimenzionalnost kolona definisati u odnosu na broj piksela, dok će redovi prikazivati klase. Vrijednost izlaza se računa izvršavanjem operacija početne funkcije.

$$\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix} \cdot \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} + \begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix} = \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}$$

Na osnovu vrijednosti izlaza dobijamo neku informaciju o tome kojoj klasi pripada data slika na ulazu. Iz podataka nije intuitivno odrediti da li slika pripada nekoj od datih klasa. Prvi element predstavlja pripadnost klasi mačka, drugi element pripadnost klasi pas, a treći element pripadnost klasi brod. Potrebno je dodatno odrediti na osnovu čega se klase razlikuju.

Isti problem preko pojednostavljenog modela $f(x, W) = Wx$ prikazujemo na

sljedeći način:

$$\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 & 1.1 \\ 1.5 & 1.3 & 2.1 & 0.0 & 3.2 \\ 0 & 0.25 & 0.2 & -0.3 & -1.2 \end{bmatrix} \cdot \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}$$

Dakle, otklon priključimo matricu W na poziciju zadnje kolone. Pored toga, potrebno je proširiti i ulazni vektor tako da ćemo dodati 1 kao posljednji element.

Za sliku 5.4. smo kao izlazni vektor (engl. *output feature vector*) dobili rezultat klasifikacije:

$$\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}$$

Ako istu ulaznu sliku pokušamo modifikovati na način da ju množimo sa 0.5, postavlja se pitanje da li možemo utvrditi rezultat klasifikacije te slike. Odgovor je potvrđan, te kaže da ukoliko isti model primijenimo nad modifikovanom slikom, dobiti ćemo i predikcije skalirane istim koeficijentom. Samim time, dovoljno je prethodni rezultat pomnožiti sa 0.5:

$$0.5 \cdot \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} = \begin{bmatrix} -48.4 \\ 218.9 \\ 30.97 \end{bmatrix}$$

Generalno možemo reći da ukoliko imamo neki model $f(x, W) = Wx + b$ i ulaze modifikujemo nekim koeficijentom c $f(cx, W) = Wx + b$, na izlazu ćemo dobiti iste predikcije kao nemodifikovan model skaliran istim koeficijentom c :

$$f(cx, W) = W(cx) = c \cdot f(x, W)$$

Iz navedenog donosimo zaključak da su i **predikcije linearne**.

Postoje razni načini prikazivanja matrice W . Jedan od drugih načina jeste da zadržimo oblik matrice koji odgovara ulaznoj slici (u našem slučaju 2×2), s tim da ćemo imati onoliko matrica W koliko imamo klase. Dakle, sada se matrica definije preko tri matrice koje imaju dimenzije kao i ulazna slika, pri čemu su vrijednosti unutar matrice definisane nešto drugačije. Matrica W je prije imala sljedeći oblik:

$$\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}$$

Kako bismo sada definisali prvu matricu W , uzeti ćemo vektor koji je prije predstavljao jedan red matrice W , te ćemo isti organizovati unutar nove

matrice dimenzija 2x2. Prva matrica W ima sljedeći oblik:

$$W_1 = \begin{bmatrix} 0.2 & -0.5 \\ 0.1 & 2.0 \end{bmatrix}$$

Preostale dvije matrice se organizuju na isti način, i one imaju sljedeće oblike:

$$W_2 = \begin{bmatrix} 1.5 & 1.3 \\ 2.1 & 0.0 \end{bmatrix}$$

$$W_3 = \begin{bmatrix} 0 & 0.25 \\ 0.2 & -0.3 \end{bmatrix}$$

Vektor kolone otklona je prije imao sljedeći oblik:

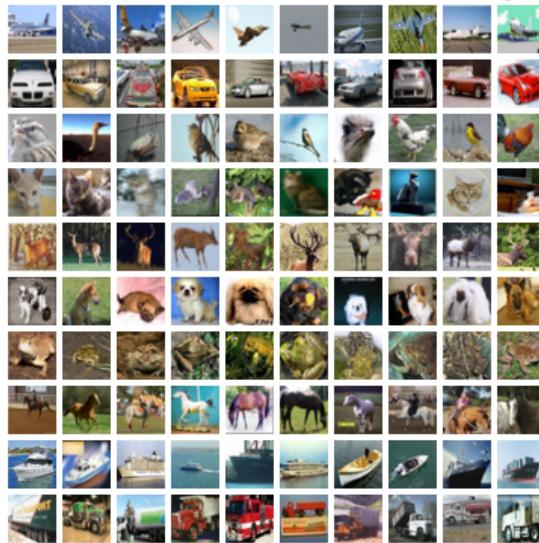
$$\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}$$

Jasno je da će se prva vrijednost vektora otklona dodavati na matricu W_1 , druga vrijednost otklona na matricu W_2 i treća vrijednost otklona na matricu W_3 .

Ova vrsta zapisa matrice W se koristi jer omogućava interesantan način prikaza rezultata. U ovom slučaju, matrica W može da prikaže svaku klasu pojedinačno.

Vizuelni pogled

Uzmimo kao primjer skup podataka CIFAR10.



Slika 5.5: CIFAR10

Nakon određenog procesa učenja, mi možemo da prikažemo šta je to klasifikator uočio na slikama i šta to pomaže da ispravno izvršimo klasifikaciju nekih novih slika. Za ovaj skup podataka, svaka klasa ima definisanu svoju matricu težina W i oni se prikazuju kao slike. Ovi koeficijenti predstavljaju neku usrednjenu vrijednost svih slika unutar skupa podataka za treniranje.

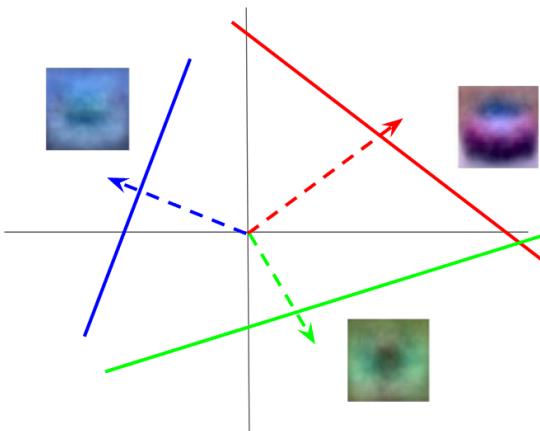


Slika 5.6: Matrice težina klase

Kod primjera automobila, možemo vidjeti da model pamti određeni oblik auta, boju, položaj i vrstu pozadine. Posmatrajući sliku matrice težina za ovu klasu možemo zaključiti da pozicija igra veliku ulogu, te su automobili unutar ovog skupa podataka pretežno centrirani. Ako bismo modelu poslali sliku sa pola automobila, postavlja se pitanje da li bi on bio u mogućnosti da napravi predikciju. Iz ovoga što vidimo, možemo pretpostaviti da ova predikcija ne bi bila najbolja. Kod klase srne, uočavamo da je model osjetljiv na zelenu boju i može da efikasno prepozna objekte u prirodnom ambijentu. Međutim, kod ove klase ne vidimo jasno položaj glave srne što je nastalo iz razloga što na različitim slikama srne gledaju u različitim smjerovima. Dakle, model ima neka ograničenja i ne mora značiti da će za veliku količinu slika dati zadovoljavajuće rezultate. Određene osobine će uticati na formiranje predikcije kod ovakvih modela.

Geometrijski pogled

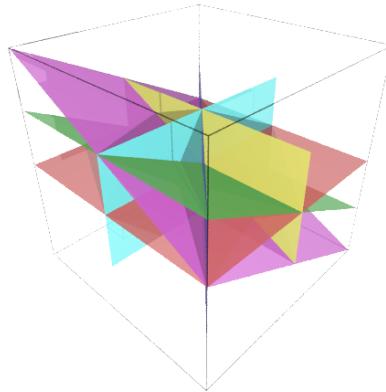
Vrlo često možemo da vizuelno predstavimo predikcije nekog klasifikatora. U nastavku je dat pojednostavljen primjer gdje imamo dvodimenzionalni koordinatni sistem (što znači da možemo prikazati samo dva piksela).



Slika 5.7: 2D vizuelno predstavljanje klasifikatora

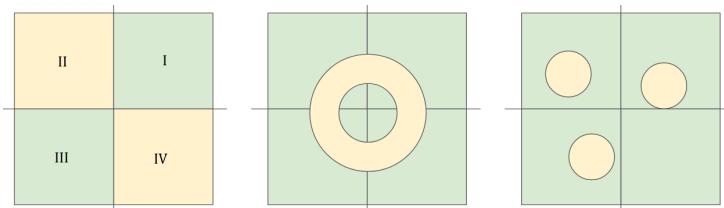
Pomoću 2D Euklidskog prostora, pokušavamo da napravimo klasifikaciju određenih objekata. Recimo za klasu automobila možemo definisati neki linearni klasifikator koji će reći da iznad nekih vrijednosti x i y možemo reći da se nalazi klasa automobila. Ukoliko neka nova slika automobila zauzima položaj u koordinatama koji odgovaraju ovoj klasi, slika će biti ispravno predviđena. U suprotnom možemo reći da neće biti ispravno predviđena. Isti postupak možemo definisati za neke druge klase, s tim što se regije klasa ne bi trebali presijecati.

Kod podataka sa više dimenzija, tačnije bi bilo definisati neki N-prostor gdje će svaka klasa biti predstavljena preko neke hiperravnih koja taj prostor dijeli na dva dijela.



Slika 5.8: Vizuelno predstavljanje klasifikatora preko N-prostora

Linearni klasifikatori imaju ograničenja. Data su tri primjera koja predstavljaju tri različita skupa sa određenim karakteristikama. Podaci se nalaze u samo jednom ili više dijelova ovog prostora.



Slika 5.9: Primjer nedostatka linearnog klasifikatora

U prvom slučaju, imamo dvije klase. Klasa 1 ima elemente u prvom i trećem kvadrantu, dok klasa 2 ima elemente u drugom i četvrtom kvadrantu. Ovakvi podaci se ne mogu podijeliti linearnim klasifikatorom.

Drući primjer sadrži klasu 1 koja je definisana pojasom u određenom opsegu (npr. $1 \leq L_2 \text{ norma} \leq 2$) i klasu 2 koja sadrži ostale podatke. Jasno je da se ni u ovom slučaju ne može naći podjela nekim linearnim klasifikatorom.

U trećem slučaju, imamo tri regije koje označavaju određene podatke i predstavljaju klasu 1, dok ostali podaci pripadaju klasi 2. Ni ovaj primjer se ne može podijeliti linearnim klasifikatorom. U sva tri slučaja zahtijevamo korištenje nekog nelinearnog klasifikatora.

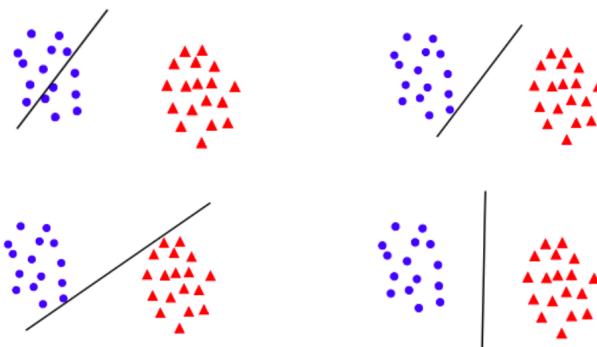
5.3 Funkcije gubitka

Sada se vraćamo na spomenuti primjer izračunavanja izlaznih vrijednosti iz nekog modela na osnovu linearne funkcije. Međutim, možemo zamijeniti čitavu funkciju neuronskom mrežom čime dobijamo nelinearnu funkciju. Koncepti ostaju isti, odnosno, neuronska mreža se može predstaviti skupom parametara s tim što će se taj skup parametara odnositi na koeficijente između dva neurona. Dakle, sve spomenuto se može primijeniti na bilo koju vrstu funkcije.

Rezultat linearног klasifikatorа nazivamo **vektorom rezultata** (engl. *score*). Postavlja se pitanje kako tumačiti rezultat i da li izabrana matrica W daje zadovoljavajuće rezultate.

Osnovno pitanje koje postavljamo sebi je da li je neki klasifikator dobar ili loš. Recimo da smo klasifikator trenirali nad određenim skupom podataka. Dalje, iz skupa za treniranje biramo neku sliku. Rezultat klasifikatora je klasa kojoj neki objekat (slika) pripada. Tu informaciju možemo koristiti kako bismo izračunali **funkciju gubitka**. U nastavku će biti opisane dvije funkcije gubitka: **Softmax** i **SVM** (engl. *support vector machines*). Na izlazu iz funkcije gubitka dobijamo vrijednost gubitka (engl. *loss*) i određena je pozitivnim brojem. Klasifikator je dobar za predikciju ako je rezultat blizu nule.

Funkcija gubitka se još naziva **funkcija cilja** ili **objektivna funkcija**. Njen cilj je pronaći matricu W na način da se minimizira funkcija gubitka i taj proces se naziva **optimizacijom**. Izbor matrice W igra veliku ulogu za klasifikaciju određenih podataka. U nastavku su prikazani primjeri koji predstavljaju različite izbore matrice W .



Slika 5.10: Različiti izbori matrice W

U prvom slučaju imamo matricu W koja neispravno vrši klasifikaciju dva

skupa podataka. Ukoliko izaberemo ovakvu funkciju gubitka, to znači da je matrica W potpuno neefikasna za podjelu ovog skupa podataka. Postoje razne mogućnosti kako podijeliti zadane podatke, te su na preostala tri primjera predstavljena tri moguća načina podjele. Postavlja se pitanje kako odrediti koji od ova tri načina je najbolji. Odgovor na ovo pitanje ovisi od vrste podataka. Ukoliko sigurno znamo da se podaci plavog skupa nalaze isključivo iznad date linije klasifikatora, onda drugi i treći primjeri klasifikatora daju zadovoljavajuće rezultate. Međutim, ukoliko se javi neki testni podatak koji će se nalaziti van određenog prostora, klasifikator neće biti dobar. Ovaj problem se može izbjegći izborom klasifikatora predstavljenog četvrtom slikom. Prepostavlja se da bi u tom slučaju upravo ovaj klasifikator imao najbolju generalizaciju između ova dva skupa. Ovakvo rješenje se naziva rješenjem **maksimalne margine** (engl. *maximum margin*) i ono je najstabilnije u slučaju pomjeranja ulaza kroz neki definisani prostor.

Funkcija gubitka se obično računa u odnosu na neki postojeći skup podataka predstavljenih kao par ulaza i izlaza

$$(x_i, y_i)_{i=1}^N$$

gdje su x_i neki ulazni podaci, a y_i neke labele (klase).

U praksi, labele se kodiraju preko integer vrijednosti i imamo ograničen broj takvih vrijednosti. Za neki klasifikator definišemo funkciju gubitka. **Apstraktnu funkciju gubitka** za jednu sliku x_i možemo definisati na sljedeći način:

$$L_i(f(x_i, W), y_i)$$

Dakle, ovakva funkcija gubitka se računa u odnosu na klasifikator f i izlaz y_i . Pored toga, i je vezano za jedan objekat unutar skupa podataka, što znači da u ovom slučaju računamo funkciju gubitka za svaki podatak koji se nalazi unutar testnog skupa podataka.

Pored ovog gubitka koji je vezan isključivo za jednu sliku, mi računamo i ukupni gubitak za neki klasifikator koji se računa na sljedeći način:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

U praksi postoji nekoliko različitih vrsta funkcija gubitaka. U nastavku će biti predstavljene **unakrsna entropija** (koja se koristi za Softmax) i **višeklasna SVM funkcija**.

Unakrsna entropija (engl. *cross-entropy loss*) se primjenjuje na vektor ili funkciju rezultata (engl. *score vector*). Funkcija rezultata se većinom označava slovom s i ona je jednaka rezultatu primjene funkcije klasifikatora nad ulaznim podacima, odnosno:

$$s = f(x_i, W)$$

Ukoliko funkciju rezultata računamo u odnosu na svaku sliku (klasu) koja postoji u datom skupu podataka, istu označavamo sa s_j . **Binarna unakrsna entropija** (engl. *binary crossentropy*) se odnosi na binarnu klasifikaciju, dok će se **kategorička unakrsna entropija** (engl. *categorical crossentropy*) odnositi na višeklasnu klasifikaciju.

Funkcija unakrsne entropije se koristi kod različitih primjera. Jedan od problema koji će se raditi u nastavku je **višeklasna klasifikacija objekata** (engl. *multiclass classification*), ali sve se može svesti i za binarnu klasifikaciju. Rezultat koji želimo postići je izračunavanje vjerovatnoće pripadnosti testne slike u odnosu na predefinisane klase.

Softmax regresija ima jako slične koncepte. Predstavlja primjer generalizirane logističke regresije koja se može koristiti za višeklasnu klasifikaciju objekata. Sinonimi koji se mogu naći u literaturi su: **Multinomial Logistic**, **Maximum Entropy Classifier** i **Multi-class Logistic Regression**. Softmax regresija je veoma popularna u primjeni neuronskih mreža. Dakle, recimo da imamo neku sliku, neki klasifikator, te određenu matricu W i otklon. Nakon toga, možemo izračunati vektor izlaza koji je u prethodno spomenutom primjeru imao sljedeći oblik (na izlazu su redom klase mačka, pas i brod):

$$\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}$$

Također znamo da su ovim vektorom dati rezultati koji su direktno vezani za ulaznu sliku. Kao što je već spomenuto, ovaj vektor treba da nam da informaciju o pripadnosti ulazne slike nekoj klasi. Dakle, postavlja se pitanje da li početna slika pripada klasi mačka ili klasi pas ili klasi brod. Recimo da je napravljeno nekakvo mapiranje koristeći neku metodu te dobijemo rezultat:

$$\begin{bmatrix} 5\% \\ 85\% \\ 10\% \end{bmatrix}$$

Možemo reći da smo mapirali broj -96.8 na 5%, 437.9 na 85% i 61.95 na 10%. Ovi rezultati bi trebali da nam kažu da je rezultat vektora izlaza rekao da ulazna slika ima vjerovatnoću od 85% da pripada klasi pas. Vidi se da u ovom slučaju rezultat predikcije nije tačan, što bi značilo da nismo napravili ispravan odabir matrice W . Ovaj podatak se dalje može koristiti da izvršimo izmjenu matrice W , te da ponovimo sav proces. Dakle, ovu izmjenu možemo da radimo nekoliko puta i to je parametar koji uvodimo kod procesa učenja. Potrebno je odrediti kako mapirati vrijednosti iz izlaznog vektora u date vjerovatnoće. U ovom slučaju znamo šta trebamo dobiti (šta je izlaz iz softmax regresije), ali ne znamo na koji način to možemo realizovati.

Softmax funkcija podrazumijeva izračunavanje vjerovatnoće na osnovu sljedeće formule:

$$P(Y = K|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Dakle, izračunavamo vjerovatnoću pripadnosti određenog elementa nekoj klasi. s_k predstavlja vrijednost funkcije rezultata (engl. *score function*) dobijene nakon primjene nekog klasifikatora u odnosu na neku klasu. Ova vrijednost će biti stepen koji koristimo za izračunavanje eksponencijalne funkcije. Zatim ćemo izračunati sumu svih tih vrijednosti u odnosu na sve klase. Količnik ove dvije vrijednosti će predstavljati vjerovatnoću. Navedeno predstavlja osnovu za formiranje softmax funkcije (nije čitava softmax funkcija).

Recimo da smo nakon primjene nekog klasifikatora imali sljedeći vektor izlaza. Elementi redom predstavljaju klase mačka, pas i brod:

$$\begin{bmatrix} 2.6 \\ -1.3 \\ 3.5 \end{bmatrix}$$

Sada je potrebno izračunati vjerovatnoću po softmax funkciji. Dakle, prije svega je potrebno izračunati eksponencijalne funkcije gdje je stepen s_k . Za dati vektor izlaza, imati ćemo $e^{2.6}$, $e^{-1.3}$ i $e^{3.5}$ čime dobijamo vektor **nenormaliziranih vjerovatnoća**:

$$\begin{bmatrix} 13.46 \\ 0.27 \\ 33.12 \end{bmatrix}$$

Dalje, potrebno je naći sumu tih vrijednosti ($\sum_j e^{s_j} = 46.85$) i izračunati količnik vektora nenormaliziranih vjerovatnoća sa dobijenom sumom, čime dobijamo **normalizirane vjerovatnoće**:

$$\begin{bmatrix} 0.29 \\ 0.00 \\ 0.71 \end{bmatrix}$$

Suma vrijednosti normaliziranih vjerovatnoća mora biti jednaka jedinici.

Navedeno predstavlja prvu fazu softmax funkcije. Razlog zašto se ova funkcija zove **softmax** je zbog toga što se radi o diferencijabilnoj funkciji maksimizacije. Dakle, prvobitne informacije smo prilagodili da bismo prikazali normalizirane vjerovatnoće. Da smo pripadnost klasi izračunavali direktno iz ulaznih vrijednosti, takvu funkciju bismo nazvali **hardmax funkcijom**.

Softmax funkcija se može koristiti za izračunavanje funkcije gubitka. Praksom

se došlo do informacije da se funkcija gubitka može izračunati na način da izračunamo vjerovatnoću preko softmax funkcije, s tim što uzimamo negiranu logaritamsku vrijednost rezultata:

$$L_i = -\log P(Y = y_i | X = x_i)$$

Za pojedinačne klase izračunavamo funkcije gubitka. Za prethodni primjer, funkcije gubitka su:

$$L_1 = -\log(0.29) = 1.79$$

$$L_2 = -\log(0.005) = 7.64$$

$$L_3 = -\log(0.71) = 0.49$$

Ukupna funkcija gubitka se računa na osnovu količnika sume pojedinačnih funkcija gubitaka i broja klasa:

$$L = \frac{1}{N} \sum_i L_i$$

Kullback-Leibler divergencija objašnjava zašto se koristi negirana logaritamska vrijednost softmax funkcije za izračunavanje funkcije gubitka. Odgovor je matematičke prirode iz razloga što definišemo Kullback-Leibler divergenciju čime pokušavamo da utvrdimo koja je mjera razlike između dvije distribucije vjerovatnoće. Ovdje imamo dvije vjerovatnoće P (normalizirana vjerovantoča) i Q (aproksimacija - poželjna vjerovatnoča). Osnovna ideja kod izračunavanja ove divergencije jeste da se izmjeri razlika između dobijene normalizirane vjerovantocu i poželjne vjerovatnoće. Za svaku sliku u skupu podataka imamo labelu čime znamo kojoj klasi ulazna slika pripada. Za posljednji navedeni primjer, znamo da ulazna slika pripada klasi mačka, što bi značilo da je rezultat u idealnom slučaju

$$\begin{bmatrix} 1.00 \\ 0.00 \\ 0.00 \end{bmatrix}$$

Ovaj vektor znači da je na slici 100% klase mačka. Za sve ostale klase (klasa pas i klasa brod), vjerovantocu je nula.

Kullback-Leibler divergencija se računa po sljedećoj formuli:

$$D_{KL}(P||Q) = \sum_y P(y) \log \frac{P(Y)}{Q(Y)}$$

Dakle, računa se logaritam količnika između date dvije vjerovatnoće i isti se množi vjerovatnoćom u odnosu na stvarne labele. Ove informacije se sumiraju u odnosu na date labele, te ćemo na osnovu toga doći do izraza za izračunavanje funkcije gubitka, odnosno:

$$D_{KL} = -\log P(Y = y_i | X = x_i)$$

Za unakrsnu entropiju minimalna vrijednost funkcije gubitka L_i je nula, a maksimalna vrijednost je $+\infty$. Naravno, što je vrijednost gubitka bliža nuli, bliže smo odgovoru.

Kao što je već spomenuto, matricu W većinom inicijaliziramo nekim nasumičnim vrijednostima. U prvoj fazi računanja funkcije gubitka, možemo koristiti malu provjeru kojom možemo utvrditi da li smo na pravom putu za računanje dobre vrijednosti gubitka. Kod samog početka treniranja, vrijednost gubitka bi trebala da se računa na sljedeći način:

$$-\log \frac{1}{C}$$

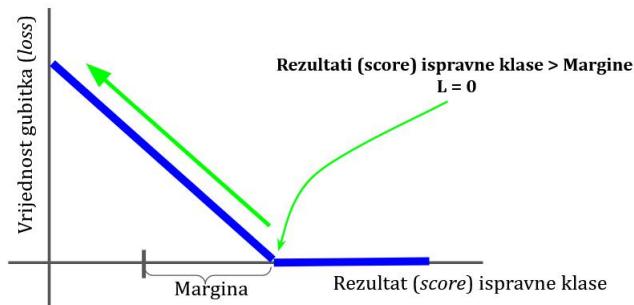
gdje je C broj klasa.

Uz ovu vrijednost, možemo provjeriti da li smo izabrali dobre hiperparametre.

Višeklasna SVM funkcija gubitka se modelira po **Hinge** funkciji gubitka. Definišimo vektor rezultata na proizvoljan način, gdje ćemo pokušati da mapiramo najveći broj na ono mjesto koje označava da slika pripada ispravnoj klasi (ovaj element stavljamo na prvu poziciju, jer ova vrijednost vektora predstavlja klasu mačka, a preostala dva elementa predstavljaju klasu pas i klasu brod):

$$\begin{bmatrix} 5.2 \\ -3.3 \\ 1.5 \end{bmatrix}$$

Hinge funkcija se predstavlja na sljedeći način:



Slika 5.11: SVM funkcija gubitka

Dakle, imamo koordinatni sistem gdje su na x-osi predstavljeni rezultati neke ispravne klase, a na y-osi je vrijednost gubitka. Recimo da znamo da je slika ispravno klasificirana i da vrijednost score funkcije iznosi 5.2. Na koordinatnom sistemu postavljamo da je $x = 5.2$. Nakon toga, reći ćemo da za neke vrijednosti x , vrijednost gubitka treba da bude jednaka nuli. Za sve

ostale slike iz datog skupa podataka za koje imamo slučaj da je funkcija rezultata veća od određene vrijednosti (u ovom slučaju 5.2), podrazumijevati će se da je vrijednost gubitka jednaka nuli. Što se više odmičemo od te vrijednosti ka lijevo, vrijednost gubitka će se povećavati. Ovakvo ponašanje je predstavljeno linearnom funkcijom koja govori da što je manji rezultat funkcije gubitka, to ćemo imati veću vrijednost gubitka. Idealan je slučaj kada je vrijednost gubitka jednaka nuli, i u ovom slučaju nema potrebe da mijenjamo matricu W za tu klasu.

Za SVM funkciju gubitka se javlja pojam margine. Ova margina je varijabilna i predstavlja vrijednost koja je prihvatljiva u odnosu na neku klasu. Kod SVM funkcije gubitka, margina većinom ima vrijednost 1. Margina govori da je funkcija gubitka u određenom području prihvatljiva (ili prihvatljivija nego nakon tog područja). Ova funkcija dalje nastavlja rasti, te će za negativne vrijednosti biti veća čime će i vrijednost gubitka biti veća. Na iznad navedenom primjeru, margina je definisana između prve dvije najveće vrijednosti score funkcije (5.2 i 1.5).

Sada je potrebno definisati SVM funkciju gubitka. Pojedinačni gubitak se računa pomoću sljedeće relacije:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

gdje s_j predstavlja rezultat koji smo dobili iz klasifikatora za neku netačnu klasu j , a s_{y_i} predstavlja vrijednost score funkcije za ispravnu labelu (klasu). Unutar sume, ne uzima se element ispravne klase predviđanja. Margina je u ovom slučaju +1

SVM funkcija gubitka za cijeli skup podataka se računa na sljedeći način:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Uzmimo neki primjer sa sljedećim vektorom rezultata (prvi element je klasa pas, drugi element je klasa mačka, a treći element je klasa konj), te da se na ulaznoj slici nalazi mačka:

$$\begin{bmatrix} 2.14 \\ 5.76 \\ -1.34 \end{bmatrix}$$

U ovom slučaju, s_j će imati vrijednosti 2.14 i -1.34 (neispravne klase - pas i konj), dok će s_{y_i} imati vrijednost 5.76 (ispravna klasa - mačka). Kao što je već definisano, za SVM funkciju gubitka nećemo sumirati ispravnu klasu zbog ograničenja $j \neq y_i$. Neka je vrijednost margine +1. Ako posmatramo linearne klasifikatore, ova vrijednost margine će odrediti odstupanje od savršenog rezultata. Ukoliko se podatak nalazi u tom pojasu, odluka o tome kojoj klasi pripada se drugačije odnosi nego ukoliko je ta margina pređena.

Izračunavanje gubitka će biti predstavljeno sljedećim primjerom. Date su tri slike i klase pas, mačka i konj. Za ove tri slike, dobili smo tri različita vektora rezultata (score vektora):

			
Pas	-1.32	2.14	-3.2
Mačka	0.98	5.76	1.3
Konj	3.72	-1.34	-2.5

Slika 5.12: Primjer vektora rezultata

U idealnom slučaju, dijagonalno bismo trebali dobiti najveće score vrijednosti koje odgovaraju klasi (vrijednosti su boldirane). Tako da za primjer prve slike, najveća score vrijednost bi se trebala nalaziti na prvoj poziciji (što nije slučaj). Analogno, za drugu sliku bi najveća vrijednost trebala biti na drugoj poziciji, a za treću sliku na trećoj poziciji. Možemo reći da je ovaj klasifikator sa nekom matriom W najbolje rezultate postigao za drugu sliku, dok za preostale dvije slike imamo neispravne predikcije. Sada je potrebno preko softmax funkcije gubitka izračunati gubitak. Na osnovu te informacije ćemo vidjeti da li smo ispravno pretpostavili kako score vrijednosti trebaju izgledati ili ne (u idealnom slučaju, gubitak je nula).

Izračunajmo gubitak za sve date slike:

$$L_1 = \max(0, 0.98 - (-1.32) + 1) + \max(0, 3.72 - (-1.32) + 1) = 9.34$$

$$L_2 = \max(0, 2.14 - 5.76 + 1) + \max(0, -1.34 - 5.76 + 1) = 0$$

$$L_3 = \max(0, -3.2 - (-2.5) + 1) + \max(0, 1.3 - (-2.5) + 1) = 5.1$$

Ukupan gubitak se računa kao suma pojedinačnih gubitaka kroz broj klasa:

$$L = \frac{9.34 + 0 + 5.1}{3} = 4.46$$

Predavanje 7

Funkcije gubitka (engl. *loss functions*) utiču na povećanje tačnosti nad podacima za treniranje. Međutim, bitnija je tačnost nad testnim podacima. To znači da je centralni problem u mašinskom učenju kako napraviti algoritam koji će imati dobre rezultate, ne samo na podacima za treniranje, već i na novim ulazima (testiranje), tj. na podacima koje model nikad do sada nije “vidio”. Mnoge strategije koje se koriste u mašinskom učenju izričito su dizajnirane da smanje grešku nad testnim podacima, i to najčešće kao posljedicu imaju povećanje greške u procesu treniranja. Strategije koje nastoje da smanje greške nad testnim podacima se nazivaju **regularizacije**. Još jedan pojam usko vezan za regularizaciju je pojam **generalizacije**. Generalizacija je sposobnost dobrog zaključivanja modela na prethodno neviđenim podacima.

6.1 Regularizacija

Bilo koja modifikacija koja se radi nad nekim algoritmom učenja sa ciljem smanjivanja njegove generalizacijske greške, ali ne i greške u treningu (idealno) naziva se **regularizacijom**. Pri ovom procesu, moramo voditi računa da nema prevelikog smanjivanja tačnosti u procesu treniranja. Postoji nekoliko strategija za regularizaciju:

- **postavljaju ograničenja** na model učenja (npr. ograničenja za određene parametre)
- **dodaju novi član u funkciji gubitka** (objektivnoj funkciji) koji utiče na promjenu parametarskih vrijednosti na blaži način nego prethodna modifikacija (engl. *soft constraint*)

Bez obzira na odabranu strategiju, krajnji cilj je postizanje boljih performansi na testnom skupu podataka. U nekim slučajevima, moguće je koristiti **heuristiku** gdje možemo da utvrdimo neka prethodno stečena znanja kroz postavljanje ograničenja i kazni. Međutim, ukoliko koristimo ovaj metod, možemo imati problem sa stvaranjem neke opće sklonosti ka nekoj vrsti podataka. Ostali oblaci regularizacije, poznati kao **ansambl** (engl.

ensemble) metode, stvaraju više hipoteza (modela) za određeni problem. Nakon toga, ove metode kombinuju rezultate novih hipoteza pomoću usrednjavanja vrijednosti ili biranja najbolje vrijednosti (ili nešto slično) kako bi se poboljšale performanse nekog skupa podataka.

Regularizacija se može koristiti za rješavanje problema underfitting-a i overfitting-a.

Funkcija gubitka sa novododanim članom regularizacije ima sljedeći oblik:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i) + \lambda R(W)$$

Novi član se označava sa **R** i kreira se u odnosu na matricu parametara **W**. Ovu funkciju množimo sa parametrom λ (hiperparametar) koji će odrediti koliko funkcija regularizacija utiče na krajnji gubitak. Ovaj parametar može biti proizvoljan (fiksna vrijednost) ili može biti adaptivna vrijednost koja se prilagođava podacima koji su vezani sa matricom **W**.

Jednostavnii primjeri funkcija regularizacije su **L1 regularizacija** i **L2 regularizacija**, odnosno:

$$R(W) = \sum_{k,l} W_{k,l}^2$$

$$R(W) = \sum_{k,l} |W_{k,l}|$$

Pomoću parametra λ , nastojimo da spriječimo model da previše dobro radi sa podacima za treniranje (na ovaj način nestaje problem overfitting-a).

Pored navedenog, postoje i dosta složenije strategije kao što su

- **dropout** - nasumično isključuje određen broj neurona u pojedinim slojevima
- **batch normalizacija**
- **cout**
- **mixup**
- **Stochastic depth**

6.2 Optimizacija

Optimizacija se odnosi na zadatku minimiziranja funkcije gubitka $f(x)$ ili maksimiziranja negativa te iste funkcije na način da se mijenja parametar x . Dakle, cilj optimizacije jeste pronaći parametre matrice **W** na način da se minimizira funkcija gubitka **L**.

Vrijednost koja minimizira ili maksimizira funkciju često označavamo sa *:

$$W^* = \operatorname{argmin}_W L(W)$$

Ovaj zadatak se može posmatrati na različite načine. Sami cilj se najbolje može objasniti ukoliko se zamislimo na vrhu u nekom 3D prostoru. Vrijednosti matrice W su predstavljene koordinatama (x, y) koje označavaju neke pozicije u samoj ravni, dok je visina predstavljena funkcijom gubitka L . U ovom slučaju, želimo doći do najniže tačke što predstavlja i sami proces optimizacije.

Optimizacija se vrlo često veže za izvod funkcije, te se taj izvod koristi za minimizaciju te iste funkcije. Postavlja se pitanje kako promijeniti x da bismo napravili neko poboljšanje u funkciji $f(x)$. Najčešće korištena metoda je **metoda gradijentnog spusta** gdje je potrebno izračunati gradijent, te se kretati u suprotnom smjeru tog gradijenta. Problem metode gradijentnog spusta je što nekada ne može da pronađe globalni minimum, već vrlo često zastane u lokalnom minimumu iz kojeg ne može izaći.

Optimizacija podrazumijeva korištenje raznih iterativnih metoda koje mogu da poboljšavaju rješenja. Postoji nekoliko algoritama:

- **nasumično pretraživanje** (engl. *random search*)
- **nasumično lokalno pretraživanje** (engl. *random local search*)
- **metoda gradijentnog spusta** (engl. *gradient descent*)

6.3 Metoda nasumičnog pretraživanja

Metoda nasumičnog pretraživanja predstavlja vrlo jednostavan algoritam koji može da radi, ali tačnost koju postiže nad podacima je jako niska. Implementacija ove metode je data u nastavku:

```

1 bestloss = float("inf") # najveća vrijednost za float
2 for num in range(100):
3     w = np.random.randn(10, 3073) * 0.0001 # generisi matricu W sa
        nasumičnim vrijednostima
4     loss = L(X_train, Y_train, w) # izracunaj funkciju gubitka
5     if loss < bestloss: # sacuvaj bolje rjesenje
6         bestloss = loss
7         bestW = w

```

Kod ove metode, matrice W u različitim iteracijama nemaju nikakvu povezanost. Tačnost ove metode za CIFAR10 skup podataka je oko 15% (tačnost najboljih algoritama je oko 95%!).

6.4 Metoda nasumičnog lokalnog pretraživanja

Za nasumično lokalno pretraživanje jednodimenzionalnog prostora, zadatak se svodi na izvod same funkcije, dok kod M-dimenzionalnog prostora definišemo gradijent u odnosu na svaku dimenziju. Nagib se definiše na način da množimo pravac sa gradijentom, te je na kraju potrebno uzeti smjer najstrmijeg spuštanja, odnosno, negativni gradijent.

Postoji nekoliko načina određivanja gradijenta. Prvi je **numeričko određivanje gradijenta** i ovaj način je dosta jednostavan, ali veoma spor za veće skupove podataka ($O(\text{brojDimenzija})$). U ovom slučaju, postupak je aproksimativan. Drugi način je **analitičko određivanje gradijenta**. Ovaj postupak je dosta brži, ali može dovesti do grešaka zbog procesa ručnog računanja određenih koraka. U praksi se najčešće koristi analitički gradijent, ali se implementacija provjerava numeričkim gradijentom (naziva se engl. *debugging tool*). Postupak se naziva još i **provjera gradijenta** (engl. *gradient check*).

```

1 w = np.random.randn(10, 3073) * 0.0001 # generisi matricu W sa
   nasumičnim vrijednostima
2 bestloss = float("inf") # najveća vrijednost za float
3 for i in range(100):
4     step_size = 0.0001
5     wtry = w + np.random.randn(10, 3073) * step_size
6     loss = L(Xtr_cols, Ytr, wtry) # izracunaj funkciju gubitka
7     if loss < bestloss: # sacuvaj bolje rjesenje
8         w = wtry
9         bestloss = loss

```

Korištenjem ove metode nad podacima iz skupa CIFAR10, dobija se tačnost oko 21.4%.

6.5 Metoda gradijentnog spusta

Metoda gradijentnog spusta se kreće iterativno u smjeru negativnog gradijenta. Ova metoda se vrlo lako može implementirati sa nekoliko linija koda.

```

1 w = initialize_weights()
2 for t in range(num_steps):
3     dw = compute_gradient(loss_fn, data, w)
4     w -= learning_rate * dw

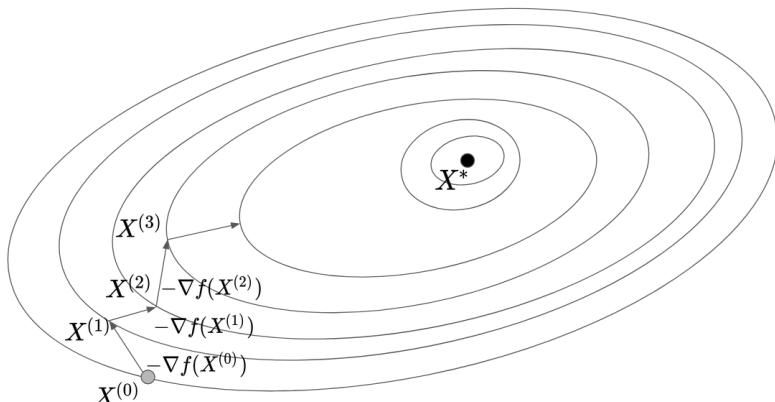
```

Potrebno je implementirati metodu `initialize_weights()` koja dodjeljuje vrijednosti matrici W . Krećemo se petljom kroz određeni niz koraka (hiperaparametar `num_steps`). Gradijent računamo koristeći metodu `compute_gradient(loss_fn, data, w)` koja se lagano može implementirati. Matrica W se dalje računa na način da oduzmemmo izračunati gradijent pomnožen sa nekom **stopom učena** (engl. *learning rate*) od

trenutne vrijednosti matrice W . Stopa učenja predstavlja novi hiperparametar koji definiše koliko daleko (odnosno blizu) pretražujemo od trenutne matrice W .

Za ovu metodu, jako je bitno napraviti dobar izbor hiperparametara. Prije svega, potrebno je ispravno inicijalizirati matricu W jer ona utiče na brzinu konvergencije. Broj koraka izvršavanja petlje unutar metode ovisi od računarskih resursa. Spomenuta stopa učenja ujedno govori koliko vjerujemo gradijentu kada se krećemo u njegovom suprotnom smjeru. Ova vrijednost je obično izuzetno mala veličina.

Primjer gradijentnog spusta je dat u nastavku.

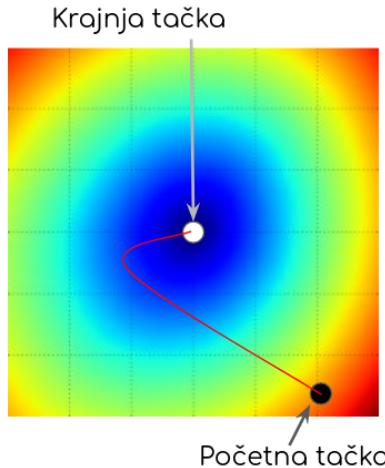


Slika 6.1: Primjer gradijentnog spusta

Dakle, predstavljena je dvodimenzionalna površina. Recimo da je sa crnom tačkom označeno neko optimalno rješenje X^* . Cilj je da bez obzira iz koje tačke krenemo iz date površine, pronađemo metodu koja na najefikasniji način dolazi do optimalnog rješenja. Ukoliko smo uzeli neku tačku $X^{(0)}$ (označena sivom bojom na primjeru), te izračunamo gradijent funkcije u njoj, potrebno je da se krećemo u negativnom smjeru tog gradijenta. Sada dobijamo drugu tačku $X^{(1)}$, te i za nju računamo gradijent. Isti postupak ponavljamo za sve sljedeće tačke sve dok ne dođemo do optimalne tačke za koju možemo garantovati da je funkcija minimuma. Postoje različite putanje koje mogu dovesti do optimalne vrijednosti, ali mi pokušavamo naći najbolju putanju. Pored toga, vrlo bitno je da metoda ne bude previše spora.

U sljedećem primjeru se može primijetiti da se najmanja vrijednost funkcije nalazi u centru. Između početne i krajnje tačke postoji beskonačno mnogo načina da nađemo putanju između njih. Metoda gradijentnog metoda se jako brzo približava krajnjoj tački. Međutim, kada dođe do određenog trenutka,

pretraživanje prostora se značajno usporava kao i proces konvergencije. Dakle, broj koraka je jako bitan kod procesa optimizacije. Ukoliko izaberemo mali korak, funkcija neće doći do određenog cilja.



Slika 6.2: Primjer gradijentnog spusta

Metoda gradijentnog spusta se zaustavlja ukoliko je ispunjen neki kriterijum koji je unaprijed poznat. U literaturi se ovaj metod naziva i **Košijevom metodom** (*Augustin-Louis Cauchy*). Osnovni koraci metode su:

0. Postaviti broj iteracija $k = 0$. Izabrati dopustivu tačku $X^{(0)} \in R^n$.
1. Izračunati $d_k = -\nabla f(X^{(k)})$.
2. Naći α_k (stopa učenja) kao rješenje problema minimizacije funkcije $g_k(\alpha) = f(X^{(k)} - \alpha \nabla f(X^{(k)}))$ za $\alpha \geq 0$
3. Izračunati $X^{(k+1)} = X^{(k)} - \alpha_k d_k$. Postaviti $k = k + 1$ ići na prvi korak.

Postoje tri verzije metode gradijentnog spusta:

1. Serijski ili ukupni gradijentni spust (engl. *Batch Gradient Descent / BGD*)
2. Stohastički gradijentni spust (engl. *Stochastic Gradient Descent / SGD*)
3. Mini-serijski gradijentni spust (engl. *Mini-batch Gradient Descent*)

Svaka verzija govori o tome koliko ulaznih vrijednosti uzimamo u obzir prilikom ažuriranja matrice W , odnosno za izračunavanje gradijenta funkcije gubitka. Prva verzija uzima sve ulazne vrijednosti i na osnovu pregleda svih ulaznih vrijednosti vrši ažuriranje. Stohastički gradijentni spust uzima samo

jednu vrijednost koja se bira na stohastičan način. Treća verzija uzima malu seriju (32/64/128 elemenata ulaznog skupa) i na osnovu izračunavanja određenih operacija nad tim skupom vršimo ažuriranje matrice W . Pravi se kompromis između tačnosti i vremena koje je potrebno da se izračuna gradijent.

6.5.1 Serijski gradijentni spust

Funkcija gubitka i njen gradijent su zadani u nastavku:

$$L(W) = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_i \nabla_W L_i(f(x_i, W), y_i) + \lambda \nabla_W R(W)$$

N je broj uzoraka u skupu za treniranje, a sumiranje po varijabli i se vrši za vrijednosti od 1 do N .

Implementacija serijskog gradijentnog spusta se definiše sa:

```
1 for i in range(nb_epochs):
2     params_grad = evaluate_gradient(loss_function, data, params)
3     params = params - learning_rate * params_grad
```

Petlja se vrši u odnosu na broj epoha. Unutar petlje vršimo evaluaciju gradijenta na osnovu svih podataka koje dolaze na ulazu, i zatim unutar svake iteracije vršimo ažuriranje parametara (varijabla PARAMS zapravo predstavlja matricu W). Realizacija ove verzije je jako skupa zbog korištenja petlje. Još jedan problem jeste što se model ne može ažurirati nakon nekoliko uzoraka, odnosno **online**, nego isključivo **offline** kada pregledamo sve podatke u nekom ulaznom skupu. Problem kod serijskog gradijentnog spusta jeste što je konvergencija prema globalnom minimumu zagarantovana za konveksne površine, dok za nekonveksne konvergira ka lokalnom minimumu.

6.5.2 Stohastički gradijentni spust

Metoda stohastičkog gradijentnog spusta ažurira parametre za svaki element unutar skupa za treniranje. Implementacija je data u nastavku:

```
1 for i in range(nb_epochs):
2     np.random.shuffle(data)
3     for example in data:
4         params_grad = evaluate_gradient(loss_function, example, params)
5         params = params - learning_rate * params_grad
```

Prednost ove verzije jeste ubrzanje zbog brže konvergencije i što je ažuriranje parametara **online**. Iz ovog razloga, ova metoda se još naziva i **Online gradijentni spust**. Mana ove verzije je što imamo česta ažuriranja zbog

kojih funkcija može da oscilira, odnosno da bude nestabilna.

Rješenje za ovaj problem je postepeno mijenjanje stope učenja (da ne bude konstantna) kako bi se ostvarila slična konvergencija kao kod serijskog gradijentnog spusta.

SGD nekada ne može pronaći globalni minimum, te se zaustavi u lokalnom minimumu. Ovaj problem se većinom javlja kod sedlastih tačaka. Iz ovog razloga, postoje razne optimizacije koje rješavaju neke od navedenih problema ove metode:

- SGD + Momentum
- Nesterov Momentum
- Adagrad
- Adadelta
- RMSprop
- Adam
- AdaMax
- Nadam
- AMSGrad

6.5.3 Mini-serijski gradijentni spust

Kod ove verzije, parametri se ažuriraju uzimajući mali skup elemenata iz skupa za treniranje nad kojima se izračunava gradijent, te se zatim ažuriraju parametri matrice W . Ovim rješenjem se smanjuje varijansa ažuriranja parametara što može dovesti do brže konvergencije.

Mini-serija (engl. *mini-batch*) obično sadrži 32/64/128 elemenata koji se analiziraju u jednom trenutku.

U nastavku je dat primjer implementacije ove metode:

```

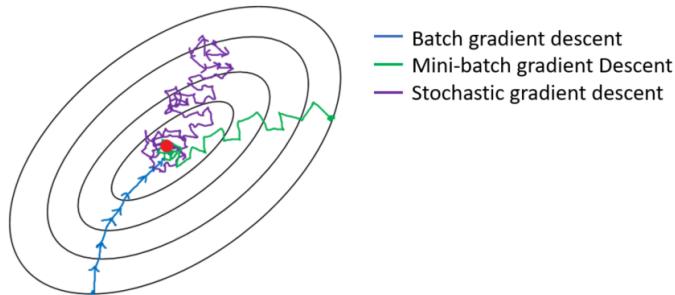
1 for i in range(nb_epochs):
2     np.random.shuffle(data)
3     for batch in get_batches(data, batch_size = 64):
4         params_grad = evaluate_gradient(loss_function, batch, params)
5         params = params - learning_rate * params_grad

```

Metodu koristimo tako što ulazne podatke X i labele Y podijelimo u skupine koje imaju isti broj ulaznih uzoraka, te dalje vršimo ažuriranje parametara. Ova metoda se može jednostavno pretvoriti u prethodne dvije predstavljene

metode postavljanjem parametra `BATCH_SIZE` na vrijednost N (serijski gradijentni spust) ili 1 (stohastički gradijentni spust).

Sljedećim primjerom je dat prikaz optimizacije koristeći prethodno opisane metode gradijentnog spusta. BGD vrlo dobro konvergira ka nekoj optimalnoj tački, ali se usporava kako se istoj približavamo. SGD pokazuje jako veliki broj oscilacija iz razloga što se parametri ažuriraju u odnosu na svaku ulaznu vrijednost. Iz ovog razloga se može desiti da odlutamo od željene tačke. Mini-serijski gradijentni spust ažurira koeficijente nakon neke skupine podataka, te konvergira značajno brže od prethodne dvije metode.



Slika 6.3: Poređenje metoda

6.6 Optimizacije SGD metode

6.6.1 SGD + Momentum

SGD se implementira na jednostavan način:

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```

1 for i in range(num_steps):
2     dw = compute_gradient(w)
3     w -= learning_rate * dw

```

Sa druge strane, **SGD + Momentum** uvodi i vektor brzine v_{t+1} i trenje ρ :

$$v_{t+1} = \rho v_t - \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

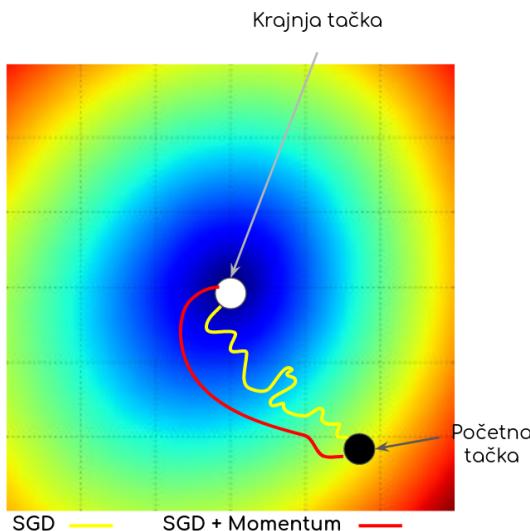
```

1 v = 0
2 for i in range(num_steps):
3     dw = compute_gradient(w)
4     v = rho * v + dw
5     w -= learning_rate * v

```

Momentum ubrzava SGD metod u najkritičnim tačkama (obično u uvalama oko lokalnog optimuma) i usmjerava ga u pravom smjeru. Na ovaj način, oscilacije koje su ranije bile problem su smanjene.

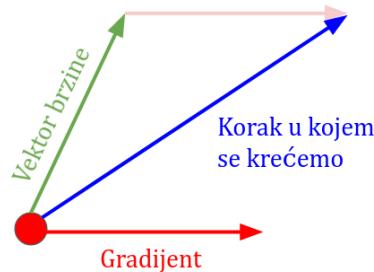
Suštinski, SGD + Momentum rješava i problem sedlaste tačke jer taj momentum omogućava da se pronađe globalni minimum. Sljedećim primjerom su prikazane različite putanje napravljene sa SGD i SGD + Momentum metodama:



Slika 6.4: Putanje SGD i SGD + Momentum metoda

SGD + Momentumom ima problem **overshooting-a**. Ovaj problem podrazumijeva da se metoda vrlo brzo kreće prema krajnjoj tački, ali u jednom trenutku počinje da se udaljava od te tačke, te se iz tog razloga javljaju oscilacije. Međutim, zagarantovana je konvergencija ka krajnjoj tački.

SGD + Momentum se može prikazati preko niza vektora. Recimo da imamo neku tačku označenu crvenom bojom u kojoj se nalazimo trenutno. U njoj možemo izračunati gradijent i vektor brzine. Zatim, napravimo projekciju gradijenta u odnosu na vektor brzine (prikazano rozom bojom). Sabiranjem ovog vektora sa vektorom brzine, dobijamo korak u kojem se krećemo.

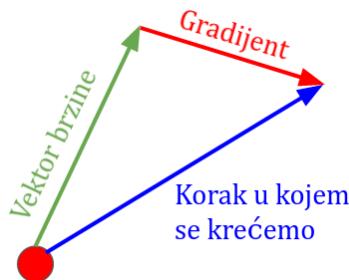


Slika 6.5: SGD + Momentum

Naučnici su uočili mogućnost modifikacije koja se još naziva **Nesterov Momentum**.

6.6.2 Nesterov Momentum

Nesterov Momentum umjesto da računa gradijent polazne tačke, računa gradijent u tački vektora brzine. Na osnovu sabiranja ova dva vektora, dobijamo korak kojim se krećemo u narednu iteraciju.



Slika 6.6: Nesterov Momentum

U literaturi se ovo često naziva "pogled u budućnost" jer se na neki način "zaviri" u budućnost kako bismo vidjeli da li je to putanja kojom se treba kretati. Iz ovog razloga, ova metoda je jako uspješna i daje dobre rezultate.

Modifikacije koje trebaju da se naprave su date u nastavku:

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

Dakle, gradijent sada računamo u odnosu na trenutnu tačku i vektor brzine. Ova relacija se može pojednostaviti određenim modifikacijama, tako što ćemo uvesti \tilde{x} :

$$\tilde{x}_t = x_t + \rho v_t$$

$$\begin{aligned} v_{t+1} &= \rho v_t - \alpha \nabla f(\tilde{x}_t) \\ \tilde{x}_{t+1} &= \tilde{x}_t - \rho v_t + (1 + \rho)v_{t+1} \\ \tilde{x}_{t+1} &= \tilde{x}_t + v_{t+1} + \rho v_t(v_{t+1} - v_t) \end{aligned}$$

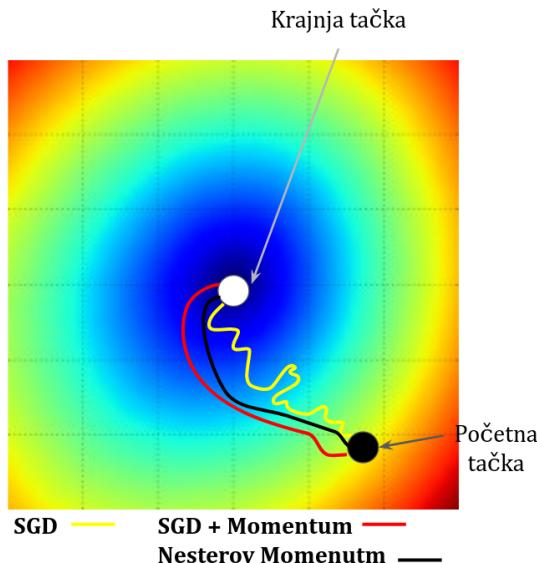
Nesterov Momentum se jako lahko implementira:

```

1 v = 0
2 for i in range(num_steps):
3     dw = compute_gradient(w)
4     old_v = v
5     v = rho * v - learning_rate * dw
6     w -= rho * old_v - (1 + rho) * v

```

Kada govorimo o poređenju tri predstavljene metode, na sljedećem primjeru su prikazane putanje koje one prave do tražene tačke:



Slika 6.7: Poređenje opisanih metoda

Nesterov Momentum ima manji problem overshooting-a. Dakle, parametre sada ažuriramo na osnovu grube skice/procjene gdje bi parametri mogli biti u sljedećem koraku. Rezultat korištenja ovog metoda je brža konvergencija.

6.6.3 AdaGrad

AdaGrad su 2012. godine objavili naučnici iz Google-a i predstavlja jedan od prvih zadataka dubinskog učenja. Na osnovu zapisa su pokušali da izvrše detekciju objekata na video klipovima.

Osnovna ideja je da se parametri prilagođavaju na osnovu parametara matrice W čime se dolazi do pojma **adaptivne stope učenja**. Ono što se promijenilo u odnosu na prethodne relacije jeste što se u ovom slučaju računa kvadrat gradijenta, te tu vrijednost koristimo za dalji proračun. Vrijednost gradijenta će kroz iteracije biti sve veća i veća, zbog čega će se vršiti manji koraci.

Implementacija ove metode je data u nastavku:

```

1 grad_squared = 0
2 for t in range(num_steps):
3     dw = compute_gradient(w)
4     grad_squared += dw * dw
5     w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

Dakle, poenta je da se postigne manje ažuriranje parametara (naročito `learning_rate`) koji se povezuju sa značajkama koje se učestalo pojavljuju, odnosno, veće ažuriranje za parametre koji se povezuju sa značajkama koje se rijetko pojavljuju. Ukoliko pokušavamo da nađemo neki objekat koji se pojavljuje često na video zapisima, isti ćemo više puta pamtitи nego one koji se javljaju rjeđe.

Ova optimizacija je pogodna za rad sa **oskudnim podacima** (engl. *sparse data*).

AdaGrad u odnosu na SGD ima značajno manji broj oscilacija, te brže konvergira ka samom rješenju. Glavna slabost AdaGrad-a je nakupljanje kvadratnih gradijenata u nazivniku. Budući da je svaka nova vrijednost pozitivna, nakupljena suma raste tokom treninga. Ova činjenica dovodi do smanjenja stope učenja koja na kraju postaje beskrajno mala, te u tom trenutku algoritam više nije u mogućnosti steći dodatno znanje. Sljedeći algoritmi imaju za cilj rješavanje ove greške.

6.6.4 RMSProp

Modifikacijom AdaGrad-a, koju je napravio Geoffrey Hinton, dolazimo do poboljšane metode **RMSProp**. Optimizacije koje je uveo su značajno jednostavnije. Cilj je da se akumulacija kvadrantnog gradijenta mijenja uvođenjem novog parametra **decay rate**. Sada možemo odrediti koliko ćemo uzimati u obzir akumulaciju gradijenta i trenutni kvadratni gradijent (većinom je to neka vrijednost 0.9 ili 0.99).

```

1 grad_squared = 0
2 for t in range(num_steps):
3     dw = compute_gradient(w)
4     grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dw *
5     dw
6     w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

6.6.5 Adam

Adam (engl. *Adaptive Moment Estimation*) predstavlja kombinaciju Momentum-a i RMSProp-a kao jednu metodu koja računa adaptivne stope učenja za svaki parametar. Razlika je što sada imamo dva momentuma koji se računaju po određenim formulama. Množenje stope učenja sa momentom koji se mijenja unutar svake iteracije će imati efekat adaptivne stope učenja. Adam optimizator daje jako dobre rezultate kod dubokih neuronskih mreža.

```

1 moment1 = 0
2 moment2 = 0
3 for t in range(num_steps):
4     dw = compute_gradient(w)
5     moment1 = beta1 * moment1 + (1 - beta1) * dw
6     moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
7     w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)

```

U praksi se preporučuje da beta2 bude 0.999, beta1 0.9, a learning_rate bi trebao da bude 1e-3, 5e-4 ili 1e-4.

6.6.6 Poređenje algoritama

Unutar ovog poglavlja su se posmatrale optimizacije prvog reda gdje se koristio izvod da bi se odredio pravac kretanja. Pored toga, postoje i metode drugog reda koje računaju kvadrat funkcije za optimizaciju, te pokušavaju da izvrše minimizaciju u odnosu na tu kvadratnu funkciju. Ove metode su kompleksnije i zahtijevaju puno više izračunavanja i vremena.

Sljedećom tabelom je prikazan pregled poređenja algoritama. Može se uočiti da Adam ima najbolje performanse, te se on najčešće bira kod treniranja neuronskih mreža.

Algoritam	Prati prvi momentum (Momentum)	Prati drugi momentum (Adaptive learning rates)	Propusni drugi momentum	Korekcija pristranosti za procjenju momentuma
SGD	x	x	x	x
SGD + Momentum	✓	x	x	x
Nesterov	✓	x	x	x
AdaGrad	x	✓	x	x
RMSProp	x	✓	✓	x
Adam	✓	✓	✓	✓

Tabela 6.1: Poređenje algoritama

Predavanje 8

Postoje različiti načini na koje možemo napraviti inteligentne sisteme koji mogu biti uspješni u raznim problemima. Inteligentni sistemi mogu istovremeno da koriste jedan ili više metoda realizacije, u koje spadaju:

- Ekspertni sistemi (engl. *Expert Systems*)
- Vještačke neuronske mreže (engl. *Artificial Neural Networks*)
- Generički algoritmi (engl. *Genetic Algorithms / GA*)
- Fuzzy sistemi (engl. *Fuzzy Systems*)
- Zaključivanje na osnovu slučajeva (engl. *Case Based Reasoning / CBR*)
- Istraživanje podataka / otkrivanje znanja (engl. *Data Mining / Knowledge Data Discovery*)
- Intelligentni agenti (engl. *Intelligent Software Agents*)
- Komunikacija u prirodnom jeziku (engl. *Language Technology / Natural Language Processing*)

Kod **zaključivanja na osnovu slučajeva** pokušavamo donijeti neki zaključak koji će uticati na donošenje neke odluke. Do tog zaključka dolazimo na osnovu slučajeva. **Slučajevi** su posebno formirane strukture koje uključuju problem i rješenje. Takvi slučajevi se definišu u odnosu na domen problema. Prije projektovanja samog sistema, uspostavlja se **baza slučajeva** iz koje se pravi metoda koja uči iz postojećih problema i rješenja. Na osnovu toga se pravi sistem koji će omogućiti da pravimo zaključivanje na novim slučajevima koje susrećemo u praksi.

Kada govorimo o inteligentnim agentima, njih možemo pronaći u raznim oblastima. Jedna od primjena je unutar **B2C** (engl. *Business-To-Consumer*) i **B2B** (engl. *Business-To-Business*) sistema.

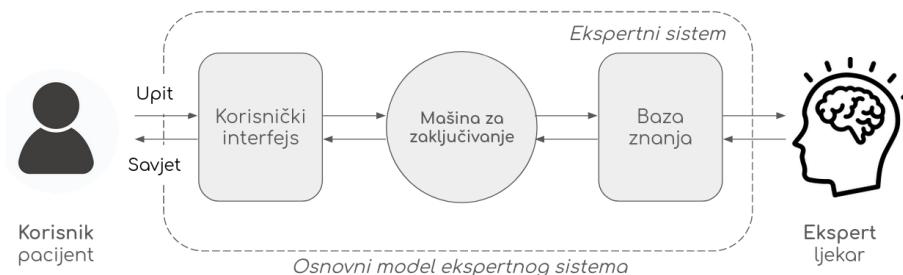
Za B2C sisteme, imamo sljedeće primjere primjene:

- **posredovanje proizvoda** (engl. *Product Brokering*) - agent predlaže kupcu proizvode (sistem Jango i Amazon)
- **trgovačko posredovanje** (engl. *Merchang Brokering*) - agent predlaže prodavca određenog proizvoda (sistem BargainFinder)
- **pregovaranje** - agent određuje cijene ili druge elemente transakcija (aukcije i ugovori)

B2B oblast pomaže trgovinama da naručuju ili daju usluge. Jedan primjer ovakvog sistema je agent koji prati potrošnju određenih proizvoda u trgovini, te automatski naručuje proizvode u zavisnosti od potrebe (engl. *Supply Chain Management*). Ostale primjene uključuju mobilne agente, evolucione agente, data mining agente i slično.

7.1 Ekspertni sistemi

Ekspertni sistemi su računarski bazirani informacioni sistemi koji koriste znanje eksperta za donošenje odluka i pokretanja akcija u svrhu rješenja problema. Ovi sistemi se isključivo koriste za proces zaključivanja, a ne donošenje odluka. **MYCIN** je prvi pravi poznati ekspertni sistem korišten za medicinsku dijagnozu, razvijen 1980. godine na Standford Univerzitetu. Ekspertni sistem je prikazan sljedećim modelom:



Slika 7.1: Model ekspertnog sistema

Osnovni elementi ekspertnog sistema su **mašina za zaključivanje** i **baza znanja**. **Baza znanja** sadrži određena znanja koja dobijamo od eksperta ili inženjera znanja. **Inženjer znanja** zna koje informacije su potrebne da bi se imala uspješna i dobra baza znanja. Jedna baza znanja se vrlo rijetko može koristiti za neki drugi sistem (jer se radi o različitim domenima problema). Međutim, postoje neki generalizirani ekspertni sistemi koji se mogu koristiti u širem kontekstu i podešavati po potrebi. Kada imamo bazu znanja, onda moramo napraviti mašinu za zaključivanje. **Mašina za zaključivanje** je suštinski skup pravila i ona se pravi posebno za svaki ekspertni sistem. Ova

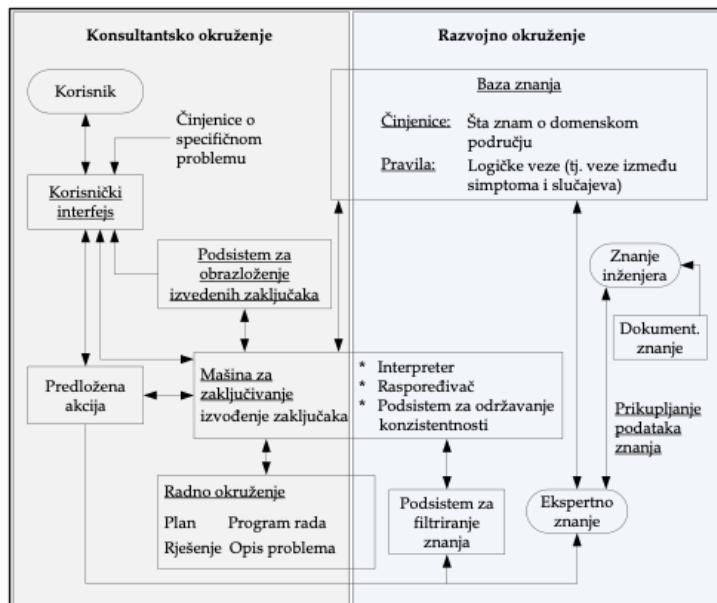
mašina se koristi za donošenje nekih novih zaključaka koje će se uputiti samom korisniku.

Kod MYCIN-a, korisnik ekspertnog sistema je pacijent, dok je ekspert ljekar ili doktor. Korisnik šalje određeni upit koji prolazi kroz korisnički interfejs gdje se može prilagoditi mašini za zaključivanje. Dalje, mašina za zaključivanje na osnovu činjenica koje se nalaze u bazi znanja generiše određeni odgovor koji se prosljeđuje samom pacijentu.

Prvi ekspertni sistemi su imali tačnost od oko 69% što se u to vrijeme smatralo dovoljno dobrom. Takvi sistemi su čak bili bolji u prognozi određenih stvari nego sami doktori!

Predstavljeni model se može proširiti na značajno kompleksniji model tako da sadrži i sljedeće komponente:

- podsistem za prikupljanje podataka znanja
- radno okruženje
- podsistem za obrazloženje izvedenih zaključaka
- podsistem za filtriranje zahtjeva



Slika 7.2: Prošireni model ekspertnog sistema

Zaključivanje (engl. *inference*) je proces izvođenja logičkih zaključaka na osnovu premeta za koje se zna ili pretpostavlja da su istinite. Zaključivanje može biti različito i poznajemo 3 pristupa:

1. **induktivno** - od pojedinačnog zaključka ka općem
2. **deduktivno** - od općeg zaključka ka pojedinačnom
3. **abudktivno** - od konkretnog zaključka ka konkretnom

Zaključivanje u uslovima neizvjesnosti se proučava u okviru matematičkih disciplina vjerovatnoće i statistike. Postoje različite forme zaključivanja (logike), a osnovne su **logika sudova** i **logika predikata**. Ova vrsta zaključivanja vrlo često podrazumijeva korištenje **Bajesove formule**. U tom slučaju, podatke posmatramo kroz statistiku i vjerovatnoću i definišemo mogućnost njihovog pojavljivanja u nekom konkretnom problemu. Na primjer, Bayesovu formulu možemo koristiti za izračunavanje vjerovatnoće neke dijagnoze d ukoliko imamo listu simptoma s.

Kreator ekspertnih sistema je profesor **Edward Feigenbaum** na Stanford Univerzitetu. Dobitnik je nagrada kao što su ACM turing Award (1994), Computer Pioneer Award, AAAI Fellow (1990), ACM Fellow (2003). Radio je doktorsku disertaciju 1960. pod vodstvom **Herbert A. Simon-a**, dobitnika Nobelove nagrade za ekonomiju 1978. i Turingove nagrade 1975. godine.



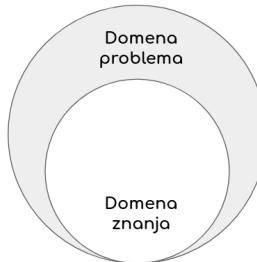
Slika 7.3: Edward Feigenbaum

Ekspertne sisteme je definisao na sljedeći način:

"**Ekspertni sistem** je računarski sistem koji oponaša ili djeluje u svim pogledima kao ljudski ekspert (stručnjak), s mogućnostima donošenja odluka ljudskog stručnjaka."

Domen problema uključuje puno širi kontekst u odnosu na **domen znanja** koji se tiče jedne specifične oblasti (kao što je medicina, finansije, nauka, inženjerstvo i slično). Unutar svakog domena imamo nekog stručnjaka (ili više

njih) koji mogu da daju ili definišu domen znanja koji je vezan za problem koji se rješava. Domen znanja je uvijek podskup domena problema.



Slika 7.4: Domen problema i domen znanja

Ekspertni sistemi se u praksi koriste kao osnovni alat za poboljšanje produktivnosti i kvalitete (obračun poreza, kreditne analize, održavanje opreme, dijagnoza grešaka i slično). Fundamentalni elementi nekog ekspertnog sistema su **ekspert** i **ekspertiza**.

Ekspert je osoba koja posjeduje specijalna znanja, iskustva, vještine i metode za savjetovanje i rješavanje problema. Zadatak jednog eksperta je da predviđa znanje koje je neophodno za izvršavanje korisnih zadataka u sistemu. Da bi ekspert ispravno izvršavao svoj posao, potrebno je da zna koje su činjenice važne i da razumije značenje veza između njih. Tipični eksperti posjeduju nekoliko osnovnih osobina. Jedna od tih osobina jeste da mogu riješiti problem i postići bolje rezultate od prosječnih ljudi u istim i/ili sličnim oblastima. Za ovu osobinu je potrebno birati eksperta u zavisnosti od problema koji se rješava. Neki zadaci koje ekspertni izvršavaju su:

- prepoznaju i formulišu problem
- brzo i korektno rješavaju problem
- definišu postupak rješavanja problema
- uče iz iskustva
- rekonstruišu znanje u svrhu rješavanja problema
- mogu da prekinu postojeća pravila ako je neophodno
- mogu odrediti prioritete i slično

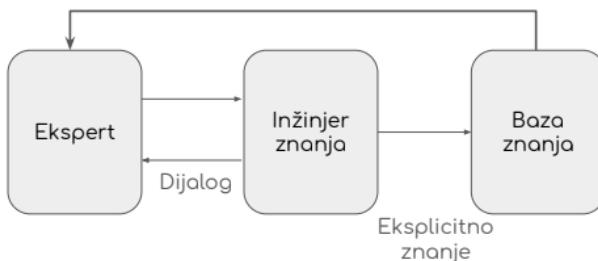
Ekspertiza (vještina) je opsežno znanje eksperta upotrijebljeno za rješavanje specifičnog zadatka. Ona proizilazi iz konstantnog usavršavanja eksperta putem treninga, svakodnevnog čitanja i znatnog iskustva koje se stiče kroz praksu. Ekspertiza podrazumijeva **eksplicitno znanje** stećeno putem

teoretskih znanja iz knjiga i **implicitno znanje** stećeno kroz svakodnevnu praksi. Ekspertiza je obično povezana sa visokim stepenom inteligencije i velikom količinom znanja, gdje eksperti uče iz prethodnih uspjeha i grešaka. Za eksperte je bitno da je njihovo znanje dobro organizirano i lako pretraživo, posebno u kritičnim momentima kada je potrebna brza odluka i zaključak.

Softverski alati za razvoj ekspertnih sistema su:

- programski jezici opšte namjene kao što su C/C++, Java, Python, F#, C#, Prolog, LISP
- ljske ekspertnih sistema kao što su Exsys CORVID, CLIPS, Jess i Experise2G0
- gotova rješenja za određene šire oblasti primjene, kao što je oblast osiguranja, medicine ili planiranje (Haley, ILOG, LPA VisiRule)

Proces izgradnje ekspertnog sistema (**inženjeriranje znanja**) je složen proces. Na prvom mjestu, **inženjer znanja** uspostavlja dijalog sa ekspertom (čovjekom) kako bi pribavio znanje. Nakon toga, inženjer znanja to stećeno znanje eksplicitno kodira u bazi znanja. Na kraju, ekspert procjenjuje ekspertni sistem i daje kritiku inženjeru znanja.



Slika 7.5: Inženjeriranje znanja

Reprezentacija znanja se tiče definisanja niza pravila. Ta pravila se definišu u kombinaciji **IF-THEN** izraza (**IF** uslov **THEN** zaključak) i **USLOV-AKCIJA** izraza (**IF** uslov **THEN** akcija). Sistem koji ćemo koristiti zavisi od same aplikacije. **Uslovi** (engl. *conditions*) predstavljaju premise, odnosno, nezavisne varijable. **Zaključak** (engl. *conclusion*) je akcija, rezultat, konsekvensa, cilj, ishod ili zavisna varijabla.

Uslovi (**IF**) su serije testova koje moramo zadovoljiti. Predstavljaju i zamjenu za "stablo odlučivanja" (engl. *decision tree*). Uslovi se kombinuju na način da se primijeni nega logička operacija (**AND**, **OR** i slično).

Posljedica (**THEN**) može biti jedna ili više njih (više zaključaka). Pored toga, posljedica može biti i akcija ili više akcija (engl. *actions*).

Dodatno, pored zaključka ili akcije može postojati i distribucija vjerovatnoće koja daje informaciju o podršci ili izvjesnosti date posljedice. Kada podaci nose neku vrstu neizvjesnosti, onda se koriste posebni sistemi koji se nazivaju "**rasuđivanje (rezonovanje) u slučaju neizvjesnosti**" pomoću kojih donosimo zaključke (koliko su podaci "tačni" i slično).

Posljedice ili konsekvene se mogu koristiti za različite situacije:

- **dovođenje u relaciju**

IF rezervoar je prazan THEN auto se ne može pokrenuti

- **davanje preporuke**

IF jesen AND oblačno AND prognoza je kiša THEN savjet je da ponesete kišobran

- **davanje direktive**

IF auto se ne može pokrenuti AND rezervoar je prazan THEN akcija je "napuni rezervoar"

- **donošenje strategija**

IF auto se ne može pokrenuti THEN provjeri rezervoar (korak1 je završen)

IF korak1 je završen AND rezervoar je pun THEN akcija je "provjeri bateriju" (korak2 je završen)

- **heuristika** (model predikcije: klasifikacija/regresija)

IF prosuta je tečnost AND PH < 6 AND miriše na sirće THEN prosuta je kisela tečnost

Pored logičkih pravila, možemo koristiti i propoziciona pravila: $<$, $>$, $=$, $'$. Dakle, možemo neku promjenljivu porediti sa nekom vrijednošću (da li je temperatura > 28). Primjer koji se može koristiti u bankarskom sektoru je sljedeći:

IF kreditni rejting visok AND plata > 15000 KM godišnje AND druge vrijednosti > 80000 AND historija plaćanja je u skladu sa obrascom THEN odobri kredit do 200000 KM i postavi pozajmnici u kategoriju 'B'.

Kod propozicionih pravila postoji problem što se ne mogu provjeravati zavisnosti između atributa. Ovakav problem se ne može definisati propozicionim pravilima, te je u ovom slučaju potrebno koristiti neku drugu vrstu pravila.

7.2 Historijat ekspertnih sistema

Prve poznate ekspertne sisteme **DENDRAL** i **MYCIN** je pokrenuo otac ekspertnih sistema **Edward Feigenbaum** na Stanfordu 1960. godine.

DENDRAL (engl. *Dendritic Algorithm*) je stvoren 1965. godine. Predstavlja ekspertni sistem za pomoć u identifikaciji molekula u organskoj hemiji na osnovu spektrograma mase. Ovaj ekspertni sistem je napisan u programskom jeziku **LISP**. DENDRAL nikada nije bio korišten u praksi.

MYCIN je ekspertni sistem napravljen 1980. godine i predstavlja prvi pravi poznati ekspertni sistem. Korišten je primarno za dijagnozu bakterijskih infekcija i propisivanje antibiotskih terapija. Sastojao se od oko 600 pravila i davao je tačnost od 69%. Na osnovu ovog ekspertnog sistema su kasnije razvijeni **KEE** i **CADUCEUS**.

XCON (engl. *eXpert CONfigurer*) (DEC / Digital Equipment Corporation) je prva poznata aplikacija korištena za sistemsku konfiguraciju. Sadrži oko 2500 pravila.

Laan Probe (Peat Marwick 1984. godine) daje preporuke o izdavanju kredita u najkraćem mogućem roku koristeći minimalnu količinu potrebnih informacija. Sadrži više od 8500 pravila.

ExperTAX (Coopers & Lybrand 1990. godine) je ekspertni sistem koji je pomogao revizorima i poreznim ekspertima u planiranju, otkrivanju problema i alternativa. Sastoji se od 3000 pravila.

Posljednja tri sistema su korištena u praksi i specifični su za određene domene.

7.3 Vrste ekspertnih sistema

Postoje različite vrste ekspertnih sistema:

- **ES na bazi pravila** - znanje se predstavlja kao serija pravila
- **ES na bazi okvira** (engl. *frame*) - koriste relacione strukture za predstavljanje složenih koncepcata, pristup OOP (slot = set atributa i aspekt = znanje i/ili proceduralna informacija)
- **Hibridni ES** - kombinacija više tipova ES-a
- **Sistem na bazi modela** - model kao potencijalni prototip budućeg sistema (npr. za simulaciju strukture i funkcionisanje sistema)
- **ES opšte namjene** (engl. *off-the-shelf*) - namijenjeni korištenju u široj oblasti primjene

- **ES posebne namjene** (engl. *custom-made*) - stvoreni prema posebnim zahtjevima krajnjih korisnika
- **ES u realnom vremenu** - striktno graničeni na sisteme koji su vremenski zahtjevni u pogledu resursa i očekivanih akcija

7.4 Osobine ekspertrnih sistema

Osobine ekspertrnih sistema se tiču ekspertize koja predstavlja najvažniji dio ekspertnog sistema. Ukoliko je ekspertiza nad sistemom dobro određena, to je vjerovatnoća da će sistem donositi dobre zaključke znatno viša. Ekspertri se razlikuju u zavisnosti od nivoa znanja koje posjeduju.

Simboličko zaključivanje podrazumijeva osnovna logička znanja i predstavlja osnovnu logičku podlogu sistema. Znanje mora biti predstavljeno simbolički, odnosno, osnovni tok zaključivanja u sistemu mora biti jasno i precizno predstavljen.

Znanje je osobina ekspertrnog sistema koja se odnosi na nivo ekspertize u **bazi znanja**. Baza znanja može da sadrži kompleksno znanje koje nije lahko pronaći među ne-ekspertima. Učenje je, također, jako bitna osobina ekspertrnih sistema. Učenje je sposobnost objašnjenja logičke odluke i detaljna interpretacija pojedinih procesa zaključivanja. Većina današnjih ekspertrnih sistema je dizajnirana tako da sadrži osobine **samostalnog učenja** putem konstantnog ažuriranja postojeće baze znanja. Na ovaj način, sistem poboljšava i optimizira procese u sistemu pri donošenju zaključaka.

Za razliku od neuronskih mreža kod kojih smo vrlo teško mogli znati kako je ona došla do nekog zaključka, sada se javljaju moderniji sistemi i oblasti **Explainable Artificial Intelligence** (XAI). Ovakve vrste sistema (u koje spadaju i ekspertri sistemi) moraju biti u mogućnosti da objasne zašto su donijeli određenu logičku odluku, te da detaljno interpretiraju svaki pojedini proces koji se javlja u procesu zaključivanja.

7.5 Baza znanja

Baza znanja je najčešći način predstavljanja znanja. Suštinski se mogu koristiti različite metode za opisivanja znanja: **produkciona pravila, logički izrazi i semantike mreže**.

Produkciona pravila (engl. *production rules*) su jednostavna pravila koja imaju oblik IF-THEN-ELSE izraza:

IF si umoran THEN se odmori

Ova pravila se koriste jer su razumljiva ljudima, odnosno, jer odgovaraju strukturi logičkog mišljenja i konstrukcijama prirodnog jezika (ako ... onda ...). Pored toga, omogućavaju i jednostavno predstavljanje neizvjesnosti i realizaciju probabilističkog zaključivanja.

Logički izrazi se koriste za predstavljanje znanja u ekspertnim sistemima:

$$H :- B_1, \dots, B_n$$

gdje je H zaključak, a B_1, \dots, B_n elementi složenog uslova tvrdnje.

7.6 Mašina za zaključivanje

U literaturi se još naziva i **mehanizam zaključivanja** (engl. *Inference Engine*). Mašina za zaključivanje predstavlja komponentu ekspertnog sistema koja realizuje operacije izvođenja zaključka na osnovu trenutnog stanja baze podataka, odnosno, radnog prostora ekspertnog sistema. Rečeno je da znanje možemo predstavljati na različite načine (logička pravila, produkcionalna pravila, semantičke mreže i pamćenje primjera). **Semantičke mreže** su suštinski složenije i predstavljaju manje formalan način za predstavljanje znanja u ekspertnim sistemima. Mogu se koristiti za predstavljanje složenih grafovskeh funkcija kao što su neke jezičke strukture.

Ako smo znanje predstavili logičkim pravilima, onda metod zaključivanja koji možemo koristiti je **princip rezolucije** (engl. *resolution principle*). Kada govorimo o produkcionalim pravilima, imamo dva metoda zaključivanja: **unaprijed** (engl. *forward*) i **unazad** (engl. *backward*). Kod semantičkih mreža se koristi **metod nasljeđivanja** i još neke posebne metode. Pamćenje primjera podrazumijeva korištenje analogije, odnosno, **koncept sličnosti**.

Predstavljanje znanja	Metod zaključivanja
Logička pravila	Princip rezolucije (<i>resolution principle</i>)
Producionalna pravila	Unaprijed (<i>forward</i>) i unazad (<i>backward</i>)
Semantičke mreže	Nasljeđivanje i posebne metode
Pamćenje primjera (slučajeva)	Analogno, koristi koncept sličnosti

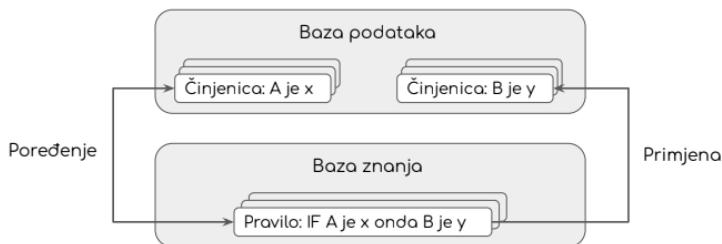
Slika 7.6: Znanje i metodi zaključivanja

Ekspertni sistemi zasnovani na pravilima, znanje predstavljaju skupom **produkcionalnih pravila i skupom činjenica**.

U sljedećem modelu su predstavljene i baza podataka i baza znanja. U bazi podataka se nalaze **činjenice** (engl. *fact*). Baza znanja sadrži skup pravila

koja su donesena na bazi intervjuja ili dijaloga sa samim ekspertom. Ukoliko je ispunjen uslov unutar baze znanja, onda to pravilo ima mogućnost da ažurira i upiše novu činjenicu u bazu podataka.

Dakle, poredi se niz pravila u bazi znanja sa činjenicama iz baze podataka, te se dalje aktivira pravilo koje odgovara određenim uslovima. Ukoliko je to pravilo ispunjeno, onda će se to pravilo dodati u bazu podataka.

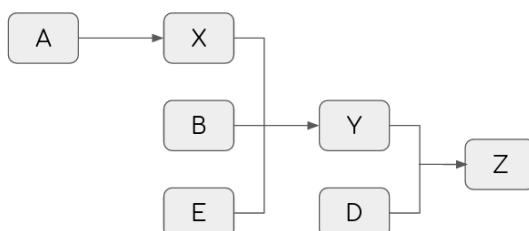


Slika 7.7: Relacija između baze podataka i baze znanja

Neka je data mašina za zaključivanje sa sljedećim pravilima:

1. IF je Y tačno AND je D tačno THEN je Z tačno.
2. IF je X tačno AND je B tačno AND je E tačno THEN je Y tačno.
3. IF je A tačno THEN je X tačno.

Kako je najjednostavnije treće pravilo, ono će se prvo aktivirati. Ukoliko prepostavimo da je A tačno, onda će se aktivirati i drugo pravilo, a zatim i prvo pravilo. Ova pravila se lakše mogu predstaviti sljedećim modelom:



Slika 7.8: Primjer

Dva osnovna načina izbora pravila koje treba primijeniti su **metod zaključivanja unaprijed** (engl. *forward chaining*) i **metod zaključivanja unazad** (engl. *backward chaining*).

7.7 Metod zaključivanja unaprijed

Metod zaključivanja unaprijed se koristi kada tražimo pravilo koje odgovara činjenicama iz baze podataka. Dakle, potrebno je aktivirati odgovarajuće pravilo iz baze znanja. Predstavlja iterativni metod koji zahtijeva određeni broj ciklusa prolaza kroz pravila da bi se izvršila promjena u bazi podataka. Pravila je moguće kompaktnije zapisati:

$$1. \ Y \ \& \ D \rightarrow Z$$

$$2. \ X \ \& \ B \ \& \ E \rightarrow Y$$

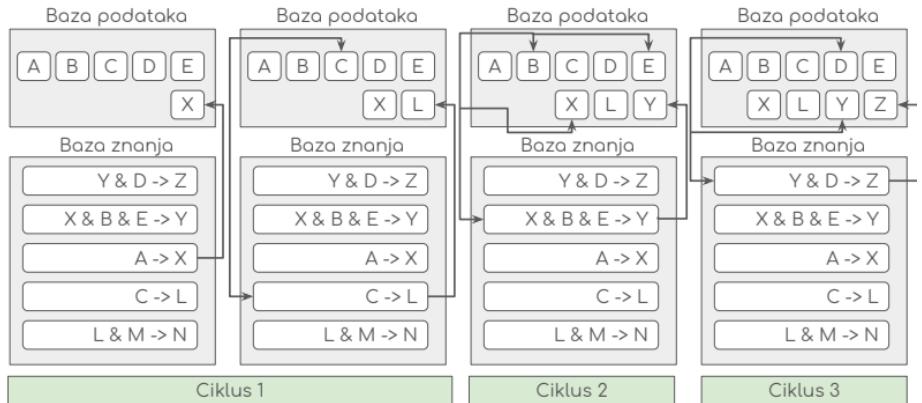
$$3. \ A \rightarrow X$$

$$4. \ C \rightarrow L$$

$$5. \ L \ \& \ M \rightarrow N$$

Čitav proces počinje od nekih poznatih činjenica i nastavlja se u skladu sa njihovim sadržajem. Proces završava ako nema više pravila koje treba primijeniti.

Recimo da je zadatak u sljedećem primjeru da se odredi Z . Da bismo izvršili prvo pravilo, u bazi podataka nam fali činjenica Y . Sada dalje prolazimo kroz pravila da bismo vidjeli unutar kojeg pravila se ona upisuje u bazu podataka. Vidimo da se to izvršava već u drugom pravilu. Međutim, da bi se drugo pravilo izvršilo, potrebno je prvo da se činjenica X nalazi u bazi podataka. Opet prolazimo kroz pravila kako bismo vidjeli unutar kojeg pravila se činjenica X upisuje u bazu i zaključujemo da je to treće pravilo. Imamo slučaj da se treće pravilo izvrši ukoliko se činjenica A nalazi u bazi podataka što je vrlo jednostavno provjeriti. Kako se tražena činjenica nalazi u bazi podataka, onda upisujemo i činjenicu X u bazu podataka. U isto vrijeme možemo izvršiti četvrto pravilo i upisati činjenicu L u bazu. Nakon toga se vraćamo na prethodnu instrukciju (druga instrukcija). Kako su svi uslovi ispunjeni, upisujemo činjenicu Y u bazu. Isti postupak radimo za prvo pravilo i upisujemo Z u u bazu podataka.

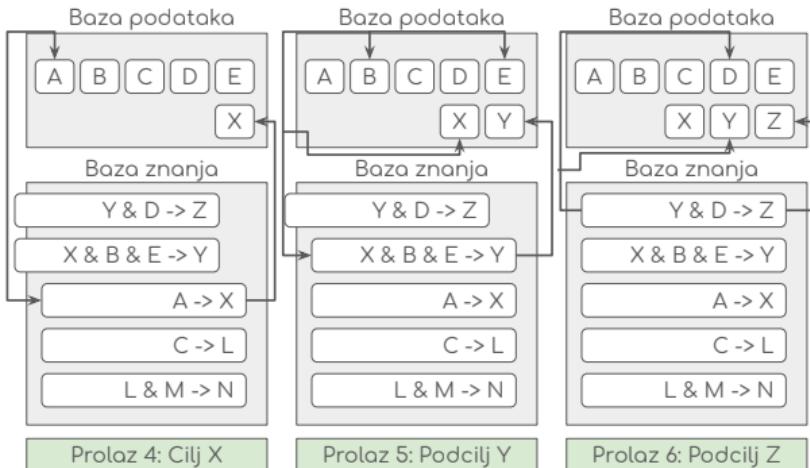
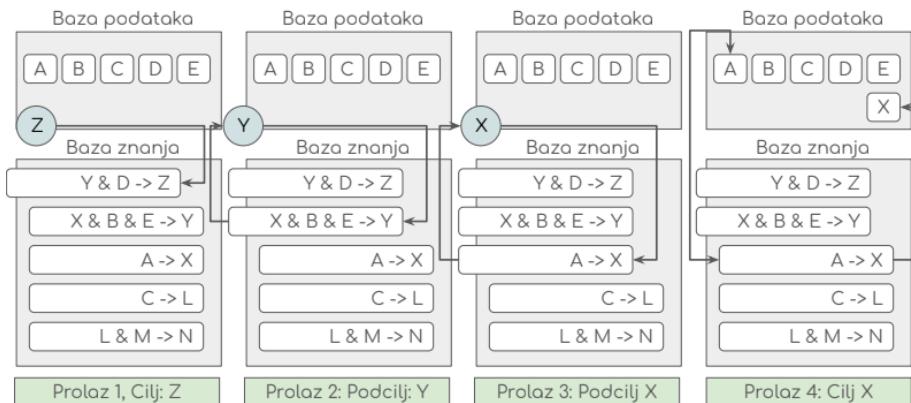


Slika 7.9: Metod zaključivanja unaprijed

7.8 Metod zaključivanja unazad

Metod zaključivanja unazad se koristi kada hoćemo da utvrdimo da li je neka hipoteza tačna ili nije. Hipoteza podrazumijeva da se zaključak prepostavi, a onda mehanizam zaključivanja pokušava da prikupi činjenice koje će potvrditi taj zaključak. Ako postoji pravilo koje je primjenljivo, ciljna tvrdnja je dokazana. Problem kod ovog pristupa je što se može desiti da se neka pravila na početku ne mogu izvršiti u datom trenutku jer nema činjenica u bazi podataka. U ovom slučaju, pravilo se stavlja na stek i postavlja se novi (pod)cilj da se dokaže istinitost uslova ovog pravila.

Kod metode zaključivanja unazad imamo **prolaze**. Unutar svakog prolaza imamo određeni cilj (u ovom slučaju, to je da se upiše činjenica **Z**). Ponovo se kreće od prvog pravila i provjerava se da li postoje činjenice **Y** i **D** u bazi podataka. Kako ne postoji činjenica **Y**, ovo pravilo stavljamo na stek i ulazimo u drugi prolaz gdje nam je podcilj upisati činjenicu **Y** te prelazimo na drugo pravilo. Isti postupak se ponavlja za ovo pravilo kako u bazi podataka fali činjenica **X**. Ulazimo u treći prolaz gdje je podcilj upisati činjenicu **X** u bazu podataka. U trećem pravilu je moguće to izvršiti, te ovaj podcilj sklanjamo sa steka nakon što upišemo činjenicu **X** u bazu podataka. Vraćajući se kroz stek, redom se upisuju činjenice **Y** i **Z**.



Slika 7.10: Metod zaključivanja unazad

U procesu zaključivanja su upotrebljena samo prva tri pravila, dok je zaključivanje unaprijed koristilo četiri pravila jer smo tu prvo tražili ona pravila koja se odmah mogu izvršiti.

Kad ekspert treba da zaključuje o prethodno prikupljenim informacijama, koristi zaključivanje unaprijed. Kad postavlja hipoteze i traži činjenice koje bi ih potvrdile ili opovrgle, koristi zaključivanje unazad.

Ekspertni sistem DENDRAL je za određivanje molekularne strukture na osnovu spektrograma mase (raspoložive informacije) koristio zaključivanje unaprijed.

Dijagostički ekspertni sistemi, kao što je MYCIN, koriste zaključivanje unazad.

Predavanje 9

8.1 Inženjering znanja

Središnji problem vještačke inteligencije je **prikazati znanje** i omogućiti proces **zaključivanja**. Rješavanje nekog problema uvijek zahtijeva veliku količinu znanja, a potrebno je znati kako to znanje iskoristiti. **Zaključivanje** je način da iz postojećeg znanja donešemo neku novu odluku na bazi činjenica koje poznajemo. Donošenje odluke podrazumijeva da pokušavamo napraviti zaključak na osnovu nekog prethodnog znanja. Ovaj proces je jako bitan za neki inteligentni sistem jer donošenje novih odluka omogućava realizaciju nekih drugih procesa ili odluka kao što je planiranje, dokazivanje teorema, detekcija, identifikacija i niz drugih problema.

Dakle, prikupljanje, predstavljanje znanja, zaključivanje i donošenje odluka su neophodni dijelovi, odnosno elementi, jednog intelligentnog sistema.

Termin **inženjering znanja** je nastao u kasnim 1970-tim godinama u vrijeme razvijanja prvih ekspertnih sistema. **Sistemi na bazi znanja** su prvi najvažniji industrijski i komercijalni proizvodi vještačke inteligencije. Koristili su se za rješavanje širokog spektra problema kao što su: otkrivanje zlonamjernih transakcija nastalim putem platnih kartica, ubrzanje izgradnje dizajna, finansijski i ekonomski sektor, pomoći pri pružanju medicinskih dijagnoza, procjena i savjetovanje o kvalitetu proizvoda i usluga, pomoći pri procjeni naučnih rezultata, podrška sistemima za isporuku električne energije, pomoći pri izvještavanju i slično. Osnovni zadatak inženjeringu znanja je pomoći ekspertima da artikuliraju svoje znanje na upotrebljiv način. Krajnji cilj je dokumentirati znanje u upotrebljiv format.

Neke od definicija inženjeringu znanja su:

- proces prikupljanja znanja od stručnjaka i izgradnje baze znanja.
- proces izgradnje intelligentno baziranog sistema.

Postoji šest osnovnih faza:

1. procjena problema

- utvrđuju se karakteristike problema
- preciziraju se ciljevi
- identifikuju se sudionici projekta
- utvrđuju se resursi potrebni za izgradnju sistema

2. prikupljanje podataka i znanja - ključne pojmove dizajna sistema treba učiniti jasnijim

- pregled dokumenata, knjiga, radova i priručnika vezanih za domen problema
- problem nepotpunih ili nedostajućih podataka
- "usko grlo sticanja znanja" (eksperti najčešće nisu svjesni svog znanja)
- proučavanje i analiziranje stečenog znanja i ponavljanje cijelog procesa dok se ne sagledaju svi aspekti

3. razvoj prototipa sistema - "manja" verzija konačnog sistema:

- uvjeriti skeptike da je odabrani alat, strategija i tehnika za prikupljanje podataka i znanja adekvatno odabrana
- pomaže kod aktivnog uključivanja domenskih eksperata
- olakšava testiranje nad testnim uzorcima

4. razvoj sistema - plan, raspored i proračun za konačni sistem

- jasno definisan kriterij izvodivosti sistema
- aktivnost dodavanja podataka i znanja
- razvoj korisničkog interfejsa
- postepeno pretvaranje prototipa u konačni (finalni) sistem (ako je moguće)

5. evaluacija i revizija sistema - procjena uspješnosti sistema u odnosu na postavljene kriterije

- procjenjuje se u kojoj mjeri sistem izvršava zadati zadatak (formalno vrednovanje sistema)
- ograničenja i slabosti sistema

6. integracija i održavanje sistema - povezivanje novog sistema s postojećim sistemom

- povlači se inženjer znanja s projekta
- sistem se prepušta korisniku

Znanje svijeta se može predstaviti putem **kategorija**. Na ovaj način možemo svrstati svo znanje koje imamo i te kategorije dalje povezivati na razne načine da bismo dobili neku novu percepciju svijeta. **Kategorije** su primarni gradivni blokovi velikih šema predstavljanja znanja.

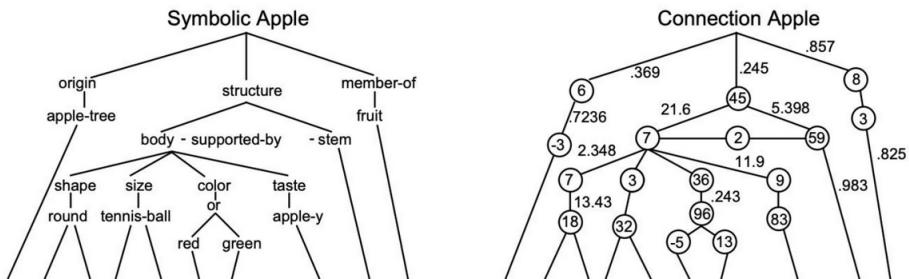
8.2 Pristup razvoju vještačke inteligencije

Postoje dva pristupa razvoju vještačke inteligencije: simbolički pristup i konektivistički pristup.

Kod **simboličkog pristupa**, znanje iz neke domene se nastoji obuhvatiti **skupom atomičkih semantičkih objekata (simbola)** i zatim se vrši manipulacija nad tim simbolima pomoću **algoritamskih pravila**. Ovaj pristup nije ispunio početna očekivanja, mada je imao određene uspjehe u oblasti ekspertrnih sistema. Pored toga, nije svako znanje moguće formulizirati formalnim pravilima.

Konektivistički pristup se zasniva na izgradnji sistema arhitekture slične arhitekturi mozga koji uči samostalno na bazi iskustva, umjesto da se eksplisitno programira. Ovaj pristup podrazumijeva da mentalna stanja i ponašanja proizilaze iz interakcije velikog broja međusobno povezanih jednostavnih procesnih jedinica (putem raznih pristupa razvoju neurona).

Rad "Logical Versus Analogical or Symbolic Versus Connectionist or Neat Versus Scruffy" Marvin Minsky-ja koji je objavljen 1991. godine, značajno doprinosi razumijevanju prethodne dvije podjele.



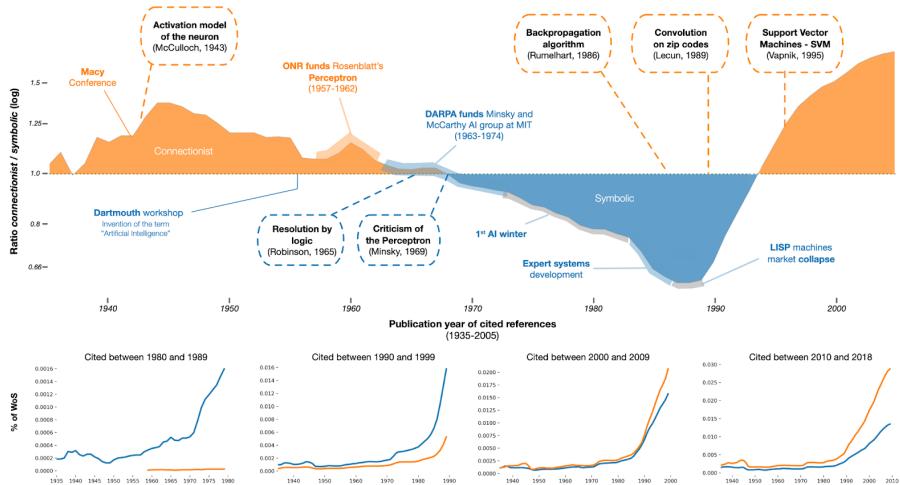
Slika 8.1: Simbolički i kontekstivistički pristup

Prva podjela predstavlja simbolički, a druga predstavlja kontekstivistički pristup. Prvi dio ("Symbolic Apple") nastoji opisati određeni objekat i predstavlja korijen objekta kojeg mi posmatramo. Ovaj objekat se kasnije razvrstava u određene potkategorije, koje se dalje razvrstavaju u drugi niz potkategorija i tako se ponavlja sve dok se pojma ne opiše.

U prvoj podjeli, jabuka se dijeli na osnovu porijekla, strukture i pripadnosti nekoj drugoj kategoriji. Struktura se kasnije klasificira na tijelo jabuke i peteljku. Nakon toga, tijelo se dalje klasificira na oblik, veličinu, boju, ukus i slično. Na ovaj način se širi znanje o određenom objektu, te se to znanje povezuje sa nekim drugim potkategorijama koje spadaju u isti graf.

S druge strane, konektivistički pristup se sastoji od osnovnih jedinica kao što su neuroni (prikazani unutar čvorova) koji predstavljaju neke sume ili vrijednosti u određenom trenutku. Pored toga, postoje i neke informacije na linijama (vezama) pomoću kojih možemo pamtiti znanje. Ove veze su značajne, ali čvorovi su značajniji.

U istom radu se nalazi sljedeća slika koja je nastala kao proces istraživanja:



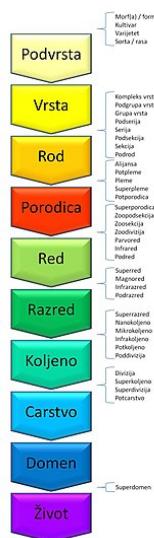
Slika 8.2: Popularnost pristupa

Na njoj se mogu uočiti zastupljenost i popularnost određenih metoda. Posmatrajući graf, vidimo da je konektivistički pristup bio više zastavljen u razdoblju 1940-1970. godine, odnosno, pristup baziran na neuronima i stvaranju modela biološkog neurona. Pojavom knjige koja je kritizovala perceptrone krajem ovog razdoblja, prelazi se na simbolički pristup. Sredinom devedesetih godina, ponovo se ulaže u polje konektivizma te se i danas smatra da je ono aktuelnije.

8.3 Ontološki inženjering

Ontologija je temeljna filozofska disciplina koja proučava biće u njegovim osnovnim odredbama, s obzirom na njegovu bit i bitak, odnosno, predstavlja nauku koja se bavi temeljnim uzrocima svega postojećega.

Ontološki inženjering (engl. *ontological engineering*) definiše način reprezentacije složenih i apstraktnih koncepata, kao što su neki događaji, vrijeme, fizički objekti, vjerovanja i slično. Predstavlja način klasifikacije apstraktnih pojmovima koji će omogućiti da kroz tu reprezentaciju predstavimo neka znanja. Dakle, potrebno je definisati šta je to fizički objekat (kategorije). Kategorije se dalje mogu dijeliti na potkategorije koje nasleđuju određene attribute od kategorije iznad. Recimo da su sve instance iz kategorije *hrana* jestive. Ako kažemo da je *voće* potkategorija kategorije *hrane* i *jabuka* potkategorija kategorije *voće*, onda donosimo zaključak da je svaka jabuka jestiva. Relacije između potklasa omogućavaju da se kategorije organizuju u **taksonomiju**. Taksonomija može biti klasifikacija živih bića:



Slika 8.3: Klasifikacija živih bića

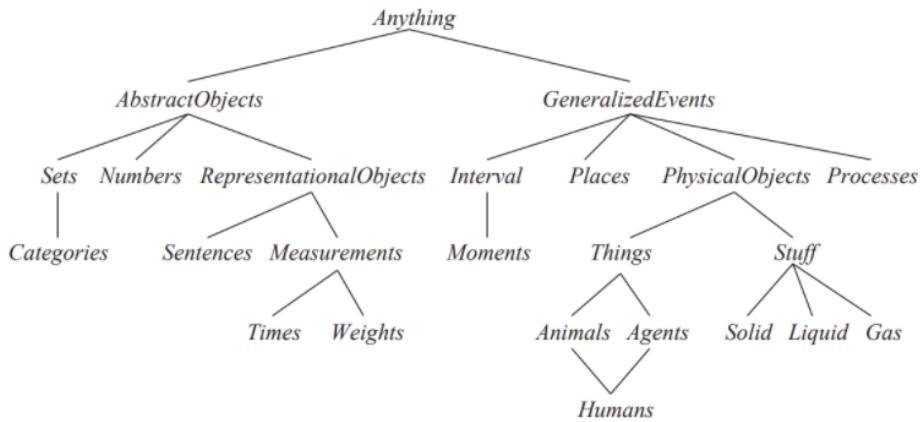
Kroz dalju gradnju ovog sistema, možemo navesti primjere nekih životinja koje ćemo klasificirati po datim klasama.

Definicija za **paradigmu istraživanja** je 1962. godine dao naučnik *Kuhn* glasi:

Paradigma istraživanja je skup zajedničkih uvjerenja i dogovora koje naučnici

dijele o tome kako probleme treba razumjeti i riješiti.

Dakle, ovo je neki generalni pristup putem kojeg su se naučnici dogovorili da predstavljaju znanje. Primjer jedne organizacije koja predstavlja **sve** (bilo šta) je data sljedećim stablom:



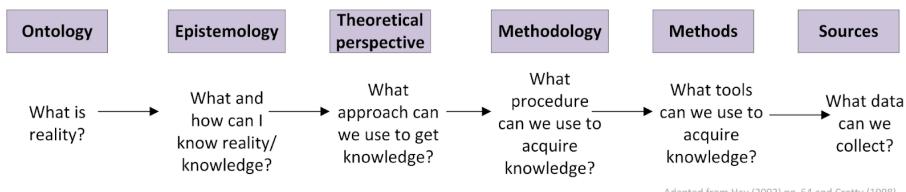
Slika 8.4: Predstavljanje znanja

Opći okvir predstavljen na ovom stablu se naziva **gornjom ontologijom** zbog konvencije crtanja grafova s općim konceptima na vrhu i specifičnijim konceptima ispod njih.

Organizacija znanja u kategorije je od vitalnog značaja za reprezentaciju znanja. Rezonovanje se vrlo često svodi na razmišljanje o kategorijama, umjesto o pojedinim fizičkim veličinama.

Epistemologija (gr. *episteme* - znanje, *logos* - logički diskurs) je grana filozofije koja se bavi teorijom znanja. Epistemologija zajedno sa ontologijom stvara holistički pogled na to kako se gleda na znanje i kako se vidimo u odnosu na to znanje, te koje metodološke strategije koristimo za otkrivanje znanja. Svest o **filozofskim pretpostavkama** povećava kvalitet istraživanja i može doprinijeti kreativnosti istraživača.

Ontologija daje odgovor na pitanje šta je to stvarnost, bitak i na neka druga filozofska pitanja. Epistemologija odgovara na pitanje kako znamo šta je to znanje. Nakon epistemologije, imamo teoretske perspektive koje nas upute na korištenje tog znanja. Pored toga, definišemo metodologije (potrebne procedure), metode (potrebni alati) i izvore (koje podatke uzimamo).



Slika 8.5: Proces predstavljanja znanja

Ultra Hal predstavlja konkretan sistem realizovan na bazi eksperimentnih sistema. Predstavlja virtuelnog asistenta gdje imamo korisnika koji može da postavlja pitanja i da daje neke činjenice. Eksperimentni sistem može da uči iz datih pravila ili da pruži odgovor na neko postavljeno pitanje deduktivnim rezonovanjem u odnosu na činjenice iz baze podataka.

8.4 Znanje

Logika prvog reda omogućava definisanje nekih činjenica u vezi kategorija i potkategorija. Primjeri nekih činjenica su:

- "Jabuka" (objekat) pripada kategoriji "Voće".
- "Voće" je potkategorija kategorije "Hrana".

Svaka kategorija, kao i svi objekti unutar jedne kategorije imaju određene osobine, te se na ovaj način članovi kategorije mogu raspozнати. Primjer objekta sa osobinama je:

$$\text{Narandža}(x) \wedge \text{Okrugla}(x) \wedge \text{Diametar}(x) = 9.5 \wedge x \in \text{Hrana} \Rightarrow x \in \text{Voće}$$

Poslije definisanja objekata/činjenica, slijedi definisanje pravila (npr. produkcionala pravila).

Pojam **znanje** je teško definisati jer se radi o širokom pojmu. Poznato je da znanje predstavlja spoznaju. Upoznati smo sa pojmovima podatka i informacije. **Podatak** je neinterpretirani signal koji može biti niz brojeva, karaktera ili drugih simbola. On nema neko specijalizirano značenje kako nije povezan sa nekim znanjem. Kada damo značenje podatku, onda takav podatak nazivamo **informacijom**. Dakle, **informacija** je podatak sa značenjem.

Za razliku od podataka i informacija, znanje se definiše kroz dva posebna aspekta:

1. **smisao svrhe/cilja** - znanje se koristi u svrhu postizanja nekog cilja

2. **generativna sposobnost** - znanje se koristi za izvođenje nekih novih zaključaka (informacija)

Sljedeći primjer opisuje karakterizaciju prethodno opisana tri pojma:

	Karakteristika	Primjer
Podatak	neinerpteriran; sirovi podatak
Informacija	značenje dato podatku	S.O.S.
Znanje	data svrha i kompetencija informaciji; poduzimanje akcije	poziv upomoć; pokretanje akcije spašavanja

Slika 8.6: Poređenje pojmljiva

Znanje može biti predstavljeno na različitim nivoima. Dva ekstremna nivoa znanja su **plitko znanje i duboko znanje**.

Za **plitko znanje** kažemo da je površinski nivo informacija koje se koriste u nekom specifičnom okruženju. Predstavlja ulazno-izlazne veze sisteme, kao što je slučaj sa IF-THEN pravilima. Plitko znanje je pogodno za manje sisteme kod kojih je potreban relativno mali skup informacija za njihovo rješavanje. Primjer plitkog znanja možemo vidjeti kod SQL alata koji su namijenjeni za ekstrakciju nekog plitkog znanja iz baze podataka.

Duboko znanje, za razliku od plitkog, podrazumijeva **unutrašnju i uzročnu strukturu** komponenti sistema. Ekspert znanja posjeduje vrhunsko poznavanje sistema. Kod dubokog znanja su definisane osnovne zakonitosti između elemenata sistema i postoji uravnotežen kompromis između apstrakcije i analogije. Pored toga, precizno je definisana struktura između objekata i procesa, veza između objekata i procesa kao i koncepcata, apstrakcija, analogija i strategija za rješavanje problema.

8.4.1 Kategorije znanja

Postoji 6 kategorija znanja:

1. **Apriori znanje** (od prije, od ranije) zavisi od toga šta osoba može da izvede iz realnosti bez potrebe da je doživi. Preduslov je postojanje neophodnog iskustva na osnovu kojeg se apriori znanje može formirati (npr. matematičke jednačine). Smatra se da je pouzdanoje od aposteriori znanja.
2. **Aposteriori znanje** (ono što dolazi poslije ili ono što slijedi) se oslanja na prethodno iskustvo i korištenje drugačije vrste zaključivanja (induktivno) za sticanje znanja. Znanje se stiče iskustvom, a zatim se koristi logika

da se dobije razumijevanje. Koristi se još i termin **empirijsko znanje**, odnosno znanje zasnovano na posmatranju.

3. **Eksplisitno znanje** je više formalno znanje u odnosu na apriorno znanje, te je zbog toga i pouzdanije. Organizuje se sistematski tako da se znanje može lako i precizno prenijeti s jedne osobe na drugu, ili na desetine hiljada ili miliona osoba. Kod eksplisitnog znanja, manje je važno šta je sadržano u odnosu na to kako je sadržano (bitnija je forma). Primjeri eksplisitnog znanja su biblioteke i baze podataka.
4. **Precutno znanje** za razliku od prethodnog znanja, jako je teško za iskommunicirati i prenijeti s jedne osobe na drugu preko bilo koje vrste medija. Primjer ovakvog znanja je kod muzičkog eksperta koji teško može prenijeti znanje na drugu osobu jer ta vrsta znanja mora biti stečena do stepena koji ide znatno dalje od teorije. Ova vrsta znanja ima najviše sličnosti sa aposteriori znanjem i teško je procijeniti kada bi ono bilo korisno i shvatiti kako ga učiniti korisnim. Potreban je niz dosljednih i opsežnih kontakata da se postigne visok procenat transfera znanja.
5. **Deklarativno znanje** se postiže tradicionalnim oblicima učenja i bazirano je na tome da budemo sigurni u ono što je istinito. Primjer ovog znanja predstavljaju matematičke jednačine ili ako se pročita (memoriše) gradivo o programiranju kako bi se moglo nešto isprogramirati na računaru. Međutim, treba razlikovati memorisanje ključnih riječi od stvarnog znanja o programiranju na računaru.
6. **Proceduralno znanje** je znanje koje se može primijeniti na nešto što se rješava (na neki problem). Stiče se radom što je suprotno od deklarativnog znanja. Karakteristika ovog znanja je **dokazivost** jer je tačno proceduralno određeno kako doći do zaključka. Znanje je veoma slično naučnim metodama gdje se hipoteze testiraju i vrše posmatranje rezultata napredovanja.

8.4.2 Predstavljanje znanja

Znanje koje je prikupljeno od strane stručnjaka u većini slučajeva nije odmah upotrebljivo. Potrebno je izvršiti oblikovanje prikupljenog znanja i predstaviti ga u nekom formatu. Koristimo različite metode za predstavljanje znanja:

- logičko predstavljanje (formalna logika)
- stablo odlučivanja
- produkciona pravila
- semantičke mreže
- okviri

- objektno-orientisano

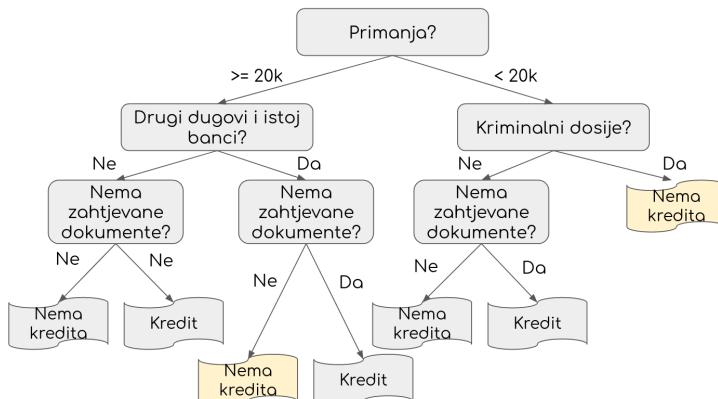
U praksi se vrlo često mogu naći i neke kombinacije ovih metoda za rješavanje složenih problema. Ovakve metode se još nazivaju **hibridnim**.

Stablo odlučivanja

Stablo odlučivanja predstavlja popularnu **klasifikacijsku tehniku** koja se koristi u oblastima mašinskog učenja (engl. *machine learning*), data mining-a, sistema za podršku odlučivanju, vještackte inteligencije ili bilo koja oblast koja uključuje logičko rasuđivanje. Koriste se za predviđanje pripadnosti različitim klasama/kategorijama uzimajući u obzir vrijednosti atributa. Stablo odlučivanja se vrlo česti koristi kao alat za sugestivnu vizuelizaciju (problem razvrstavamo u različite kategorije). Nisu svi problemi pogodni za modeliranje putem stabla odlučivanja. Postoje previše kompleksni problemi čime se povećavanjem broja grana i čvorova proces zaključivanja komplikuje. Koristi se u sljedećim domenima:

- medicina (dijagnoza)
- računarske nauke (strukture podataka)
- psihologija (teorija ponašanja)
- botanika (klasifikacija flore i faune)
- sistemi za podršku odlučivanju (logičko rasuđivanje) i slično

Primjer izgleda stabla odlučivanja je sljedeći:



Slika 8.7: Stablo odlučivanja

Stablo odlučivanja se sastoji od:

- **neterminalnih čvorova** - svaki čvor putem kojeg se izražava testiranje na osnovu određenog atributa
- **veza** - rezultat testa
- **terminalnih čvorova** (čvorovi lista) - predstavljene odluke ili klase

Postoje tri klasična pristupa korištenja stabla odlučivanja:

1. **klasifikacija** - rezultat stabla klasifikacije je poređenje i pripadnost na osnovu klase pripadnosti
2. **regresija** - kao rezultat ima realan broj
3. **klasifikacija i regresija** - kombinuje dva prethodna pristupa

Veliki izazov kod stabla odlučivanja je odabir atributa i formiranja veza prilikom izgradnje samog stabla. **Pravila razdvajanja** predstavljaju mjeru ili heuristiku za izbor kriterija podjele koji najbolje razdvaja date podskupove uzoraka u pojedinačne klase.

Izgradnja stabla je **induktivan proces** i atributi se biraju na osnovu heuristike koja najbolje razdvaja date podskupove u klase.

Postoje tri popularne mjere za izbor atributa, putem koje se određuje kriterij za izbor podjele, i to su:

1. informacijska dobit (engl. *information gain*)
2. koeficijent dobitka (engl. *gain ration*)
3. Gini indeks (engl. *Gini index*)

Gini indeks se računa po formuli:

$$Gini(D) = 1 - \sum_{i=1}^K p_i^2$$

gdje je p_i vjerovatnoća da uzorak i pripada klasi C_i , a D su podaci.

Prepostavimo da je potrebno ispitati atribut za oznakom A. Za ovaj slučaj potrebno je ispitati sve podskupove koji se mogu formirati koristeći poznate vrijednosti atributa A. Prepostaviti ćemo binarnu podjelu stabla. Ako broj ishoda označimo sa v , onda takvih skupova imamo 2^v (samo ako je stablo binarno!).

Prepostavimo da je stablo sa skupom D podijeljeno na skupove D_1 i D_2 , onda se Gini indeks za atribut A računa na sljedeći način:

$$Gini_A(D) = \frac{|D_1|}{|D|} x Gini(D_1) + \frac{|D_2|}{|D|} x Gini(D_2)$$

Tačka koja daje minimalnu vrijednost Gini indeksa se uzima kao tačka raspodjele:

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

Produkciona pravila

Predstavlja jedan od najpopularnijih načina predstavljanja znanja za ekspertne sisteme. Sastoje se od dva dijela: **uslov** i **zaključak**. Pravila se mogu definisati preko

```

IF [premisa] THEN [zaključak]
IF [prethodno] THEN [posljedično] ← deklarativno znanje
IF [situacija] THEN [akcija] ← proceduralno znanje

```

Nedostaci ove metode su što imamo veću kompleksnost s brojem pravila. Samim time, efikasnost obrade se značajno smanjuje. Pored toga, prezentacija statičkog znanja o strukturi objekta je značajno otežana i pokušava se cjelokupno znanje pohraniti u skupu pravila.

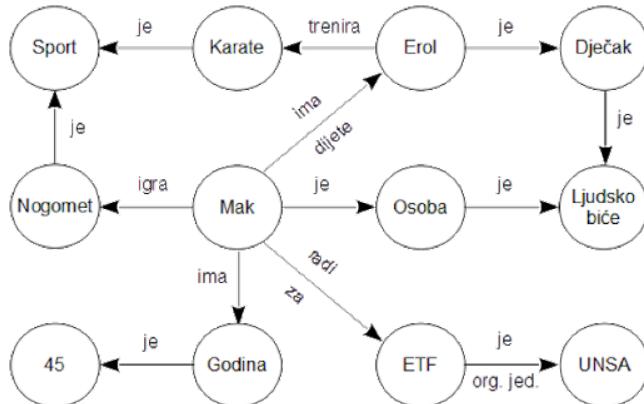
Semantičke mreže

Semantička mreža predstavlja asocijativnu mrežu koncepata. Sastoji se od **čvorova** i **lukova**. **Čvorovi** predstavljaju različite koncepte (instance, objekte) i oni mogu biti specifične instance ili generalni tipovi objekata. **Lukovi** (veze) pokazuju hijerarhijski odnos između koncepata. Na granama (vezama) su moguće neke relacije:

- **(JE)** - IS-A koja se koristi za definisanje taksonomskih veza (podklasa/klasa)
- **(IMA)** - HAS-A ILI **(JE DIO OD)** - IS-PART-OF predstavlja komponente objekta u cilju identifikacije karakteristika ili atributa

Važna osobina taksonomskih veza (JE) je **tranzitivnost**.

Primjer semantičke mreže je dat sljedećim grafom gdje je centar mreže osoba Mak. Na grafu se mogu uočiti asocijacije koje se prave između različitih objekata (čvorova) - sport, karate, osoba i slično.



Slika 8.8: Semantička mreža

Prednost semantičke mreže je **generalnost** jer se pomoću ovakve strukture mogu predstaviti kompleksni koncepti. Pored toga, imamo mogućnost sagledavanja cijelokupnog znanja. U kombinaciji sa više metoda daje značajnu prednost.

Nedostatak semantičke mreže je činjenica da efikasnost obrade zavisi od broja čvorova u mreži. Pored toga, teška je za održavanje zbog korištenja veza (pokazivača). Postupak objašnjenja je u mnogim slučajevima nejasan, odnosno, teško je zaključiti kako se došlo do zaključka.

Okviri

Okvir je struktura podataka koja se koristi za predstavljanje stereotipnih situacija. Svaki okvir predstavlja jedan objekat koji ima svoje **stanje** i **ponašanje**. Stanje je opisano skupom atributa - **slotovi** (engl. *slots*), a svaki slot može da sadrži jedan ili više aspekata (engl. *facets*) koji opisuju procedure. Oni omogućavaju da se na jedan prirođen način predstavlja struktorno znanje. Znanje u okviru je organizovano unutar slotova gdje svaki slot može opisati deklarativno ili proceduralno znanje (atributi i karakteristike). **Aspekt** sadrži kardinalnost, tip i dozvoljeni opseg vrijednosti.

Dat je primjer okvira gdje se opisuje objekat *Volkswagen Golf*. Sastoji se od slotova i aspekata:

Naziv: Volkswagen Golf	
Slot:	Aspekt:
Vlasnik	Povjeriti registracijski list
Boja	Lista, po proizvođaču
Broj cilindara	
Raspon	4, 6 ili 12
Ako je potrebno	Pitati proizvođača
Model	Sedan sport
Raspon	2 ili 4 vrata
Ako je potrebno	Pitati proizvođača

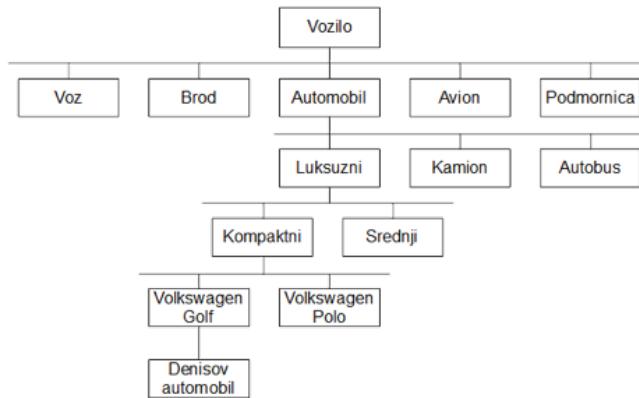
Slika 8.9: Primjer okvira

Pored ovoga, postoje određene forme koje možemo korisititi za aspekte kao što su:

Forma	Opis
Vrijednost	Koristi se za opis atributa, tj. atributu se pridružuje konkretna vrijednost
Zadata vrijednost	Karakteristika stereotipne situacije – ako je vrijedost slota prazna, tada će mu biti pridružena zadata vrijednost
Opseg vrijednosti	Dozvoljeni opseg i tip vrijednosti npr. <1000
oko-je-promijenjen	Procedura koja se pokreće kada je vrijednost slota promijenjena
oko-je-potreban	Procedura se pokreće kada je vrijednost potrebna, ali ne postoji.
oko-je-izbrisani	Procedura se pokreće kada se izbriše vrijednost iz slota.

Slika 8.10: Forme okvira

Okviri imaju vezu sa taksonomijom (hijerarhijski pristup objekata):



Slika 8.11: Taksonomija

Zaključivanje na osnovu znanja

Deduktivno zaključivanje (od općeg ka konkretnom) je zaključivanje novih činjenica na bazi poznatog (znanja). Kod ovog zaključivanja donosimo odluku na bazi činjenica koje već poznajemo.

Recimo da imamo sljedeće dvije činjenice:

- C_1 : električni aparat A je uključen u električnu mrežu
- C_2 : električni aparat A ne radi iako je osigurač ispravan

Iz ove dvije činjenice, implikacijom možemo doći do nove činjenice Z :

$C \rightarrow Z$: IF električni aparat A = 'uključen'
 AND električni aparat \leftrightarrow 'ispravan' AND osigurač = 'ispravan'
 THEN Z = električni aparat A treba servisirati

Na osnovu ovoga, imamo dva osnovna načina zaključivanja: **Modus Ponens** i **Modus Tolens**.

Modus Ponens kaže:

Ako je C i $C \rightarrow Z$ tačno, tada slijedi da je i Z tačno.

Ovo pravilo zaključivanja se naziva univerzalno instanciranje koje kaže:

Ako je nešto tačno za sve vrste, tada vrijedi da je to tačno i za bilo koji trenutak vremena te vrste.

Modus Tolens je pravilo zaključivanja koje kaže:

Ako je $C \rightarrow Z$ tačno i ako Z nije tačno, tada slijedi da i C nije tačno.

Zaključivanje je prirodno svojstvo čovjeka koji kreće od nekih hipoteza (zaključaka) i pronalazi činjenice koje su prethodile datoј hipotezi (zaključku). Ovo se naziva **hipotetskom dedukcijom**. Alternativni metod koji se koristi za ovu svrhu zaključivanja je baziran na zaključku s određenim stepenom vjerovatnoće tačnosti (**zaključak s vjerovatnoćom**).

Predavanje 10

9.1 Algoritmi evolucijskog računarstva

Algoritmi evolucijskog računarstva su inspirisani biološkom evolucijom. Motivacija za korištenje evolucijskog računarstva leži u tome da su sva živa bića kroz proces evolucije došla do određenog nivoa inteligencije, te se postavlja pitanje da li je moguće imitirati oblik evolucije za projektovanje inteligentnog sistema. Odgovor je potvrđan i veže se za određenu grupu problema.

Inteligencija jata/roja (engl. *Swarm Intelligence*) je podoblast vještačke inteligencije. Prikazan je jedan primjer ovakve inteligencije (fenomen jata):



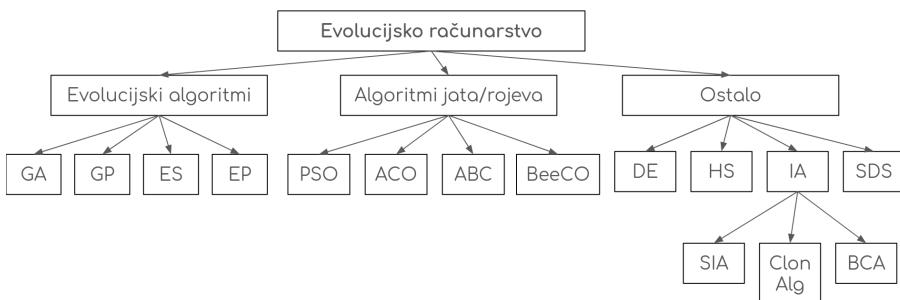
Slika 9.1: Inteligencija jata

Ovakvo jato je samoorganizirajuće i sastoji se od manjih jedinki koje prateći pokrete drugih jedinki mogu da se kreću uz neku svrhu ili cilj. Ovakav fenomen

se javlja u prirodi (kod ptica, riba, mrava i slično). Na osnovu analize ovakvih sistema možemo stvarati neke inteligentne sisteme ili sisteme koji se ponašaju na sličan način. Suština ideje inteligencije jata je u tome da imamo jednostavne jedinke (koje ne moraju biti izrazito intelligentne) čijim grupisanjem možemo dobiti intelligentniji sistem koji lakše preživljava u prirodi.

9.2 Evolucijsko računarstvo

Evolucijsko računarstvo je grana vještice inteligencije koja se najvećim dijelom bavi rješavanjem optimizacijskih problema. Ove metode su nastale u kasnim pedesetim godinama, te se mogu podjeliti na nekoliko podgrupa



Slika 9.2: Podjela evolucijskog računarstva

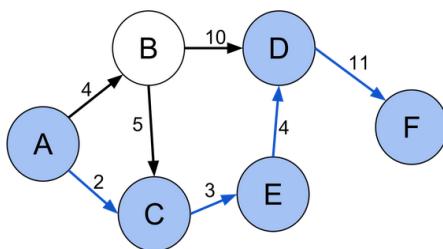
Optimizacija je postupak pronalaženja najboljeg rješenja nekog problema. Rješavanje problema posmatramo kao pretraživanje nekog prostora rješenja koji može biti manji ili veći (ovisno o problema). Potrebno je posmatrati aspekte problema i birati određena rješenja koja smatramo optimalnim. Kod optimizacijskog problema imamo postavku problema i eventualno viziju mogućeg rješenja. Rješenje će pri tome imati definisanu **funkciju sposobnosti** (engl. *fitness-function*) koju će optimizacijski postupak maksimizirati, ili **funkciju gubitka** (engl. *cost-function*) koju će optimizacijski postupak minimizirati.

Postoje dvije klase problema koje se pokušavaju optimizirati: kombinatorički i kontinualni.

Kombinatorički problemi su definisani nad diskretnim domenom koji može (ali ne mora) imati uredaj između elemenata (npr. poređenje domene cijelih brojeva i činjenice da vrijedi $1 < 5$ sa domenom tropskog voća i činjenicom da ne možemo reći da li je banana manja, veća ili jednaka ananasu). Rješenja koja koristimo za ove probleme su **pretraživanje u širinu**, **pretraživanje u dubinu** i A^* .

S druge strane, klasa **kontinualnih problema** je definisana nad kontinualnim domenama poput realnih ili kompleksnih brojeva (i slično). Ova činjenica povlači beskonačan prostor pretraživanja koji grubom silom (engl. *brute force*) nikada ne možemo pretražiti. Brute force je koristan ukoliko nemamo nikakvu informaciju o nekom problemu, ali ne ukoliko je potrebno riješiti problem za kratko vrijeme.

Neki problemi koje rješavamo sa optimizacijom su navedeni u nastavku. **Pretraživanje prostora stanja** traži neku najkraću putanju između dvije tačke. Ovakvih puteva može biti jako puno, te je iz tog razloga bitno ograničiti broj čvorova. Da bismo odrediti najkraći put, moramo imati informaciju o tome koliko je potrebno resursa da bismo prešli iz jedne tačke u neku drugu. Jedan primjer ovakve vrste problema je sljedeći, gdje je potrebno pronaći najkraći put između čvorova A i F:



Slika 9.3: Najkraći put

Problem pretraživanja prostora stanja se svodi na problem pronašlaska puta od nekog početnog stanja s_p do ciljnog stanja s_c koristeći dozvoljene poteze.

Kod **problema zadovoljavanja ograničenja** (engl. *Constraint Satisfaction Problem*) se pretražuje prostor stanja gdje put od početnog do krajnjeg stanja nije bitan. Postoje određena ograničenja koja se moraju ispoštovati i ona mogu biti gruba ili meka. **Gruba (čvrsta) ograničenja** nužno moraju biti ispunjena da bi rješenje bilo prihvatljivo, dok **meka ograničenja** definišu poželjne (ili nepoželjne) osobine rješenja. Jedan od primjera za ovakav problem je rješavanje sljedeće puzzle. Postoje određena stanja i pravci kretanja, gdje nam je konačni cilj da poredamo brojeve od 1 do 8.



Slika 9.4: Problem puzzle

Algoritmi koji se vežu za **heuristiku** se nazivaju približni algoritmi, heurističke metode, heuristički algoritmi ili jednostavno **heuristike**. Radi se o algoritmima koji pronalaze rješenja koja su zadovoljavajuće dobra, ali ne nude nikakve garancije da će uspjeti pronaći optimalno rješenje. Imaju relativno nisku računsku složenost (tipično govorimo o polinomijalnoj složenosti). Ovakvi algoritmi se dijele na:

- **konstrukcijske algoritme** - rješenje problema grade dio po dio sve dok ne naprave cijelo rješenje
- **algoritme sa lokalnom pretragom** - rješavanje započinje od nekog početnog rješenja koje se pokušava inkrementalno poboljšati

Metaheuristika je skup algoritamskih koncepcija koje koristimo za definisanje heurističkih metoda primjenljivih na širok skup problema. Metaheuristika predstavlja heuristiku opće namjene čiji je zadatak usmjeravanje problemski specifičnih heuristika prema području u prostoru rješenja u kojem se nalaze dobra rješenja. Primjeri metaheuristike predstavljaju simulirano kaljenje (Monte Carlo kaljenje), tabu pretraživanje, algoritmi evolucijskog računanja i slični. Mogu se podijeliti na dvije skupine:

- **algoritmi koji rade nad jednim rješenjem** (gdje spadaju algoritmi simuliranog kaljenja i algoritam tabu pretrage)
- **populacijski algoritmi** - rade sa skupovima rješenja (gdje spadaju algoritmi evolucijskog računanja)

Inteligencija se definiše kao sposobnost sistema da se adaptira u okruženju koje se konstantno mijenja. Ako se zna da je ljudska vrsta proizvod evolucije, onda se postavlja pitanje da li se **simuliranjem evolucije** može kreirati inteligentni sistem. Oblast nauke koja se bavi simulacijom evolucije se naziva **evolucijsko računarstvo**.

Evolucijsko računarstvo je oblik mašinskog učenja koji podrazumijeva korištenje modela **prirodne selekcije i genetike**. Postoje različite tehnike u praksi, a neke od njih su **genetički algoritmi, evolucione strategije i**

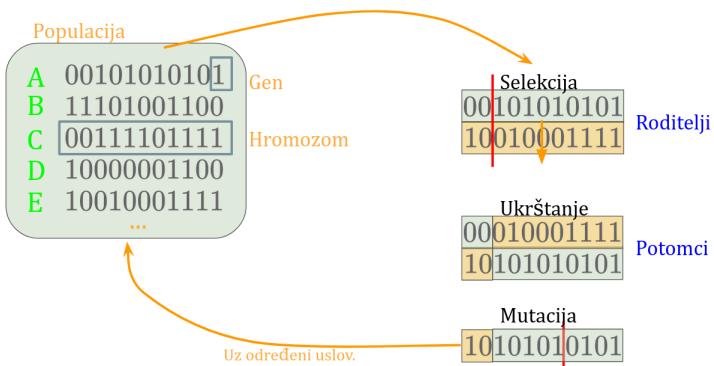
genetičko programiranje.

Recimo da imamo problem koji definišemo sa sljedećom matematičkom relacijom:

$$f(x) = x_1^2 + \log(x_2) + \frac{\sin(x_3)}{1 - x_4}$$

Cilj je pronaći maksimalnu vrijednost funkcije $f(x)$ ukoliko su postavljeni neki uslovi preko nekog genetičkog algoritma. Ovakve probleme možemo povezati sa neuronskim mrežama gdje su se tražile funkcije sa minimalnim gubitkom. Dakle, optimizacijske algoritme možemo korisiti i za optimizaciju neuronskih mreža i predstavljaju posebnu klasu problema.

Kod ovakvih problema se koriste pojmovi iz biologije kao što su **populacija**, **gen** i **hromosom**. Recimo da imamo neki problem predstavljen skupinom nekih rješenja. Skup rješenja se naziva **populacijom**. Pored ovog pojma, postoji i pojam **generacije** koji predstavlja populacije u različitim vremenskim momentima. Kako koristimo iterativan pristup za pronađazak rješenja, svaki put ćemo imati bolje i bolje populacije za rješavanje određenog problema. Moguća rješenja predstavljamo nekom kombinacijom **gena**. Svaki gen se može kodirati u binarnom sistemu pomoću nula i jedinica (iako mogu biti i kompleksni brojevi). Skupinu gena nazivamo **hromosomom**. Broj hromosoma zavisi o problemu koji rješavamo.



Slika 9.5: Pojmovi

Prvi korak kod korištenja genetičkih algoritama je proces **kodiranja**. Unutar ovog procesa, pokušavamo kodirati rješenje sa određenim brojem bita. Svako rješenje možemo povezati sa nekim konkretnim brojem ili stringom.

Da bismo došli do druge generacije, potrebno je primijeniti određene operacije nad populacijom koju imamo. Potrebno je izabrati parove hromosoma nad kojima ćemo primijeniti određene operacije. Proces izbora parova se naziva

selekcijom. Prvi par koji je izabran se naziva **roditeljima**. Nad ovim parom primijenimo određene operacije, kao što je naprimjer **ukrštanje** (dijelimo par na određen broj bita). Na predstavljenom primjeru par je podijeljen između drugog i trećeg bita. Proces ukrštanja je jednostavan, te je potrebno uzeti prvi dio iz prvog hromosoma i spojiti ga sa drugim dijelom drugog hromosoma. Rezultat ove operacije su **potomci**. Pored ukrštanja, postoje i operacije **mutiranja i kopiranja**. Kopiranje koristimo ukoliko smo zadovoljni sa određenim parom (ima zadovoljavajuću ocjenu). Mutiranje podrazumijeva uzimanje para hromosoma i mijenjanje vrijednosti određenih bita (gdje je bila nula, biti će jedan). U praksi se često uzima mutacija jednog bita. Ukoliko uočimo da ova operacija dovodi do boljih rezultata, moguće je povećati broj bita za izmjenu.

Broj hromosoma unutar populacije mora biti isti. Dakle, ukoliko smo dobili potomke koji su bolji od roditelja ili nekih drugih pripadnika unutar populacije, onda moramo odlučiti da izbacimo neke od tih hromosoma iz populacije.

Kod primjera matematičke funkcije, postavlja se pitanje kako generisati neka rješenja uz neki genetički algoritam

$$f(x) = x_1^2 + \log(x_2) + \frac{\sin(x_3)}{1-x_4} \quad \begin{matrix} \xrightarrow{\hspace{1cm}} & 00011 \\ \downarrow & \\ 00101 & 11101 \end{matrix} \quad \begin{matrix} \xrightarrow{\hspace{1cm}} & 10001 \\ \downarrow & \\ 00101 & 11101 & 00011 & 10001 \end{matrix}$$

Slika 9.6: Kodiranje matematičke funkcije

Prvi korak je **kodiranje** koje podrazumijeva kodiranje svake promjenljive na određen broj bita. Ukoliko ne znamo kako izabrati broj bita, onda isti biramo nasumično. U primjeru je fiksno odabранo da se svaka promjenljiva kodira na pet bita. Na ovaj način, generisana su određena rješenja koja predstavljaju vrijednosti promjenljivih. Moguće je generisati beskonačno mnogo rješenja koje možemo koristiti kao člana određene populacije.

Sljedeći korak je **kreiranje hromosoma**. Sva moguća rješenja formiramo u vektor. U ovom slučaju, jedan hromosom je predstavljen sa pet bita. Na ovaj način se formiraju populacije. Zatim je potrebno odrediti određene parametre koji su vezani za genetičke algoritme i na ovaj način doći do optimalnog rješenja. Dakle, potrebno je naći hromosom koji daje najbolje rješenje za dati problem (problem maksimizacije funkcije $f(x)$).

9.3 Genetički algoritmi

Genetički algoritmi se definišu kao grupe računarskih procedura koje konceptualno prate korake inspirisane biološkim procesima evolucije. Ove metode se još nazivaju i **evolucionim algoritmima**. Dakle, rezultati algoritma se evoluiraju kroz nekoliko generacija sve dok se ne dođe do optimalnog rješenja ili blizu optimalnog rješenja. Imaju veoma slične karakteristike kao najsposobniji biološki organizmi, poput osobina **samoorganizacije i adaptacije**. Metod ima mogućnost da uči na način da generiše sve bolje i bolje potomke, čija se sposobnost mjeri **funkcijom sposobnosti ili fitness funkcijom**. Genetički algoritmi korišteni su u velikom broju aplikacija kao što je usmjeravanje ili rutiranje vozila, predviđanje stečaja i pretraživanje web-a. Koncept genetičkih algoritama je prvi uveo **John Holland** 1975. godine.

Recimo da imamo igru sa dva igrača. Cilj igre je da se pogodi broj koji je zamislio protivnik u što manjem broju pokušaja na način da postoji određena povratna sprega od strane protivnika (postoji informacija o tome koliko je pokušaj blizu rezultata). Ova povratna informacija se koristi kao znanje iz kojeg se generiše novi pokušaj. Za demonstraciju igre, uvedimo ograničenja da je broj koji se pogada niz od šest cifara gdje svaka cifra može biti nula ili jedan (kao kod hromosoma). Dakle, igra je završena kada igrač pogodi broj.

Za rješenje postoji $2^6 = 64$ mogućih kombinacija. Najjednostavnija metoda je **metoda nasumičnog pokušaja i greške** (engl. *trial and error*) gdje je u prosjeku potrebno 32 puta da se pogodi broj. Međutim, postoji brži način za pogadanje ovog broja ukoliko se ispravno interpretira povratna informacija od protivnika (mjera valjanosit ili fitness funkcija).

Recimo da je broj koji se treba pogoditi **001010**.

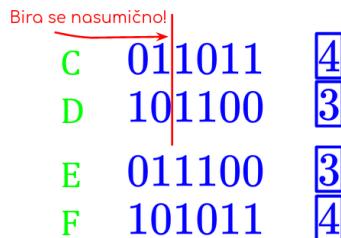
Ako rješavamo ovaj problem sa nekim genetičkim algoritmom, potrebno je pratiti određene korake. Naprimjer, prvi korak bi mogao biti da ponudimo četiri nasumično generisana odgovora. Recimo da smo uzeli brojeve 110100, 111101, 011011, 101100. Za svaki od pokušaja, protivnik odgovara kakvo je **bodovanje**, odnosno koliko cifara je pogodeno. Navedeni pokušaji bi imali bodovanja 1, 1, 4 i 3 respektivno.



Slika 9.7: Početno stanje

Drugi korak bi bio da se uklone sve kombinacije koje imaju najmanje bodovanje, a dvije kombinacije koje imaju najveće bodovanje proglašiti roditeljima. Kako prva dva pokušaja imaju mala bodovanja njih možemo izbaciti iz razmatranja, a preostala dva proglašavamo roditeljima.

Treći korak je da se izvrše određene operacije (mutiranje, kopiranje, ukrštavanje) nad odabranim roditeljima. Recimo da operacijom ukrštavanja podijelimo roditelje na dva dijela na proizvoljnom mjestu (neka je to između druge i treće cifre). Rezultat operacije nam daje dva nova niza (dva potomka): 011100 i 101011. Protivnik vraća bodovanja 3 i 4 za ova dva pokušaja. Kako nije došlo do poboljšanja bodovanja u odnosu na roditelje, potrebno je pokušati rascijepiti početne kombinacije roditelja. U slučaju dobijanja boljih rezultata, moguće je odbaciti roditelje i ubaciti potomke nad kojima bi se dalje vršile operacije.



Slika 9.8: Ukrštavanje roditelja

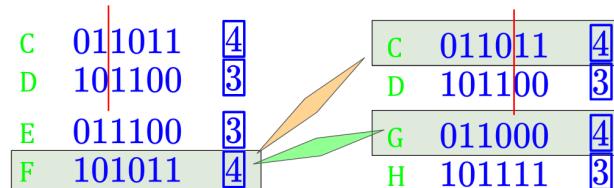
U ovom slučaju, četvrti korak je kopirati prethodno odabrane roditelje (011011 i 101100) kako bismo ponovo vršili operacije nad njima. Peti korak je ponovo upariti C i D na način da se kombinacije rascijepi na različitom mjestu (npr. između četvrte i pete cifre). Rezultati ukrštavanja ova dva niza daju potomke: 011000 i 101111. Protivnik vraća bodovanja 4 i 3 za ova dva pokušaja.

C	011011	4
D	101100	3
G	011000	4
H	101111	3

Slika 9.9: Ukrštavanje roditelja

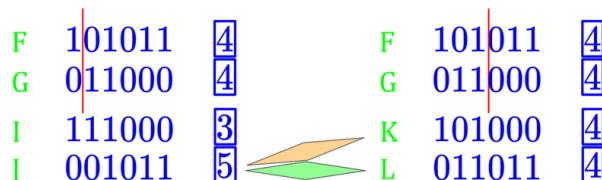
Dalje je potrebno ponoviti drugi korak kako bismo izabrali najbolje parove iz prethodnih rješenja, pri čemu postoji nekoliko kombinacija. Kako smo imali 3 niza sa bodovanjem 4 (011011, 101100 i 011000), mogući parovi su:

- 011011 i 011000
- 101100 i 011000



Slika 9.10: Mogući parovi

Posmatrajmo drugi navedeni par: 101011 i 011000. Ukoliko izvršimo operaciju ukrštavanja nad ovim parom gdje je podjela napravljena između prve i druge cifre, dobijamo novi par 111000 i 001011 sa bodovanjima 3 i 5. Izvršimo još jednu operaciju ukrštanja nad ovim parom gdje je podjela napravljena između treće i četvrte cifre. Kao rezultat dobijamo par 101000 i 011011 sa bodovanjima 4 i 4.



Slika 9.11: Ukrštavanje parova

U ovom slučaju, ponovo možemo da vršimo izbor najboljih potomaka na osnovu bodovanja. Moguća su dva para:

- 001011 i 101000
- 001011 i 011011

Ukoliko nad prvim parom izvršimo operaciju ukrštanja sa podjelom između pete i šeste cifre, dobijamo novi niz 001010 čime smo došli do rješenja ovog problema.

J	00101 1	5
K	101000	4
M	001010	6

Slika 9.12: Rješenje problema

Genetički algoritam je iterativna procedura koja moguća rješenja predstavlja kao nizove gena ili hromosoma, a zatim daje mjeru održivosti ili sposobnosti za preživljavanje koja se dobije putem **fitness funkcije**. **Fitness funkcija** je mjera cilja koja se želi postići. Obično se koristi funkcija minimuma ili maksimuma. Jedno moguće rješenje u jednoj iteraciji naziva se generacija. Iz jedne generacije roditelja i potomaka biraju se samo oni najsposobniji i oni postaju roditelji sljedećih potomaka.

9.3.1 Genetičke operacije

U postupku generisanja sljedećih potomaka koriste se posebne genetičke operacije.

Reprodukcijski operacije

Genetički algoritam kroz proces **reprodukcijski operacije** (engl. *reproduction*) generiše nove generacije poboljšanih rješenja na način da bira roditelje sa najvećim bodovanjem ili dodjeljivanjem veće vjerovatnoće za preživljavanje roditelja.

Ukrštanje

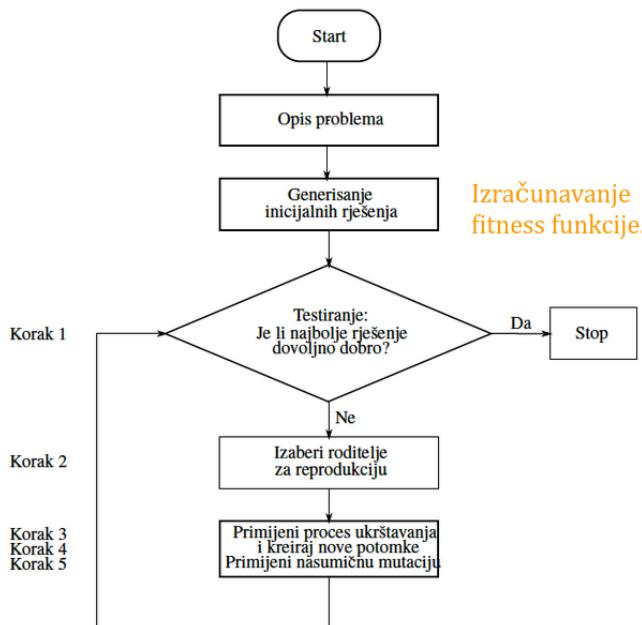
Ukrštanje (engl. *crossover*) podrazumijeva izbor nasumičnog rascjepa u nizu i zamjenju segmenata dva niza. Mogu se zamijeniti segmenti lijevo ili desno od rascjepa, sve dok je jedan segment iz jednog broja, a drugi iz drugog.

Mutacija

Mutacija (engl. *mutation*) je proizvoljna promjena u određenoj situaciji. U nekim situacijama se koristi kako algoritam ne bi došao u stanje iz kojeg ne može nastaviti dalje. Naprimjer, procedura može podrazumijevati zamjenu cifara 0 u 1 ili 1 u 0.

9.3.2 Proces

Proces genetičkog algoritma je prikazan sljedećim grafom:



Slika 9.13: Proces genetičkog algoritma

Na samom početku, potrebno je opisati problem a zatim generisati inicijalna rješenja. Postoji čitava teorija o tome kako generisati inicijalna rješenja jer ovaj korak predstavlja bitan aspekt kod korištenja genetičkih algoritama. Prije rješavanja samog problema, potrebno je napraviti procjenu da li se problem može riješiti genetičkim algoritmom. Ako se može riješiti genetičkim algoritmom, onda inicijalno rješenje kodiramo sa nulama i jedinicama. Nakon generisanja inicijalnih rješenja, izračunavaju se fitness funkcije svakog rješenja (**fitness populacije**) koje će dati određeno bodovanje na osnovu kojeg možemo zaključiti da li se približavamo rješenju. Ukoliko izračunamo sumu svih pojedinačnih fitness funkcija, dobiti ćemo određene vjerovatnoće za

potomke. Pomoću ovih vjerovantoća, moguće je rangirati potomke unutar jedne populacije.

Nakon navedenih početnih faza, iterativno generišemo neko novo rješenje te ga upoređujemo sa prethodnim. Ukoliko to novo rješenje nije dobro, potrebno je izabrati nove parove nad kojima ćemo primijeniti određene operacije kako bismo došli do nekog novog rješenja. Postupak se ponavlja dok rješenje ne bude dovoljno dobro.

9.3.3 Parametri genetičkog algoritma

Parametri genetičkog algoritma su:

- broj inicijalnih rješenja za generisanje
- broj potomaka za generisanje
- broj roditelja i potomaka koji će se zadržati za sljedeću generaciju
- vjerovatnoća mutacije (koja bi trebala da bude niska)
- distribucija vjerovatnoće pojave tačaka ukrštanja

Drugi problem koji se veže za genetičke algoritme je **problem ranca**. Problem ranca je jednostavan organizacijski problem koji se može riješiti direktnom primjenom analitičkih metoda. Ovaj problem se u praksi vrlo često koristi za rješavanje nekih kritičkih zadataka kao što su:

- Šta ponijeti u šatlu na nekoj svemirskoj misiji?
- Šta spakovati na robota koji se šalje na Mars?

Dakle, problem ranca se koristi kada je potrebno optimizirati broj stavki koje želimo ponijeti u rancu po nekom kriteriju. Kriterij može biti ukupna težina (kg) stavki u odnosu na njihovu vrijednost/cijenu ili drugu vrijednost u cilju postizanja određenog cilja.

Recimo da imamo problem ranca gdje je težina ranca ograničena na 22kg, a dostupne stvari (stavke) zajedno sa težinom i beneficijom su date sljedećom tabelom:

Stavka	1	2	3	4	5	6	7
Beneficija	5	8	3	2	7	9	4
Težina	7	8	4	10	4	6	4

Tabela 9.1: Problem ranca

Primjer jednog rješenja za ovaj problem je [1010100] pri čemu su izabrane stavke 1, 3 i 5. Ukupna težina ranca je 15kg. Koristeći genetički algoritam, može se doći do znatno boljeg rezultata.

Da bismo konkretno riješili ovaj problem, potrebno je uvesti sljedeće parametre:

- veličina populacije → 20
- broj generacija → 20 (pokušava se riješiti problem u 20 iteracija)
- vjerovatnoća ukrštanja → 0.7
- vjerovatnoća mutacije → 0.4
- kapacitet ranca → 7 stavki
- maksimalna težina ranca → 22kg

Nakon određenog broja iteracija, genetički algoritam bi mogao da dođe do rješenja [0 1 0 0 1 1 1] pri čemu su odabране stavke 2, 5, 6 i 7 i ukupna težina je 22kg. Ukupna beneficija iznosi 28. Na osnovu težine ili beneficije se može izračunati bodovanje problema, te izabратi ono bolje rješenje.

Veličina populacije se bira u odnosu na sami problem i na bazi iskustva. Ukoliko se izabere veći broj, može se pojaviti problem sa korištenjem velikog broja resursa. S druge strane, ukoliko se izabere manji broj, onda nema puno opcija za ukrštavanje.

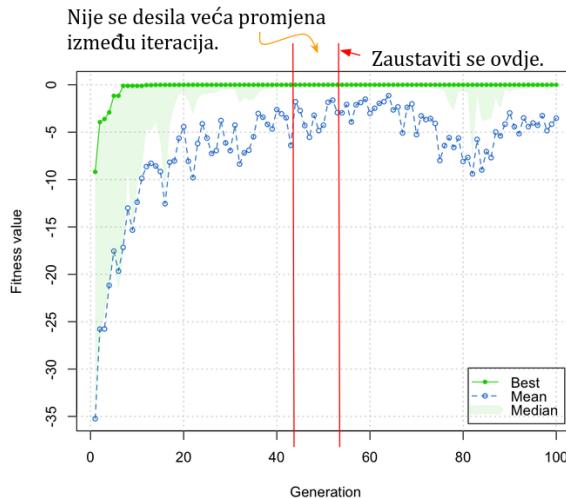
Unutar svake generacije, potrebno je izabrati potomke (roditelje) koji će se dalje ukrštati. Roditelji se biraju u odnosu na vrijednost fitness funkcije. Kada su vrijednosti fitness funkcije veoma različite, onda je potrebno rangirati hromosome po bodovanju, i biramo najbolje rješenje. Ako su vrijednosti fitness funkcije veoma slične, moguće je koristiti **metod ruleta** gdje će potomci biti reprezentirani u odnosu na neke vjerovatnoće. Dakle, jedinke su predstavljene na nekoj kružnici pri čemu će jedinke sa većom vjerovatnoćom zauzimati veću površinu. Ovakve jedinke će imati veću šansu da budu izabrane kao potomci od ostalih.

Inicijalna rješenja za svaku od populacija je moguće generisati na više načina. Jedan od najjjednostavnijih rješenja je da se ove vrijednosti nasumično generišu. Također, moguće ih je generisati koristeći **domensko znanje** o problemu (ako su nam poznata rješenja na nekim pozicijama). Treće rješenje bi bilo preko uniformne kombinacije mogućih vrijednosti.

Za obustavljanje traženja rješenja (evolucije) postoji nekoliko opcija. Moguće je koristiti maksimalan broj operacija ili neki minimalni opseg diverziteta (mjeru različitosti među jedinkama). Zatim, moguće je ciljati neku vrijednost

ukupne fitness populacije (a ne samo pojedinačnu vrijednost). Četvrto rješenje je da se zaustavi evolucija onda kada nema značajne promjene u fitness funkciji.

Neka smo dobili sljedeće rješenje nekog problema genetičkim algoritmom:



Slika 9.14: Rješenje problema

Problem ima sto generacija (iteracija). Analiziranjem rješenja, može se uočiti da je rješenje postignuto i stagnira u određenom broju iteracija. Na ovaj način, moguće je zaključiti da je nekada potrebno izvršiti manji broj iteracija od nekog predefinisanog broja. Ovaj broj se određuje pravljenjem grafa za problem. Nasumično se odaberu dvije lokacije (na grafu označene crvenom bojom) i u ovom intervalu se provjerava da li je došlo do neke značajne promjene u generisanju novih rješenja. Ukoliko nije došlo do velike promjene, algoritam se može zaustaviti u toj iteraciji.

9.4 Prednosti i mane

Prednosti genetičkih algoritama su:

- jednostavnji za kodiranje
- daju mnogo rješenja i moguće je izbjegći lokalne ekstreme
- pogodni za paralelizaciju

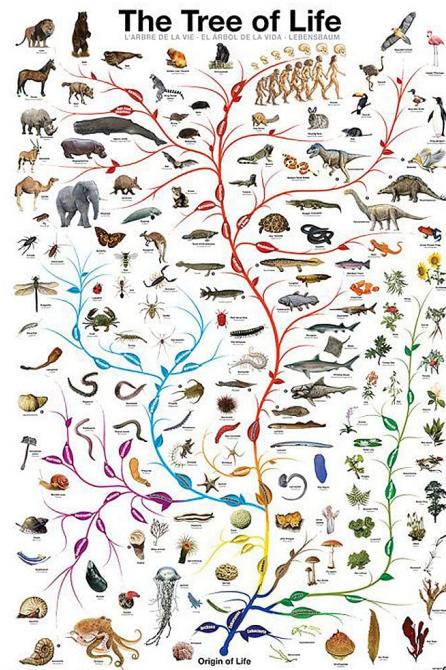
Mane genetičkih algoritama su:

- veoma spori
- izbor fitness funkcije nije trivijalan proces (sigmoid, radial basis function)

Predavanje 11

10.1 Evolucija

Evolucija je algoritam koji generiše sva živa bića na planeti Zemlji. **Drvo života** (engl. *The Tree of Life*) je slika na kojoj je prikazan proces evolucije. Ovakav sistem je nakon dužeg vremenskog perioda generisao ljudsku inteligenciju koja danas može da stvori sisteme koji su približno kompleksniji kao i sistemi koje je i sama evolucija kreirala.



Slika 10.1: Drvo života

Smatra se da je biologija pronašla rješenja za teške računske probleme i ova činjenica predstavlja ideju evolucijskog računarstva. Evolucija predstavlja algoritam koji nema kraja i smatra se da je **open-ended problem**.

Inteligencija jata povećava vjerovatnoću preživljavanja određene grupe jedinki u odnosu na individualnu jedinku. Postoje određena istraživanja i kod formiranja određenih grupa u ljudskoj populaciji gdje se neki problemi prevazilaze u takvoj grupi. Također, postoje određeni algoritmi koji se baziraju na kopiranju ponašanja grupnih jedinki u prirodi (naprimjer navođenje ljudi da glasaju na određen način).

Dakle, inteligencija jata predstavlja mozak mozgova. Suštinski, neuroni pojedinačnog sistema se kombinuju u jedan novi sistem kako bi se došlo do nove inteligencije koja se naziva **superinteligencija**. Evidentno je da su ovakvi sistemi bolji od pojedinačnih sistema za rješavanje nekih kompleksnijih problema. Algoritmi inteligencije jata se jako često koriste za rješavanje optimizacijskih problema.

Definicija **inteligencije roja** je sljedeća:

"Intuitivni pojam "**inteligencije roja**" je pojam "roja" agenata (bioloških ili vještačkih) koji bez centralne kontrole kolektivno (i samo kolektivno) izvršavaju (nesvjesno i na pomalo slučajan način) zadatke koji obično zahtijevaju neki oblik "inteligencije"."

Potencijalne prednosti upotrebe inteligencije roja su:

- ekonomičnost - jedinke (engl. *units*) koje čine roj su jednostavne, pa su u principu jednostavne za masovnu proizvodnju, modularizaciju, mogu mijenjati uloge i lako se odlažu (engl. *disposable*)
- pouzdanost - roj čini veliki broj jedinki/komponenti pa je destrukcija/uništavanje nekih jedinki zanemariva za izvršavanje cilja, jer roj ima mogućnost adaptiranja
- sposobnost - bolja sposobnost izvršenja zadataka u odnosu na centralizirane sisteme (npr. izbjegavanje detekcije od strane neprijatelja)

Potencijalne sfere korištenja inteligencije roja su: tehnologije vojne odbrane i svemirske tehnologije (npr. kontrola bezpilotnih letjelica na tlu, u vodi, u zraku, itd.), sistemi za proizvodnju, napredni računarski sistemi (biocomputing), medicinske tehnologije, telekomunikacije i slično.

Sistemi su bazirani na ograničenom broju jedinki koji se kreće od

$$10^2 \rightarrow 10^s, s << 23$$

Postoje neke tehnologije i nauke koje su direktno vezane za inteligenciju roja:

- vještački život
- etologija (grana biologije koja proučava ponašanje životinja uključujući ljude)
- robotika
- vještačka inteligencija
- izračunljivost (engl. *computing*)
- samoorganizacija
- kompleksnost
- ekonomija
- sociologija

U svim poljima javlja se potreba za proučavanjem, modeliranjem i predviđanjem funkcionalnosti grupe jedinki koje samo kada rade zajedno (na veoma nestruktuiran i nesvjestan način) izvršavaju neki "inteligentni" zadatak.

Eksperiment dvostrukog mosta je vezan za problem pronalaženja najkraćeg puta od jedne tačke do druge. Ovaj eksperiment je nastao na bazi posmatranja mravlje kolonije. Dakle, postoje dvije tačke: S (start) i F (finish). Pri tome, postoje dvije putanje od tačke S do tačke F. Jedna je direktna, dok je druga preko luka (duža putanja). Ukoliko se vještački naprave ovakve prepreke gdje se kod F nalazi hrana, a kod S određeni broj mrvava, nakon određenog vremena će se svi mrvavi kretati isključivo po kraćem putu.



Slika 10.2: Dvostruki most

Analizirajući ovaj problem, naučnik Goss sa grupom autora 1989. dolazi do spoznaje da mrvavi pronalaze najbolji put putem **stigmergije**. **Stigmergija** je nastala od dvije riječi: stigma + ergon (označiti + raditi/akcija). Zaključak je da mrvavi komuniciraju na način da modifiraju okolinu kojom se kreću. Mrvavi zapravo ostavljaju hemijske tragove preko **feromona**. Ostali mrvavi čitaju tu informaciju i na osnovu toga modifikuju ili nastavljaju tom

putanjom u zavisnosti od toga koliko feromona ima koja staza. Pored toga, unutar algoritma postoji i vjerovatnoća zbog koje se mrav neće uvijek kretati pravom stazom, ali će nakon određenog broja iteracija doći do kraćeg rješenja. Pretežno svi algoritmi vezani za inteligenciju roja su bazirani na populaciji i stvaranju određenih pravila koje unose određenu vjerovatnoću u cilju stvaranja nekog rješenja.

Stigmergija je pojava koju viđamo i kod ljudi na različite načine. Naprimjer, ukoliko na nekoj mapi sa različitim brojem stanica pokušavamo naći najkraći put od jedne stanice do druge stanice, određenom logikom biramo najoptimalniji put.

Naučnici su posmatranjem raznih bioloških sistema, došli do nekoliko bitnih koncepata inteligencije roja:

- višestruke komunikacije različitih vrsta (između jedinki)
- slučajnost (slučajne fluktuacije u ponašanju/kretanju)
- pozitivna povratna sprega koja ojačava slučajne fluktuacije
- negativna povratna sprega za stabilizaciju

Postoje različiti načini komunikacije između jedinki unutar roja:

- indirektna komunikacija (stigmergija) kroz modifikaciju okoline
- direktna komunikacija između dvije jedinke (preko antena ili donje vilice)
- emitovanje akustično ili hemijski u određenom opsegu u prostoru (engl. *broadcasting*).

Izbor komunikacije je izuzetno bitan (kritičan) za zadatak koji se rješava, pa se bira od zadatka do zadatka.

Robotika roja opisuje dizajn grupe robotskih jedinica koje izvode kolektivni zadatci. Jedna robotska jedinica ne može sama riješiti zadatci, te robotske jedinice zajedno pokušavaju ostvariti zajednički cilj bez centralizirane kontrole. Ovakvi sistemi zahtjevaju:

- dizajn mehanike
- upravljanje (kontrolu)
- komunikaciju

Trenutno se najviše istražuju zadnje dvije komponente: upravljanje i komunikacija.

10.2 Vještački život

Vještački život (engl. *Artificial Life*) ili skraćeno **A-Life** je područje proučavanja u kojem istraživači ispituju sisteme povezane sa prirodnim životom, njegovim procesima i njegovom evolucijom koristeći simulacije s računarskim modelima, robotikom i biohemijom. A-Life je konceptualno smješten negdje između nauke i tehnologije te između biologije i robotike. Cilj je imitacija procesa evolucije (odnosno nekog biološkog sistema) koristeći neke odredene nauke bazirane na vještačkim elementima.

U svojoj trenutnoj upotrebi, pojam vještački život (A-Life) je uveo Langton 1989. godine, koji ga je izvorno definirao kao „**život koji stvara čovjek, a ne priroda**“, odnosno, to je proučavanje sistema napravljenih od strane čovjeka koji pokazuju ponašanja karakteristična za prirodne životne sisteme.

Teorije koje su temelj A-Life koncepta, a vezane su i za inteligenciju roja su **samoorganizacija i složenost**.

Samoorganizacija omogućava da se sistemi koji se inicijalno ponašaju bez nekog reda, uspiju organizovati u struktuirane sisteme. Kao rezultat dobijamo robusne sisteme otporne na smetnje (perturbacije).

Složenost se očituje u samoj interakciji osnovnih jedinki.

10.3 Optimizacijski problemi

Svaki optimizacijski problem se sastoji od nekoliko komponenti:

- **Funkcija cilja ili objektivne funkcije** - jedan ili više ciljeva (minimalna cijena, emisija, gubitak, profit, najkraći put i slično)

$$\min f_i(x), i \in 1, 2, 3, \dots, N$$

- **Skup promjenljivih** - u nekoj relaciji sa funkcijom cilja

$$x = (x_1, x_2, x_3, \dots, x_n), n = (1, 2, 3, \dots, N)$$

- **Skup ograničenja** - jednakosti ili nejednakosti

$$g_k(x) \leq 0 (k = 1, 2, 3, \dots, K)$$

Primjer jednog optimizacijskog problema:

$$f(x) = x_1^2 + \log(x_2) + \frac{\sin(x_3)}{1 - x_4}$$

sa ograničenjima:

$$x_1 \leq 10, x_2 \geq 10, x_3 \geq 3, x_4 \geq 1$$

gdje su (x_1, x_2, x_3, x_4) promjenljive, a $f(x)$ funkcija cilja.

Optimizacijski algoritam pretražuje moguće vrijednosti promjenjivih (parametara) da pronađe minimalnu vrijednost funkcije cilja uz zadana ograničenja.

10.3.1 Optimizacijske metode

Postoje različite metode koje se koriste za rješavanje optimizacijskih problema.

Konvencionalna rješenja

- Linearno programiranje
- Dinamičko programiranje
- Cjelobrojno programiranje
- Kvadratno programiranje
- Lagrange metod
- Gradijentne metode
- Newton-Raphson
- ...

Problem ovakvih metoda je što ne mogu pronaći globalni minimum.

Metaheuristički algoritmi

Ovi algoritmi su inspirisani prirodnim fenomenima (ljudskim i životinjskim ponašanjem). Karakteristični su iz razloga što ne računaju izvod kod optimizacije. Bazirani su na populacijama, odnosno, podrazumijeva se formiranje populacija za rješavanje nekog problema. Ovakvi algoritmi mogu da pronađu globalni minimum.

- Genetički algoritmi (GA)
- Particle Swarm Optimization (PSO)
- Firefly Algorithm (FA)
- Ant Colony Algorithm (ACA ili ACO)
- Artificial Bee Colony Algorithm (ABC)
- ...

Hibridni algoritmi

Ovakvi algoritmi nastaju kao kombinacija dva ili više algoritama. Cilj je ukloniti slabosti pojedinačnih algoritama.

- GA-FA
- GA-PSO
- PSO-FA
- ...

10.4 Roj

Inteligencija roja je **inteligencija** postignuta **kolektivno, slučajno i nesvjesno**. Osnovni roj koji zadržava ova četiri elementa može se definisati kao:

- Uređeni skup od N jedinica opisanih pomoću N komponenata koje se označavaju sa v_i ($i = 1, 2, \dots, N$) pri čemu bilo koja jedinica može ažurirati vektor u bilo koje vrijeme t_i koristeći funkciju f vektorskih komponenata K_i :

$$\forall i \in N : v_i(t+1) = f(v_{k \in K(i)}(t))$$

Slučajnost se postiže kroz vrijeme ažuriranja vektora. Proces evolucije se dešava "nesvjesno" jer jedinke nemaju moć procesiranja. Inteligencija će se postići na način da se pokreću odgovarajući algoritmi preko funkcije ažuriranja f .

10.4.1 Optimizacija roja

Optimizacija roja je isto što i optimizacija bilo koje funkcije. Dva algoritma koja se najčešće koriste za optimizaciju roja su **Ant Colony Optimization** (ACO) i **Particle Swarm optimization** (PSO). Cilj ovih metoda je isti kao kod bilo koje optimizacijske metode a to je optimizacija funkcije (npr. minimizacija). Vrši se pretraživanje prostora na sistematičan ili optimizovan način. Najčešće se koriste metode koje koriste neku vrstu slučajnog pretraživanja prostora.

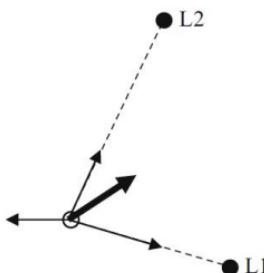
Optimizacija roja čestica

Optimizacija roja čestica ili **Particle Swarm Optimization** (PSO) je opisana 1995. godine od strane Kennedy i Eberhart-a, te je inspirisana socijalnim ponašanjem ptica i riba.

Kreće se od populacije koja je sačinjena od određenog broja jedinika. Potrebno je definisati broj jedinki (čestica) unutar populacije i broj iteracija. U prvom koraku se vrši inicijalizacija čestica pri čemu se te čestice nasumično

postavljaju u nekom prostoru po kojem se vrši pretraživanje. U narednom koraku se vrši modifikacija položaja svih čestica, pri čemu je bitno na koji način vršimo ovu modifikaciju (koje informacije se uzimaju da bi pomjerili neku česticu).

Na sljedećem primjeru, "praznom" tačkom je prikazana neka čestica. Ova čestica se trenutno kreće lijevo (prikazano strelicom). Pored ove tačke, postoje još dvije karakteristične tačke: L1 i L2. Čestice se vrlo često organizuju u susjedstva, odnosno, u tačno određene strukture. Ove strukture će omogućiti kretanje u određenim smjerovima. Prikazani su i vektori koji će odrediti smjer kretanja prema ove dvije tačke. Dalje, poznato je da je u lokaciji L1 funkcija imala minimalnu vrijednost, dok je za L2 poznato da se tu nalazi minimalna vrijednost funkcije poznata susjednim jedinkama. Na osnovu ovih informacija, formira se smjer kretanja čestice koju posmatramo u narednoj iteraciji. Podebljana strelica predstavlja kombinaciju svih dobijenih informacija dodavajući i određen aspekt slučajnosti kretanja (određeni koeficijenti slučajnosti se množe sa vektorima, čija ukupna suma određuje smjer kretanja čestice u narednoj iteraciji).



Slika 10.3: Primjer

Dakle, pozicija svake jedinke u roju je tačka u promjenljivom prostoru funkcije koja se treba optimizirati. Svaka jedinica pokušava da dosegne poziciju koja odgovara minimumu funkcije. Svaka jedinka može imati grupu susjeda kao parametar, dok u ekstremnom slučaju grupa može biti čitav roj. Algoritam staje onda kada je algoritam dovoljno blizu globalnog minimuma ili kada se dosegne određeni broj iteracija.

Optimizacija roja čestica spada u kategoriju stohastičkih algoritama zasnovanih na populaciji (kao što su genetički algoritmi). U mnogim slučajevima nadmašuje genetičke algoritme i dosta je jednostavan za primjenu. Slično svim algoritmima za optimizaciju ovog tipa, konvergencija optimizacije roja čestica se oslanja na upotrebu heuristike (ne mora biti konvergencija ka optimumu). Ovaj algoritam ne garantira konvergenciju čak ni do lokalnog minimuma, ali se mijenjanjem parametara može kontrolisati. Navedeni

algoritam je također neučinkovit u problemima dinamičke optimizacije, odnosno, u problemima u kojima se optimalna lokacija mijenja.

Glavne klase aplikacija koje mogu koristiti ovu optimizaciju su u područjima kao što su:

- neuronske mreže (trening, učenje pod nadzorom i bez nadzora, odabir arhitekture i slično)
- učenje igara (engl. *game learning*)
- grupisanje / klastering
- dizajn (avionskih krila, antena, sklopova ...)
- raspoređivanje i planiranje (održavanje, prodavac, prenos snage ...)
- kontroleri (putanja leta, temperatura vazduha ...)
- data mining

Čestica predstavlja pojedinačno (moguće) rješenje u prostoru rješenja. Svaka čestica ima ocjenu sposobnosti koja se dobije primjenom funkcije sposobnosti. Čestice imaju svoju brzinu koja ih usmjerava u nekom prostoru i kreću se na način da slijede trenutno optimalne čestice.

Osnovni koraci algoritmi su navedeni u nastavku.

Prvi korak je inicijalizirati nasumične čestice gdje je potrebno odrediti broj čestica i broj iteracija. U drugom koraku vršimo pretraživanje prostora za optimalno rješenje kroz generacije/iteracije. Čestice se kreću kroz prostor rješenja i ocjenjuje se njihova sposobnost preko **fitness funkcije** gdje se javlja problem izbora ispravne fitness funkcije. U svakoj iteraciji, čestica se ažurira sa dvije najbolje vrijednosti:

- pbest: vektor najbolje lokalne pozicije čestica
- gbest: vektor najbolje globalne pozicije čestica

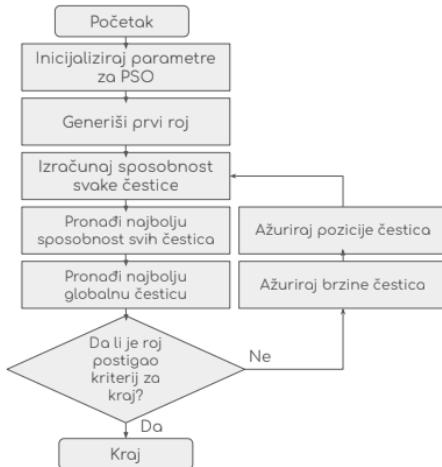
Ako su vrijednosti sposobnosti u trenutku t bolje od prethodne najbolje vrijednosti (pbest), potrebno je ažurirati trenutnu vrijednost kao novi pbest. Pored toga, potrebno je izabrati česticu s najboljom fitness vrijednošću svih čestica (gbest). Svaka čestica ažurira vektor brzine po formuli:

$$\begin{aligned} v_i^{t+1} &= v_i^t + c_1 \cdot rand1 \cdot (p_{best_i} - x_i^t) + c_2 \cdot rand1 \cdot (g_{best_i} - x_i^t) \\ x_i^{t+1} &= x_i^t + v_i^{t+1} \end{aligned}$$

gdje je v_i^t brzina u i-toj iteraciji, c_1 i c_2 faktori akceleracije/povjerenja, $rand1$ slučajan broj u opsegu $[0,1]$ i x_i^t pozicija trenutne čestice.

Postupak se ponavlja sve dok roj ne postigne kriterij za kraj.

Dijagram toka je sljedeći:



Slika 10.4: Dijagram toka

Prednosti algoritma su:

- prilično neosjetljiv na skaliranje varijabli dizajna
- jednostavna implementacija
- lako paraleliziranje za konkurentnu obradu
- nema derivacija (optimizaciju crne kutije - ne mora znati puno o problemu koji rješava)
- vrlo malo parametara
- efikasan za globalno pretraživanje

Mane algoritma su:

- teži ka brzom i preranom približavanju u srednjim optimalnim tačkama
- spora konvergencija u profinjenoj fazi pretraživanja (slaba sposobnost lokalnog pretraživanja)

Optimizacija mravlje kolonije

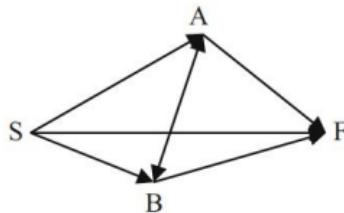
Kreće se od osnovnog problema pronalaženja najkraćeg puta. Rad je baziran na istraživanju dva naučnika: Marco Dorigo i Jean-Louis Deneubourg. Osnovna ideja algoritma je apstrakcija i generalizacija ideje koju je 1989. godine predložio Goss. Prva generalizacija podrazumjeva pronalazak najkraćeg puta između startne pozicije S i finalne tačke F.



Slika 10.5: Dvostruki most

Problem se može predstaviti preko grafa sa tri čvora (S, A, F) i tri veze ($S \rightarrow A$, $A \rightarrow F$, $S \rightarrow F$). Pri tome, najkraći put bi bio $S \rightarrow F$, a najduži $S \rightarrow A \rightarrow F$.

Ukoliko se navedeni problem proširi na način da dodamo tačku B i povežemo sve moguće putanje, pojavit će se tri nove veze: $S \rightarrow A \rightarrow F$, $S \rightarrow B \rightarrow F$ i $S \rightarrow B \rightarrow A \rightarrow F$.



Slika 10.6: Optimizacija mravlje kolonije

Da bismo formirali algoritam, zamislimo N mrvava čiji je cilj da pronađu najkraći put u grafu od čvora S do čvora F. Ovi mrvavi nasumično biraju čvor koji će posjetiti. U svakoj novoj iteraciji, svaki mrvav odlučuje po kojoj putanji će putovati do sljedećeg čvora, u odnosu na vjerovatnoću proporcionalnu količini feromona na putu relativnu u odnosu na ukupnu količinu feromona na svim mogućim putanjama koje mrvav može izabrati. Kada mrvav dode do cilja, pamti dužinu puta L_k . Zatim, mrvav se vraća istom putanjom i ostavlja feromonske tragove u omjeru $\frac{1}{L_k}$. Na ovaj način modificira se graf i povećava se vjerovatnoća da će sljedeći mrvav izabrati najkraći put. Na kraju će svih

mravi pratiti istu putanju (ako algoritam konvergira). Kao kod svih optimizacijskih metoda, ne postoji garancija da će se ovo desiti.

Postoje neki parametri koje je važno spomenuti:

- Svaki luk ima apriornu tendenciju (vjerovatnoću) da se pređe (bez obzira na sadržaj feromona)
- Svaki mrav zadržava u sjećanju tabu-listu luka kojima se ne smije prelaziti, kako bi se izbjegle petlje
- Feromoni isparavaju zadanom brzinom

Ovakvi algoritmi se primjenjuju na sljedeće vrste optimizacijskih problema:

- kontinualne i diskretne
- ograničene i neograničene
- statičke i dinamičke
- probleme sa jednostrukim ili višestrukim ciljevima

Prva primjena ovog algoritma je bila problem trgovackog putnika (NP-hard optimizacijski problem).

Glavne klase ostalih aplikacija su:

- naručivanje (raspoređivanje, usmjeravanje)
- dodjela (treniranje neuronskih mreža, segmentacija slika, dizajn)
- pronalaženje podskupova (maksimalno neovisni skup)
- grupisanje (grupisanje, pakovanje u kante)

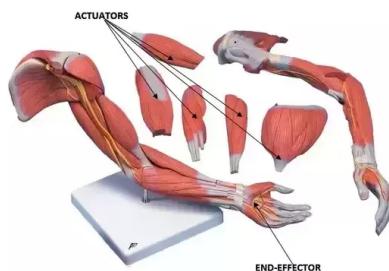
Sistemi bazirani na ovim algoritmima imaju memoriju, mogu osjetiti stanje okoline ukoliko je potrebno, koriste diskretno vrijeme i predstavljaju optimizacijske probleme.

Predavanje 12

11.1 Agent

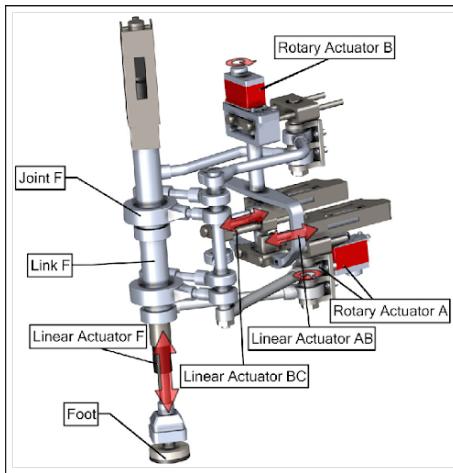
Agent predstavlja bilo šta za što možemo reći da može percipirati svoju okolinu putem **senzora** i djelovati na tu okolinu putem **aktuatora**. Kao primjer agenta možemo uzeti čovjeka koji putem pet čula (senzora) može da mozgom (aktuator) izvrši percepciju objekata u okolini. **Objekat zapažanja** (engl. *percept*) je objekat koji agent zapaža u svojoj okolini i uzima u obzir kao ulazni podatak. Agent može biti programiran da uočava samo određene objekte i da zanemaruje druge. **Sekvenca objekata zapažanja** predstavlja historiju svega što agent vidi (npr. sve što vidimo u toku jednog dana).

Aktuatori su naprave ili uređaji kojim se na pobudu upravljačkog (kontrolnog) signala pokretni dijelovi sistema dovode u željeni položaj, ostvaruje se njihovo gibanje ili razvija sila ili moment sile (zakretni moment) kojim ti dijelovi djeluju na okolinu. Kod čovjeka određeni signali iz mozga pokreće aktuatore (mišiće).



Slika 11.1: Aktuatori kod čovjeka

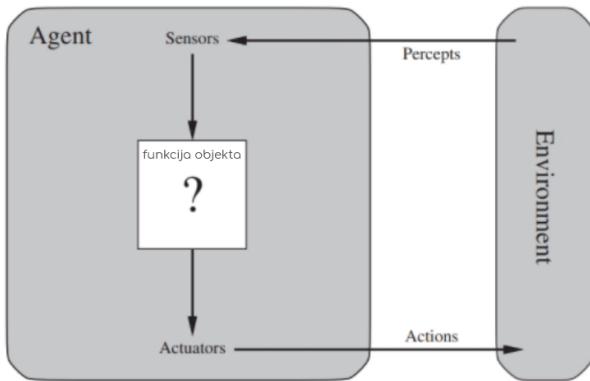
Aktuatori mogu biti i razne vrste motora (linearni, rotacijski):



Slika 11.2: Aktuatori kod motora

Inteligentni agent je onaj agent koji može donijeti odluku na osnovu nekih informacija (percepcije) iz okoline (objekata zapažanja). Definisanjem svih mogućih odluka, izbora i akcija se definiše kako jedan agent **funkcioniše**. Definisanjem **funkcije agenta** moguće je mapirati ili preslikati percepciju u akcije. **Program agenta** je interna implementacija funkcije agenta.

Ukoliko pokušavamo konceptualno da predstavimo agenta, moguće je za primjer uzeti sljedeću sliku:



Slika 11.3: Model agenta

Agent je definisan u odnosu na neku okolinu koja može biti poznata, ali i ne mora biti. Ukoliko nam okolina nije poznata, onda agent nema nikakve

senzore. Neka je ovaj agent napravljen za rješavanje nekog problema i definisimo funkciju objekta. Iz okruženja agent dobija određene informacije putem senzora. Ove informacije šalje programu (implementaciji) koji treba da napravi odluku koja će proizvesti neku akciju. Akcija se dalje prosljeđuje aktuatorima koji realizuju akciju u odnosu na okolinu.

Suštinski, da bismo napravili dobar inteligentni sistem, potrebno je poznavati okolinu i razmisliti o mogućim načinima implementacije funkcije agenta.

11.1.1 Koncept racionalnosti

Racionalni agent je onaj agent koji radi ispravan ili dobar posao. Ispravnost se definiše u odnosu na sami zadatak jer za svaki zadatak znamo kada agent izvršava posao dobro, a kada loše. Samim time, tabela preslikavanja percepције u akcije treba da sadrži sekvence ispravnih stanja.

Kod svakog agenta postoji sekvenca događaja koje treba pratiti:

1. agent percipira (opaža) svoju okolinu
2. percepcija okoline će dovesti do nekih akcija
3. akcije dovode do promjene okoline

Ukoliko je sekvenca promjene agenta poželjna, onda smatramo da je agent napravio ispravne korake (npr. robot-usisivač koji je očistio svoju okolinu). Performanse agenta je potrebno mjeriti putem neke metrike (**mjera performansi**). Jedan od načina mjerenja performansi jeste nagrađivanje agenta sa određenim bodovima, te na osnovu ove akumulacije određujemo da li agent radi dobro ili loše.

Racionalnost zavisi od:

- mjere performansi
- agentovog prethodnog znanja
- akcija koje agent može da poduzme
- agentove trenutne percepције (ili liste objekata zapažanja)

Racionalni agent može da se definiše na sljedeći način:

"Za svaku moguću sekvencu opažanja, racionalni agent trebao bi odabratи onu akciju (radnju) za koju se očekuje da bi mogla **maksimizirati mjeru performansi**, s obzirom na dokaze pružene sekvencom opažanja i bez obzira na ugrađeno (prethodno) znanje koje agent ima."

Suštinski, agent ima zadatak da izvrši maksimizaciju mjere performansi.

Robot-usisivač može biti isprogramiran na način da za svaki očišćeni blok dobije po jedan bod. U ovom procesu, možemo da ograničimo broj radnji u vremenu. Nakon toga, zadatak robota je da stane. Na ovaj način sprječavamo stanje ulaska u petlju. Ovakav agent može da ima akcije da se kreće desno, lijevo ili da usisa. Robot-usisivač može da percipira da li je lokacija čista i da odredi gdje se trenutno nalazi.

Agent može postati iracionalan uvođenjem mjere kazne, ali iz ovakvog stanja on treba biti u mogućnosti da odabere korake koji će dovesti do pozitivnih bodova. Također, potrebno je napraviti mogućnost detekcije finalnog stanja i ugasiti usisivač ili postaviti isti u stanje mirovanja.

Racionalne akcije nisu uvijek savršene ili korektne i ne moraju da imaju pozitivan ishod, iako na prvu ruku mogu da izgledaju korektne. Perfektnost nije racionalnost jer ona vrši maksimiziranje stvarnih perfomansi. Nije moguće projektovati perfektnog agenta jer poznavanje okoline nije uvijek potpuno definisano. S druge strane, moguće je minimizirati neefikasne akcije i na ovaj način spriječiti agenta da pravi neadekvatne akcije. Agent može prikupljati informacije koje će omogućiti zapaženje objekata za bolje odlučivanje u narednim koracima (posmatra sekvencu prethodnih događaja).

Racionalni agenti trebaju imati **autonomiju** jer se u tom slučaju ne moraju eksplicitno programirati za svaki novi slučaj.

11.2 Okruženje agenta

Okruženje zadatka (engl. *task environment*) je okolina koja se definiše za rješavanje nekog problema. Okolina se vrlo često određuje u odnosu na četiri parametra (**PEAS**):

1. performanse (engl. *performance*)
2. okruženje (engl. *environment*)
3. aktuatori (engl. *actuators*)
4. senzore (engl. *sensors*)

Navedeno se vrlo često definiše pomoću tabele:

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Slika 11.4: PEAS

Postoje različite klasifikacije okruženja i za svaki problem definišemo niz karakteristika tog okruženja.

Okruženje može biti stvarno ili virtuelno. Potrebno je što bolje razumijeti okruženje kako bi ponašanje agenta bilo što bolje.

Softverski agenti (softbot) predstavljaju virtuelne agente. Kao primjer možemo uzeti agenta koji pretražuje Twitter za interesantne novosti. U ovakvim slučajevima, okruženje je izuzetno dinamično.

11.2.1 Osobine okruženja

Osobine okruženja su jako bitne jer postoji veliki raspon zadataka. Mogu se klasificirati na različite, gdje okruženje može biti:

- **potpuno definisano** - ako agent može da dobije sve informacije iz okruženja za donošenje odluka preko svojih senzora kroz čitav period vremena
- **djelimično definisano** - važi suprotno
- **nedefinisano** - agent koji nema senzore

Okruženja se mogu klasificirati u odnosu na:

- broj agenata
- vrste komunikacije (kompetitivnost, kooperativnost)
- dinamičnost
- ...

11.2.2 Vrste okruženja

Okruženje nije jednostavno definisati. Uspješnost projekta u velikom mjeri zavisi od uspješno definisanog okruženja. Primjeri različitih okruženja su dati sljedećom tabelom (zajedno sa napisanim vrstama okruženja):

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle Chess with a clock	Fully Fully	Single Multi	Deterministic Deterministic	Sequential Sequential	Static Semi	Discrete Discrete
Poker Backgammon	Partially Fully	Multi Multi	Stochastic Stochastic	Sequential Sequential	Static Static	Discrete Discrete
Taxi driving Medical diagnosis	Partially Partially	Multi Single	Stochastic Stochastic	Sequential Sequential	Dynamic Dynamic	Continuous Continuous
Image analysis Part-picking robot	Fully Partially	Single Single	Deterministic Stochastic	Episodic Episodic	Semi Dynamic	Continuous Continuous
Refinery controller Interactive English tutor	Partially Partially	Single Multi	Stochastic Stochastic	Sequential Sequential	Dynamic Dynamic	Continuous Discrete

Slika 11.5: Vrste okruženja

U odnosu na broj agenata

U odnosu na broj agenata, okruženje može biti:

- sa jednim agentom - agent samostalno rješava neki problem (npr. zagonetka)
- sa više agenata - agensi komuniciraju međusobno u istom okruženju (npr. igranje šaha)

Agenti mogu da rade jedan protiv drugog gdje imamo **kompetitivno okruženje**. S druge strane, ukoliko agenti rade zajedno onda imamo **kooperativno okruženje**.

U odnosu na stanja

Okruženje može biti:

- **Determinističko** - bazirano samo na predefinisanim stanjima i akcijama agenata
- **Stohastičko** - suprotno od determinističkog, postoji određena vjerovatnoća djelovanja
- **Neizvjesno** - ne može se potpuno opažati i nije determinističko

U odnosu na prethodne akcije

U odnosu na prethodne akcije, okruženje može biti:

- **Epizodno** - akcije su diskretne (atomske), ne uzimaju se obzir prethodne akcije, nego samo trenutna opažanja
- **Sekvencijalno** - uzimaju se u obzir prethodne akcije i na bazi toga se donosi nova odluka

U odnosu na stanje

Posmatrajući stanje okruženja, ono može biti:

- **Dinamičko** - ukoliko se okruženje mijenja u vremenu donošenja (razmišljanja o) naredne odluke (vožnja autonomnog vozila)
- **Statičko** - suprotno od dinamičkog (rješavanje puzle)
- **Poludinamičko** - okruženje se ne mijenja, ali se mijenja bodovanje performansi (igranje šaha sa aspektom vremena)

U odnosu na vrijeme

U odnosu na pojam vremena u okruženju (način opažanja objekata i agentovih akcija), okruženje može biti:

- **Diskretno** - konačan broj koraka i stanja (igra dame u šahu)
- **Kontinualno** - agent prelazi iz jednog okruženja u drugo vrlo glatko (brzina autonomnog vozila koja se stalno mijenja)

U odnosu na poznavanje okoline

Ova podjela se odnosi na agentova poznavanja okruženja u kojem se nalazi, kao i na stvari fizičke prirode:

- **Poznato** - ishodi i vjerovatnoća ishoda akcija je poznata ili zadana
- **Nepoznato** - suprotno od poznatog

11.3 Struktura agenta

Zadatak svakog programera koji se bavi vještackom inteligencijom jeste da napiše ili dizajnira **program agenta** na način da izvršava **agentsku funkciju** (mapiranje percepcije u akcije ili radnje). Program se izvršava na nekom hardverskom uređaju sa senzorima i aktuatorima, i takve hardverske komponente nazivamo **arhitekturom**.

11.4 Program agenta

Program treba da uzme jedan objekat opažanja i da vrati akciju koju izvode aktuatori. Ako agent donosi odluke na osnovu sekvence prethodnih koraka, onda mora imati memoriju.

```

function TABLE-DRIVEN-AGENT(percept) returns an action
  persistent: percepts, a sequence, initially empty
    table, a table of actions, indexed by percept sequences, initially fully specified
  append percept to the end of percepts
  action  $\leftarrow$  LOOKUP(percepts, table)
  return action

```

Slika 11.6: Pseudokod agenta

Upotreba agenta koji su bazirani na lookup tabelama je veoma ograničena jer je teško predvidjeti svaku moguću sekvencu zapažanja. Kako bismo napisali program agenta koji će koristiti skup manjih programa umjesto jedne velike tabele, možemo koristiti jedan od sljedeća četiri pristupa:

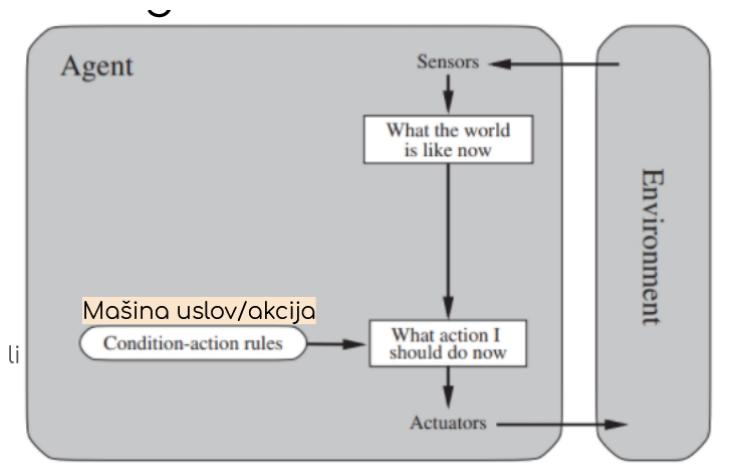
1. Jednostavnii refleksni agenti
2. Model-bazirani refleksni agenti
3. Ciljno-bazirani agenti
4. Korisno-bazirani agenti

11.4.1 Jednostavnii refleksni agenti

Jednostavnii refleksni agenti nemaju nikakvog pamćenja i rade na principu refleksa. Ova implementacija može sagledati samo jedno opažanje i na osnovu toga djelovati (pokrenuti akciju). Ignorišu se sva prethodna opažanja kako ovi agenti nemaju memoriju. Beneficije ovakvog pristupa su što imamo kraći programski kod i što nemamo tabele. Većinom koriste IF-THEN-ELSE logiku (detektuj objekat i reaguj). Kao primjer možemo uzeti autonomno vozilo:

IF auto ispred THEN koči

Dijagram je predstavljen sljedećom slikom:



Slika 11.7: Dijagram toka

Dakle, imamo agenta, senzore, aktuatore i okruženje. Pored toga, postoji i **mašina uslova i akcija**. Senzori daju informaciju o tome šta se dešava u okolinu, te se na osnovu ove informacije i pravila (RULE-MATCH(state, rules)) donosi odluka koja se prosljeđuje aktuatorima. Ovakav agent ima svoje interno stanje (INTERPRET-INPUT(percept)) koje zavisi od trenutnih informacija koje se dobijaju od senzora (npr. auto koje smanji brzinu ukoliko pada kiša).

```

function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition-action rules
  state  $\leftarrow$  INTERPRET-INPUT(percept)
  rule  $\leftarrow$  RULE-MATCH(state, rules)
  action  $\leftarrow$  rule.ACTION
  return action
  
```

Slika 11.8: Program agenta

Mašina može da se programira za različite situacije. Implementacija je veoma jednostavna, ali se javljaju ograničenja u nivou inteligencije jer je potrebno predvidjeti dosta pravila. Ovakvi agenti rade dobro jedino u okruženju koje se ne mijenja. Idealno bi bilo da agent uči iz sekvence slika (iz prethodnih koraka) na osnovu kojih generiše pravila.

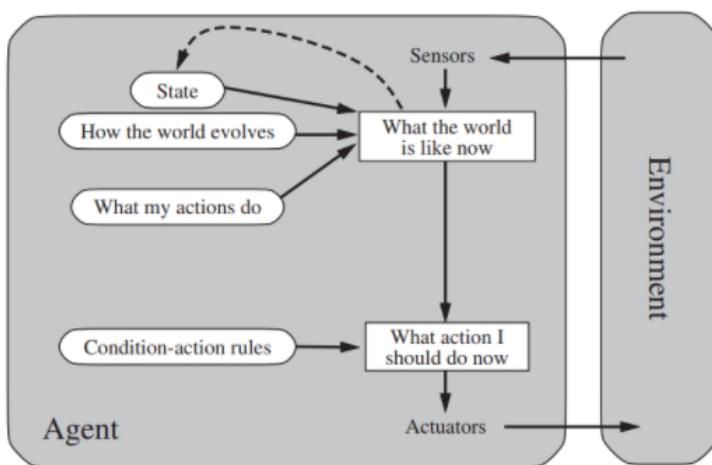
11.4.2 Model-bazirani refleksni agenti

Ukoliko je okruženje djelimično definisano onda je najbolje sačuvati historiju prethodnih dešavanja (opažanja, akcija). U ovom slučaju, agent ima definisano interno stanje.

Model predstavlja znanje kojim se opisuje kako svijet funkcioniše. Agent će biti uspješniji što je model bolji, odnosno, što više zna o nekom okruženju. Agenti koji se baziraju na ovom pristupu se nazivaju **model-bazirani agenti**.

Kao primjer možemo uzeti autonomno auto koje ima trenutnu sliku okruženja u kojem se nalazi. Ovu sliku može uporediti sa slikama koje je prethodno pohranio i na osnovu toga detektovati promjene u okolini.

Dijagram agenta je sljedeći:



Slika 11.9: Dijagram toka

U ovom slučaju, stanje se može mijenjati. Dakle, sada možemo da definišemo trenutno stanje, pratimo kako se okruženje mijenja i posmatramo šta bi se desilo ukoliko poduzmemos neku akciju. Ove informacije se koriste kako bismo promijenili informaciju o okruženju i ovo se sve dešava unutar agenta.

Ovakvo okruženje se naziva **ažurirano okruženje** i ono predstavlja kombinaciju originalnog stanja, kako se svijet može promijeniti (fizika) i kako određene akcije utiču na svijet.

Odluka se donosi na bazi pravila uslov-akcije i ažuriranog okruženja.

Pseudokod je dat u nastavku:

```

function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
    model, a description of how the next state depends on current state and action
    rules, a set of condition-action rules
    action, the most recent action, initially none

  state  $\leftarrow$  UPDATE-STATE(state, action, percept, model)
  rule  $\leftarrow$  RULE-MATCH(state, rules)
  action  $\leftarrow$  rule.ACTION
  return action

```

Slika 11.10: Program agenta

Stanje se dobija na osnovu metode UPDATE-STATE(*state*, *action*, *percept*, *model*), odnosno na osnovu trenutnog stanja, akcija, percepције и модельа. Затим се правило добија на основу аžuriranог stanja i dobijamo određenu akciju.

11.4.3 Ciljno-bazirani agenti

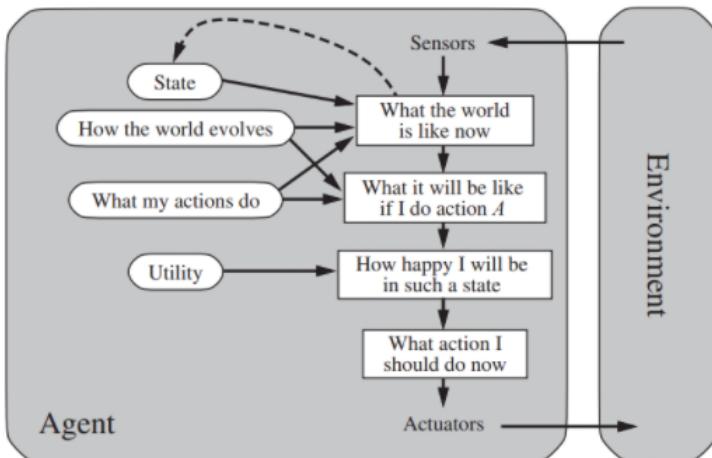
Kod određenih problema nije dovoljno poznavati само okruženje. Agenti понекад морaju имати одређени циљ (нпр. циљ autonomnog auta је да дође до одредене destinacije). Циљ се може комбиновати са окolinом, како би се дошло до што боље одлуке. **Pretraživanje** за секвенком акција које доводе до циља је посебно поле истраживања, као и **planiranje** долaska до циља (stanja система се могу представити као чворови дрвета, при чему путanja представља промјену stanja).

Agenti базирани на достизању циља су више fleksibilni, али мање efikasni jer заhtijevaju више resursa. Ovakvi agenti могу модификовати одлуке на бази претходних деšавања и понашање агента се може лако модификовати (нпр. промјеном destinacije).

11.4.4 Korisno-bazirani agenti

Ciljevi nisu dovoljni да би се дошло до неког kvalitetnog rješenja. Potrebno je uzimati u obzir korisnost određene putanje ili sekvence akcija. Iz ovog razloga се definiše **funkcija korisnosti** (engl. *utility function*) и уводи се мјера корисности која се користи за poređenje različitih puteva. Sekvena prelaza stanja се оцjenjuje како би се створила слика о tome који је put bolji. Agent се сматра racionalnim ukoliko napravi najbolju odluku u odnosu на zadati cilj или postigne **очекивану корисност**.

Predstavljen je dijagram ovakvog agenta:



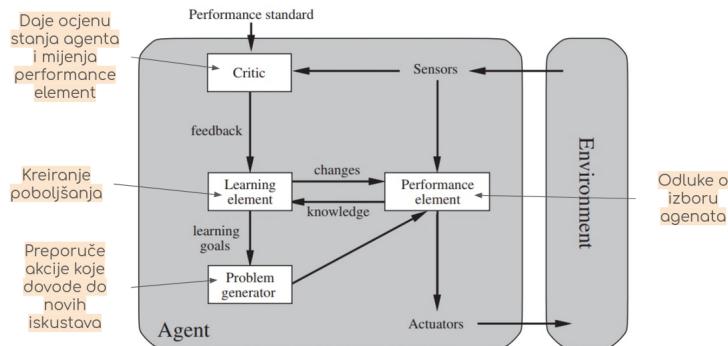
Slika 11.11: Dijagram toka

Umjesto mašine uslova i akcija, sada se javlja ocjena korisnosti. Također, postavljamo pitanje kojim provjeravamo da li smo zadovoljni sa određenom odlukom ili ne. Navedeno se može posmatrati u smislu korisnosti ili iskoristivosti koju provjeravamo putem funkcije korisnosti.

11.5 Agenti za učenje

Agente za učenje nije potrebno programirati. Ova ideja dolazi od Turinga koji je predložio kreiranje maštine koja može da uči na način da se sama programira. Mašinu je zatim moguće dalje podučavati. Na ovaj način dobijamo sistem koji kontinuirano uči i mijenja se kroz vrijeme.

Model ovakvog agenta je dat u nastavku:



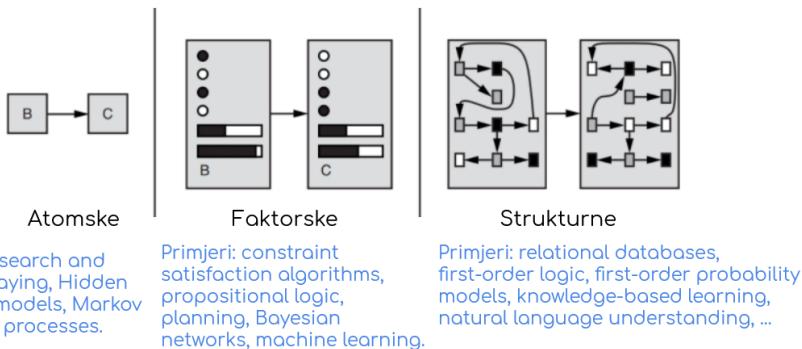
Slika 11.12: Dijagram toka

Javlja se drugi set elemenata vezan za agenta. Senzorski ulazi se šalju do kritičara (engl. *critic*) i do elementa izvedbe (engl. *performance element*). Element izvedbe treba da donese odluke o izborima akcija. Na njegove odluke utiče element učenja (engl. *learning element*). Element učenja kreira poboljšanja na osnovu određenih znanja koja dolaze od prethodnih odluka i povratne sprege kritičara. Kritičar daje ocjenu stanja agenta i može da mijenja element izvedbe. Ove performanse mogu da mijenjaju stanje agenta i da generišu neke nove akcije. Generator problema (engl. *problem generator*) je komponenta koja odgovara na pitanja "šta ako?". Preporučuje akcije koje dovode do novih iskustava i ove akcije je potrebno probati prije nego što se implementiraju.

Na ovaj način, ukinute su tabele i eksplisitno kodiranje. Agent može u potpunosti da mijenja svoje interno stanje na osnovu opisanih elemenata. Inteligentni agenti uče kroz proces modifikacije svake komponente na način da se slože kroz povratne informacije. Na ovaj način je moguće doći do bolje odluke.

Komponente agenta se mogu implementirati na različite načine:

- **atomske komponente** - svako stanje predstavljeno putem crne kutije bez interne reprezentacije (najčešće putem boolean vrijednosti zbog čega je stanje nedjeljivo)
- **faktorske komponente** - stanje je vektor parametara (boolean, realne vrijednosti, skup simbola)
- **strukturne komponente** - stanje uključuje objekte, a svaki objekat može da ima atribute kao i relacije između objekata (komponente su povezane)



Slika 11.13: Komponente agenta