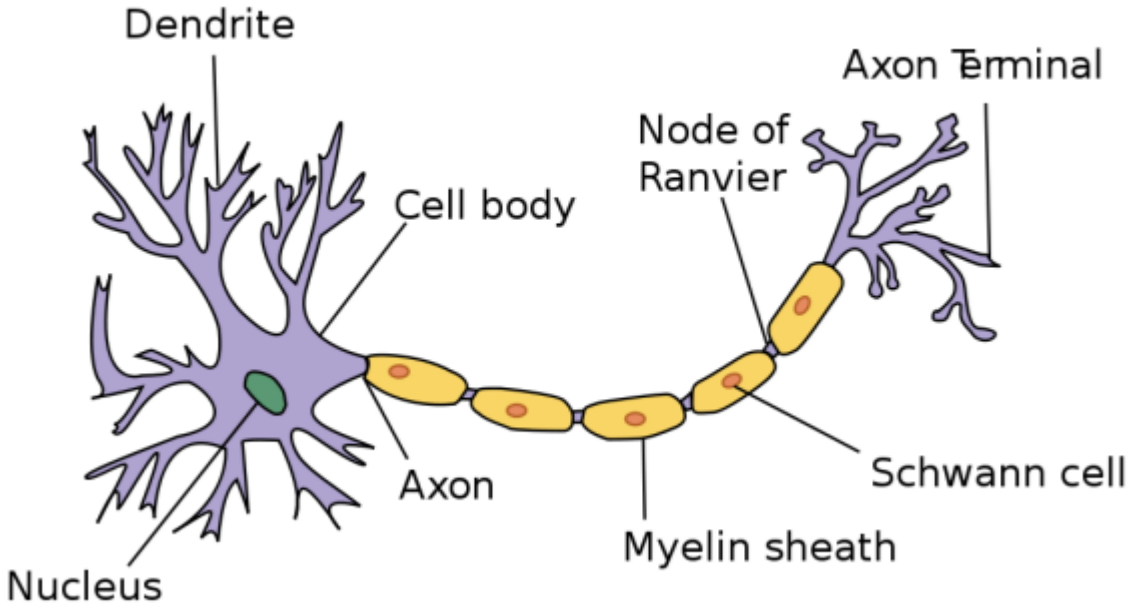
Računarska inteligencija

1. Računarska inteligencija, definicija i paradigme

Inteligencija se definiše kao mogućnost razumevanja i ostvarivanja koristi od razumevanja. Uključuje kreativnost, veštine, svesnost, emocije i intuiciju. **Računarska inteligencija** (**Computational Inteligence** - **CI**) je podgrana veštačke inteligencije (Artificial Intelligence - AI) koja izučava mehanizme inteligentnog ponašanja u složenim i promenljivim okruženjima. To su mehanizmi koji mogu da uče, da se prilagođavaju, uopštavaju i slično. Srodan termin je **Soft Computing** koji podrazumeva upotrebu višestrukih CI paradigmi i verovatnosnih metoda. **Paradigme računarske inteligencije** su:

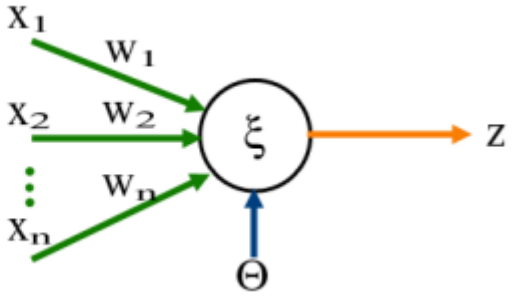
- **veštačke neuronske mreže** (**Artificial Neural Networks** - **ANN**)
- **evolutivna izračunavanja** (**Evolutionary Computation** - **EC**)
- **inteligencija grupa** (**Swarm Intelligence** - **SI**)
- **veštački imuni sistem** (**Artificial Immune System** - **AIS**)
- **rasplinuti** (**fazi**) **sistemi** (**Fuzzy Systems** - **FS**)

2. Veštačke neuronske mreže, definicija, biološki neuron, veštački neuron

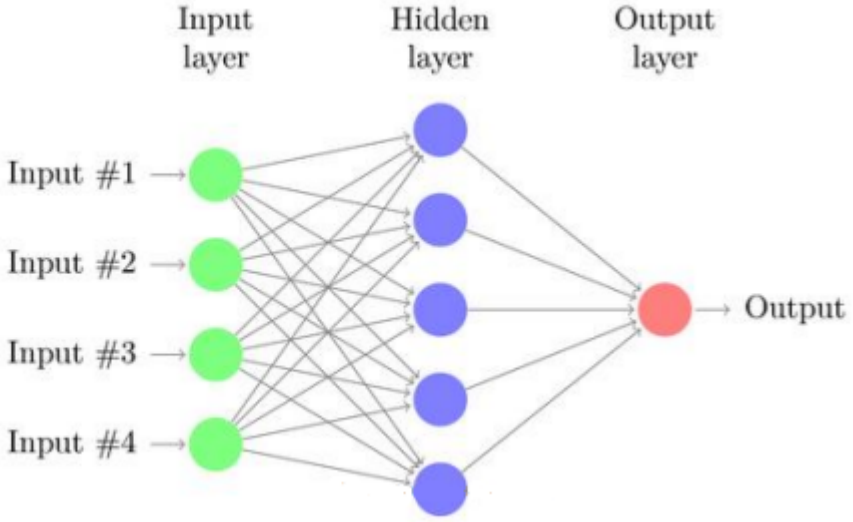


Ljudski mozak ima mogućnost prepoznavanje oblika, percepciju, motoriku i slično. U tome je mnogo efikasniji od bilo kog računara. Sadrži između 10 i 500 milijardi neurona i 50 triliona sinapsi. Pitanje je da li će ikada biti moguće modelovati ljudski mozak u potpunosti. Današanje veštačke neuronske mreže omogućavaju rešavanje pojedinačnih zadataka, dok mozak može da rešava mnogo zadataka istovremeno. **Biološki neuroni** su međusobno povezani vezama koje se otvaruju između aksona jednog neurona i dendrita drugog

neurona. Ta veza naziva se **sinapsa**. Signali se kreću od dendrita, preko tela ćelije do aksona. Slanje signala dešava se povremeno - u trenutku kada ćelija "ispali" signal. Neuron može da suzbije ili pojača jačinu signala. **Veštački neuron** (**Artificial Neuron - AN**) je model biološkog neurona. AN prima signal od okruženja ili od drugog AN, a nakon toga ga prenosi svim povezanim AN. Ispaljivanje signala i njegova jačina su kontrolisani formulom.



Veštačka neuronska mreža (**Artificial Neural Network - ANN**) je mreža slojevito poređanih AN. Obično sadrži ulazni, izlazni i nula ili više središnjih slojeva neurona.



ANN mogu biti:

- **jednoslojne**
- **višeslojne sa propagacijom unapred**
- **temporalne i rekurentne** - predviđanja iz prethodnih koraka utiču na trenutni korak
- **samoorganizujuće**
- **kombinovane**

ANN se primenjuju u medicinskoj dijagnostici (npr. prepoznavanje tumora), prepoznavanju zvuka, procesiranju slika, predviđanju u analizi vremenskih serija, kontroli robota, klasifikaciji i kompresiji podataka i tako dalje.

3. Uvod u fazi sisteme i fazi skupovi

Klasična binarna logika često nije primenjiva u rešavanju realnih problema. Realni problemi se često opisuju neprecizno i nekompletno. Na primer, rečenica "delimično je oblačno" nije binarna. Potrebno je razviti drugačiji logički sistem kako bismo mogli zaključivati na osnovu ovakvih izjava. Za te potrebe, kreće se od drugačije definicije skupova i funkcije pripadnosti skupu. Na primer, na koji način možemo definisati skup visokih ljudi? U klasičnoj teoriji

skupova, možemo postaviti neku granicu (npr. 175cm). Problem je u tome što su i osoba od 178cm i osoba od 210cm samo "visoke", kao i to što postoji oštra granica na prelazima. **Fazi skupovi** omogućavaju da se pripadnost skupu definiše sa nekom numeričkom vrednošću koja je između 0 i 1. Ako je X domen, a $x \in X$ konkretni element tog domena, onda se fazi skup A opisuje **fazi funkcijom pripadnosti** $\mu_A : X \to [0,\ 1]$. Fazi funkcija pripadnosti skupu je jednoznačna za svaki element domena. Fazi skupovi mogu biti definisani nad diskretnim ili realnim domenima.

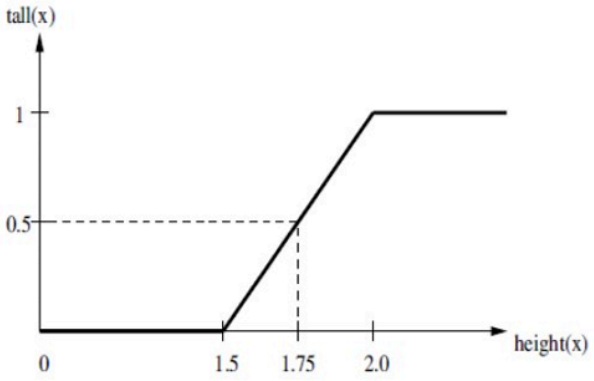
Postoje dve notacije za predstavljanje diskretnog fazi skupa:

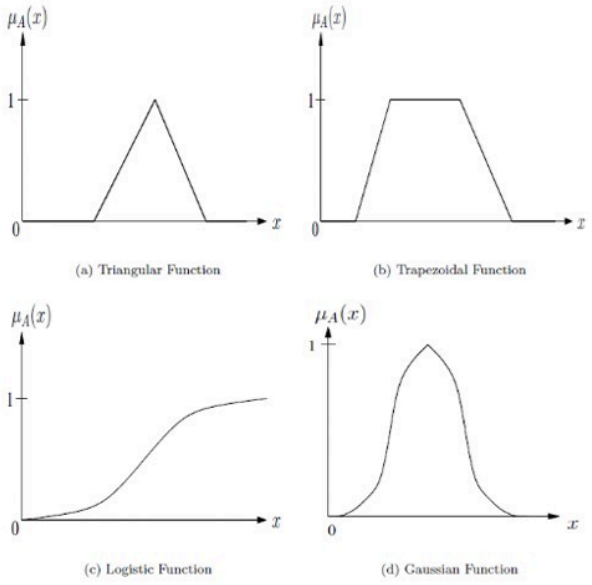
- Preko skupa uređenih parova: $A = \{(\mu_A(x_i),\ x_i \mid x_i \in X,\ i = 1,\ \ldots,\ n_X\}$
- Preko "sume": $A = \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \ \ldots\ + \frac{\mu_A(x_{n_X})}{x_{n_X}}$. Ova suma nije prava, tj. ne podrazumeva stvarno sabiranje, već je samo vid redefinisane notacije koja je u saglasnosti sa notacijom za realni domen.

Notacija za realni fazi skup je data preko "integrala": $A = \int_X \frac{\mu(x)}{x}$. Opet, ovo nije pravi integral već samo pogodna notacija.

Primer mogućeg načina definisanja skupa visokih ljudi, kao i nekih standardnih fazi funkcija pripadnosti:

$$tall(x) = \begin{cases} 0,\ height(x) < 1.5m \\ (height(x) - 1.5m) \cdot 2m,\ 1.5m \leq height(x) \leq 2m \\ 1,\ height(x) > 2m \end{cases}$$





4. Fazi skupovne operacije

- **Jednakost skupova** - fazi skupovi su jednaki ako imaju isti domen i pritom za svaki element domena imaju istu funkciju pripadnosti: $A = B$ akko $\mu_A(x) = \mu_B(x)$ za sve $x \in X$.
- **Podskupovi** - skup A je podskup skupa B akko $\mu_A(x) \leq \mu_B(x)$ za sve $x \in X$.

- **Komplement** - ako je A^C komplement skupa A, onda za sve $x \in X$ važi $\mu_A(x) = 1 - \mu_{A^C}(x)$. Ne važe identiteti kao u klasičnoj teoriji skupova da je $A^C \cap A = \emptyset$ i $A^C \cup A = X$.
- **Presek** - postoji mnogo načina, a standardni su:
  - preko minimuma: $\mu_{A \cap B}(x) = min\{\mu_A(x),\ \mu_B(x)\},\ \forall x \in X$.
  - preko proizvoda: $\mu_{A \cap B}(x) = \mu_A(x)\mu_B(x),\ \forall x \in X$. Sa proizvodom treba biti oprezniji jer već nakon nekoliko uzastopnih primena funkcije pripadnosti teže 0.
- **Unija** - postoji mnogo načina, a standardni su:
  - preko maksimuma: $\mu_{A \cup B}(x) = max\{\mu_A(x),\ \mu_B(x)\},\ \forall x \in X$.
  - preko sume i preseka: $\mu_{A \cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x),\ \forall x \in X$. Sa sumacijom i presekom treba biti oprezniji jer funkcije pripadnosti teže 1 čak iako su polazne funkcije bliske 0.

5. Karakteristike fazi skupova

- **Normalnost** - fazi skup A je normalan ako ima bar jedan element koji pripada skupu sa stepenom 1: $(\exists x \in A)\ \mu_A(x) = 1$, odnosno $sup_X \mu_A(x) = 1$
- **Visina** - supremum po funkciji pripadnosti: $height(A) = sup_X \mu_A(x)$
- **Podrška** - skup svih elemenata koji imaju pripadnost veću od 0: $support(A) = \{x \in X \mid \mu_A(x) > 0\}$
- **Jezgro** - skup svih elemenata koji pripadaju skupu sa stepenom 1: $core(A) = \{x \in X \mid \mu_A(x) = 1\}$
- α**-rez** - skup svih elemenata koji imaju pripadnost najmanje α: $A_\alpha = \{x \in X \mid \mu_A(x) > \alpha\}$
- **Unimodalnost** - fazi skup je unimodalan ako je njegova funkcija pripadnosti unimodalna.
- **Kardinalnost** - u zavisnosti od tipa domena definiše se kao:
  - $card(A) = \sum_{x \in X} \mu_A(x)$
  - $card(A) = \int_{x \in X} \mu_A(x) dx$
- **Normalizacija** - fazi skup se normalizuje tako što se funkcija pripadnosti podeli visinom fazi skupa: $normalized(A) = \frac{\mu_A(x)}{height(A)}$
- Ostala bitna svojstva - **komutativnost**, **asocijativnost**, **tranzitivnost** i **idempotencija**.

6. Fazi i verovatnoća

Postoji česta zabuna između fazi koncepta i verovatnoće. Oba termina referišu na (ne)sigurnost događaja, ali tu sve sličnosti prestaju. Statistička verovatnoća se vezuje za posmatranje događaja koji treba tek da se dese i za donošenje zaključaka o tome kolika je njihova šansa da će se desiti. Ako posmatramo uzastopna bacanja novčića i na velikom broju bacanja utvrdimo da je u 50% situacija pala glava, tada možemo govoriti o sigurnosti toga da će u narednom bacanju da padne glava. Kod fazi skupova, funkcija pripadnosti ne govori o sigurnosti da se neki događaj iz budućnosti desi. Na primer, imamo dve osobe, jedna je visoka 220cm i pripada skupu visokih ljudi sa stepenom 1, a druga je visoka 195cm i pripada skupu visokih ljudi sa stepenom 0.95. Interpretacija ne znači da će u ponovljenim

eksperimentima prva osoba biti češće veća od druge osobe. To samo znači da su obe osobe visoke ali sa različitim stepenom pripadnosti skupu visokih osoba. Dakle, fazi se vezuje za stepen istinitosti, dok se verovatnoća vezuje za mogućnost predviđanja nekog ishoda.

7. Fazi logika

Ključni elementi **fazi logike** su lingvističke promenljive i **fazi if-then pravilo zaključivanja**. **Lingvističke (fazi) promenljive** su promenljive čije su vrednosti reči prirodnog jezika (npr. visok). Tipovi lingivističkih promenljivih:

- **kvantifikatori**: sve, većina, mnogo, nijedan, itd.
- **promenljive za učestalost**: ponekad, često, uvek, itd.
- **promenljive za šansu**: moguće, verovatno, sigurno, itd.

Modifikatori su dodatne reči koje pojačavaju ili slabe efekat lingvističkim promenljivim. Najčešće su u pitanju neki pridevi, npr. veoma, malo, srednje, itd. Mogu biti dovedeni u relaciju sa originalnom lingvističkom promenljivom putem funkcije. Na primer, $\mu_{verytall}(x) = \mu_{tall}(x)^2$. Ako neko pripada skupu visokih sa sigurnošću 0.9, onda će pripadati skupu veoma visokih sa manjom sigurnošću od 0.81. Modifikatori za pojačavanje obično imaju formu $\mu_{A'}(x) = \mu_A(x)^{\frac{1}{p}}$ za $p > 1$.

8. Fazi zaključivanje

Na primer, neka važi $\mu_{tall}(Peter) = 0.9$, $\mu_{goodathlete}(Peter) = 0.8$ i $\mu_{tall}(Carl) = 0.9$, $\mu_{goodathlete}(Carl) = 0.5$. Ako se zna da je dobar košarkaš visok i atleta, koji od ove dvojice je bolji? Primenom pravila minimuma za presek dobijamo $\mu_{goodplayer}(Peter)$ = $min\{0.9,\ 0.8\}$ = 0.8 i $\mu_{goodplayer}(Carl)$ = $min\{0.9,\ 0.5\}$ = 0.5 pa zaključujemo da je Peter bolji igrač. U realnim okolnostima, zavisnosti su mnogo složenije pa govorimo o skupu **if-then pravila**. Na primer, "ako je snaga motora velika i auto je nov, onda je cena visoka" i "ako je snaga motora mala i auto je star, onda je cena niska". **Mamdanijev metod** je jedan od najpoznatijih i najčešće korišćenih metoda u fazi zaključivanju i koristi se za pretvaranje fazi if-then pravila u konkretan izlaz. Sastoji se od fazifikacije ulaza, primene fazi pravila nad dobijenim fazi skupovima i defazifikacije rezultata.

Fazifikacija je prevođenje ulaznih podataka iz ulaznog prostora u fazi reprezentaciju. Predstavlja primenu funkcije pripadnosti nad ulaznim podatkom. Na primer, ako su A i B fazi skupovi nad domenom X, proces fazifikacije prihvata elemente $a,\ b \in X$. Dodeljuju se stepeni pripadnosti svakom od fazi skupova - $\{(\mu_A(a),\ a),\ (\mu_B(a),\ a),\ (\mu_A(b),\ b),\ (\mu_B(b),\ b)\}$

Nakon fazifikacije, cilj je primeniti **pravila zaključivanja** nad fazifikovanim ulazima. Neka su A i B definisani nad domenom X_1, dok je fazi skup C definisan nad domenom X_2. Neka je pravilo "if A is a and B is b then C is c". Na osnovu fazifikacije znamo $\mu_A(a)$ i $\mu_B(b)$. Prvo je potrebno izračunati stepen pripadnosti skupu premise: $min\{\mu_A(a),\ \mu_B(b)\}$. Ovo se radi za svako od pravila zaključivanja. Neka je α_k stepen pripadnosti premisa za k-to pravilo. Sledeći korak je računanje stepena pripadnosti zaključku c_i: β_i = $max\{\alpha_{k_i}\}$, za svako pravilo

k u kojem figuriše c_i. To znači da je na izlazu iz zaključivanja stepen pripadnosti za svaki od fazi skupova zaključaka.

Defazifikacija je poslednji korak i predstavlja prevođenje fazi zaključaka u ne-fazi prostor. Podrazumeva određivanje lingvističkih promenljivih za prethodno određene stepene pripadnosti zaključcima. Pristupi zasnovani na računanju **centroide**:

$$\text{output} = \frac{\sum_{i=1}^{n} x_i \mu_C(x_i)}{\sum_{i=1}^{n} \mu_C(x_i)} \ \vee \ \text{output} = \frac{\int_{x \in X} x \mu_C(x) dx}{\int_{x \in X} \mu_C(x) dx}$$

Nakon računanja centroide, pročita se ona lingvistička promenljiva koja joj odgovara prema nekom od sledećih pravila:

- **max-min**: uzima se centroida ispod lingvističke promenljive koja odgovara zaključku sa najvišim stepenom.
- **uprosečavanje**: računa se centroida za sve lingvističke promenljive i na osnovu toga određuje konačna lingivstička promenljiva.
- **skaliranje**: funkcije pripadnosti se skaliraju prema dobijenim zaključcima i nakon toga se računa centroida.
- **isecanje**: funkcije pripadnosti se seku na mestima koja odgovaraju zaključcima i potom se računa centroida.

9. Optimizacija, definicija, izazovi, ključni pojmovi

Optimizacioni algoritmi pripadaju grupi **algoritama pretrage**. Cilj je pronaći rešenje problema takvo da je:

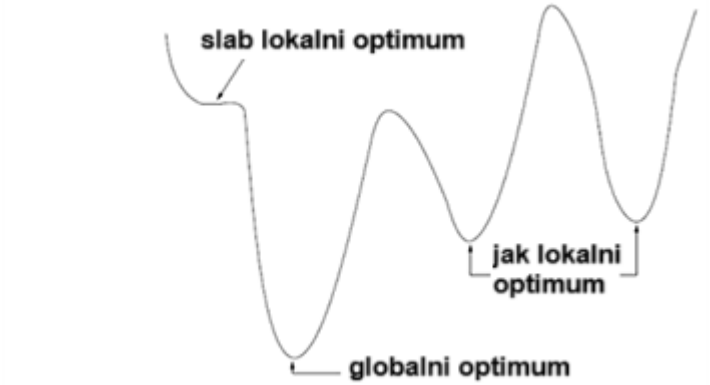
- neka ciljna funkcija (**funkcija cilja**) maksimalna/minimalna.
- neki **skup ograničenja** zadovoljen. Ovaj uslov je opcion, odnosno nekada nećemo imati ograničenja.

Optimizacija je sama po sebi teška, pogotovo kod rešavanja NP teških problema gde se veruje da su algoritmi u najboljem slučaju eksponencijalne složenosti. Još neki od izazova optimizacije su:

- Rešenje može biti predstavljeno kao kombinacija vrednosti iz različitih domena.
- Ograničenja, a i sama funkcija cilja, mogu biti nelinearna.
- Karakteristike problema mogu varirati tokom vremena.
- Funkcija cilja može biti u konfliktu sa ograničenjima.

Neka je S **domen** posmatranog problema. **Funkcija cilja** je funkcija $f : S \to R$ koju želimo da minimizujemo ili maksimizujemo. Ona govori o kvalitetu rešenja. Važi da je minimum funkcije f isto što i maksimum funkcije $-f$, pa po konvenciji uvek možemo govoriti o minimizaciji. Skup x predstavlja **nezavisne promenljive** koje utiču na vrednost funkcije f i za date vrednosti promenljivih funkcija ima vrednost $f(x)$. Skup ograničenja najčešće postavlja zavisnosti između nezavisnih promenljivih, ali može i da ograničava nezavisne

promenljive na neki interval ili skup vrednosti. Ukoliko ne postoji funkcija cilja, već samo ograničenja koja treba zadovoljiti, onda se radi o **programiranju ograničenja (Constraint programming)**, a ne o optimizaciji. Klasični primeri su problem n dama i SAT problem.



Pri optimizaciji razlikujemo nekoliko tipova optimuma:

- **globalni optimum** - najbolje rešenje na čitavom dopustivom skupu rešenja S. Moguće je da postoji više globalnih optimuma.
- **jak lokalni optimum** - najbolje rešenje u nekoj okolini $N \subseteq S$. Predstavlja najveću prepreku za optimizacione algoritme.
- **slab lokalni optimum** - jedno od najboljih rešenja u okolini $N \subseteq S$. Većina optimizacionih algoritama može lako da savlada ovaj tip lokalnog optimuma.

Postoje još neki vidovi optimizacionih problema:

- **problemi sa višestrukim optimumom** - cilj je pronaći sva rešenja koja su optimalna ili dovoljno blizu optimuma. Na primer, detekcija otiska na ekranu.
- **višeciljna optimizacija** - problem ima više funkcija cilja. Na primer, želimo da dođemo od tačke A do tačke B tako da istovremeno utrošimo i najmanje vremena i najmanje goriva.
- **dinamička optimizacija** - funkcija cilja i ograničenja mogu se menjati tokom vremena. Na primer, autopilot aviona ili automobila.

Prema tipu domena nad kojim se vrši, optimizacija se deli na:

- **kombinatornu (diskretnu)** - dopustivi skup vrednosti promenljivih je najviše prebrojiv, odnosno konačan ili prebrojivo beskonačan. Specijalni slučaj su problemi **binarne optimizacije** gde promenljive uzimaju vrednosti 0 ili 1.
- **globalnu (kontinualnu)** - dopustivi skup vrednosti promenljivih je neprebrojiv, iz skupa realnih brojeva.

Optimizacione metode traže optimum u prostoru **dopustivih rešenja** - rešenja koja ispunjavaju ograničenja problema. Metode se prema fokusu pretrage dele na:

- **lokalne** - nisu sposobne da nađu globalne optimume. Jedna velika grupa lokalnih metoda su **gradijentne metode** koje na neki način prate gradijent. Lokalne metode mogu naći i globalni optimum ako imaju sreće, odnosno ako su početni uslovi dobri.

- **globalne** - imaju mehanizme koji im omogućavaju da izbegnu lokalne optimume, na primer mehanizam razmrdavanja.

Optimizacione metode se prema pristupu pretrage dele na:

- **stohastičke** - koriste elemente nasumičnosti u procesu pretrage i ne garantuju uvek isto rešenje za iste ulazne parametre. Globalne metode su češće stohastičke jer im to omogućava bolju pretragu prostora rešenja i pronalaženje globalnog optimuma.
- **determinističke** - koriste precizno definisana pravila i algoritme za pretragu, što znači da će za iste ulazne parametre uvek dati iste izlazne rezultate. Lokalne metode su češće determinističke.

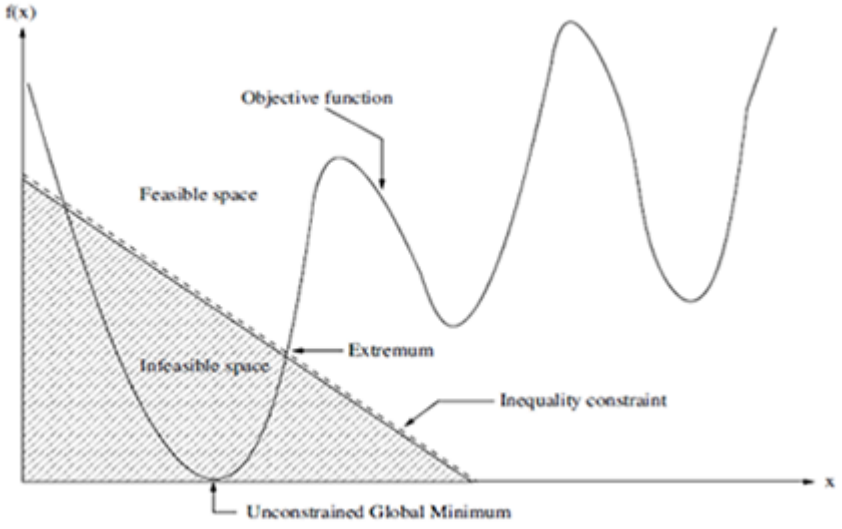
10. Optimizacija bez ograničenja, definicija, primer

Optimizacija bez ograničenja podrazumeva da se funkcija cilja minimuzuje na celom domenu S, tj. formulacija problema glasi "minimizovati $f(x)$", $x = (x_1, \ldots, x_{n_X})$, $x \in S$. Na primer:

- pronaći minimum funkcije $f(x, y) = x^2 + y^2$ na domenu realnih brojeva. Analitičkim rešavanjem nalazimo da je minimum $(0, 0)$ i da je on jedinstven, pa je gradijentni spust pogodna metoda za rešavanje ovog problema.
- pronaći minimum Rosenbrock funkcije $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$ na domenu realnih brojeva. Funkcija nema očigledno analitičko rešenje, ali na grafičkom prikazu vidimo da ona ima više lokala, pa je pogodna neka metoda koja kombinuje stohastičku i deterministčku pretragu.

11. Optimizacija sa ograničenjima, definicija, slika sa objašnjenjima ključnih pojmova, rad sa nedopustivim rešenjima

Optimizacija sa ograničenjima podrazumeva minimizaciju funkcije cilja uz domenska ograničenja promenljivih, kao i ograničenja zasnovana na jednakostima i/ili nejednakostima. Forma funkcija ograničenja u opštem slučaju nije linearna.



Nedopustiva rešenja mogu se obraditi na razne načine:

- **odbacivanje**
- **dodeljivanje penala** - negativni faktor u slučaju maksimizacije ili pozitivan faktor u slučaju minimizacije. Omogućava veću fleksibilnost jer možda je nedopustivo rešenje jako blizu optimuma pa je korisno iskoristiti ga.
- **svođenje na rešenje bez ograničenja**, a potom konvertovanje u rešenje koje poštuje ograničenja.
- **održavanje dopustivosti** dizajnom metoda - krećemo se po prostoru rešenja tako da nikada ne dolazimo u nedopustiva rešenja.
- **uređivanje nedopustivih rešenja** prema stepenu nedopustivosti
- **popravljanje nedopustivih rešenja** - na primer krećemo se do najbližeg dopustivog rešenja.

12. Kombinatorna optimizacija i optimizacioni algoritmi

Formulacija problema **kombinatorne optimizacije** je "minimizovati $f(x)$", $x = (x_1, \ldots, x_{n_X})$, $x \in S$, pri čemu je S konačan ili beskonačan i diskretan skup. Primer kombinatorne optimizacije je **problem trgovačkog putnika (Travelling Salesmen Problem - TSP)**: Neka je zadat skup C od m gradova i funkcija udaljenosti $d(c_i, c_j) \in N$ za svaki par gradova. Pronaći permutaciju $p : [1..m] \to [1..m]$ takvu da je ukupna suma udaljenosti grana koje se prolaze obilaskom minimalna. Funkcija udaljenosti može biti proizvoljna, na primer rastojanje između gradova ili vreme potrebno da se stigne iz jednog grada u drugi. Moguće je rešiti TSP na više načina:

- **Brute-force** - totalna enumeracija. Veličina dopustivog skupa je $m!$, pa pomoću Stirlingove formule uočavamo da bi rešenje bilo eksponencijalne složenosti.
- **Dinamičko programiranje (tehnika memoizacije)** - može se ostvariti malo poboljšanje, ali asimptotski rešenje ostaje eksponencijalne složenosti.
- **P-aproksimativni algoritam** - sa stepenom sigurnosti P moguće je matematički dokazati da se posmatrani problem svodi na neki drugi. Za TSP je utvrđeno da postoji 2-aproksimativni algoritam baziran na minimalnom razapinjućem stablu. To znači da je rešenje dobijeno ovim algoritmom najviše 2 puta gore od optimalnog, što i nije baš dobar faktor aproksimacije. Obično imaju asimptotski manju složenost od prva dva pristupa.

Prva dva pristupa su neadekvatna kada dimenzija problema naraste, ali prednost je što daju optimalna rešenja ako se završe. Aproksimativni pristup je efikasan (polinomijalan) i imamo garanciju da rešenje ne može da bude više od P puta lošije, ali mana je što dobijeno P nekada nije zadovoljavajuće. **Metaheuristike** su pristupi koji će nam omogućiti da vreme izvršavanja bude razumno, ali i da kvalitet rešenja bude "očekivano" blizak optimalnom. Mana u odnosu na aproksimativni pristup je što nemamo nikakve garancije koliko će rešenje zapravo biti dobro. Metaheuristike pripadaju široj grupi algoritama pretrage.

Opšta forma algoritma **lokalne pretrage** je:

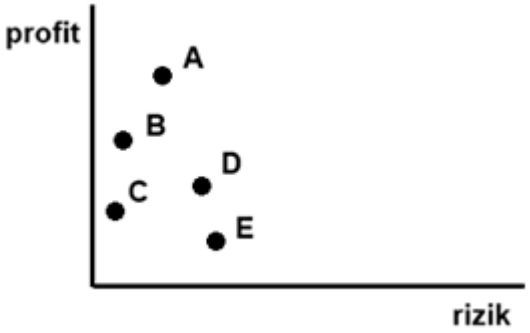
- Započni od polaznog rešenja $x(0) \in S,\ t = 0$
- Dok nije zadovoljen kriterijum završetka ponavljaj:
  - Izračunaj vrednost $f(x(t))$
  - Izračunaj pravac i smer pretrage $q(t)$
  - Izračunaj dužinu koraka pretrage $\eta(t)$
  - Pređi u naredno rešenje $x(t+1) = x(t) + \eta(t)q(t)$
  - $t = t + 1$
- Vrati $x(t)$ kao konačno rešenje

Dužina koraka pretrage predstavlja stepen poverenja u izračunati pravac i smer pretrage. On se povećava tokom vremena jer želimo da budemo sve sigurniji. **Kriterijum završetka** može biti fiksan broj iteracija, stabilizacija pomeranja i slično. **Pravac i smer pretrage** u kontinualnoj optimizaciji se može prirodno računati preko izvoda. U diskretnom slučaju mogu se koristiti metode koje će malo izmeniti rešenja i imitirati izvod. Na primer, ako rešenja predstavljamo kao niz bitova, invertovanje nasumičnog bita može predstavljati funkciju koja imitira izvod. Metaheuristike koriste principe nasumičnosti i determinizma kako bi izbegle problem lokalne pretrage, a to je zaglavljivanje u lokalnom minimumu. Dakle, metaheuristika se svodi na dva koraka: **diverzifikacija** (slučajno se pomeramo na neko rešenje, odnosno istražujemo prostor rešenja) i **intenzifikacija** (u okolini tog rešenja dolazimo do lokalnog optimuma lokalnom pretragom). Metaheuristike se dele na:

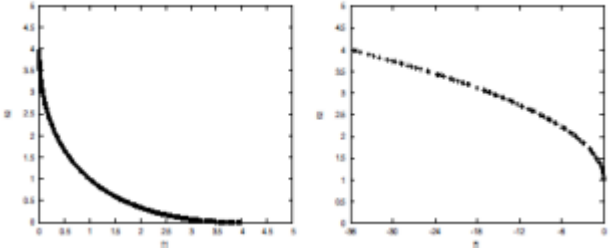
- **S metaheuristike** - rade samo sa jednim rešenjem. Primer je **simulirano kaljenje** gde u svakom koraku sa nekom verovatnoćom prihvatamo i lošije rešenje. Ta verovatnoća opada vremenom jer želimo da diverzifikacija bude mala pri kraju pretrage.
- **P metaheuristike** - rade sa čitavom populacijom. Primer su genetski algoritmi koji simuliraju evoluciju.

13. Višeciljna optimizacija

U praksi je često potrebno zadovoljiti više funkcija cilja, odnosno izvršiti **višeciljnu optimizaciju**. Na primer, potrebno je pronaći akciju sa maksimalnim prihodom i minimalnim rizikom ili je potrebno pronaći putanju sa minimalnim utroškom goriva i minimalnim utroškom vremena. Često poboljšanje jedne funkcije cilja izaziva pogoršanje druge, na primer poboljšanje strukturalne stabilnosti mosta izaziva povećanje troškova izgradnje. U tom slučaju potrebno je na neki način napraviti balans, tj. kompromis. Trivijalan način rešavanja jeste pravljenje ponderisanih proseka odnosno agregacija. Za svaku funkciju cilja određuje se težina koja predstavlja njen značaj, pa se na ovaj način problem svodi na jednociljnu optimizaciju. Ovaj pristup je efikasan, ali ne možemo uvek znati da li su težine koje smo zadali zaista dobre. Postoje i drugi načini rešavanja. Posmatrajmo primer traženja akcija sa najvećim profitom i najmanjim rizikom. Možemo prikazati zavisnost rizika i profita u svakoj tački:



Posmatrajmo tačke A i B. Tačka A ima veći profit, ali i veći rizik od tačke B. Odnosno, profit je bolji kod tačke A, a rizik je bolji kod tačke B. Sa druge strane, ako posmatramo tačke A i D, vidimo da tačka A ima i veći profit i manji rizik od tačke D. Kažemo da rešenje x **dominira** nad rešenjem y ako nijedna vrednost funkcije cilja rešenja y nije bolja od odgovarajuće vrednosti funkcije cilja rešenja x. Rešenje x je **Pareto-optimalno** ako ne postoji nijedno drugo rešenje koje dominira nad njim. Dakle, rešenja D i E su dominirana rešenja, a rešenja A, B i C su Pareto-optimalna jer međusobno nijedno ne dominira nad ostalima. Skup svih Pareto-optimalnih rešenja naziva se **Pareto-optimalni skup**. **Pareto-optimalna površ** predstavlja površ koju formiraju funkcije cilja kada se primene nad Pareto-optimalnim skupom rešenja. Problem se dalje rešava korišćenjem algoritama pretrage koji efikasno pretražuju Pareto-optimalnu površ. To su na primer populacione strategije koje iterativno poboljšavaju skup dobrih rešenja upotrebom svojstva dominacije. Primer Pareto-optimalnih površi:



14. Klase složenosti izračunavanja i rešavanje NP teških problema

Algoritam za rešavanje nekog problema je konačan spisak pravila čijim praćenjem dolazimo do rešenja bilo kojeg partikularnog problema iz zadate klase. Praćenje pravila traje konačno mnogo koraka. **Problemi odlučivanja** su problemi čije se rešenje sastoji u potvrđivanju ili opovrgavanju neke osobine. Različiti optimizacioni problemi mogu se svesti na problem odlučivanja, Na primer, možemo se pitati da li za konkretan TSP problem postoji rešenje sa troškom manjim od C, a onda na osnovu odgovora možemo menjati granicu i tako iterativno tražiti najbolje rešenje. Problem pripada **klasi polinomsko rešivih problema (klasi P)** ako postoji algoritam A za rešavanje tog problema i polinom $p(n)$ takav da A završava izvršavanje za ne više od $p(n)$ koraka za svaku instancu tog problema, pri čemu je n dimenzija problema. Polinomijalne algoritme smatramo "dobrim" algoritmima. Problem odlučivanja pripada klasi P. Algoritme čije vreme ne možemo da ograničimo polinomom podrazumevano ograničavamo eksponencijalnom funkcijom $c^n,\ c > 1$. Problem pripada **klasi eksponencijalno rešivih problema (klasi EXPTIME)** ako je za njega dokazano da ne može biti rešen algoritmom bržim od eksponencijalnog. To su na primer problemi evaluacije poteza u uopštenom šahu i igri GO ili problem pronalaženja skupa svih razapinjućih stabala

u potpunom grafu, gde je dokazano da je broj razapinjućih stabala n^{n-2}. Problem pripada **klasi nedeterminističih polinomskih problema (klasi NP)** ako se za njega ne zna da li postoji polinomski algoritam za njegovo rešavanje, ali se za neko konkretno ponuđeno rešenje problema odlučivanja može utvrditi da li je odgovor potvrdan u polinomijalnom broju koraka. Za TSP se za bilo koju datu permutaciju gradova i dati trošak može dati potvrdan ili odričan odgovor u polinomskom vremenu.

Neka su data dva problema odlučivanja A_1 i A_2. Pretpostavimo da se za A_1 može konstruisati polinomski algoritam u kojem se kao jedan od koraka pojavljuje algoritam za rešavanje problema A_2. Ako je pritom algoritam za rešavanje problema A_2 polinomski onda i za A_1 postoji polinomski algoritam. Kažemo da se problem A_1 **redukuje** na problem A_2 ako se za svaki specijalan slučaj X problema A_1 može u polinomskom vremenu pronaći specijalan slučaj Y problema A_2, takav da je za problem X odgovor potvrdan akko je i za Y odgovor potvrdan. Jasno je da je problem A_2 teži. Problem je **NP potpun (kompletan)** ako za svođenje bilo kog NP problema na posmatrani problem postoji polinomski algoritam. To znači da ako se za bilo koji NP potpun problem pronađe polinomski algoritam time se dokazuje postojanje polinomskog algoritma za svaki NP problem. Odnosno, pokazalo bi se da važi $P = NP$. U kontekstu optimizacije spominju se **NP teški problemi** - problemi čije su odlučive varijante NP potpuni problemi. Da bi se za neki novi problem pokazalo da je NP težak (kompletan) dovoljno je redukovati neki od postojećih NP teških (kompletnih) problema na njega, jer to znači da je novi problem težak bar koliko i taj koji je redukovan na njega. Postoje dva opšta pristupa za rešavanje problema. **Egzaktno rešavanje** podrazumeva postupke koji dovode do garantovano optimalnog rešenja ako se završe. Polinomski problemi se rešavaju egzaktno. **Približno (aproksimativno) rešavanje** podrazumeva postupke koji čak i kada završe ne garantuju optimalnost. Ovde se ubrajaju i metaheuristike i algoritmi sa garancijom kvaliteta. NP teški problemi se rešavaju približno.

15. Evolutivna izračunavanja - opšti koncepti

Evolucija se može smatrati optimizacionim procesom sa ciljem poboljšanja prilagođenosti organizma (ili sistema) dinamičnom i takmičarski nastrojenom okruženju. Domeni: hemijski, kosmički, biološki. **Prirodna selekcija** podrazumeva da se svaka jedinka takmiči sa ostalima u cilju preživljavanja. Najbolje jedinke imaju veću šansu da prežive i ostave potomstvo. Zbog ovoga češće prenose svoje gene, tj. karakteristike. Tokom vremena, ovakve pogodne karakteristike postaju dominantne u populaciji. Tokom stvaranja potomaka ulogu igraju i slučajni događaji: kroz **ukrštanje** (bira se gen oca ili majke) ili kroz procese **mutacije** (nasumične izmene zbog spoljnih događaja). Neki slučajni događaji mogu dodatno da unaprede organizam. **Evolutivna izračunavanja** imitiraju proces evolucije kroz prirodnu selekciju, ukrštanje, mutaciju itd. Umesto organizama i njihove borbe za preživljavanjem, jedinke u populacijama kodiraju rešenja nekog problema. Nakon nekog vremena, rešenja evoluiraju u smeru poboljšanja. Evolutivni algoritmi traže optimalna rešenja putem stohastičke pretrage nad prostorom rešenja. Jedinke (hromozomi) predstavljaju pojedinačne tačke u prostoru rešenja. Ključni aspekti EC su:

1. **Kodiranje rešenja** u vidu hromozoma, npr. niz celih brojeva
2. **Fitnes funkcija** - ocena kvaliteta jedinke
3. **Inicijalizacija** početnog skupa jedinki (početnih rešenja)
4. **Operatori selekcije** - kako biramo one koji se reprodukuju
5. **Operatori ukrštanja** - kako se vrši stvaranje novih jedinki

Opšti oblik **evolutivnog algoritma (EA)**:

- inicijalizuj broj generacija na t = 0
- kreiraj i inicijalizuj n_X-dimenzionalnu populaciju $C(0)$ od n_S jedinki
- dok nisu zadovoljeni uslovi za završetak:
  - izračunaj fitnes $f(x_i(t))$ za svaku jedinku $x_i(t)$
  - izvrši ukrštanje i formiraj potomke
  - odaberi novu populaciju $C(t+1)$
  - pređi u narednu generaciju, $t = t + 1$

Vrste evolutivnih algoritama:

- **Genetski algoritmi** - evolucija nad linearnim genotipom, nizom.
- **Genetsko programiranje** - evolucija nad stabloidnim genotipom.
- **Evolutivno programiranje** - evolucija fenotipa tj. ponašanja. Kodiranje sekvencama ponašanja, a ne nizovima. Nema ukrštanja. Fitnes je relativan u odnosu na druge jedinke.
- **Evolutivne strategije** - evolucija evolucije, tj. evolucija genotipa + evolucija paramatera evolucije genotipa.
- **Diferencijalna evolucija** - kao standardni EA samo se mutacija bira iz unapred nepoznate slučajne raspodele prilagođene populaciji. Biraju se vektori pomeraja koji su relativni u odnosu na ostale jedinke populacije.
- **Kulturna evolucija** - evolucija kulture u populacijama. Jedinke prihvataju verovanja iz populacije, ali i utiču na populaciju srazmerno svojoj prilagođenosti.
- **Koevolucija** - evolucija i preživljavanje jedinki kroz saradnju i takmičenje, npr. simbioza biljaka i insekata.

16. Kodiranje rešenja evolutivnog algoritma, fitnes funkcija i inicijalna populacija

Hromozomi su sačinjeni od molekula DNK, odnosno od velikog broja gena. **Gen** je jedinica nasleđivanja. Određuje anatomiju i fiziologiju organizma jer kodira i kontroliše proces izgradnje proteina. Jedinka je sačinjena od sekvence gena. Vrednost (sadržaj) gena se zove **genski alel**. U kontekstu EA hromozom predstavlja rešenje problema dok su pojedinačni geni karakteristike rešenja. Odabir pogodnog kodiranja je ključno za rešavanje problema. Kodiranje je najčešće zasnovano na nizu vrednosti nekog tipa, osim u slučaju genetskog programiranja gde je kod nelinearan (stablo). Klasičan primer reprezentacije: binarni vektor

fiksne dužine. Kodiranje može biti zasnovano i na nizu celih brojeva fiksne duzine. Ponekad se domen hromozoma i domen rešenja ne poklapaju. Na primer, možemo koristiti niz realnih vrednosti za hromozom, a da rešenje bude binarne prirode. Primer: p-median problem definisan nad ravni. Potrebno je odrediti p tačaka, tako da je ukupna udaljenost od zadatih tačaka u ravni do najbliže od p tačaka najmanja. Rešenje predstavljamo podskupom od p odabranih tačaka.

Prema Darvinovom modelu evolucije, jedinke sa najboljim karakteristikama imaju šanse da prežive i ostave potomstvo. Kvantifikacija ovih karakteristika se izražava putem **fitnes funkcije**. Fitnes funkcija se primenjuje nad jedinkom. Ova vrednost je obično apsolutna mera kvaliteta jedinke. Međutim, nekada može biti i relativna u odnosu na druge jedinke. Fitnes funkcija je obično, ali ne i nužno, jednaka funkciji cilja. U prethodnom primeru, za p-median, fitnes funkciju definišemo na isti način kao i funkciju cilja, a to je ukupna udaljenost svih tačaka do najbliže odabrane tačke. Za validaciju kvaliteta predloženog algoritma dobro je imati uporedni algoritam. Idealno je ako je taj algoritam egzaktan. U slučaju NP teških problema dimenzija koju rešavamo uporednim algoritmom je očekivano mala. Drugi način je poređenje sa već postojećim rezultatima iz literature. Često uporedni algoritmi koriste istu funkciju cilja i kodiranje.

Standardni pristup kod **inicijalizacije rešenja** podrazumeva da inicijalnu populaciju formiramo od nasumično odabranih dopustivih rešenja. U slučaju nedopustivih rešenja verovatno će biti potrebna popravka zbog boljeg pokrivanja celog skupa dopustivih rešenja. Dovoljno velik slučajan uzorak ima dobru reprezentativnost. U slučaju da neki deo prostora rešenja nije pokriven u početku, velika je šansa da se kasnije uopšte neće obići. Veličina se obično određuje empirijski za konkretnu metodu. Velika populacija omogućava veću pokrivenost i povećava šansu za nalaženjem globalnog optimuma (diverzifikacija). Mala populacija je efikasnija i omogućava brzu konvergenciju ka lokalnom optimumu (intenzifikacija).

17. Operator selekcije kod evolutivnog algoritma i elitizam

Selekcija predstavlja odabir rešenja koja treba da ostave potomstvo. Načelna ideja je davanje veće šanse boljim rešenjima. **Selekcioni pritisak (selection pressure)** je vreme potrebno da se proizvede uniformna populacija jedinki, odnosno da najbolje jedinke ostave svoje gene svuda. Operatori selekcije sa visokim selekcionim pritiskom smanjuju raznovrsnost gena u populaciji brže, što se naziva **preuranjena konvergencija**. Vrste selekcije:

- **Slučajna** - svaka jedinka ima istu šansu. Najniži selekcioni pritisak, ali spora konvergencija.
- **Proporcionalna** - daje veću šansu boljim jedinkama. **Ruletska** selekcija je standarni način implementacije ovog mehanizma.
- **Turnirska** - turnir između slučajnog podskupa jedinki. Ako je podskup jednak populaciji strategija je elitistička. Ako je podskup veličine jedan strategija je slučajna. Variranjem veličine podskupa menja se selekcioni pritisak.

- **Rangovska** - umesto vrednosti fitnes funkcije koristi se samo redni broj u uređenju populacije. Smanjuje selekcioni pritisak, jer se jako dobrim rešenjima relativizuje značaj.

Elitizam je tehnika koja sprečava gubljenje dobrih jedinki. Iako se najbolje jedinke najverovatnije nalaze u skupu odabranih roditelja, primenom ukrštanja i mutacije može doći do toga da potomci imaju lošiji kvalitet. Vrši se tako što biramo najbolju ili nekoliko najboljih jedinki i direktno ih prekopiramo u narednu generaciju. Broj elitistički odabranih jedinki ne sme biti preveliki, jer podrazumeva visok selekcioni pritisak.

18. Operator ukrštanja, mutacije i kriterijum zaustavljanja - ukratko

Ukrštanje je proces kojim se kreiraju nove jedinke - potomci. Podrazumeva upotrebu sledećih operatora:

- **Operatori ukrštanja** - rekombinacija gena
- **Operatori slučajne mutacije** - promene nasumičnih gena

Kriterijumi zaustavljanja:

1. Nakon isteka unapred fiksiranog broja generacija
2. Nakon isteka unapred fiksiranog vremena
3. Kada nema unapređenja u p poslednjih generacija
4. Kada u p poslednjih generacija nema promene u genotipu
5. Kada je nađeno prihvatljivo resenje (samo ako znamo šta nam je prihvatljivo)
6. Kada se nagib fitnes funkcije više ne povećava (potrebno je pratiti kretanje fitnes funkcije kroz vreme)

19. Genetski algoritmi - uvodni koncepti, kanonski genetski algoritam

Genetski algoritam (GA) razvijen je u Americi 1970-ih. Ključni autori su Holland, DeJong i Goldberg. Glavne primene nalazi kod problema u diskretnom domenu. Nije preterano brz kao i većina populacionih metaheuristika. Predstavlja dobru heuristiku za rešavanje kombinatornih problema. Ima dosta varijanti (npr. različiti mehanizmi ukrštanja, mutacija, itd). **Jednostavni (kanonski) genetski algoritam (SGA)** je originalni GA koji je razvio Holland. Za **reprezentaciju** jedinki koristi se niz bitova. Drugi GA se razlikuju u reprezentacijama (kodiranjima i dekodiranjima), mutacijama, ukrštanju, selekciji. Algoritam:

- inicijalizuj populaciju
- evaluiraj populaciju
- dok nije zadovoljen uslov za završetak:
  - odaberi roditelje za ukrštanje
  - izvrši ukrštanje i mutaciju

- evaluiraj populaciju

Ukrštanje kod SGA odvija se na sledeći način:

1. Biramo roditelje u skup za ukrštanje. Veličina tog skupa jednaka je veličini populacije. Ideja kod **selekcije** jeste da bolje jedinke imaju veću šansu, tako što su šanse srazmerne fitnesu. Implementira se kao ruletski točak, tako što se svakoj jedinki dodeli isečak točka i okreće se n puta za odabir n jedinki.
2. Razbacamo (shuffle) skup za ukrštanje.
3. Za svaki par uzastopnih hromozoma primenjujemo ukrštanje sa verovatnoćom p_c (obično 0.6 do 0.9), a inače kopiramo roditelje. Koristi se **SGA operator ukrštanja sa jednom tačkom**, odnosno bira se slučajna pozicija (manja od broja gena) i svaki od roditelja se razdvoji po toj poziciji na dva dela. Deca se kreiraju razmenom delova između roditelja.
4. Za svako dete primenjujemo mutaciju sa verovatnoćom p_m po svakom bitu nezavisno. Vrednost p_m naziva se **stopa mutacije** i obično ima vrednost između $\frac{1}{veličina\ populacije}$ i $\frac{1}{dužina\ hromozoma}$
5. Menjamo celu populaciju novodobijenom populacijom dece.

SGA je i dalje relevantan metod za poređenje sa drugim GA. Neka od ograničenja su previše restriktivna reprezentacija, mutacija i ukrštanje koji su primenljivi samo na bitovsku ili celobrojnu reprezentaciju i osetljivost selekcije kada su fitnes vrednosti bliske.

20. Ostali tipovi reprezentacija kod genetskih algoritama i mutacije nad njima

Grejovo kodiranje celih brojeva (binarni hromozomi) - ponekad je pogodnije, jer malim promenama u genotipu se prave i male promene u fenotipu (za razliku od standardnog binarnog koda). Glatkije genotip-fenotip preslikavanje može da poboljša rad GA. Danas je opšte prihvaćeno da je bolje kodirati numeričke vrednosti direktno kao cele brojeve ili realne brojeve u fiksnom zarezu. Ovo zahteva da i operatori ne budu dizajnirani da rade sa binarnim brojevima, već sa odgovarajućim tipom. **Direktna celobrojna reprezentacija** - neki problemi prirodno imaju celobrojnu reprezentaciju rešenja, npr. vrednosti parametara u procesiranju slika. Neki drugi mogu imati kategoričke vrednosti iz fiksiranog skupa, npr. plavo, zeleno, žuto. n**-poziciono** ili **ravnomerno ukrštanje** radi i u ovim situacijama. Binarna mutacija se mora proširiti, ne može biti samo invertovanje bitova. Potrebno je omogućiti mutiranje u bliske (slične) vrednosti, kao i mutiranje u nasumične vrednosti.

Mutacija za direktno realno kodiranje: Opšta šema za brojeve u fiksnom zarezu je

$$x =< x_1,\ \ldots,\ x_l > \ \rightarrow\ x' =< x'_1,\ \ldots,\ x'_l >,\ x_i,\ x'_i \in [LB_i,\ UB_i]$$

Ravnomerna mutacija podrazumeva da se x'_i bira ravnomerno iz $[LB_i,\ UB_i]$. Analogna je izvrtanju bitova (binarni kod) ili nasumičnom mutiranju (kod celih brojeva). **Neravnomerne mutacije** su mutacije čija se verovatnoća menja sa vremenom, pozicijom, itd. Standardni pristup je dodeljivanje slučajne devijacije svakoj promenljivoj, a potom izvlačenje

promenljivih iz $N(0,\ \sigma)$. Standardna devijacija kontroliše udeo promena ($\frac{2}{3}$ devijacija će se nalaziti u opsegu $(-\sigma, \sigma)$.

Postoje mnogi problemi koji za rešenje imaju uređenu strukturu: linearnu, kvadratnu, hijerarhijsku, itd. Zadatak je organizovati objekte u odgovarajućem redosledu (npr. problem sortiranja ili problem trgovačkog putnika). Ovakvi problemi se generalno izražavaju posredstvom permutacija: ako postoji n promenljivih, onda je reprezentacija sačinjena od n celih brojeva, takvih da se svaki pojavljuje tačno jednom. Primer **TSP problem**: Neka je dato n gradova. Pronaći rutu sa minimalnom dužinom. Kodiranje: Označimo gradove sa 1, 2,..., n. Jedna kompletna ruta je jedna permutacija. Prostor pretrage je velik, npr. za 30 gradova imamo 30! mogućih ruta. Normalni operatori mutacije nad permutacijama dovode do nedopustivih rešenja. Na primer, operator koji gen i sa vrednošću j menja u neku vrednost k bi mogao da dovede do toga da se vrednost k pojavljuje više puta u rešenju, dok vrednosti j više nema uopšte u resenju. Zbog toga se moraju menjati vrednosti bar dvema promenljivama. Verovatnoća mutacije sada opisuje verovatnoću primene operatora nad celim rešenjem, a ne nad pojedinačnim pozicijama. Vrste mutacija:

- **Mutacija zasnovana na umetanju** - izaberu se dve vrednosti (dva alela) na slučajan nacin. Drugi se umeće tako da bude posle prvog, pri čemu se svi ostali pomeraju ukoliko je potrebno. Ovo zadržava veći deo uređenja odnosno informacije o prethodnom susedstvu. To je dobro, jer mutacija ne treba da izaziva dramatične promene.
- **Mutacija zasnovana na zameni** - izaberu se dve vrednosti na slučajan način i zamene se. Zadržava većinu uređenja.
- **Mutacija zasnovana na inverziji** - izaberu se dve vrednosti na slučajan način, a potom se obrne redosled vrednosti između njih. Ovo je malo intenzivnija promena uređenja od prethodna dva pristupa.
- **Mutacija zasnovana na mešanju** - izabere se podskup pozicija na slučajan način. Slučajno se reorganizuju vrednosti na tim pozicijama.

21. Ostali operatori ukrštanja kod genetskih algoritama

Kvalitet **jednopozicionog ukrštanja** zavisi od redosleda promenljivih u reprezentaciji rešenja. Veća je šansa da će geni koji su blizu biti zadržani u potomstvu. Takođe, geni koji su na različitim krajevima hromozoma se ne mogu naći u istom potomku. Ovo se zove **poziciona pristrasnost**. Može biti korisna ukoliko znamo strukturu problema, međutim, u opštem slučaju je nepoželjna. Kod n**-pozicionog ukrštanja** bira se n slučajnih pozicija i razdvajanje se vrši po tim pozicijama. Spajaju se alternirajući delovi. Ovo je uopštenje jednopozicionog ukrštanja u kojem i dalje postoji poziciona pristrasnost. **Ravnomerno ukrštanje** prvo dodeljuje glavu jednom roditelju, a pismo drugom. Baca se novčić za svaki gen prvog deteta i uzima gen iz odgovarajućeg roditelja. Drugo dete je inverz prvog. Nasleđivanje je stoga nezavisno od pozicije.

Postavlja se pitanje da li koristiti mutaciju ili ukrštanje. Odgovor zavisi od konkretnog problema, ali najbolje je da postoje oba pošto imaju različite uloge. Samo mutacijski EA su mogući, dok samo ukrštajući EA ne bi radili. Ukrštanje radi **eksploataciju**, tj. optimizaciju u

okviru postojećih oblasti, tako što pravi kombinacije između roditeljskih hromozoma. Ako neki alel potreban za globalni optimum ne postoji, onda globalno rešenje nikad neće biti dostignuto. Mutacija dominantno radi **eksploraciju**, tj. otkrivanje novih oblasti u prostoru pretrage, pošto uvodi novu informaciju i time proširuje prostor pretrage. Mutacija vrši i eksploataciju, jer menja lokalnu okolinu trenutnog rešenja. Samo ukrštanje može da kombinuje informacije dva roditelja. Samo mutacija može da uvede nove informacije. Ukrštanje ne menja frekvenciju genskih alela u okviru populacije. Da bi se pogodio optimum, obično je potrebna srećna mutacija.

Ukrštanje za direktno realno kodiranje: Kod diskretnog domena (binarni ili celobrojni) svaki alel deteta z je direktno nasleđen od nekog od roditelja (x, y) sa jednakom verovatnoćom: z_i = $x_i \vee y_i$. U realnom kodiranju nema smisla koristiti n-pozicionо ili ravnomerno ukrštanje, već se koristi **aritmetičko ukrštanje** - formiranje dece koja su između roditelja: $z_i = \alpha x_i + (1 - \alpha)y_i$, $0 \leq \alpha \leq 1$. Parametar može biti konstanta (**ravnomerno aritmetičko ukrštanje**), promenljiva (npr. zavisi od starosti populacije) ili odabran slučajno svaki put. Neki tipovi aritmetičkog ukrštanja:

- **Jednostruko aritmetičko ukrštanje** - samo jedan slučajno odabran gen se aritmetički ukršta, a ostali se kopiraju od jednog roditelja ili kombinuju nasumično.
- **Jednostavno aritmetičko ukrštanje** - slučajno se bira jedan gen i svi geni pre ili posle njega se aritmetički ukrštaju.
- **Celovito aritmetičko ukrštanje** - svi geni se aritmetički ukrštaju.

U **permutacijskim problemima** normalni operatori ukrštanja dovode do nedopustivih rešenja. Nije validno da na primer ukrstimo dve permutacije $[123-45]$ i $[543-21]$ i dobijemo $[12321]$ i $[54345]$. Potrebni su operatori ukrštanja koji se fokusiraju na kombinovanje poretka i susedstva, kao na primer:

- **Ukrštanje prvog reda (OX)**: Biramo segment hromozoma prvog roditelja i taj segment kopiramo u prvo dete. Preostale vrednosti kopiramo tako da kopiranje počinje desno od kopiranog segmenta, pri čemu se koristi redosled dat drugim roditeljem. Slično se radi i za drugo dete , ali obrnemo roditelje.
- **Delimično ukrštanje (PMX)**: Biramo slučajni segment i kopiramo ga od roditelja P_1. Počev od pozicije početka segmenta tražimo elemente u tom segmentu za roditelja P_2 koji nisu bili kopirani i za svaki od ovih elemenata i pronalazimo vrednost j iz P_1 koja je na njegovom mestu u P_1. Ostale pozicije redom popunjavamo iz P_2 tako što prvo proverimo da li se vrednost već nalazi u detetu. Ako ne, postavljamo tu vrednost i idemo dalje, a inače na to mesto postavljamo vrednost iz P_1 koju smo mapirali u prethodnom koraku. Drugo dete se kreira analogno.

22. Populacioni modeli i selekcija

GA se razlikuju od bioloških modela po tome što su veličine populacija fiksne. Ovo nam omogućava da se proces selekcije opisuje pomoću dve stvari: selekcija roditelja i strategija

zamene koja odlučuje da li će potomci zameniti roditelje, kao i koje će roditelje zameniti. Dve klase GA se razlikuju u zavisnosti od odabrane strategije:

1. **Generacijski genetski algoritmi (GGA)** - svaka jedinka preživi tačno jednu generaciju. Ceo skup roditelja je zamenjen svojim potomcima nakon što su svi potomci stvoreni i mutirani. Kao rezultat ovoga nema preklapanja između stare i nove populacije (pretpostavljamo da se ne koristi elitizam). SGA koristi ovaj model.
2. **Genetski algoritmi sa stabilnim stanjem (Steady State GA, SSGA)** - jedno dete se generiše po generaciji i jedan član populacije biva zamenjen njime. Čim se potomak stvori i mutira donosi se odluka o tome da li će roditelj ili potomak preživeti i dospeti u sledeću generaciju. Postoji preklapanje između stare i nove generacije.

Generacijski jaz je količina preklapanja između trenutne i nove generacije. Za GGA je generacijski jaz 1, dok je za SSGA $\frac{1}{N}$, gde je N veličina populacije. Selekcija se može javiti u dva navrata - selekcija roditelja za ukrštanje i selekcija preživelih, tj. biranje iz skupa roditelja i dece onih koji će preći u narednu generaciju. Razlike među selekcijama prave se na osnovu operatora (definišu različite verovatnoće) i algoritma (definišu kako su verovatnoće implementirane). Neke vrste selekcija su:

- **Fitnes srazmerna selekcija** - problem nastaje jer jedna visoko kvalitetna jedinka može brzo da preuzme čitavu populaciju ako su ostale jedinke značajno lošije, tj. dolazi do rane konvergencije. Kada su fitnesi slični (pred kraj), selekcioni pritisak je loš i smanjuje se i favorizacija. Skaliranje može da pomogne u rešavanju prethodna dva problema. Skaliranje prema najgorem: $f'(i) = f(i) - \beta^t$, gde je β najgori fitnes u poslednjih n generacija.
- **Rang-bazirana selekcija** - pokušaj da se prevaziđu problemi fitnes srazmerne selekcije. Vrednost fitnesa nema apsolutni već relativni značaj. Najbolja jedinka ima najviši rang, a najgora rang 1. Trošak primene sortiranja je obično zanemarljiv.
- **Turnirska selekcija** - prethodne metode se oslanjaju na opšte populacione statistike. Ovo može biti usko grlo, npr. na paralelnim mašinama. Oslanjaju se na prisustvo eksternih fitnes funkcija koje možda ne postoje uvek (npr. evolucija botova za igrice - ovde ne znamo koji je fitnes, ali možemo da utvrdimo ko bolje igra). Opšta šema je sledeća: odabrati k članova na slučajan način, a potom odabrati najboljeg od njih. Nastaviti proces za odabir još jedinki. Verovatnoća odabira jedinke i zavisi od njenog ranga, vrednosti k (veće k znači veći selekcioni pritisak), kao i od toga da li se takmičari biraju sa vraćanjem (odabir bez vraćanja pojačava selekcioni pritisak).

Selekcija preživelih koristi metode slične onima koje se koriste za odabir roditelja. Postoje dve grupe pristupa:

- **Selekcija zasnovana na starosti** - kao kod SGA. SSGA može da implementira brisanje slučajne (potomak menja slučajno izabranu jedinku iz tekuće populacije, ne preporučuje se) ili brisanje najstarije (FIFO, potomak menja najstariju jedinku iz populacije, često se dešava da to bude jedna od najboljih jedinki).

- **Fitnes srazmerna selekcija** - primena neke od ranije pomenutih metoda. Specijalan slučaj je **elitizam**. Često se koristi i kod GGA i kod SSGA. Uvek se zadržava kopija najboljeg rešenja do sada.

23. Teorema o shemama

Teorema o shemama je teorijska osnova iza genetskih algoritama i genetskog programiranja koju je postavio Holland. Predstavlja nejednakost koja objašnjava evolutivnu dinamiku. Neformalno glasi: "Kratke sheme sa natprosečnim fitnesom postaju eksponencijalno učestalije tokom generacija." **Shema** je šablon koji identifikuje podskup niski koje su slične na pojedinačnim pozicijama. Na primer, za binarne niske dužine 6, primer sheme je $1 * 10 * 1$ gde ovakva shema opisuje sve niske dužine 6 sa fiksiranim bitovima na 4 opisane pozicije. **Red sheme** $o(H)$ je definisan kao broj fiksiranih pozicija. Vrednost $\delta(H)$ je **dužina sheme** - udaljenost između prve i poslednje fiksirane pozicije. **Fitnes sheme** $f(H)$ je prosečni fitnes svih niski koje pripadaju shemi. Formalno:

$$E(m(H,\ t+1)) \geq \frac{m(H,\ t)f(H)}{a_t}[1-p],$$

gde je $m(H,\ t)$ broj niski koje pripadaju shemi H u generaciji t, a_t prosečni fitnes u generaciji t, a p verovatnoća da će ukrštanje ili mutacija "razbiti" shemu i iznosi:

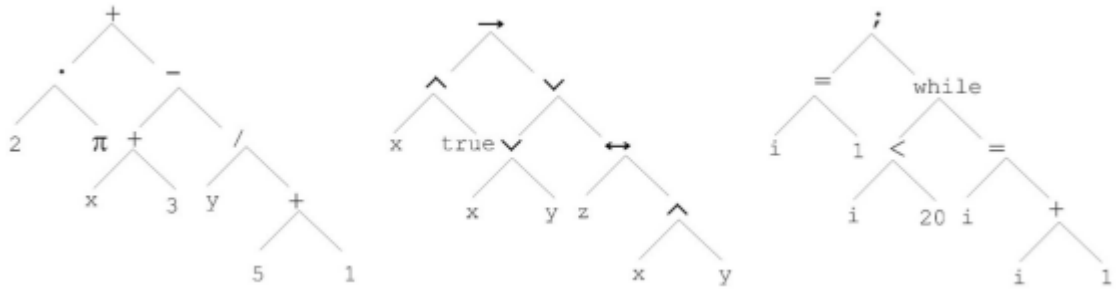
$$p = \frac{\delta(H)}{l-1}p_c + o(H)p_m,$$

gde je l dužina genotipa, a p_c i p_m verovatnoće ukrštanja i mutacije.

24. Genetsko programiranje - pregled koncepata i opštih shema

Genetsko programiranje je razvijeno u Americi 90-ih godina, autor je J. Koza. Obično se primenjuje u mašinskom učenju i konkurentno je neuronskim mrežama i sličnim metodama, ali zahteva velike populacije jedinki (hiljade) i relativno je sporo. Smatra se specijalizacijom genetskog algoritma. Slično kao GA, genetsko programiranje se koncentriše na evoluciju genotipova, a osnovna razlika je u reprezentaciji: tamo gde GA koristi linearnu reprezentaciju (string ili vektor), GP koristi stablo. Primer primene je određivanje kreditne sposobnosti - banka želi da razlikuje dobre i loše kandidate za davanje pozajmica koristeći istorijske podatke. Mogući model: $IF\ (\text{broj dece}\ = 2)\ AND\ (\text{plata}\ \geq 80000)\ THEN\ \text{dobar}\ ELSE\ \text{loš}$. Opšti pristup je: $IF\ \text{formula}\ THEN\ \text{dobar}\ ELSE\ \text{loš}$. Prava formula je nepoznata, a prostor pretrage (fenotip) je skup svih mogućih formula. **Fitnes formule** se meri procentom dobro klasifikovanih primera. Prirodna reprezentacija formule (genotip) je **stablo**, koje omogućava predstavljanje velikog broja formula. Na primer:

- Aritmetička formula: $2\pi + ((x+3) - \frac{y}{5+1})$
- Logička formula: $(x \wedge true) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$
- Program: $i = 1; while(i < 20) i++;$



U genetskim algoritmima (GA), evolutivnim strategijama (ES) i evolutivnom programiranju (EP), hromozomi su linearne strukture, npr. nizovi bitova, nizovi celih brojeva, nizovi realnih brojeva ili permutacije. Stablo-hromozomi su nelinearne strukture. Kod GA, ES i EP veličina hromozoma je fiksna, dok stabla u GP mogu imati proizvoljnu dubinu i širinu. Simbolički izrazi mogu biti definisani pomoću **skupa termova** T i **skupa funkcija** F. Skup termova određuje sve promenljive i konstante, dok skup funkcija sadrži sve funkcije koje se mogu primeniti na elemente skupa termova. Funkcije mogu uključivati matematičke, aritmetičke, logičke funkcije, kao i strukture odlučivanja poput if-then-else i petlji. Elementi skupa termova čine listove stabla, dok su svi ostali čvorovi elementi skupa funkcija. Za neki problem, prostor pretrage se sastoji od skupa svih mogućih stabala koja se mogu konstruisati korišćenjem definisane gramatike. Rekurzivna definicija korektnog izraza glasi:

1. Svaki $t \in T$ je korektan izraz.
2. $f(e_1, \ldots, e_n)$ je korektan izraz ako je $f \in F$, $arity(f) = n$ i $e_1, \ldots, e_n$ su korektni izrazi.
3. Ne postoje druge korektne forme izraza.

U opštem slučaju, izrazi u GP nisu tipizirani, što znači da svaka funkcija $f \in F$ može uzeti bilo koji element $g \in F$ kao argument.

Generisanje potomaka u GP se obavlja slučajno ukrštanjem ili mutacijom. Za **ukrštanje** se koristi razmena podstabala, a za **mutaciju** slučajna promena u drvetu. **Selekcija roditelja** je obično fitnes srazmerna. Kod veoma velikih populacija, rangira se populacija prema fitnesu i deli se u dve grupe: grupa 1 čini najboljih $x\%$ populacije, a grupa 2 ostalih $(100 - x)\%$ populacije. Oko 80% operacija selekcije vrši se nad grupom 1, dok preostalih 20% dolazi iz grupe 2. Za populacije veličine 1000, 2000, 4000, 8000, x je empirijski određen: 32%, 16%, 8%, 4%. **Selekcija preživelih** može biti generacijska ili, u novije vreme, model sa stabilnim stanjem i elitizmom. **Inicijalizacija populacije** se vrši slučajno unutar ograničenja maksimalne dubine i semantike izražene datom gramatikom. Za svaku jedinku bira se koren nasumično iz funkcijskog skupa. Broj dece korena i svakog čvora koji nije list određuje se pomoću $arity$-a odabrane funkcije. Za svaki čvor koji nije koren, nasumično se bira element iz skupa terminala ili funkcija. Kada se izabere element iz skupa terminala, taj čvor postaje list i više se ne širi. **Maksimalna dubina stabla (D_{max})** se postavlja na sledeći način:

- **Balansirani pristup:** teži se ka balansiranom stablu dubine D_{max}. Čvorovi na dubini $d < D_{max}$ biraju se iz funkcijskog skupa F, dok se čvorovi na dubini $d = D_{max}$ biraju iz terminalnog skupa T.
- **Ograničeni pristup:** teži se ka stablu ograničene dubine $\leq D_{max}$. Čvorovi na dubini $d < D_{max}$ biraju se iz skupa $F \cup T$, a na dubini $d = D_{max}$ iz skupa T.

Standardna GP inicijalizacija koristi kombinovani pristup: pola populacije se kreira balansiranim pristupom, a pola ograničenim. Tokom evolucije javlja se problem poznat kao **bloat** - tendencija stabala da postaju prevelika. Potrebne su mere za kontrolu, poput sprečavanja operatora koji dovode do prevelike dece i penalizacije velikih jedinki.

25. Operatori mutacije i ukrštanja kod genetskog programiranja

Najčešće korišćeni operator **mutacije** funkcioniše tako da se slučajno odabrano podstablo zameni novim, nasumično generisanim stablom. Mutacija ima dva parametra:

- **Verovatnoća** p_m odabira mutacije. Ako mutacija nije odabrana, izvršava se ukrštanje.
- **Verovatnoća odabira unutrašnje tačke** kao korena podstabla koje će se menjati.

Preporučuje se da p_m bude 0 ili veoma blizu nule, na primer 0.05. Veličina potomka može biti veća od veličine roditelja. Najčešće korišćeni operator **ukrštanja** funkcioniše tako da se dva slučajno odabrana podstabla zamenjuju između roditelja. Ukrštanje takođe ima dva parametra:

- **Verovatnoća** p_c za odabir ukrštanja. Ako ukrštanje nije odabrano, izvršava se mutacija.
- **Verovatnoća odabira unutrašnje tačke** (korena podstabla) kao pozicije za ukrštanje kod svakog roditelja.

Veličina potomka može biti veća od veličine roditelja.

26. Inteligencija rojeva - uopšteno

Roj se formalno definiše kao grupa (uglavnom mobilnih) agenata koji međusobno komuniciraju direktno ili indirektno. Interakcije među agentima dovode do **distributivnih kolektivnih strategija** za rešavanje problema. **Inteligencija rojeva (Swarm Intelligence, SI)** odnosi se na ponašanje koje rešava problem, a proizilazi iz interakcije među agentima. **Computational Swarm Intelligence (CSI)** označava algoritamske modele takvog ponašanja. SI se često naziva i **inteligencijom kolektiva**. Kompleksno kolektivno ponašanje nastaje iz interakcija pojedinaca kroz vreme, bez centralne kontrole. Najvažniji faktor je **interakcija i saradnja**. Ona može biti:

- **Direktna**: fizički kontakt, vizuelni, zvučni ili hemijski signali.
- **Indirektna**: preko promena u sredini (**stigmergija**).

Primeri iz prirode:

- **Pčele** - kooperacija u koloniji, regulacija temperature, podela rada, komunikacija o hrani.
- **Ose** - traženje hrane i vode, gradnja složenih gnezda.
- **Termiti** - kompleksna gnezda sa konusnim zidovima i ventilacijom.
- **Mravi** - feromonski tragovi (autoputevi), formiranje mostova telima, podela posla.

Osobine socijalnih insekata:

- **Fleksibilnost** - kolonija se prilagođava promenama.
- **Robusnost** - zadaci se obavljaju i ako neke individue zakažu.
- **Decentralizovanost** - nema lidera ni centralne kontrole.
- **Samoorganizovanost** - rešenja nastaju spontano, nisu unapred definisana.

Algoritamski modeli:

- **PSO (Particle Swarm Optimization)** - jedinke se kreću prema svom najboljem susedu i najboljoj poznatoj poziciji.
- **ACO (Ant Colony Optimization)** - inspirisano mravima i feromonskim tragovima. Najbolji putevi imaju najveću koncentraciju feromona.

Primeri primene: optimizacija ruta (TSP), rutiranje u komunikacionim mrežama, klasterovanje i sortiranje, podela posla, kooperativni transport i izgradnja složenih struktura.

27. Optimizacija rojevima čestica - opšti koncepti i osnovni algoritam

Optimizacija rojevima čestica (PSO) je metoda inspirisana socijalnom psihologijom i ponašanjem živih bića u grupama. Osnivači metode su James Kennedy i Russell Eberhart. Ideja je da čestice, koje predstavljaju jedinke u roju, imitiraju društveno ponašanje: interaguju međusobno i uče iz sopstvenog iskustva, što ih postepeno vodi ka boljim rešenjima. Čestice u PSO se mogu posmatrati slično kao ćelije u celularnim automatima. Svaka jedinica ažurira svoje stanje paralelno sa ostalima, novo stanje zavisi od sopstvenog prethodnog stanja i od suseda, a ista pravila važe za sve čestice. Upravo zbog toga što se uzimaju u obzir brzine i ubrzanja, koristi se naziv „čestice“ umesto „tačke“. PSO se može primeniti na različite vrste problema: na realne, diskretne ili mešovite prostore pretrage, zatim na probleme sa više lokalnih optimuma i ograničenjima, kao i na višeciljnu i dinamičku optimizaciju. U originalnoj verziji PSO-a svaka čestica ima svoju **trenutnu poziciju** i **brzinu**, zatim **lično najbolje rešenje** koje je do sada pronašla, kao i **globalno najbolje rešenje** u celom roju. Brzina čestice ažurira se kao kombinacija tri komponente:

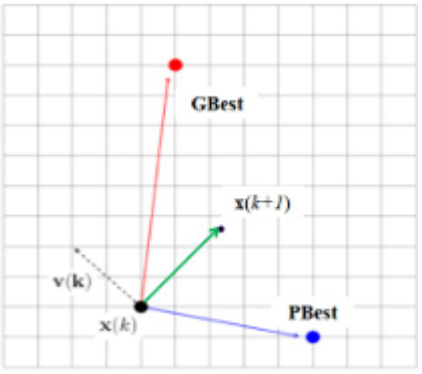
- **momenat** - prethodno stanje brzine se prenosi na novo
- **kognitivna komponenta** - povratak na lično najbolje rešenje
- **socijalna komponenta** - kretanje ka globalno najboljem rešenju

Nakon toga, nova pozicija se dobija pomeranjem na osnovu izračunate brzine. Različite topologije se mogu koristiti da bi se usmerio tok informacija između čestica, na primer prsten, zvezda ili fon Nojmanova mreža. Da bi se kontrolisao uticaj prethodne brzine na novu, uvodi se **inercijalna težina** W. Ako je $W > 1$, brzina raste tokom vremena i roj divergira. Kada je $0 < W < 1$, čestice usporavaju i algoritam može da konvergira, dok za $W < 0$ brzina postepeno opada do nule i algoritam se zaustavlja. Empirijski rezultati pokazuju da konstantna vrednost $W = 0.7298$ i koeficijenti ubrzanja $c_1 = c_2 = 1.49618$ daju dobre rezultate. Ipak, preporučuje se korišćenje promenljive inercije koja se smanjuje tokom vremena, najčešće od 0.9 do 0.4, čime se algoritam postepeno prebacuje iz režima

intenzivne eksploracije u režim jače eksploatacije. Na visokom nivou, pseudokod PSO algoritma izgleda ovako:

- inicijalizuj nasumičnu populaciju
- ponavljaj dok se ne ispuni kriterijum zaustavljanja:
  - za i od 1 do veličine populcije radi:
    - ako je $f(X_i) < f(P_i)$ onda $P_i = X_i$, gde je P_i lični najbolji položaj
    - P_g = $\min P_{\text{neighbours}}$, gde je P_g globalni najbolji položaj
    - za d od 1 do broja dimenzija radi:
      - ažuriraj brzine
      - ažuriraj pozicije

28. Geometrijska interpretacija optimizacije rojevima čestica i primeri



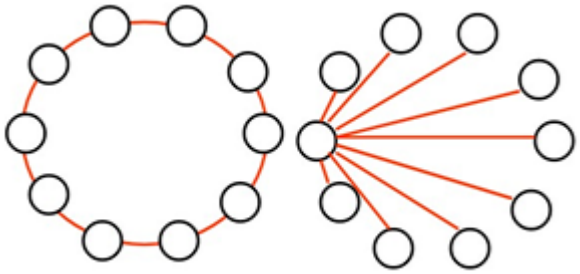
Jedno od važnih pitanja u optimizaciji rojevima čestica jeste koliko su interakcije između čestica presudne za ponašanje algoritma. Da bismo to razumeli, možemo posmatrati pojednostavljeni PSO sa jednom dimenzijom, bez stohastičkih komponenti i sa unapred zadatim početnim pozicijama i brzinama. U ovom modelu brzina i pozicija čestice ažuriraju se formulom:

$$v \to Wv + c_1(p_i - x) + c_2(p_g - x), \quad x \to x + v$$

U primerima se uzima da su parametri $W = 0.7$ i $c_1 = c_2 = 0.7$. Funkcija koju optimizujemo je $f(x) = x^2$, definisana na intervalu [-20, 20]. Pretpostavimo da imamo dve čestice sa početnim pozicijama sa iste strane u odnosu na minimum ($x = 0$). Njihova lična najbolja rešenja (p_i) ostaju na početnim pozicijama dok se ne nađe bolje rešenje. Čestice su pod uticajem globalnog najboljeg rešenja (p_g), ali budući da su sve na istoj strani, kretanje može biti ograničeno i postoji rizik od **prerane konvergencije** ili sporijeg dostizanja globalnog minimuma. Ako su sa različitih strana minimuma, lična najbolja rešenja čestica se ažuriraju dok osciluju oko minimuma. Čestice utiču jedna na drugu kroz globalno najbolje, što omogućava bržu i pouzdaniju konvergenciju ka globalnom minimumu. Opšti zaključak je da početne pozicije i međusobna interakcija čestica značajno utiču na ponašanje PSO algoritma.

29. Varijante gbest i lbest algoritam i topologije uticaja

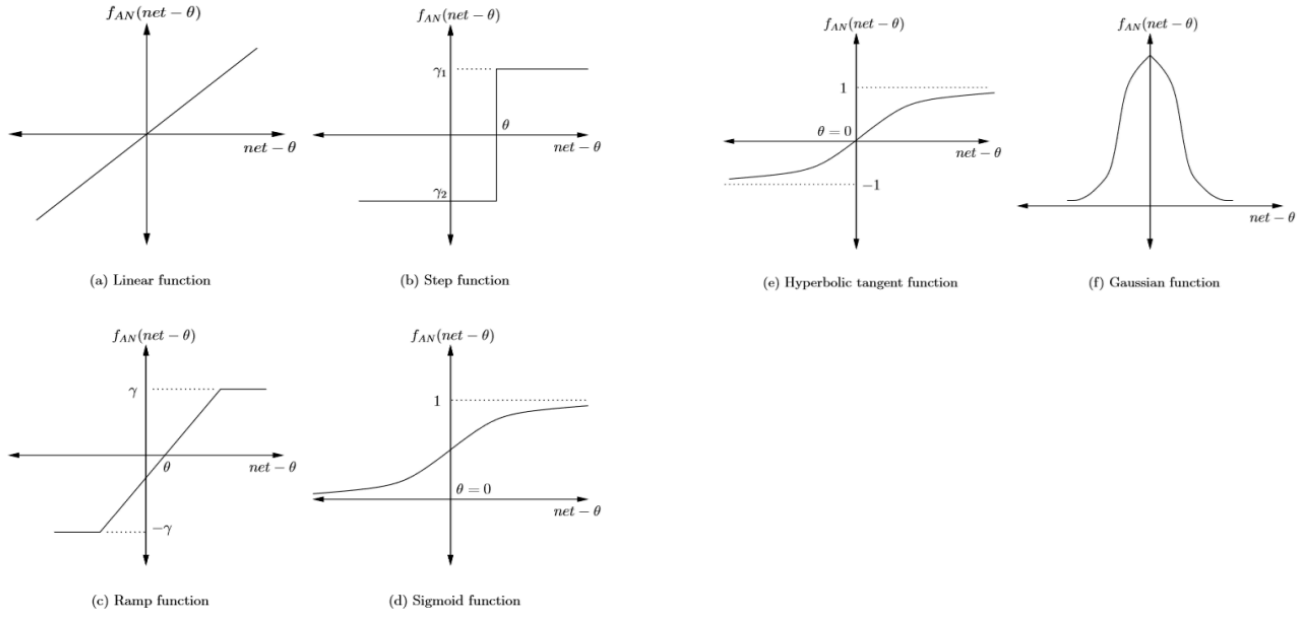
U praksi se najčešće koriste dva modela: **gbest** i **lbest**. U gbest modelu svaka čestica je pod uticajem najbolje jedinke iz čitavog roja, dok je u lbest modelu svaka čestica pod uticajem najboljih jedinki iz svoje lokalne okoline.



Izbor između ova dva modela predstavlja kompromis između eksploatacije i eksploracije. Gbest model najbrže širi informaciju širom populacije, dok lbest model koji koristi topologiju prstena to čini najsporije. Za složene višemodalne funkcije brza propagacija informacija često nije poželjna, ali sa druge strane sporije širenje informacija usporava konvergenciju. U tom smislu, Fon Nojmanova topologija se pokazuje kao najbolje rešenje. Ona podrazumeva da su čestice raspoređene u "rešetku" i svaka čestica komunicira sa svojim neposrednim susedima.

30. Funkcija aktivacije

Funkcija aktivacije je matematička funkcija koja određuje izlaz neurona na osnovu njegovog ulaza. Drugim rečima, ona definiše da li i koliko će jedan neuron "aktivirati" signal i proslediti ga dalje ka sledećim slojevima mreže. Najčešće varijante:



(a) Linear function (b) Step function (e) Hyperbolic tangent function (f) Gaussian function

(c) Ramp function (d) Sigmoid function

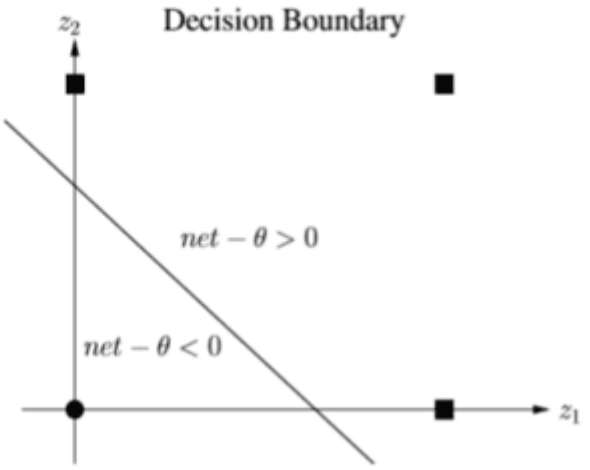
31. Linearna i nelinearna razdvojivost

Linearno razdvojivi podaci su oni kod kojih je moguće pronaći jednu hiperravan (pravac u 2D, ravan u 3D) koja ih tačno deli na dve klase bez greške. To znači da se svi podaci sa različitim izlazima mogu potpuno odvojiti linearnom granicom. Klasičan primer linearno razdvojivog problema je logička disjunkcija (OR), gde je moguće povući pravac koji jasno

razdvaja tačke koje daju izlaz 0 od onih koje daju izlaz 1.

Truth Table

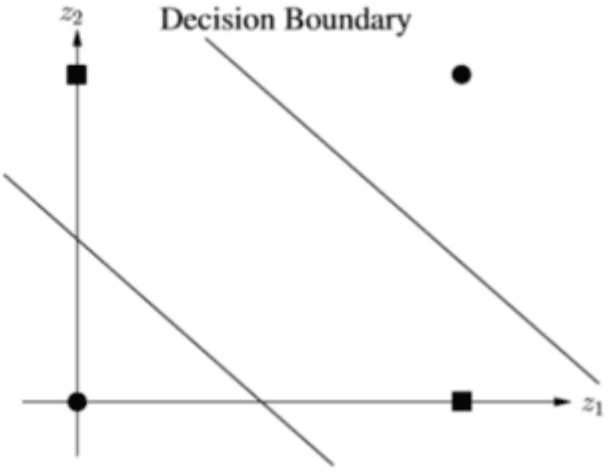
z_1	z_2	z_1 OR z_2
0	0	0
0	1	1
1	0	1
1	1	1



Nelinearno razdvojivi podaci su oni koje nije moguće odvojiti jednom hiperravni. Takvi problemi zahtevaju složenije strukture neuronske mreže, na primer dodavanje skrivenog sloja. Tipičan primer je logička ekskluzivna disjunkcija (XOR), gde je potrebno postojanje najmanje jednog skrivenog sloja sa više neurona da bi se postigla tačna klasifikacija, jer linearna granica ne može da razdvoji klase bez greške.

Truth Table

z_1	z_2	z_1 XOR z_2
0	0	0
0	1	1
1	0	1
1	1	0



32. Učenje veštačkog neurona

Učenje veštačkog neurona zasniva se na ideji da se podešavanjem težina postigne što bolja aproksimacija veze između ulaznih i izlaznih signala. Cilj je da razlika između željenog izlaza (ciljne vrednosti) i stvarnog izlaza neurona bude što manja, odnosno da se minimizuje greška:

$$\mathcal{E} = \sum_{p=1}^{P_r} (t_p - o_p)^2,$$

gde su t_p i o_p redom ciljna i aproksimirana vrednost izlaznog signala, a P_r broj podataka sprovedenih na ulaz. **Metod gradijentnog spusta** koristi se za postepeno smanjivanje ukupne greške. Algoritam koriguje težine tako što ih pomera u suprotnom smeru od gradijenta:

$$v_i(t) = v_i(t-1) + \Delta v_i(t),\ \Delta v_i(t) = \eta(-\frac{d\mathcal{E}}{dv_i})$$

$$v_i(t) = v_i(t-1) + 2\eta(t_p - o_p)z_{i,p},$$

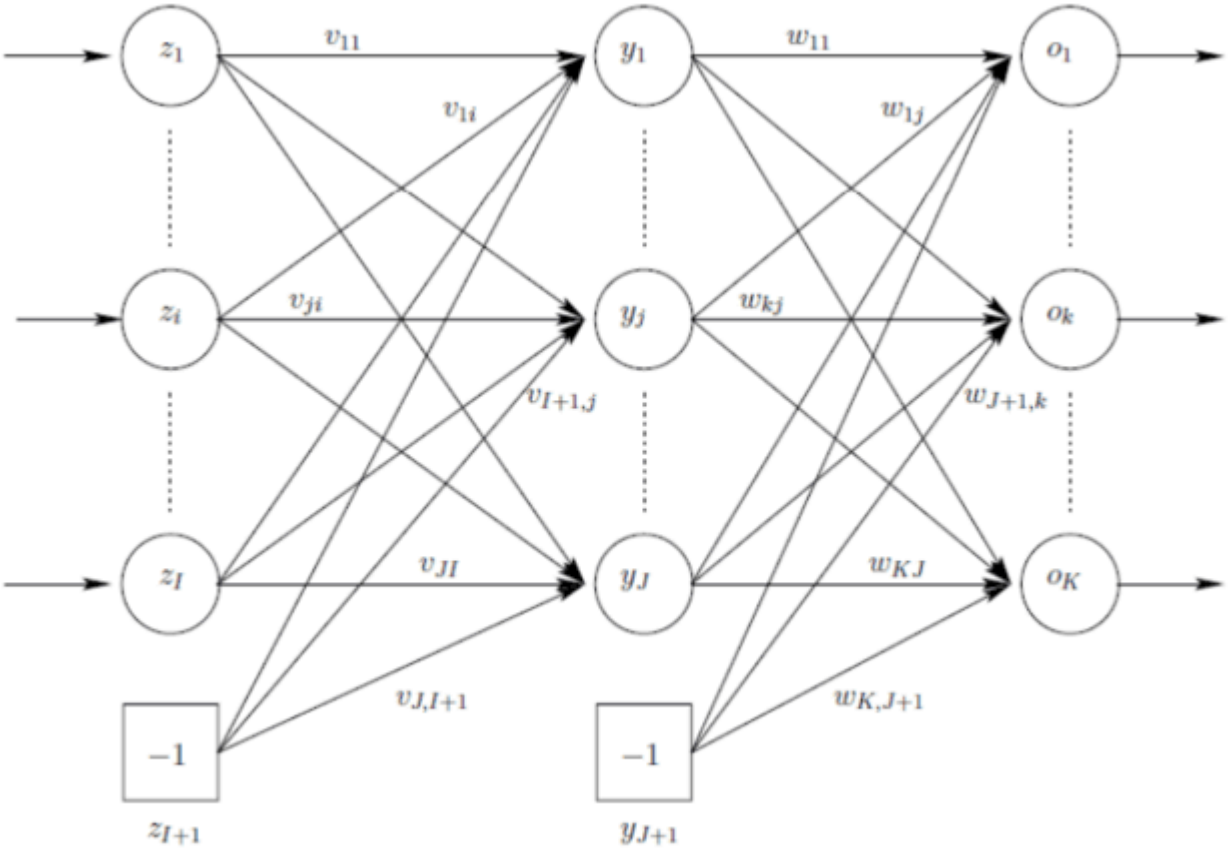
gde je $z_{i,p}$ i-ti ulaz podatka p. Na ovaj način, težine se iterativno prilagođavaju i mreža uči da daje sve preciznije rezultate. Parametar **brzine učenja** određuje koliki korak se pravi pri svakoj promeni težina.

33. Tipovi i organizacija veštačkih neuronskih mreža, slike sa objašnjenjima

Jedan veštački neuron može da nauči samo linearno razdvojive funkcije. Grupisanjem neurona u mreže dobijamo mogućnost rešavanja nelinearnih problema. Međutim, učenje neuronskih mreža je značajno kompleksnije i računski zahtevnije. Postoje dva osnovna pristupa:

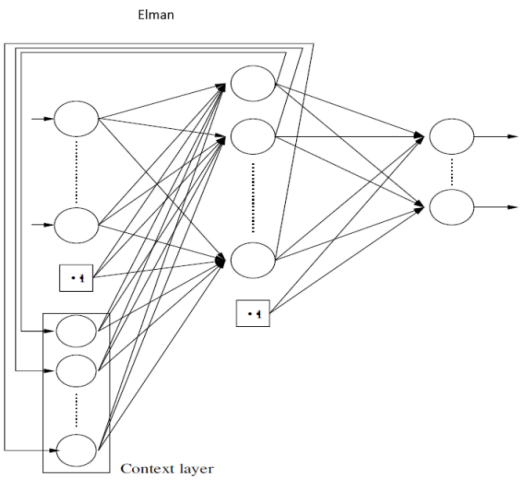
- **Nadgledano učenje** - koristi skup podataka za trening, pri čemu svaki podatak ima pridruženu ciljnu promenljivu.
- **Nenadgledano učenje** - nemamo ciljne vrednosti, već želimo da otkrijemo obrasce i strukturu u podacima.

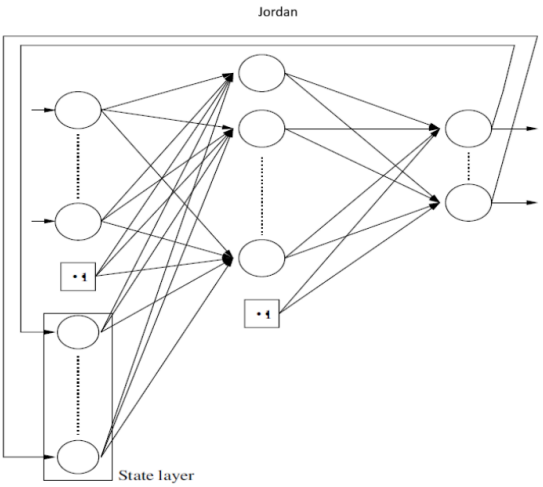
Mreže sa propagacijom unapred (Feedforward Neural Network - FFNN) sastoje se najmanje od tri sloja: **ulaznog**, **skrivenog** i **izlaznog**. Signal prolazi kroz mrežu samo u jednom smeru, a izlaz se računa pomoću jednog prolaska kroz mrežu. Najčešće se koriste za klasifikaciju i regresiju.



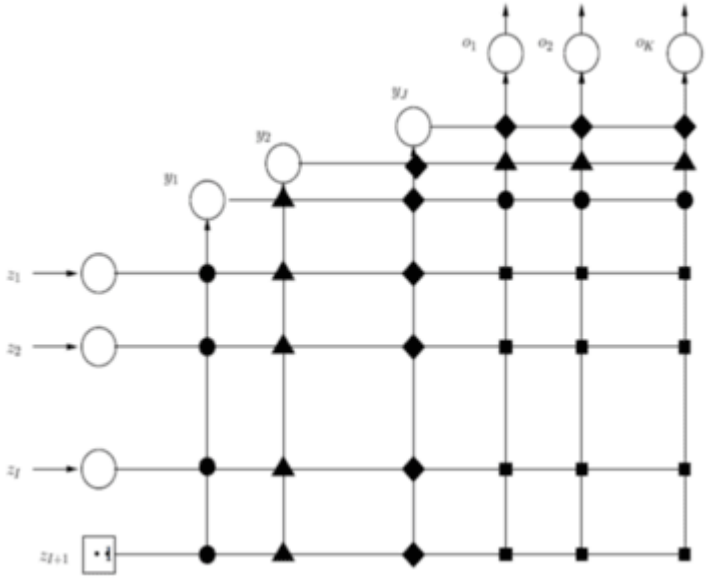
Rekurentne neuronske mreže (RNN) imaju povratne veze koje omogućavaju modelovanje sekvenci i zavisnosti u vremenu. Vrste:

- **Elman SRNN (Simple Recurrent Neural Network)** - kopija skrivenog sloja vraća se na ulaz (**kontekstni sloj**). Na taj način mreža koristi prethodno stanje, što omogućava učenje temporalnih zavisnosti.
- **Jordan SRNN** - kopija izlaznog sloja vraća se na ulaz (**sloj stanja**). Ovaj pristup takođe omogućava pamćenje prethodnih izlaza.





Kaskadne neuronske mreže (Cascade Neural Network - CNN) podrazumevaju da su svi ulazi povezani sa svim skrivenim i svim izlaznim neuronima. Elementi skrivenog sloja povezani su sa svim izlazima, ali i sa svim narednim elementima u skrivenom sloju. Na taj način se gradi složenija struktura koja omogućava bogatije reprezentacije.



34. Pravila nadgledanog i nenadgledanog učenja

U **nadgledanom učenju** imamo konačan skup uređenih parova ulaznih vrednosti i pridruženih ciljnih vrednosti: $D = \{(z_p,\ t_p) \mid p = 1,\ \ldots,\ P\}$, gde su $z_{i,\,p},\ t_{k,\,p} \in R$ za $i = 1,\ \ldots,\ I$ i $k = 1,\ \ldots,\ K$. I je broj ulaznih signala, K broj izlaznih signala, a P je broj trening podataka. Imamo zavisnost $t_p = \mu(z_p) + \xi_p$, gde je ξ_p **šum (greška)**. Cilj učenja je aproksimirati nepoznatu ciljnu funkciju $\mu(z_p)$ na osnovu podataka iz skupa D. Skup D se obično deli na tri disjunktna podskupa:

- D_T za trening i aproksimaciju
- D_V za validaciju
- D_G za testiranje

Tokom faze učenja minimizuje se empirijska greška podešavanjem težina W:

$$\mathcal{E}_T(D_T;\ W) = \frac{1}{P_T}\sum_{p=1}^{P_T}(F_{NN}(z_p,\ W) - t_p)^2$$

Za optimizaciju se koriste lokalne metode (gradijentni spust) i globalne metode (metaheuristike). Glavni izazovi su **preprilagođavanje (overfitting)** kada model previše dobro uči trening podatke, uključujući šum, pa loše generalizuje na nove podatke i **potprilagođavanje (underfitting)** kada je model previše jednostavan i ne uspeva da uhvati obrasce u podacima, pa loše radi i na trening i na test podacima. **Gradijentni spust za učenje neuronskih mreža** sastoji se iz dve faze:

1. signal se propagira unapred kroz mrežu (FFNN) i računa izlaz
2. greška se propagira unazad kako bi se izvršila promena težina

Kao funkcija cilja često se koristi **suma kvadrata grešaka (Sum of Squared Errors - SSE)**:

$$E = \frac{1}{2}\sum_{k=1}^{K}(t_k - o_k)^2$$

Ako se koriste sigmoidne funkcije aktivacije, izlaz neurona se računa kao

$$o_k = f_{o_k}(net_{o_k}) = \frac{1}{1 + e^{-net_{o_k}}}$$

Stohastički gradijentni spust ažurira težine iterativno, uz mogućnost dodavanja momenta koji prenosi deo prethodne promene težina, čime se ubrzava konvergencija i izbegava zaglavljivanje u lokalnim minimumima. Algoritam:

- inicijalizuj težine i broj epoha $t = 0$
- dok nije ispunjen uslov za završetak radi:
  - $E = 0$
  - za svaki trening podatak p radi:
    - propagiraj podatak unapred i računaj y i o
    - računaj signale grešaka Δ_o i Δ_y
    - ažuriraj težine w i v (propagacija grešaka unazad)
    - $E +=\ E_p$
  - $t = t + 1$

U **nenadgledanom učenju** ne postoji očekivani izlaz. Mreža mora sama da otkrije pravilnosti u ulaznim podacima. Ovakve mreže omogućavaju stvaranje asocijacija između obrazaca (**pattern association**) i često se nazivaju asocijativna memorija ili **asocijativne**

neuronske mreže. Cilj je da mreža samostalno identifikuje strukturu podataka i povezanost između različitih ulaznih šablona.

35. Asocijativna neuronska mreža i Hebovo učenje

Asocijativne neuronske mreže su obično dvoslojne i imaju cilj da omoguće stvaranje asocijacija bez upotrebe "učitelja". Razvoj ovakvih mreža inspirisan je studijama vizuelnog i zvučnog korteksa kod mozga sisara. Topološka organizacija neurona omogućava povezivanje ulaznih i izlaznih šablona, dok dodatna poželjna osobina mreže jeste zadržavanje prethodnih informacija i nakon pristizanja novih podataka, što kod nadgledanog učenja obično nije moguće. Funkcija koju ovakva mreža uči je preslikavanje ulaznog šablona u izlazni: $f_{NN} : R^I \rightarrow R^K$. **Hebovo učenje**, nazvano po neuropsihologu Donaldu Hebu, zasniva se na principu korelacije između aktivacionih vrednosti neurona. Prema Hebovoj hipotezi, potencijal neurona da generiše signal zavisi od potencijala njegovih susednih neurona. Težina između dva korelisana neurona se pojačava proporcinalno njihovoj aktivaciji:

$$u_{ki}(t) = u_{ki}(t-1) + \Delta u_{ki}(t),\ \Delta u_{ki}(t) = \eta o_{k,p} z_{i,p}$$

Izmena težina je veća za ulazno-izlazne parove kod kojih ulazna vrednost snažnije utiče na izlaz. Problem kod ponovnog ubacivanja istih ulaznih šablona je eksponencijalni rast težina, što se rešava postavljanjem limita na maksimalnu vrednost težina. Jedan primer je primena nelinearnog faktora zaboravljanja:

$$\Delta u_{ki}(t) = \eta o_{k,p} z_{i,p} - \gamma o_{k,p} u_{ki}(t-1),$$

gde je γ pozitivna konstanta koja kontroliše smanjenje težina tokom vremena.

36. Kvantizacija vektora 1

LVQ-1 (Learning Vector Quantizer-1) klasterovanje je nenadgledana metoda učenja koja služi za klasterovanje podataka. Cilj je grupisati skup od n podataka u m grupa tako da su elementi iz iste grupe što sličniji jedni drugima. Za meru sličnosti ili razlike između podataka obično se koristi **Euklidsko rastojanje**. Izlazne vrednosti, odnosno oznake klastera, "takmiče se" za ulazne podatke. Algoritam:

- inicijalizuj težine mreže, brzinu učenja i prečnik susedstva
- dok uslov za završetak nije zadovoljen radi:
  - za svaki ulazni podatak p radi:
    - izračunaj Euklidsko rastojanje d između ulaznog vektora z i svakog vektora težine u
    - pronađi izlaznu vrednost o za koju je rastojanje d najmanje
    - ažuriraj sve težine u susedstvu k prema odgovarajućoj formuli
  - ažuriraj bazu učenja
  - smanji prečnik susedstva

37. Samoorganizujuće mape

Samoorganizujuće mape (SOM) razvio je Kohonen kako bi modelirao karakteristike ljudskog cerebralnog korteksa. Metoda vrši projekciju višedimenzionog ulaznog prostora u diskretni izlazni prostor, što predstavlja neku vrstu kompresije. Izlazni prostor je obično dvodimenziona mreža vrednosti. Glavna ideja je očuvanje topološke strukture ulaznog prostora: ako su dva podatka blizu u ulaznom prostoru, biće blizu i u izlaznom prostoru, a slične moždane aktivnosti aktiviraju bliske neurone. SOM koristi **stohastičko pravilo učenja** zasnovano na kompetitivnoj strategiji, slično LVQ-1 klasterovanju. Ulazni podaci su povezani sa neuronima u mapi, koja je često kvadratnog oblika. Broj neurona je manji od broja trening podataka, a idealno bi broj neurona bio jednak broju nezavisnih trening primera. Vektor težina za svaki neuron na poziciji $(k,\ j)$ inicijalno je nasumično postavljen: $w_{kj} = (w_{kj1},\ w_{kj2},\ \ldots\ w_{kjI})$. Dimenzija vektora težina odgovara dimenziji ulaznog podatka. Za svaki ulazni podatak pronalazi se neuron čiji je vektor težina najsličniji ulaznom podatku (npr. po Euklidskom rastojanju). Nad tim **pobedničkim neuronom** vrši se korekcija težina u skladu sa ulaznim podatkom, a takođe se koriguju i težine susednih neurona proporcionalno njihovoj udaljenosti od pobednika. Tokom treninga, opseg susednih neurona se smanjuje, dok na kraju neuroni praktično više nemaju suseda. Primene SOM uključuju analizu slika, prepoznavanje zvuka, obradu signala, telekomunikacije i analizu vremenskih serija. Prednost SOM-a je to što omogućavaju laku vizualizaciju i interpretaciju podataka, jer oblasti koje klasifikuju podatke postaju vidljive na mapi.