NEIZRAŽITI SKUPOVI

⇒ **klasični skup:**

- definicija pomoću karakteristične funkcije:

$$\mu_A(x) : X \longrightarrow \{0, 1\}$$

$$\mu_A(x) = \begin{cases} 1, & \text{ako je } x \in A \\ 0, & \text{ako je } x \notin A \end{cases}$$

- drugim riječima, karakteristična funkcija odgovara na pitanje „Je li x iz tog skupa ili ne?"

⇒ **neizraziti skup:**

- definiramo ga „karakterističnom" funkcijom koja ne preslikava na $\{0, 1\}$ nego na $[0, 1]$, to su zove funkcija pripadnosti:

$$A = \{(x, \mu_A(x)) \mid x \in U, \mu_A(x) \in [0, 1]\}$$

U ⇒ univerzalni skup (svi mogući elementi)
A ⇒ neizraziti skup

- prinjet da je neizraziti skup, skup uređenih parova („element", „stupanj pripadnosti")

- kraći zapis elementa neizrazitog skupa: $\frac{\mu_A(x)}{x}$
- zapis neizrazitog skupa:
  - a) prebrojiv: $A = \sum_x \frac{\mu_A(x)}{x}$
  - b) neprebrojiv: $B = \int_x \frac{\mu_A(x)}{x}$

NAPOMENA: ako po nekoj tvrdnji X pripada neizrazitom skupu A sa mjerom „a", a po nekoj drugoj tvrdnji sa mjerom „b", tada uzimamo maksimum:

$$\frac{a}{x} + \frac{b}{x} \Rightarrow \frac{\max(a,b)}{x}$$

⇒ sama definicija neizrazitog skupa (tj. njegove funkcije pripadnosti) ovisi o problemu koji rješavamo te je subjektivna

↳ no, to nas ne muči previše jer sama mjera pripadnosti nije odviše bitna nego su bitni odnosi među njima

⇒ JEDNAKOST NEIZRAZITIH SKUPOVA:

↳ A, B su definirani nad univerzalnim skupom U, za svaki $x \in U$ vrijedi:

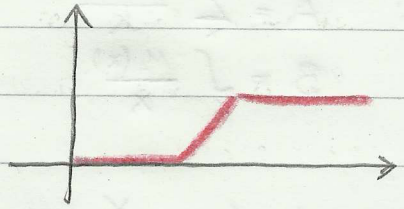
$$A = B \Rightarrow \mu_A(x) = \mu_B(x)$$

⇒ postoji i definicija podskupa, univerzalnog neizrazitog skupa i praznog (neizrazitog skupa

(VIDI SLAJDOVE!)

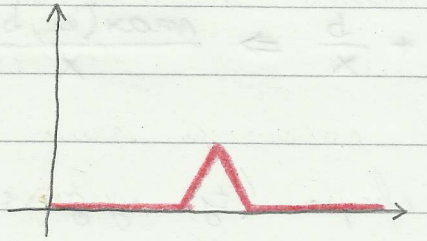
⇒ najčešće funkcije pripadnosti:

a) Γ funkcija („gama funkcija"):

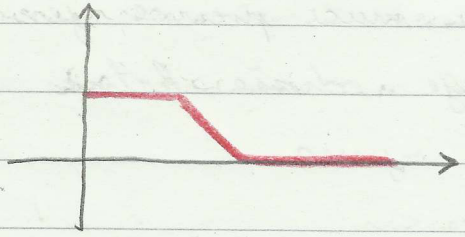


(opis visokih ljudi)

b) Λ funkcija

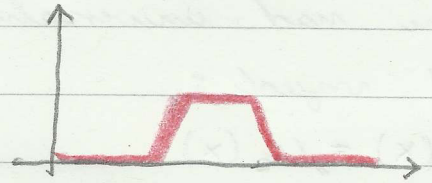


c) L funkcija



(opis niskih ljudi)

d) Π funkcija



(intervalni opis)

⇒ SVOJSTVA NEIZRAZITIH SKUPOVA:

1) JEZGRA NEIZRAZITOG SKUPA

↳ klasičan skup koj je podskup univerzalnog skupa sa svojstvom $\mu_A(x)=1$

2) POTPORA NEIZRAZITOG SKUPA

↳ klasičan skup koj je podskup univerzalnog skupa sa svojstvom $\mu_A(x)>0$

3) VISINA NEIZRAZITOG SKUPA

↳ najveća mjera pripadnosti elemenata izvarstog skupa

↳ normalan neizrazit skup je neizrazit skup visine jedan ($hgt(A)=1$)

4) λ-PRESJEK NEIZRAZITOG SKUPA

↳ klasičan skup koj sadrži elemente iz neizrazitog skupa koj pripadaju tom neizrazitom skupu sa mjerom pripadnosti od barem „λ"

5) PRODUKT λA:

↳ A je klasičan skup, a $\lambda \in [0,1]$

↳ produkt λA je neizrazit skup od elemenata iz A od kojih svaki pripada tom neizrazitom skupu sa λ

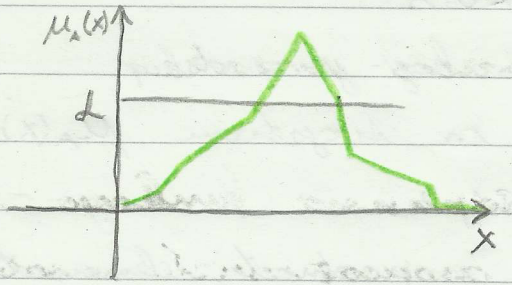
⇒ svaki neizrazit skup se može prikazat kao unija / suma njegovih λA_λ produkata

BITAN TEOREM!

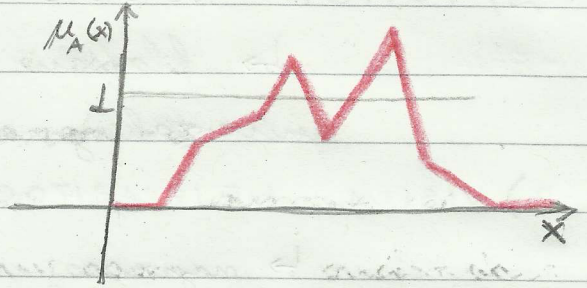
(VIDI SLAJDOVE!)

⇒ KONVEKSNOST NEIZRAZITOG SKUPA

↳ neizrazit skup je konveksan ako i samo ako se svi njegovi L presjeci definiraju nad jednim intervalom



KONVEKSAN



NEKONVEKSAN

(jer se prikazan L-presjek raspada na dva disjunktna skupa)

↳ konveksnost neizrazitog skupa je bitno svojstvo i koristit će nam za definiranje neizrazitih skupova

⇒ DRUGE DEFINICIJE OSNOVNIH OPERACIJA:

PRESJEK

↳ ne mora nužno presjek biti definiran kao minimuma nyega pripadnost već to može biti proizvoljna funkcija:

$$f : [0,1] \times [0,1] \longrightarrow [0,1]$$

↳ najopćenitiji oblik operacije presjeka:

t-norme

↳ t-norma je gore napisana funkcija koja zadovoljava: komutativnost, asocijativnost, monotonost i rubni uvjeti

UNIJA

↳ najopćenitiji oblik operacije unije:

s-norme

↳ ista domena i kodomena kao i t-norme, ali se pravila zadovoljavaju na drugi način (VIDI SLAJDOVE?)

KOMPLEMENT

↳ možemo i drugačije definirati operaciju komplementa, ali mora zadovoljavat određena pravila (VIDI SLAJDOVE?)

⇒ pomoću općenitih (generalizirani) pravila možemo i definirati poopćene **DeMorganove** zakone:

$$s(a,b) = c(t(c(a), c(b)))$$
$$t(a,b) = c(s(c(a), c(b)))$$

⇒ no, ne možemo uzet bilo koju definiciju s-norme, t-normi i komplementa i očekivat da su one DeMorganov sustav (tj. da zadovoljavaju DeMorgana)

↳ možemo birati dvije, a treća će proizaći iz DeMorganovih zakona

↳ napomena, uz komplement biramo „s-" ili „t-" normu ▽

⇒ PARAMETRIZIRANE „t-" i „s-" NORME:

↳ uvodi se parametar za dodatnu kontrolu nad funkcijom pripadnosti

↳ pretpostavit da još uvijek moraju zadovoljit općenite definicije „t-" i „s-" normi tako da parametar utječe samo na „sivo područje"

⇒ NEIZRAZITI SKUPOVI TIPA „M":

↳ neizrazit skup tipa 2 je neizrazit skup čija je mjera pripadnosti neizrazit skup

↳ neizrazit skup tipa 3 je neizrazit skup čija je mjera pripadnosti neizrazit skup tipa 2 (rekurzivna definicija)

ovime se nećemo baviti !

NEIZRAZITE RELACIJE

⇒ standardna matematička relacija:

- skup uređenih parova
- npr. pogledajmo sljedeću domenu:

$$D = \{0, 1, 2, 3\}$$

↳ relacija „manje-il-jednako":

$$\leq \equiv \{(0,0), (0,1), (0,2), (0,3), (1,1), (1,2), (1,3), (2,2), (2,3), (3,3)\}$$

- vidimo da relacija nije ništa više od skupa uređenih parova iz domene koji su „u duhu" te relacije

⇒ NEIZRAZITA RELACIJA:

- neizrazit skup u kojem za svaku n-torku (uređeni par) definiramo koliko je ta n-torka „u duhu" te relacije

$$R = \{((x_1, x_2, \ldots, x_m), \mu_R(x_1, x_2, \ldots x_m)) \mid x_1 \in U_1, x_2 \in U_2, \ldots, x_m \in U_m, \mu_R : U_1 \times U_2 \times \ldots \times U_m \to [0,1]\}$$

⇒ ove relacije su prenijerene za modeliranje stvarnog svijeta jer svijenja nekad nisu apsolutno precizna i stroga

⇒ **kartezijev produkt neizrazitih skupova:**

$$\mu_R(x,y) = \min\left(\mu_A(x), \mu_B(y)\right) \quad \forall x \in A,\ \forall y \in B$$

↓

ovakva relacija je dobra jer:

- možemo ovo koristiti kao operator implikacije jer se visoke mjere pripadnosti gomilaju tamo gdje su obje mjere pripadnosti visoke

⇒ **cilindrično proširenje:**

↳ želim relaciju između skupa A i B, al za B ne znam mjeru pripadnosti:

$$A = \left\{\frac{1}{1} + \frac{0.7}{2} + \frac{0.3}{3} + \frac{0}{4}\right\} \qquad B = \left\{\frac{?}{audi} + \frac{?}{fića} + \frac{?}{opel}\right\}$$

	audi	fića	opel
1	1		
2	0.7		
3	0.3		
4	0		

popunil smo sve što znamo za kartezijev produkt

↳ pretpostavimo optimistično: $B = \left\{\frac{1}{audi} + \frac{1}{fića} + \frac{1}{opel}\right\}$

- sada radimo kartezij, pa samo preslikavamo sve ostale od lijeva nadesno

↳ stoga, cilindrična projekcija primjera

	audi	fiat	opel
1	1	1	1
2	0.7	0.7	0.7
3	0.3	0.3	0.3
4	0	0	0

⇒ **projekcija:**

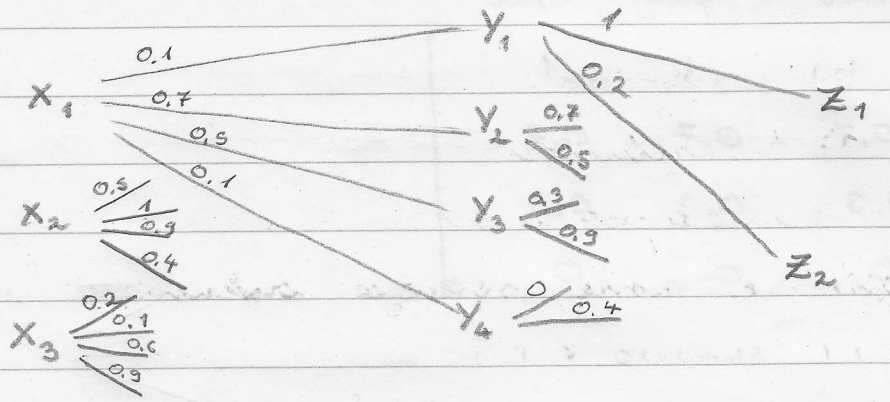
↳ ovisna o osi, to je vektor koj govori o maksimalnoj mjeri u kojoj element sudjeluje u kartezijevom produktu

(VIDI PRIMJER NA SLAJDOVIMA!)

⇒ nova: spoj relacija
neinteraktivnost binarne relacije
kompozicija binarne relacije

} koristno za smanjenje broja informacija

PR: Kompozicija linearne relacije II.



↳ tražimo veru od X do Z:

	Z_1	Z_2
X_1	?	?
X_2	?	?
X_3	?	?

↳ veru između „X_i" i „Z_j" tražimo ovako:

1) naći minimum svakog puta / vere

2) naći maksimum minimuma svih vera

⇒ SVOJSTVA RELACIJA:

↳ klasične relacije definiraju četiri tipična svojstva:
1) refleksivnost
2) simetričnost
3) tranzitivnost
4) antisimetričnost

↳ neizrazite binarne relacije definiraju ista svojstva (VIDI SLAJDOVE!)

• bitna je tranzitivnost! (shvat ono što piše na slajdovima!)

↳ EKVIVALENCIJA:

• svojstvo izvedeno iz četiri osnovna svojstva
• ne postoji jedinstvena definicija jer ova relacija ovisi o definiciji t-norme
• ako ne piše eksplicitno, koristi se Zadehova definicija t-norme!

NAPOMENA:

• vidi primjer sa sličnostima automobila (vrijedi razumjeti ekvivalenciju na tom primjeru)

⇒ RELACIJA „KOMPATIBILNOSTI":

PR: Promatramo bliskost (geografsku) gradova ▽:

DUGO SELO ← --- 20km --- ZAGREB --- 25km ---> SAMOBOR

↳ „bliskost" ⇒ udaljenost do 30km

↳ u relaciji su:

(DUGO SELO, ZAGREB)

(SAMOBOR, ZAGREB)

↳ u relaciji nisu:

(DUGO SELO, SAMOBOR)

⇒ kompatibilnost: relacija kojoj nemamo tranzitivnost, a relacija je refleksivna i simetrična

⇒ matematički „teška" definicija sa slajdova:

- znamo koliko koraka trebaš spojat da spojiš svakoga sa svima
  - ↳ u najgorem slučaju, to je „n - 1" koraka
- npr. na početku:

- prvi korak: spojamo na udaljenost 2

• drugi korak: spajamo na udaljenost 3

• u najgorem slučaju, to će biti „n - 1" koraka

⇒ <u>Zatvaranje binarne relacije:</u>

↳ ∃ jedna bitno svojstvo

↳ pogledaj definiciju na slajdovima i u knjizi

⇒ neka od ovih svojstva biti će korisna kada naša početna relacija neće biti simetrična:

• npr. ispitivanje ljudi o sličnosti automobila može dovesti do toga da:

$$\text{SLIČNOST}(A_1, A_2) \neq \text{SLIČNOST}(A_2, A_1)$$

↳ takva nešto se u stvarnosti javlja, pa trebamo alat da to normaliziramo

⇒ sada smo upoznali teoriju neizrazitih skupova i relacija, pa možemo krenuti na raključivanje i neizrazitu logiku

NEIZRAZITA LOGIKA

⇒ trebamo standardne vernike:

- negacija: $\neg a = 1 - a$
- konjunkcija: $a \wedge b = \min(a, b)$
- disjunkcija: $a \vee b = \max(a, b)$
- implikacija: $a \rightarrow b = \min(\neg b, \min(1, \ldots$

⇒ <u>ne vrijedi</u>:

- zakon kontradikcije $(a \wedge \neg a = 0)$
- zakon isključenja trećega $(a \vee \neg a = 1)$

⇒ NEIZRAZITI PREDIKATI:

- preslikavaju neku tvrdnju u mjeru istinitosti između nula i jedan $[0, 1]$
- opisuju svojstvo objekata domene

„ X je P "

↳ $\mu_P(x)$ ⇒ koliko je istina da je x P ?

⇒ MODUS PONENS:

↳ klasičan modus ponens:

- činjenica: x je A
- pravila: Ako x je A tada y je B
- zaključak: y je B

↳ <u>generalizirani modus ponens</u> za neizrazitu logiku:

- činjenica: x je A'
  (x „nagnje" prema A)
- pravilo: Ako x je A tada y je B
- zaključak: y je B'
  (y „nagnje" prema B)

⇒ generalizirani modus ponens omogućava izvođenje zaključaka iz rečenica prirodnog jezika

⇒ NEIZRAZITA PRODUKCIJSKA PRAVILA:

↳ ako - onda:

AKO (neizrazita propozicija) ONDA (neizrazita propozicija)

↳ implikacija ima više oblika što nam je u neizrazitoj super jer možemo na različite načine računati implikaciju

⇒ INTERAKCIJA IMPLIKACIJE:

- kad izračunamo implikaciju zapravo dobijemo veru između antecedenta i konsekvensa
- no, kada napravimo kompoziciju antecedenta i izračunate implikacije, dobijemo relaciju koja je u duhu konsekvensa, ali nije ista
  ↳ to nazivamo interaktivnost implikacije

⇒ <u>vrste implikacije</u>:

a) s lokalnim djelovanjem:

- nije slično implikaciji iz klasične logike
- tvrdi da je istina samo ono što implikacija tvrdi
- njih spajamo sa unijom (s-normom)
- primjer je Mamdani implikacija
- mjere pripadnosti „crnih" područja (tamo gdje nema definiranih pravila) su niske

b) s globalnim djelovanjem:

- odgovara implikaciji u klasičnoj logici
- tvrdi samo da nije istina ako je A onda nije B, a u ostalim slučajevima je istina (visoke mjere pripadnosti)
- njih spajamo presjekom (t-normom)
- mjere pripadnost „svih" podmnožja (tamo gdje nema definiranih pravila) su visoke

NEIZRAZITO UPRAVLJANJE

⇒ ne znamo analitički oblik prijenosne funkcije sustava
↳ umjesto toga, sustav opisujemo AKO-ONDA pravilima
↳ pravila su dobivena eksperimentom ili iskustvom

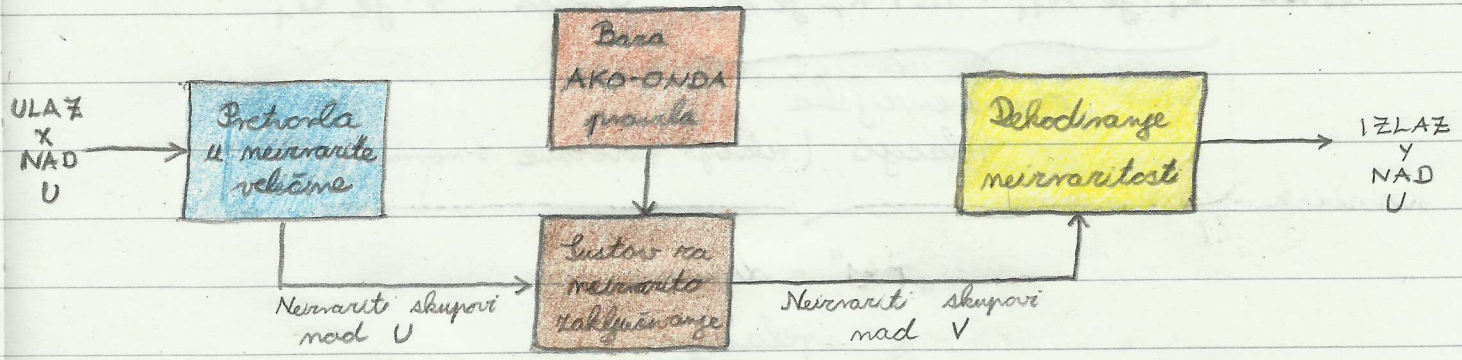
⇒ upravljanje: na temelju trenutnog stanja sustava treba donijet odluku o akciji

⇒ sustav neizrazitog upravljanja:
- na ulazu ima „oštre" („crisp") vrijednosti, al ih možemo „fuzzyficirati"
- izlaz tog sustava mora bit opet „oštra" vrijednost, pa ovdje moramo iz „fuzzy" zaključka odabrat jednu vrijednost
↳ to se naziva DEKODIRANJE NEIZRAZITOSTI

⇒ DEKODIRANJE NEIZRAZITOSTI:
↳ postoj mnogo načina **kako iz neizrazitog skupa odabrat reprezentanta**
↳ drugim riječima, trebamo način prelaska iz „fuzzy" svijeta u „crisp" svijet
↳ vidi slajdove za različite metode defuzzyfikacije

⇒ ZAKLJUČIVANJE U SUSTAVIMA NEIZRAZITOG UPRAVLJANJA:

- razmatramo produkcijske sustave
  - ↳ oni se sastoje od niza AKO-ONDA pravila (produkcija)



⇒ GRAĐA SUSTAVA NEIZRAZITOG UPRAVLJANJA:

- oznake:
  - r ⇒ broj podataka
  - n_R ⇒ broj pravila u bazi pravila
  - R ⇒ „ništa" (samo indeks za oznaku n_R)
  - n ⇒ broj neizrazitih skupova

⇒ dva glavna načina zaključivanja:

1) zaključivanje temeljeno na kompoziciji
2) zaključivanje temeljeno na pojedinačnim pravilima

⇒ **rakljućivanje temeljeno na kompoziciji:**

- kompliciramo, pa se u praksi ne implementira, ali je dobra podloga za druge načine rakljućivanja

AKO x_1 je A_{11} | ... | x_r je A_{1r} ONDA y je B_1

"r" dimenzijska relacija (relacije povezane t-normom)

"r+1" - dimenzijska relacija

↳ takvih relacija imamo R, pa ih želimo iskombinirat u jednu koja mapira ulaz i izlaz

- tu ukupnu relaciju možemo dobit kombiniranjem, ali je ta kombinacija ovisna o vrsti implikacije (s lokalnim il globalnim djelovanjem):

GLOBALNO ⇒ presjek

LOKALNO ⇒ unija

↳ problem je da je složenost pre-velika:

PR: recimo da imamo 5 x-eva ($r=5$) i svaki od njih je iz domene 100 brojeva (takozter, y je isto takav)

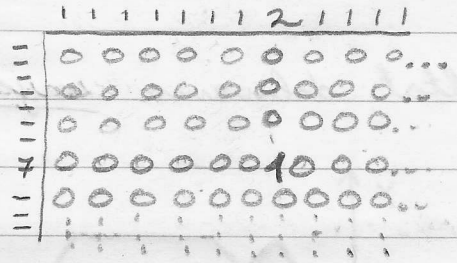
↳ stoga sama implikacija je dimenzija 100^6 (prevelika za pohranu a kamoli za rad kompozicije nad njima, pa tako ne možemo rakljućivat)

⇒ **bolja ideja**:

- umjesto da pamtimo cijele definicije izvarstog skupa (pripadnosti svih elemenata), koristimo SINGLETON ulaze:

↳ samo prikačujemo onaj izmjereni ulaz u svakoj relaciji i on pripada sa 1 tog relacij

PRI: Dobivena relacija za izmjereni $X_1 = 2$, $X_2 = 7$!



↳ stoga, umjesto pohranjivanja 100^5 kombinacija za antecedente, znamo da je za $100^5 - 1$ mjesta relacija jednaka nula, a za jednu „M"-torku (izmjerene podatke) je ona jednaka 1

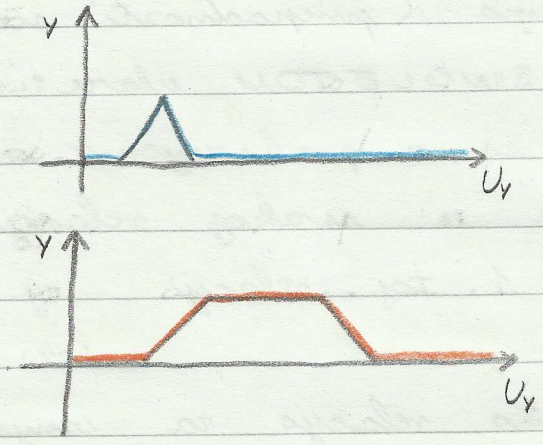
↳ tako, trebamo čitavat samo po domeni raključaka !

BITNO!

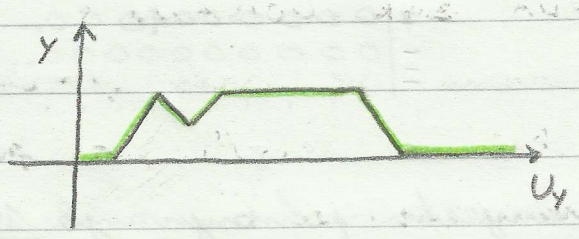
(vidi slajdove!)

⇒ <u>raključivanje temeljeno na pojedinačnim pravilima:</u>

↳ svako pravilo daje svoj rakljućak:



↳ rezultat dobivamo <u>unijom</u> ovih parcijalnih rakljućaka:



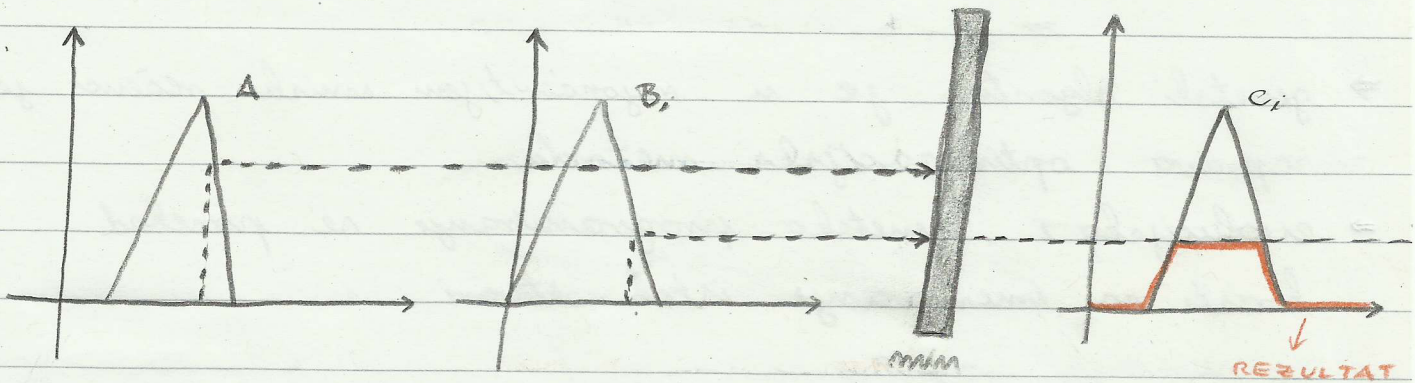
↳ izabrali smo umjesto zbroj po pravilu parcijalnih rakljućaka vidimo da su oni dobiveni implikacijom lokalnog djelovanja

↳ ovo ćemo implementirati u drugoj laboratorijskoj vježbi ☺

⇒ PROUČI I RAZUMI S PREZENTACIJAS

• varijante strojeva za raključivanje !

⇒ grafički prikaz raklapanja singleton vrijednost strojem za raklapanje koji se temelji na minimumu



⇒ LARSENOVA METODA ZAKLJUČIVANJA

- na slajdovima je malo pogrešno jer je ovo stroj koji se temelji na produktu, a ne na minimumu!

BITNO!

EVOLUCIJSKO RAČUNARSTVO

⇒ genetski algoritam je u najopćenitijem smislu riječi zapravo optimizacijska metoda

⇒ evolucijsko i genetsko programiranje se ponekad koriste za imenovanje istih stvari

```
Genetski algoritam {
    t = 0;
    generiraj početnu populaciju početnih rješenja P(0);
    sve dok nije zadovoljen uvjet zaustavljanja {
        t = t + 1;
        selektiraj P'(t) iz P(t-1);
        križaj jedinke iz P'(t), djecu spremi u P(t);
        mutiraj jedinke iz P(t);
    }
    ispiši rješenje;
}
```

⇒ priprema za genetski algoritam:

↳ trebamo odabrat:

a) PRIKAZ RJEŠENJA

↳ prikaz jedinke / kromosoma

↳ može biti npr. binarni, floating point, matrica, stablo, ...

b) VREDNOVANJE JEDINKI

↳ evaluacija pojedinog rješenja

↳ može dugo trajat

⇒ o operatorima genetskog algoritma:

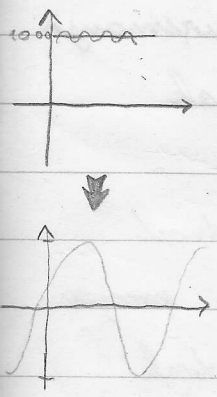
a) SELEKCIJA

↳ koristi se na dva mjesta u genetskom algoritmu:

1) odabir jedinki za reprodukciju

2) odabir jedinki za iduću generaciju

↳ ponekad; selekcija zahtjeva pripremu



(npr. $\sin(x) + 1000$ ⇒ treba oduzet 1000, tako translatirat na manje područje [-1, 1] gdje genetski algoritam bolje vidi razliku među jedinkama)

↳ vrste selekcija:

1) generacijske (biraj koje umiru)

2) eliminacijske (biraj koje odbacuješ)

↳ druga podjela:

1) proporcionalne („roulette wheel")

↳ jednostavne - vrti svaki put za odbor

↳ stohastičke - vrti samo jednom

2) k-turnirske

↳ cijel algoritam selekcije izgrađuje se korištenjem raznih operatora selekcije

↳ SELEKCIJSKI PRITISAK:

• ako je velik, onda s velikom vjerojatnošću biva one bolje jedinke

↳ cilj je imati selekcijski pritisak, ali što je manji moguć, kako bi išli prema rješenju, ali ne prebrzo kako neb zaglavili u lokalnom optimumu

↳ mjerenje selekcijskog pritiska:

1) trajanje preuzimanja

↳ ostavimo samo operator selekcije i izračunamo prosječan broj generacija nakon kojih se populacija sastoji od N duplikata najbolje jedinke

↳ manje trajanje preuzimanja znači veći selekcijski pritisak

MANJE TRAJANJE PREUZIMANJA ⇓ VEĆI SELEKCIJSKI PRITISAK

2) selekcijska razlika

↳ razlika prosječne vrijednosti dobrote preživjelih jedinki i prosječne vrijednosti dobrote svih jedinki:

$$s(t) = \overline{d_p}(t) - \overline{d}(t)$$

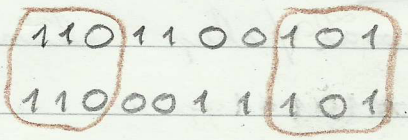
VEĆA SELEKCIJSKA RAZLIKA ⇓ VEĆI SELEKCIJSKI PRITISAK

b) REPRODUKCIJA

↳ ponekad se misli na samo križanje, a u većini slučajeva se misli na križanje i mutaciju kada govorimo o operatoru reprodukcije

↳ ovaj operator ovisi o prikazu rješenja jer treba izbjeći nemoguća rješenja

↳ PR: Nemoguća križanja smisla jer mogu dati iste jedinke:



ova dijelova su ista, pa križanje ovdje daje iste jedinke (stoga valja paziti na izbor točke križanja)

↳ mutacija je jednostavna:

- za binarni kromosom, imamo vjerojatnost „p" s kojom mutiramo svaki bit
- za realne brojeve, mutaciju radimo tako da svaki broj s vjerojatnošću „p" izbrišemo i umjesto njega nasumično iz nekog intervala

TEOREM SHEME I
HIPOTEZA BLOKOVA

⇒ ovdje je dano objašnjenje zašto uopće jednostavni genetsk algoritam radi

⇒ TEOREM SHEME:

- **shema** - uzorak / skup sličnih rješenja

*101*1

↓ shema 4 rješenja

010101
010110
110101
110111

- ako imamo „n" bitova, postoj 3^n različitih shema jer na svakom mjestu može biti 0, 1 il *
- **red sheme:**
  - ↳ broj fiksnih gena
  - ↳ oznaka: $o(S)$
- **dulјina sheme:**
  - ↳ udaljenost između prve i zadnje nepromjenjene znamenke
  - ↳ oznaka: $\delta(S)$

⇒ napišimo sada teorem sheme:

$$N(S, t+1) \geq N(S, t) \frac{\overline{D_s}}{\overline{D}} \cdot \left[1 - \frac{\delta(S)}{m-1} p_c - o(S) p_m \right]$$

SELEKCIJA KRIŽANJE MUTACIJA

⇒ prosječna dobrota jedinki sheme koje jesu u populaciji u omjeru sa prosječnom dobrotom svih jedinki u populaciji je brzan broj?

↳ kovar o vjerojatnosti uništava jedinke iz sheme

⇒ mutacija kvari shemu, pa trebamo vjerojatnost da se shema pokvari

* 1 { 0 { 1 } * } 0

tu se shema kvari na 4 moguća mjesta

⇒ također, mutacija sa svojom vjerojatnošću kvari početnu shemu

↳ pa, to uvrstimo u formulu

⇒ broj jedinki koje sadrže shemu niskog reda, raste eksponencijalno

⇒ napišimo hipotezu građevnih blokova:

- genetski algoritam pretražuje prostor rješenja mrežući sheme niskog reda

⇒ pogledaj primjer decepcijskog problema na slajdovima

↳ to su problemi na kojima genetski algoritam naglovljuje u lokalnom optimumu

PR: Bitni registar koji vraća broj jedinica, no za sve nule vraća 2^n !

↳ školski decepcijski problem

POSTAVLJANJE PARAMETARA GENETSKOG ALGORITMA

⇒ djelotvoran genetski algoritam ima uravnoteženo slučajno i usmjereno pretraživanje

↳ ovo je optimizacija hiperparametara koje treba pametno postaviti

↳ stoga, trebamo eksperimentirat s parametrima i gledat što se događa sa krivuljom genetskog algoritma

⇒ PREPORUKE ZA POČETNE VRIJEDNOSTI:

$M \approx 50\%$

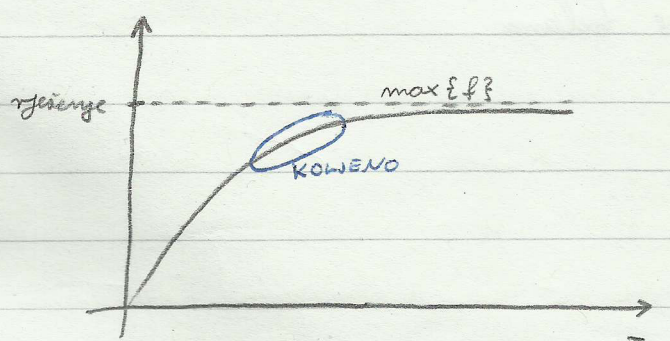
$p_m \approx 1\%$

veličina_populacije $\in [20, 100]$

$k = 3$

broj_evaluacija $\in [10^5, 10^7]$

⇒ genetski algoritam asimptotski teži k maksimumu funkcije s povećanjem broja iteracija:



ideja podešavanja parametara:

- podesi parametre da se pomaknes malo desno od koljena

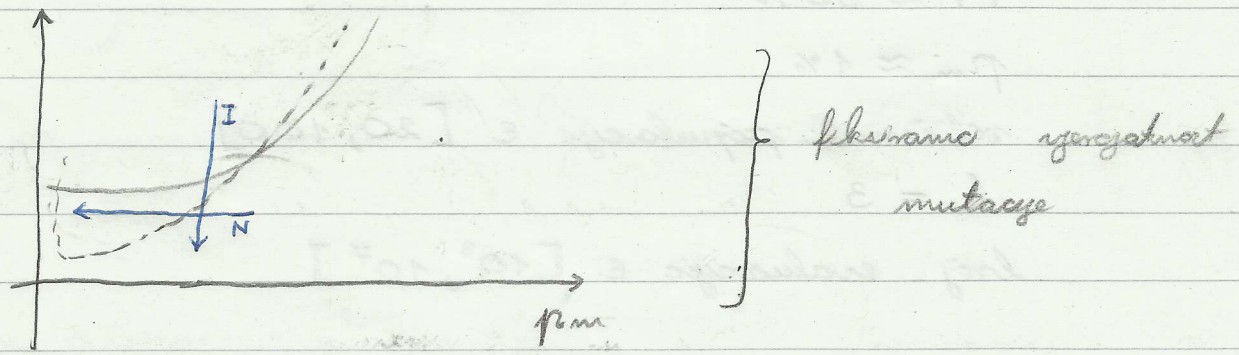
⇒ ELITIZAM: čuvanje skupa najboljih jedinki prilikom prelaska u novu populaciju

↳ npr. k-turnirska selekcija inherentno čuva k-1 najbolju jedinku, pa je rata korisna

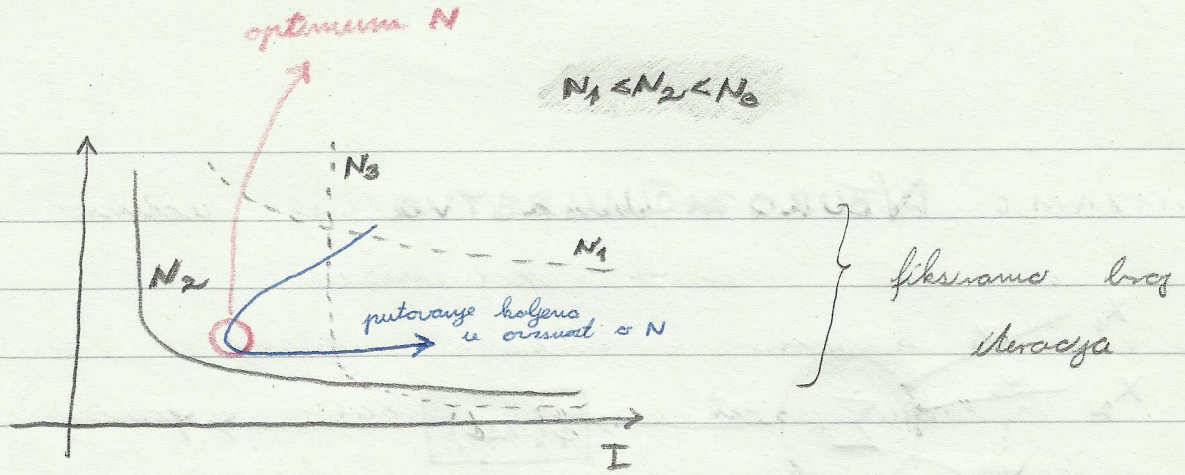
↳ elitizam eliminira rubne na hrvulj genetskog algoritma

⇒ najmanji broj parametara koje možemo imati je 3 kod osnovnih genetskih algoritama

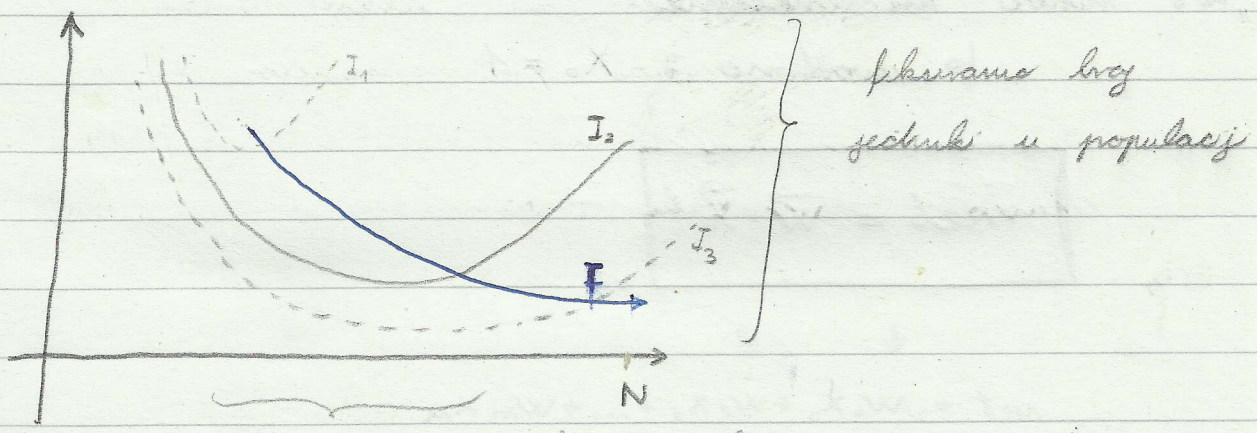
↳ njihovu osjetljivost promatramo tako da jednog mijenjamo, a gledamo što se događa s preostala dva



dno je lijevo jer ako nam je velika populacija, a već slučajno generiramo više jedinki, pa ne trebamo veliku mutaciju



veličina kolijena koja je prije bila bi trebali napraviti što manje iteracija



povećanjem broja jedinki u populaciji dobivamo veći broj potrebnih iteracija da se dosegne optimum, al je optimum bolji!

⇒ "ako povećavate N, I, trebamo smanjiti p_m"

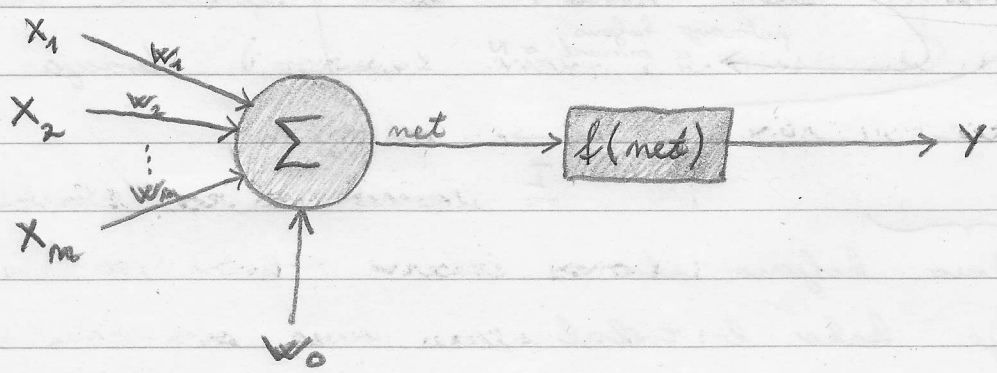
↳ generalno pravilo?

↳ vrijedi, obratno?

⇒ NAPOMENA: par na trajanje evaluacije rješenja!

(po potrebi paralelizirati evaluaciju ili čak i cijeli algoritam)

NEURORAČUNARSTVO



- težine matricama, pa idemo na to da se i „w_0" množi sa nečime:
  - ↳ uvodimo: $x_0 = 1$

$$net = \vec{w} \cdot \vec{x}$$

$$\downarrow$$

$$net = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m$$

- „f" je prijenosna funkcija i obično se koriste ove:
  a) step funkcija
  b) linearna funkcija
  c) sigmoida

⇒ NAPOMENA: ako samo slažemo linearne neurone, svaki računa težinsku sumu, a težinska suma težinske sume je opet težinska suma, pa nismo dobili ništa složenij model (zato trebamo ne-linearne prijenosne funkcije)

BITNO!

⇒ TIPIČNA GRADA:

- ulazni sloj (obično identitet)
- skriveni slojevi (razne funkcije)
- izlazni sloj (ovis o primjeni:
  - ↳ sigmoida za klasifikaciju
  (onaj izlazni neuron zadužen za tu klasu daje najveću vjerojatnost)
  - ↳ identitet za predviđanje realnih vrijednosti (npr. regresija)
)

⇒ UČENJE NEURONSKE MREŽE:

- na temelju izlaza i željenog izlaza, izračunaj pogrešku i modificiraj težine na temelju te pogreške

⇒ NAPOMENA:

- više neurona u sloju je manje ekspresivno od više slojeva neurona
  - ↳ da bi dobili bolje rješenje s više slojeva (rješenje koje generalizira), trebamo više podataka za učenje da bi ukrotili prenaučenost

⇒ matematički je pokazano da se „gotovo" svaka funkcija može naučiti neuronskom mrežom s jednim skrivenim slojem

↳ no, taj teorem ništa ne govori o sljedećem:

a) koliko neurona u skrivenom sloju uopće trebamo?

b) koje aktivacijske funkcije neuron u skrivenom sloju trebamo imati?

(pokazano je da same sigmoide nisu dovoljno dobre) //

UČENJE NEURONSKE MREŽE

⇒ UČENJE GRADIJENTNIM SPUSTOM:

↳ mičemo se u smjeru suprotnom od gradijenta funkcije pogreške

↳ naravno, gradijent funkcije pogreške računamo po težinama $\vec{w}$ jer njih jedino možemo mijenjati

↳ znači, težine mijenjamo u smjeru negativnog gradijenta

⇒ **BACKPROPAGATION:**

↳ ovo ćemo koristiti, trebat će znat izvesti korak po korak

↳ ulaz: skup uzoraka za učenje

↳ definiramo **funkciju pogreške:**

$$E = \frac{1}{2N} \sum_{\Delta=1}^{N} \sum_{o=1}^{m} \left(t_{\Delta,o} - Y_{\Delta,o}^{(k+1)} \right)^2$$

$o \Rightarrow$ „output neuron iterator" $k \Rightarrow$ iterator skupa

$\Delta \Rightarrow$ „sample iterator"

$t_{\Delta,o} \Rightarrow$ „true output"

$Y_{\Delta,o} \Rightarrow$ „calculated output"

↳ E je funkcija:

a) težina $\vec{w}$

b) skupa uzoraka za učenje

↳ stoga, E je zapravo:

$$E(\vec{w}; D)$$

↳ no, D je za nas konstanta jer nećemo mijenjat skup za učenje sve dok se mreža s danim težinama ne ponaša dobro

↳ mi želimo minimum po težinama jer tako dobijemo mrežu s težinama koja se najbolje ponaša na skupu uzoraka za učenje

↳ promotrimo težinu w_{ij} u izlaznom sloju:

$$\frac{\partial E}{\partial w_{ij}^{(k)}} = \ldots = -\frac{1}{N}\sum_{\Delta=1}^{N}\sum_{o=1}^{m}\left(t_{\Delta,o} - y_{\Delta,o}^{(k+1)}\right)\cdot\frac{\partial y_{\Delta,o}^{(k+1)}}{\partial w_{ij}^{(k)}}$$

↳ primjetit da izlaz ovisi samo o svoj težini, pa „$y_{\Delta,j}$" izlaz ovisi samo o $w_{ij}^{(k)}$; a ostalo su konstante za derivaciju po $w_{ij}^{(k)}$, pa su to nule:

$$\sum_{o=1}^{m}\frac{\partial y_{\Delta,o}^{(k+1)}}{\partial w_{ij}^{(k)}} = \frac{y_{\Delta,j}^{(k+1)}}{\partial w_{ij}^{(k)}}$$

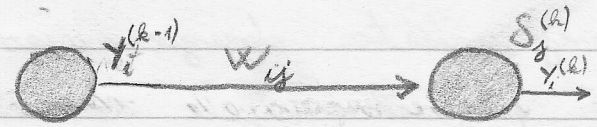
↳ na slican način izvodimo $y_{\Delta,j}$ preko ulaznih slojeva (samo promatramo „net" umjesto $y_{\Delta,o}$) te time dobivamo:

$$\frac{\partial E}{\partial w_{ij}^{(k)}} = -\frac{1}{N}\sum \delta_{\Delta,j}^{(k+1)} \cdots\cdots$$

} VIDI KNJIGU!

↳ nadalje, trebamo invest korekciju težina za unutarnje slojeve

- ovo detaljno prođi u knjizi (nešto je složenije nego za izlazne neurone, ali si puno pomogneš ako shvatiš što je nula (što ovisi o čemu kada se derivira))



↓

derivacija pogreške po ovoj težini je „ova delta puta ovoj izlaz"

$$\frac{\partial E}{\partial w_{ij}} = \delta_j^{(k)} \cdot y_i^{(k-1)}$$

↓

koliko slijedeći gnjež, puta moj izlaz ∇.

⇒ ovo sve moramo napravit za sve primjere iz D, pripadne parcijalne derivacije po promjenama zbrojimo

↳ tek tada imamo pravi gradijent i onda radimo update težina ∇.

- ovo je BATCH varijanta (sporo je jer za 100000 uzoraka imamo 100000 forward i backpropagationa za 1 update težina ∇.)

↳ možemo koristit STOHASTIC varijantu, al je ona loša aproksimacija gradijenta, pa se danas koriste male grupe primjera (MINI-BATCHERS)

↳ dodatni problem stroge BATCH varijante je mogućnost zaglavljivanja u lokalnim optimumima jer funkcija greške umjetničke mreže ima malo plitkih optimuma!

⇒ znači, Backpropagation nije ništa drugo nego nego minimizacija funkcije pogreške cijele mreže gradijentnim spustom

↳ jer smo stavili sigmoidu u mrežu, možemo dobit gradijent - funkcije pogreške (u zatvorenoj formi)

⇒ NAPOMENA: implementacija backpropagation algoritma treba primat listu - lista i tako ove unutarnje liste zapravo predstavljaju „mini - batchere"

↳ za veličinu unutarnje liste jednaku 1, imamo stohastički backpropagation

↳ za veličinu unutarnje liste jednaku cijelom skupu za učenje imamo strog backpropagation jer mijenjamo težine nakon cijelog skupa za učenje

NEURO-FOZZY SUSTAVI

⇒ koja spoj neizrazitog upravljanja i neuroračunarstva?

- neuronska mreža je dobra jer je robusna i dobro može učiti iz podataka
- neizrazito računarstvo je dobro jer je interpretabilno, dok je neuronska mreža crna kutija

↓

želimo ovo spojiti!

⇒ mi ćemo se baviti modelom spoja neizrazite logike i neuronskih mreža koj se zove ANFIS

⇒ RAZLIČITE VRSTE ZAKLJUČIVANJA:

TIP 1 zaključak / konsekvent mora imati monotonu funkciju pripadnosti (mora biti invertibilna) kako bi postojao samo jedan element zaključka

↳ za svak zaključak imamo težinu s kojom smo sigurni da je taj zaključak ispravan, pa za finalni zaključak radimo težinski prosjek

TIP 2 zaključivanje odsjecanjem je standardno kao u prvom sustavu

↳ lošije od tipa 1 po tome što ne moramo raditi defuzzifikaciju na način da tražimo po cijeloj domeni!

TIP 3 zaključak svakog pravila je oblika
o nekoj proizvoljnoj funkciji (tu funkciju
mi dajemo za svako pravilo)

↳ tu imamo veliku slobodu modeliranja

↳ za svako pravilo imamo:

1) smjeru u kojoj je zadovoljen
antecedent pravila

2) vrijednost proizvoljno definirane
funkcije

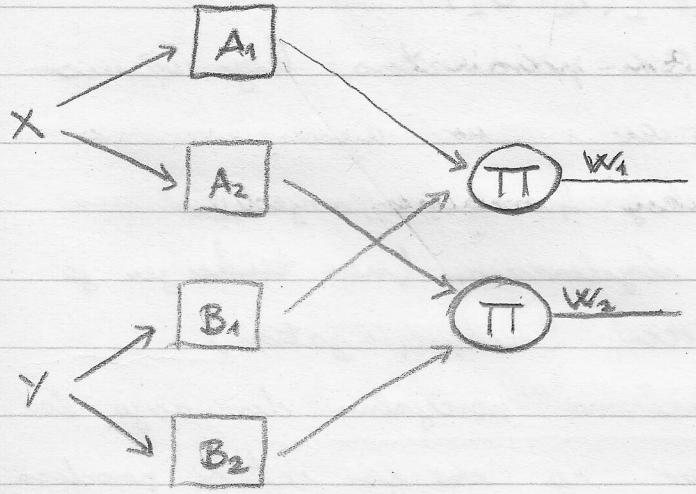
↳ ovaj tip nam je super za
neuronsku mrežu jer lagano može biti
ugrađen u neuronsku mrežu, a koeficijente
pravila će naučiti neuronskom mrežom

⇒ KORISTAN MATEMATIČKI ZAKLJUČAK:

• svaku funkciju možemo aproksimirati na
proizvoljno malu pogrešku dovoljnim brojem
linearnih funkcija

⇒ ANFIS mreža (raključivanje tipa 3):

- ulaznom sloju damo oblik odabrane funkcije pripadnosti (do na parametre, a te parametre će mreža učiti)
- obrnuti neuroni na slici nemaju parametre, već samo računaju neke stvari (oni su fiksirani).
  ↳ na tim obrnutim neuronima trebaju biti S i T norme, no treba ureći nešto što nam je lagano derivirati kako bi ovo mogli napasti s backpropagation algoritmom!
  ↳ stoga, u najvišoj slučajevima ćemo koristiti algebarski produkt i algebarsku sumu!



⇓

ako želimo veći kapacitet modela, moramo dodati više pravila (pazi na prenaučenost!)

⇒ učenje ANFIS sustava (učenje tipa 3)

↳ pojednostavljenje:

• za konsekvente pretpostavljamo konstante (prvo pravilo ⇒ r_1, drugo pravilo ⇒ r_2, ⋮ } „r"-ove učimo

• premise izpitujemo samo jednu varijablu:

↳ stoga imamo „m" raznih pravila:

R_i: Ako x je A_i tada z je $const_i$ (neka konstanta)

↳ dodatno, funkcije pripadnosti skupovima $A_1, \ldots, A_m$ modeliramo eksponencijalnom zvonolikom krivuljom (vidi na slajdovima!)

↳ definiramo pogrešku za „k"-ti uzorak:

$$E_k = \frac{1}{2}(y_k - o_k)^2$$

↳ imamo ukupno 3m parametara koje učimo (vidi raščlanjeno!)

↳ za ažuriranje nekog parametra vrijedi:

• vidi se slajdovima (nije teško jer je iste ideje kao i za backpropagation)

↳ primjetit da ne trebamo parcijalnu derivaciju loš za svaki parametar jer su one sve istih porodica (mislim da imamo samo 3 različita oblika)

$$z_i(t+1) = z_i(t) + \eta (y_k - o_k) \frac{\perp_i}{\sum_{j=1}^{m} \perp_j}$$

↓

to je za sve „z"-ove, pa za „a"-ove i „b"-ove izvodimo analogno!

↳ na slučaj kompliciranijih pravila (imamo složeniji antecedent):

$$\underbrace{X \text{ je } A_i}_{\alpha_i} \quad I \quad \underbrace{Y \text{ je } B_i}_{\beta_i} \quad \text{tada} \quad Z \text{ je } Z_i$$

$$w_i = t(\alpha_i, \beta_i)$$

$$a_i \xrightarrow[\text{na}]{\text{utječe}} \alpha_i \xrightarrow[\text{na}]{\text{utječe}} w_i \xrightarrow[\text{na}]{\text{utječe}} o_k$$

↳ stoga:

$$\frac{\partial E}{\partial a_i} = \frac{\partial E_k}{\partial o_k} \cdot \frac{\partial o_k}{\partial w_i} \cdot \frac{\partial w_i}{\partial \alpha_i} \cdot \frac{\partial \alpha_i}{\partial a_i}$$

} PRAVILO ULAZNOG SLOJA (lijevo!)

⇒ napomena:

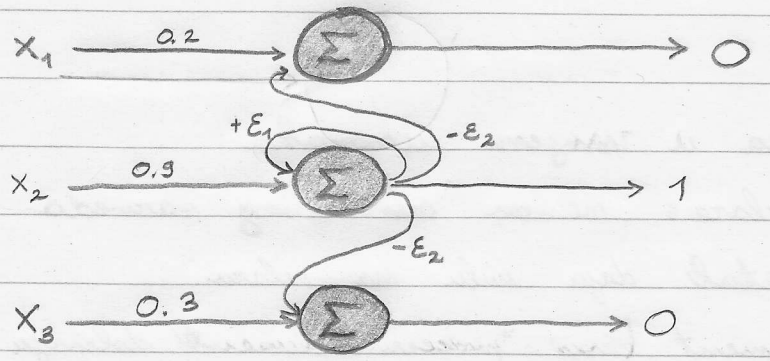
- nacrtaj sl. pravila koje se uči
- imamo ponovo X, Y i Z
  - ↳ nacrtaj pogrešku sustava nakon što se uči

SAMO-ORGANIZIRAJUĆE NEURONSKE MREŽE

⇒ UČENJE S UČITELJEM: („Supervised Learning")
↳ regresija i klasifikacija
↳ podaci su u paru „ulaz - izlaz"

⇒ UČENJE BEZ UČITELJA: („Unsupervised Learning")
↳ dani su podaci bez specificiranog preslikavanja koje će mreža morati naučiti neuronskom mrežom
↳ mreža obavlja grupiranje (engl. „clustering")
↳ ovdje spada i potporno učenje jer se dobiva povratna informacija (engl. „feedback") puno češće nego kod učenja s učiteljem
• npr. feedback nakon svake evaluacije
• korisno za mreže koje upravlja likom iz igrice te dobiva feedback nakon svakog koraka

⇒ **NATJECANJE:**
↳ svaki neuron sebe potiče da generira što veći izlaz, a druge neurone mora gušiti (smanjiti njihove izlaze
↳ cilj je modelirati natjecanje koje opažamo u prirodi



PRIKAZANO SAMO ZA JEDAN NEURON:

↳ ε_1 je težina kojom neuron sebe potiče (povratna težina)

↳ ε_2 je težina kojom neuron guši ostale neurone

WINNER - TAKES - ALL

↳ izlaz je jedan tamo gdje je ulaz maksimalan (ostala su nule)

⇒ ZAHTJEVI NA SAMOORGANIZIRAJUĆE NEURONSKE MREŽE:

1) težine u neuronu moraju biti predstavnici razreda uzoraka, tako da svaki neuron predstavlja drugi razred

2) ulazni uzorak predočava se svim neuronima i svaki neuron je izlazni - vrijednost izlaza je mjera sličnosti između ulaznog uzorka i uzorka pohranjenog u neuronu

3) koristi se **natjecateljska strategija** koja odabire neuron s najvećim izlazom

4) postoje metode potiskanja najvećeg izlaza

⇒ TIP NATJECANJA:

a) čvrsto natjecanje

↳ uči samo pobjednik (ova se dribla koristi)

b) meko natjecanje

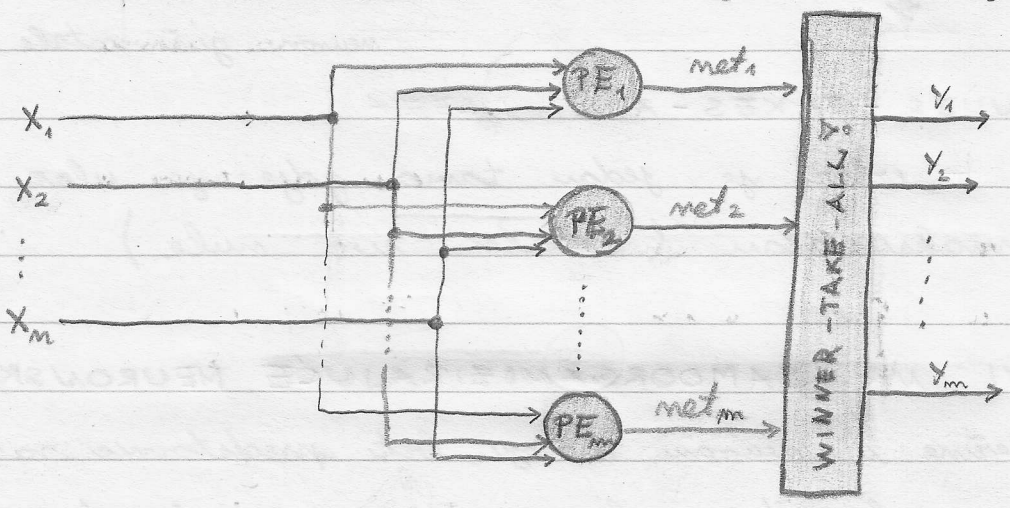
↳ postoji samo jedan pobjednik, al u nekoj mjeri uči i njegova okolina

⇒ INSTAR MREŽA:

↳ rad se kovekcija u smjeru uzoraka

↳ želimo binarne izlaze: neuron odredenog razreda daje jedinicu, a ostali daju nulu na izlazu

↳ PE ⇒ procesni element (svi procesni elementi dobivaju iste inpute, ali se natječu za svoj output)



↳ „net$_i$" je tim veći što uzorak više pripada razredu i^{th}

↳ onaj neuron (PE) koji je pobjednik za dani uzorak, modificira svoje težine tako da postane sličnije predočenom uzorku ⇒ TAKO INSTAR MREŽA UČI!

↳ stoga, pravilo učenja je:

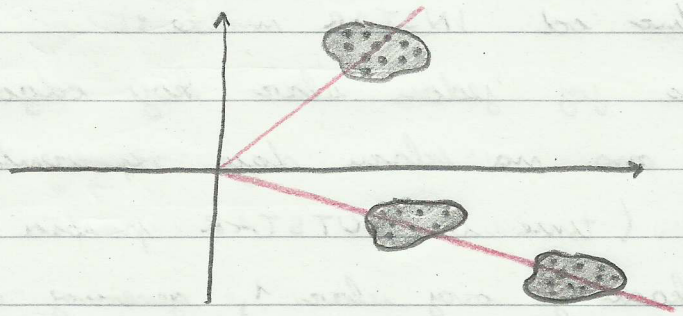
$$w_{ij} \leftarrow w_{ij} + \eta (x_i - w_{ij}) \cdot y_j$$

($y_j = 1$, ako je „j"-ti neuron pobjednik)

↳ mjera sličnosti:

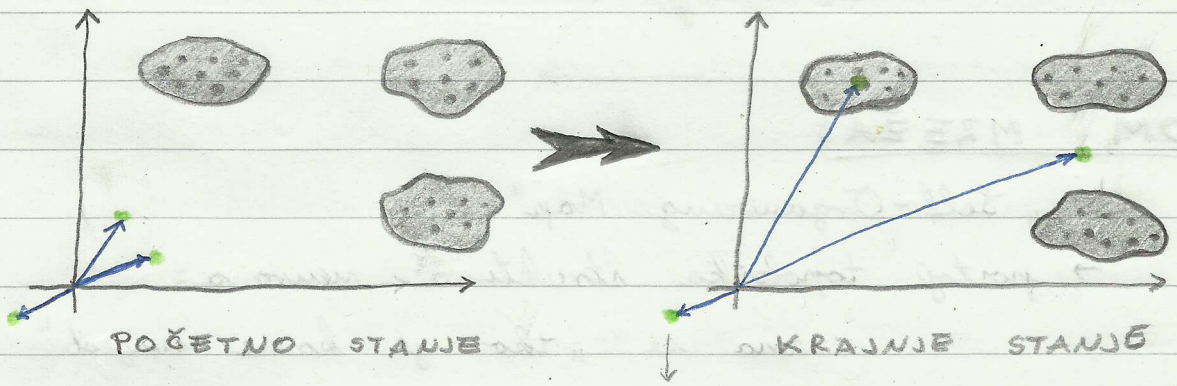
- može se uzeti kosinus kuta ili neka mjera izvedena iz udaljenosti vektora
- sam kosinus nije dobar baš uvijek jer se uzorci preslikavaju na jedinične kvadrante, pa

duže grupe uzoraka mogu postati jedva:



ove duže grupe
dozu ist kosinus, pa
će sličnost bit ista
za uzorke iz obje grupe

↳ loša inicijalizacija može dovesti do toga da jedan neuron uvme obje grupe i pozicionirat se na sredinu grupa, a drug neuron će ostat uvjeti ⇒ MRTAV NEURON



mrtav neuron - to ne želimo!

↳ mrtvi neuron se sprječavaju dodavanjem svojstva koje nazivamo SAVJEST:

- ako uzek pobjeduje, neuron počne pet savjest i on smanjuje svoju sličnost da drug neuron dobe šansu

↳ pretpostavit da je INSTAR mreža napravo dekoder jer na ulaze stavimo broj (binarni), a izlaz je jedinica na mjestu koje predstavlja taj broj

⇒ OUTSTAR MREŽA:

↳ radi suprotno od INSTAR mreže:

- damo joj jedan ulaz koj odgovora nekoj klasi, a ona na izlazu daje reprezentanta tog razreda (time se OUTSTAR ponaša kao koder)

↳ učenje: „kada je ovaj ulaz 1, generiraj ovaj podatak"

↳ težine sada pamte pojedine komponente uzorka kojeg želimo pohraniti (vidi objašnjenje na slajdu!)

↳ formula učenja:

$$w_{ij} \leftarrow w_{ij} + \eta(y_j - w_{ij})x_i$$

⇒ SOM MREŽA:

↳ „Self-Organizing Map"

↳ postoj topološka struktura neurona:

- zna se „tko je kome susjed"

↳ očekujemo da će susjedni neuroni biti reprezentanti susjednih razreda u ulaznom prostoru

- time dobivamo ekstra informaciju

↳ zbog topologije, ne uči samo pobjednik nego i cijelo njegovo susjedstvo (MEKO UČENJE)

- vidi pravilo učenja na slajdu!

↳ najčešće topologije:

a) lanac

b) mreža

NEURO-EVOLUCIJSKI SUSTAVI

⇒ zašto želimo takav spoj grana računarstva?

- jer postoj jako puno varijacija neuronskih mreža koje imaju varijacije algoritme učenja
- jer gradijentni postupci (npr. Backpropagation koj minimizira funkciju pogreške) pretpostavljaju strukturu neuronske mreže i pretpostavljaju deriviabilnu funkciju pogreške te prijenosne funkcije
- visoka više-modalnost neke funkcije pogreške može uzrokovati da algoritam učenja jako brzo zapne u lokalnom optimumu
- puno hiperparametara koje pogađamo:
  - ↳ broj neurona po sloju
  - ↳ broj slojeva
  - ↳ prijenosne funkcije po slojevima/neuronima
- dodatno, izabrano pravilo učenja ne mora biti najbolje (najefikasnije), pa možemo genetski algoritam upotrijebiti za učenje algoritma učenja

⇒ NAPOMENA: učenje hiperparametara može biti jako sporo pomoću korištenja evolucijskih algoritama (ograničeni smo hardverom)

⇒ sada ćemo pogledati varijante stvari koje se uče!

⇒ EVOLUCIJA (UČENJE) TEŽINSKIH FAKTORA:

↳ klasični algoritmi temeljeni na gradijentu idu prema najbližem optimumu jer tamo pokazuje negativni smjer gradijenta (stoga, globalni optimum dobivamo ako nam se posreći)

• otpornost na lokalni optimum dobivamo stohastičkom procjenom gradijenta, faktorom inercije, ...

↳ **svojstvo simetričnosti**: postoj mnoštvo različitih neuronskih mreža koje imaju različite težine, a obavljaju iste matematičke funkcije (na iste ulaze daju iste izlaze)

↳ svojstvo simetričnosti može biti problem prilikom evolucije jer iz **dva dobra roditelja vrlo vjerojatno dobivamo loše dijete**

• ovo se može riješiti tako da izbacimo operator križanja, već koristimo samo mutaciju (**algoritmi evolucijske strategije**)

↳ kako bi ubrzali rad genetskog algoritma, njega „hibridiziramo" s nekim algoritmom koj nije algoritam slijepe pretrage (npr. Backpropagation jer koristi gradijent)

• problem je to što imamo još dodatnih parametara (kako odlučit kada primjenit Backprop)

⇒ EVOLUCIJA (UČENJE) ARHITEKTURA:

↳ tipično se pretražuju porodice arhitektura

↳ **heuristički algoritmi**: („AKO OVO, ONDA ONO!")

• IZGRAĐUJUĆI - kreću od nepotpune mreže uče te temeljem praćenja funkcije pogreške (pada li dovoljno brzo?) dodaju neurone i slojeve neuronske mreže

• RAZGRAĐUJUĆI - kreću od opće mreže, pa uklanjaju nepotrebne dijelove

↳ **mjerenje dobrote arhitekture**:

• jednostavnost mreže

• brzina učenja nekim klasičnim algoritmom

• negativna vrijednost generalizacijske pogreške

• kombinacije ovih i drugih mjera

↳ **implementacija evolucije arhitektura**:

- svaki kromosom je jedna arhitektura (KODIRANJE!)
- radi sljedeće:

① generiraj početnu populaciju mreža

② dekodiraj svaki kromosom u mrežu

③ svaku mrežu treniraj **puno** puta

④ temeljem kombinacije rezultata treniranja dodijeli svakoj mreži dobrotu

⑤ biraj roditelje i generiraj djecu

↳ **načini kodiranja kromosoma**:

a) **direktno** - sve se upisuje (iznos svake težine)

b) **indirektno** - grubi nacrt mreže

ako direktno kodiramo, ne trebamo korak ③ (težine se naučene)

VIDI SLAJDOVE ZA PRIMJERE NAČINA KODIRANJA!

⇒ EVOLUCIJA (UČENJE) PRAVILA UČENJA:

↳ cilj je doći do algebarskog izraza koji mijenja težine neuronske mreže

↳ bilo je pokušaja, ali ne jako uspješnih

↳ ne treba znati, samo znaj da postoji

NEURO-FUZZY-EVOLUCIJSKI SUSTAVI

⇒ u okviru ovakvih sustava, nećemo koristiti genetski algoritam (iako možemo), nego koristimo **algoritam roja čestica** (PSO)

⇒ ALGORITAM ROJA ČESTICA:

↳ evolucijski populacijski algoritam

↳ rješenja ravnamo česticama

↳ svaka čestica pamti svoju trenutnu poziciju (tj. prijedlog rješenja) te najbolju ikad pronađenu poziciju

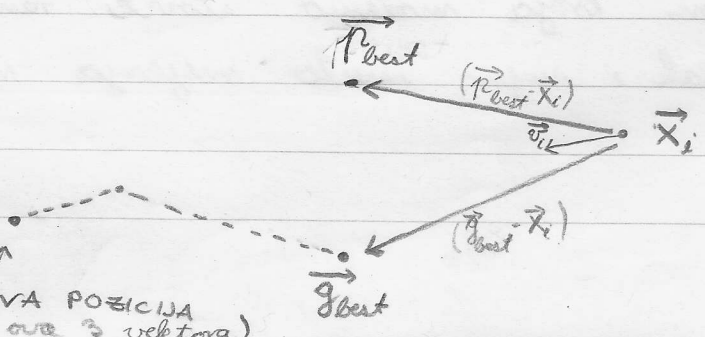
↳ algoritam iskorištava populaciju kroz socijalnu komponentu:

· za određivanje nove pozicije, čestica može imati pristup najboljoj viđenoj poziciji čitavog roja („**global-best**") ili najboljoj viđenoj poziciji svojeg susjedstva („**local-best**")

PR: Pomicanje čestice $\vec{x}_i$ na temelju „global-best" principa:

↳ NAPOMENA: čestica ima svoju brzinu $\vec{v}_i$

$\vec{p}_{best}$ ⇒ personal best

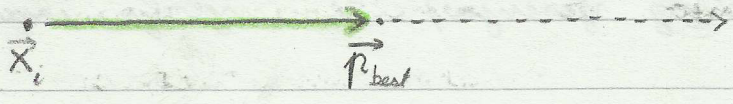


↳ vrlo često se vektor $(\vec{p}_{best} - \vec{x}_i)$ i $(\vec{g}_{best} - \vec{x}_i)$ množe s konstantama (skalirosu se) te je ta konstanta najčešće 2:

$c_1(\vec{p}_{best} - \vec{x}_i)$ $(c_1 = 2)$

$c_2(\vec{g}_{best} - \vec{x}_i)$ $(c_2 = 2)$

↳ nadalje, sada se oba ova izraza množe sa slučajnim brojem nekog proizvoljnog očekivanja (najčešće uzimamo slučajan broj očekivanja 0.5 kako bi pomnožen s konstantom koja je najčešće jednaka 2, dobili upravo slučajan vektor s očekivanjem jednakim $(\vec{p}_{best} - \vec{x}_i)$ odnosno $(\vec{g}_{best} - \vec{x}_i)$)



$2(\vec{p}_{best} - \vec{x}_i)$

⇓

ako pomnožimo ovo s random brojem između 0 i 2 očekivanja 0.5, dobivamo neki vektor manji od $2(\vec{p}_{best} - \vec{x}_i)$, a očekujemo upravo vektor $(\vec{p}_{best} - \vec{x}_i)$

↳ dodatno, ako želimo veću stohastičnost, umjesto random broja možemo staviti random vektor $(\vec{r})$, pa se čak i smjer malo mijenja na slučajan način

⇒ ALGORITAM PSO ZA ANFIS:

↳ svi parametri iste vrste se drže u zasebnoj kolekciji

↳ u jednoj iteraciji mijenjamo samo parametre iz jedne kolekcije - to je zato što se takvo mijenjanje jednog tipa parametra u praksi pokazalo bolje (npr. nemoj u jednoj iteraciji mijenjati i centar i širinu zvonolike krivulje jer je to jako stohastično i destruktivno - umjesto toga promjeni samo centar u jednoj iteraciji, a u drugoj samo širinu zvona)

↳ dodatna promjena je da uvedemo malo ponašanja iz genetskog algoritma:

- nakon svake iteracije, izdvoji čestica s najgorim P_{best} i zamijeni je s nekom česticom koja je dobivena selekcijom i križanjem roditelja

⇒ ALGORITAM SAPSO ZA ANFIS:

↳ iz podrojeva, random broj pravila te i taka dobic sustav ANFIS - imamo jedno pravila za svaki roj, pa možemo usporedit prolaz kroz podatke

↳ ukupno imamo N sustava ANFIS koj međusobno suradnju tako da:

- za svaki sustav ANFIS odredimo koliko je taj sustav dobar
- vidim koja pravila konstim u svakom sustavu ANFIS i na temelju dobrote tog sustava dodjeli dobrotu svim pravilima koje sam konstio u tom sustavu
- na kraju, kako bi odredio dobrotu pravila, uprosječujem dobrotu tog pravila kroz one sustave ANFIS u kojima se je to pravilo pojavilo (moramo osigurat dovoljno sustava ANFIS da svako pravilo bude izabrano barem nekoliko puta)

NEIZRAZITO GRUPIRANJE

⇒ rezultat klasičnog grupiranja je rastaviti skup uzoraka X na uniju „c" disjunktnih podskupova A
(vidi matematički zapis ovoga)

⇒ primjeti da je INSTAR mreža napravo on-line algoritam grupiranja - radimo korekciju položaja nakon svakog uzorka
↳ algoritmi koo što su k-means i ov algoritmi koje ćemo sada obraditi su BATCH - radi se korekcija nakon što smo prikazali čitav skup uzoraka za učenje

RAZRED
1 2 3
UZORAK

$$M = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

rezultat klasičnog grupiranja

$$M = \begin{bmatrix} 0.7 & 0.1 & 0.2 \\ 0.1 & 0.9 & 0 \\ 0.4 & 0.4 & 0.2 \end{bmatrix}$$

rezultat fuzzy grupiranja

↳ μ_{ij} je član matrice M na mjestu „ij"

⇒ **ALGORITAM C-MEANS:**

↳ definiramo funkciju cilja:

$$J(M, v) = \sum_{i=1}^{n} \sum_{j=1}^{c} \mu_{i,j}^{m} \, d(i,j)^2$$

- što je „m" veći, grupiranje postaje više fuzzificirano (za $m = 1$ imamo klasično grupiranje)
- $d_{i,j}$ je neka udaljenost (neka matrična norma)

$$\sqrt{\vec{x}_i^T A \vec{v}_j}$$

↳ kako bi dobili kome razredu uzorak pripada, potrebno je odredit udaljenost (sličnost) do svakog centra

- sličnost nekom centru:

$$\frac{\text{sličnost tom centru}}{\text{zbroj sličnosti svim centrima}}$$

⇓

tako dobijemo da je suma po retcima matrice M jednaka 1

↳ kad imamo mjere pripadnosti, krećemo sa ažuriranjem pozicija centara

- jedina varijabla je do sada ~~ažuriranje težinski~~:

↳ što mu više pripadaš, to ga jače privuci k sebi

↳ za zaustavljanje, najjeftinije je konstatirati neku nepromjenjivost kao minimalni uvjet pomaka centara koji se može dogoditi da bi algoritam nastavio (drugim riječima, provjeri jesu li centri „stali")

↳ kako svaki put računamo udaljenost svih primjera od svih centara, složenost je $O(c \cdot n)$

- „c" je broj centara
- „n" je broj primjera