Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

MEHKO RAČUNANJE

za modeliranje, razpoznavanje in regresijo

Andrej Dobnikar in Branko Šter

Ljubljana, september 2008

CIP - Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

510.644.4(075.8)
004.032.26(075.8)

DOBNIKAR, Andrej
Mehko računanje za modeliranje, razpoznavanje in regresijo / Andrej Dobnikar, Branko Šter. - 1. izd. - Ljubljana : Fakulteta za računalništvo in informatiko, 2008

ISBN 978-961-6209-64-7
1. Šter, Branko
241035008

Recenzenta: dr. Uroš Lotrič, doc. dr. Kukar Matjaž

Založila: Fakulteta za računalništvo in informatiko, 2008
Urednik: mag. Peter Šega

Natisnil: KOPIJA Mavrič, Ljubljana
Naklada: 200 izvodov
1. izdaja

Kazalo

Poglavje 1

Uvod

Čeprav je računalniška znanost še zelo mlada, pa je že od njenega rojstva prisotno spogledovanje z naravo in predvsem s človeškimi možgani, njihovimi strukturnimi značilnostmi in načini delovanja. Že von Neumann je ob samem nastanku klasične računalniške arhitekture razmišljal o celičnih strukturah, ki bi pohitrile delovanje in hkrati približale način delovanja živčnemu sistemu. Ne dosti pozneje je Holland kot prvi povezal Darwinovo teorijo z računalništvom s t.i. veliko zanko genetskih algoritmov in s tem odprl povsem novo področje računanja, ki sledi naravni evoluciji in ki jo je mogoče zelo uspešno uporabiti pri reševanju številnih optimizacijskih problemov. Približno v istem času je Zadeh definiral mehke množice, iz katerih se je kmalu razvila mehka logika, ki vključuje simbolično izražanje in računanje, podobno človeškemu. Čeprav so ob koncu prve polovice 20. stoletja nastajali tudi že zametki modelov za živčne celice oz. nevronskih mrež, pa se je ta teorija afirmirala šele v osemdesetih letih prejšnjega stoletja. Od tedaj pa do danes so se omenjene teorije razvijale do današnjih izjemnih razsežnosti. Poleg tega so se pojavile številne interdisciplinarne študije, ki so omenjena področja povezovala med seboj in s tem izboljševala njihovo uspešnost reševanja problemov, ki jih s klasičnimi računalniškimi prijemi ni bilo mogoče rešiti. Hibridni postopki so postali nujnost, združevati so se začele tudi panoge, ki so bile do nedavna konkurenčne, n.pr. umetna inteligenca in evolucijsko računanje, nevronske mreže ali mehka logika. Seveda je pri razvoju novih področij pomembno vlogo igrala tudi tehnologija. Danes smo priče nenavadnim situacijam, ko so kompleksna vezja bistveno cenejša od enostavnih in ko vseh sposobnosti tehnologije (n.pr. reprogramiranje v realnem času) sploh ne poznamo ali pa ne znamo do-

volj dobro izkoristiti.

Z današnjega zornega kota je težko napovedati, v katero smer bo šel prihodnji razvoj računalniške znanosti. Ne moremo reči, ali je mikroelektronska tehnologija že v zatonu in kakšne bodo v bližnji prihodnosti tehnologije nanostruktur: na bazi biologije, optike, ali kakšne nove vede. Tudi vsenavzoča prisotnost računalnikov nam ne omogoča jasnejše slike v prihodnost. Zagotovo je le, da poti nazaj ni in da bodo specializirane naloge prevzemali najboljši algoritmi in najboljše tehnologije ter da bo pri tem navzoče vsesplošno sodelovanje in komuniciranje med različnimi vedami. Prav tako se bo brez dvoma poglabljalo znanje o naravi in o ustroju ter funkcioniranju živih bitij, kar bo zagotovo bogatilo zakladnico znanja, iz katere bo največ črpalo prav računalništvo.

Knjiga mehko računanje predstavlja osnovni študijski pripomoček pri uvajanju v nova področja adaptivnih sistemov na osnovi umetnih in naravnih algoritmov. Vpeljuje nas v svet računanja, ki se zgleduje po obnašanju človeka in po načinu, kako rešuje probleme. Nove metode računanja je mogoče s pridom uporabiti pri številnih problemih modeliranja, razpoznavanja, regresije in krmiljenja. Zaradi precejšne kompleksnosti algoritmov, ki spremljajo metode računanja, je pomembno paralelno procesiranje oz. programiranje. Zato se tematika dobro navezuje na področje porazdeljenih sistemov, ki omogočajo pohitritev delovanja mnogih algoritmov in s tem povečanje njihove uporabnosti pri realnih in časovno zahtevnih problemih.

Poglavje 2

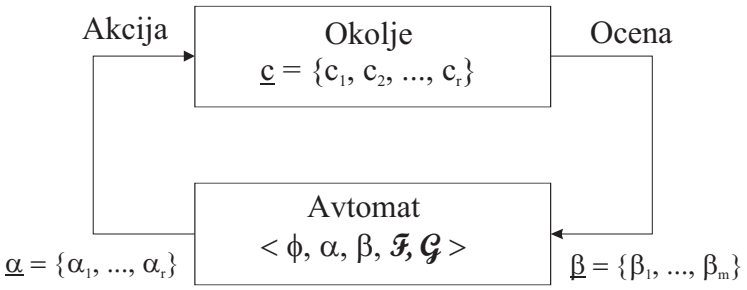
Učeči avtomati

2.1 Osnovni pojmi

Učeči avtomati (UA) [1] so posplošitev klasičnih končnih avtomatov, ki so lahko deterministični ali stohastični [2]. Učeči avtomati delujejo v neznanih stohastičnih okoljih (stacionarnih ali nestacionarnih), tako da s spreminjanjem lastnih strukturnih lastnosti težijo k izboljšanju svojega obnašanja v okolju. Okolje se odziva na akcije avtomata z oceno, ki je lahko v najpreprostejšem primeru zgolj dvovrednostna (pozitivna ali negativna). Če delovanje avtomata vodi k vedno boljši oceni okolja, potem pravimo, da se je avtomat učil oz. prilagodil (adaptiral) na okolje. Takšna adaptacija je lahko posledica strukturnih značilnosti avtomata (primer determinističnega oz. stohastičnega avtomata s fiksno strukturo), lahko pa je vzrok za adaptacijo korekcijska shema, ki sproti spreminja funkcijske lastnosti avtomata. Tedaj govorimo o stohastičnemu avtomatu s spremenljivo strukturo. Obnašanje učečega avtomata v okolju se torej s časom praviloma izboljšuje.

Učeči avtomat je definiran s petorčkom $< \underline{\phi}, \underline{\alpha}, \underline{\beta}, \mathcal{F}, \mathcal{G} >$. To je algebraični sistem, katerega elementi po vrsti pomenijo množico stanj $\underline{\phi} = \{\phi_1, ..., \phi_s\}$, množico izhodov iz avtomata $\underline{\alpha} = \{\alpha_1, ..., \alpha_r\}$, množico vhodov v avtomat $\underline{\beta} = \{\beta_1, ..., \beta_m\}$, operator prehajanja stanj $\mathcal{F} : \phi(n+1) = \mathcal{F}(\phi(n), \beta(n))$ in izhodni Moorov operator $\mathcal{G} : \alpha(n) = \mathcal{G}(\phi(n))$, kjer je n oznaka za diskretni čas. Izhodom avtomata običajno rečemo akcije, vhodu v avtomat pa ocena okolja.

Učeči avtomati običajno delujejo v okolju, s katerim skupaj tvorijo zaprt sistem, ki ga prikazuje Slika 2.1. Okolje določa $\underline{c} = \{c_1, c_2, ..., c_r\}$, to je množica



Slika 2.1: Kombinacija okolja in avtomata tvori zaprt sistem

verjetnosti kaznovanja za vse možne akcije iz izhodne množice $\underline{\alpha}$. β kot ocena (kazen) iz okolja je vhod v avtomat.

Če sta oba operatorja, $\mathcal{F}$ in $\mathcal{G}$, deterministična, je tudi avtomat deterministočen, če pa je vsaj eden stohastičen, je tudi avtomat stohastičen.

2.1.1 Deterministični avtomat

Operator prehajanja stanj $\mathcal{F}$ je običajno podan z nizom m binarnih kvadratnih matrik $\mathbf{F}(\beta_i),\ i = 1, ..., m$, reda $s \times s$, kjer je m število vhodov v avtomat in kjer vrstice matrik določajo stanja v času n, stolpci matrik pa stanja v času $n + 1$. Pri deterministočnem avtomatu (DA) enica v matriki pomeni prehod med dvema stanjema, ničla pa, da prehoda ni:

$$f_{ij}^{\beta} = \begin{cases} 1, & \text{če } \phi_i \ \rightarrow \ \phi_j \text{ pri } \beta, \\ 0, & \text{sicer}\,, \end{cases}$$

kjer je $f_{ij}^{\beta} \in \mathbf{F}(\beta)$.

Podobno je izhodni operator $\mathcal{G}$ podan z matriko $\mathbf{G}$ reda $s \times r$, kjer vrstice pomenijo stanja v času n, stolpci pa akcije v istem času. Enica v matriki pomeni prisotnost izhoda pri danem stanju, ničla pa njegovo odsotnost:

$$g_{ij}(\beta) = \begin{cases} 1, & \text{če } \mathbf{G}(\phi_i) = \alpha_j \\ 0, & \text{sicer} \end{cases}$$

V vsaki vrstici deterministične matrike je natanko ena enica, ostalo pa so ničle.

Primer
Deterministično avtomat ima 4 stanja, 2 izhoda in 2 vhoda: $\underline{\phi} = \{\phi_1, \phi_2, \phi_3, \phi_4\}$, $\underline{\alpha} = \{\alpha_1, \alpha_2\}$, $\underline{\beta} = \{\beta_1, \beta_2\}$. Operatorja $\mathcal{F}$ in $\mathcal{G}$ sta podana z matrikami:

$$\mathbf{F}(\beta_1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{F}(\beta_2) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \mathbf{G} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

2.1.2 Stohastični avtomat

Pri stohastičnem avtomatu (SA) so elementi matrik **F** in **G** verjetnosti. Avtomat velja za stohastičnega, če je vsaj eden od obeh operatorjev stohastičen. Tedaj ima vsaj ena komponenta njegove matrike (verjetnost p) vrednost med 0 in 1 ($0 < p < 1$). Tudi v tem primeru velja, tako kot pri determinističnem avtomatu, da je vsota vseh komponent v eni vrstici matrike enaka 1 (zakon totalne verjetnosti). Taki matriki rečemo tudi stohastična matrika.

Primer
Stohastičen avtomat ima naslednji matriki prehajanja stanj

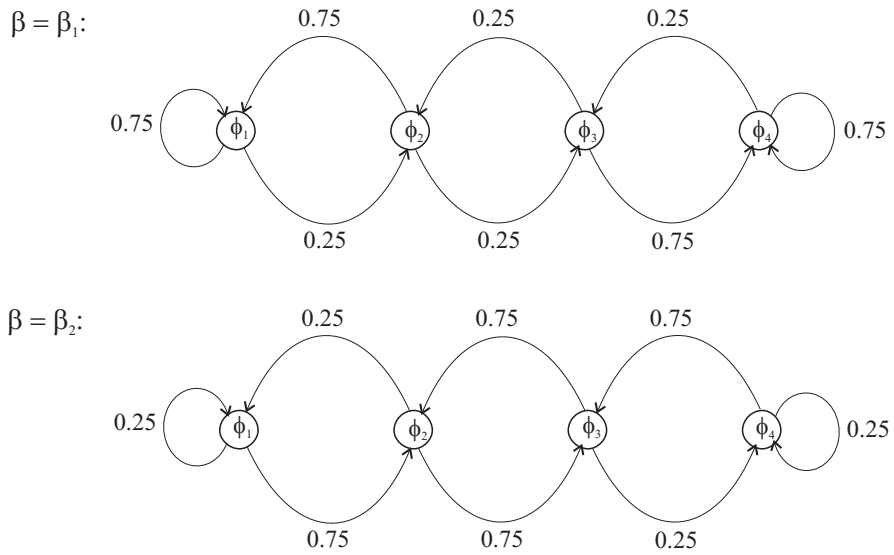
$$\mathbf{F}(\beta_1) = \begin{bmatrix} 0.75 & 0.25 & 0 & 0 \\ 0.75 & 0 & 0.25 & 0 \\ 0 & 0.25 & 0 & 0.75 \\ 0 & 0 & 0.25 & 0.75 \end{bmatrix}, \quad \mathbf{F}(\beta_2) = \begin{bmatrix} 0.25 & 0.75 & 0 & 0 \\ 0.25 & 0 & 0.75 & 0 \\ 0 & 0.75 & 0 & 0.25 \\ 0 & 0 & 0.75 & 0.25 \end{bmatrix}$$

ter izhodno matriko

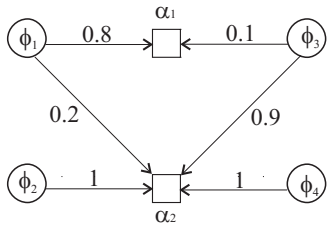
$$\mathbf{G} = \begin{bmatrix} 0.8 & 0.2 \\ 0 & 1 \\ 0.1 & 0.9 \\ 0 & 1 \end{bmatrix}$$

Operatorje prehajanja stanj lahko predstavimo tudi z diagramom, kot kaže Slika 2.2, prav tako izhodni operator (Slika 2.3).

V primeru, da so verjetnosti f_{ij} matrike **F** in verjetnosti g_{kl} matrike **G** konstantne oz. neodvisne od diskretnega časa n, govorimo o stohastičnih avtomatih s fiksnimi strukturami (SAFS). Vkolikor pa se po vsakem koraku n



Slika 2.2: Diagrama prehajanja stanj, ki ustrezata matrikama $\mathbf{F}(\beta_1)$ in $\mathbf{F}(\beta_2)$.



Slika 2.3: Izhodni diagram, ki ustreza matriki $\mathbf{G}$.

glede na vhodno akcijo α_i in odziv okolja β_j verjetnosti operatorjev $\mathbf{F}$ in/ali $\mathbf{G}$ spremenijo, govorimo o stohastičnih avtomatih s spremenljivimi strukturami (SASS).

Primer

Problemov zaprte zanke je zelo veliko, tudi v vsakodnevnem življenju. Vzemimo, da smo v vlogi avtomata in da predstavljajo naše okolje izkušnje, ki jih imamo z izbiro restavracije. Če se omejimo na izbor restavracij A in B, potem se v nekem času n odločimo npr. za A. Če je bila hrana in postrežba dobra, se bomo tudi v času $n + 1$ odločili za A. Sicer se odločimo za B. To je primer deterministočnega odločanja (DA). Če se odločimo ponovno za isto restavracijo, v primeru, da smo bili ob zadnjem obisku zadovoljni, sicer mečemo

kovanec (za izbor med A in B, $p = 0,5$), je to primer stohastičnega avtomata s fiksno strukturo, če pa spremenimo verjetnosti odločanja (o naslednjem obisku restavracije) vsakič, ko zapustimo restavracijo, v odvisnosti od kvalitete uslug (okolja), potem je to primer stohastičnega avtomata s spremenljivo strukturo.

Stohastični avtomat z deterministčnim izhodnim operatorjem

Vsak stohastični avtomat, ki ima izhodni operator stohastičen, je mogoče modificirati tako, da postane izhodni operator deterministčen, ob nespremenjeni funkciji njegovega delovanja. To dosežemo tako, da vse pare (ϕ_i, α_j) smatramo za nova stanja, ki jih označimo z $\hat{\phi}_{ij} = (\phi_i, \alpha_j)$. Na ta način število stanj naraste na $s \times r$. Pogojno verjetnost prehoda med $\hat{\phi}_{ij}$ in $\hat{\phi}_{kl}$ pri vhodu β opišemo z operatorjem $\hat{f}^{\beta}_{(ij)(kl)}$, nove matrične operatorje pa z $\hat{F}(\beta)$ in $\hat{G}$. Ni se težko prepričati, da sedaj velja:

$$\hat{f}^{\beta}_{(ij)(kl)} = g_{kl} \cdot f^{\beta}_{ik} \,.$$

in da ta verjetnost ni odvisna od izhoda α_j. Zato dobimo izhodno matriko z:

$$\hat{g}_{(ij)(l)} = \begin{cases} 1, & \text{če } j = l, \\ 0, & \text{sicer}\,. \end{cases}$$

Verjetnosti stanj in akcij

Izhodno obnašanje stohastičnega avtomata pri dani vhodni sekvenci lahko opazujemo, če so znane verjetnosti prehodov med stanji. Pogosto pa je pomembno vedeti tudi, kolikšne so verjetnosti, da se avtomat v določenem trenutku nahaja v posameznih stanjih. Takšne verjetnosti imenujemo verjetnosti stanj. Z njihovo pomočjo je mogoče podati delovanje avtomata tudi drugače. Vzemimo vektor verjetnosti stanj v času n:

$$\boldsymbol{\pi}(n) = [\pi_1(n), \pi_2(n), ..., \pi_s(n)]^T,$$

kjer je ^{T} oznaka za transpozicijo in so verjetnosti stanj podane s:

$$\begin{aligned} \pi_j(0) &= P(\phi(0) = \phi_j) \\ \pi_j(n) &= P(\phi(n) = \phi_j | \beta(0), ..., \beta(n-1))\,. \end{aligned}$$

Če je dan vhod in vektor $\boldsymbol{\pi}(0)$, potem lahko komponente vektorja verjetnosti stanj pri $n = 1$ določimo s:

$$\begin{aligned}\pi_j(1) &= P(\phi(1) = \phi_j|\beta(0)) = \\ &= \sum_{i=1}^{s} P(\phi(1) = \phi_j|\phi(0) = \phi_i, \beta(0)) \cdot P(\phi(0) = \phi_i) = \\ &= \sum_{i=1}^{s} f_{ij}^{\beta(0)} \cdot \pi_i(0)\,.\end{aligned}$$

oz. v vektorski obliki:

$$\boldsymbol{\pi}(1) = F^T(\beta(0)) \cdot \boldsymbol{\pi}(0)\,.$$

Če postopek rekurzivno ponavljamo, dobimo:

$$\boldsymbol{\pi}(n) = F^T(\beta(n-1))\, F^T(\beta(n-2)) \ldots F^T(\beta(0)) \cdot \boldsymbol{\pi}(0)\,.$$

Podobno lahko definiramo vektor verjetnosti akcij $\mathbf{p}(n)$, katerega i-ta komponenta je podana z:

$$p_i(n) = P(\alpha(n) = \alpha_i|\beta(0), ..., \beta(n-1)), \quad i = 1, ..., r\ .$$

Ob upoštevanju:

$$\begin{aligned}p_i(n) &= \sum_{j=1}^{s} P(\alpha(n) = \alpha_i|\phi(n) = \phi_j) \cdot P(\phi(n) = \phi_j|\beta(0), ..., \beta(n-1)) \\ &= \sum_{j=1}^{s} r_{ji} \cdot \pi_j(n),\end{aligned}$$

sledi še matrična oblika:

$$\mathbf{p}(n) = \mathbf{G}^T \cdot \boldsymbol{\pi}(n)\ .$$

Zadnji izraz pove, da je dovolj, če poznamo le en verjetnostni vektor. Drugega si lahko vedno izračunamo s pomočjo prvega in obratno.

2.2 Naključno okolje

V zaprtem sistemu, ki ga tvorita avtomat in okolje, prvi s svojimi akcijami vpliva na okolje, le-to pa s svojim odgovorom na akcijo posledično vpliva na

avtomat. Na Sliki 2.1 je bilo prikazano, kaj določa okolje. To je množica verjetnosti $\underline{c}$, ki ima toliko komponent, kot je akcij avtomata, torej r. Komponenta c_i, ki ustreza akciji α_i, je verjetnost, da bo okolje odgovorilo s kaznijo oz. z $\beta = 1$. Tega dogovora, da bomo kazen (angl. penalty) iz okolja zaradi akcije avtomata označevali z 1, nagrado (angl. reward) pa z 0, se bomo držali tudi v nadaljevanju. Če je množica $\underline{c}$ sestavljena iz samih ničel in enic, potem je okolje deterministično, sicer pa je stohastično. Pri popolnoma naključnem okolju so vse komponente vektorja c enake $1/r$, saj so verjetnosti za nagrado in kazen za vse akcije avtomata enake.

Čeprav avtomat vedno sprejme iz okolja le 0 ali 1, pa je v primeru determinističnega okolja odziv na posamezno akcijo vedno enak, pri stohastičnem okolju pa je odziv odvisen od ustrezne verjetnostne komponente vektorja okolja c. Če je npr. odziv okolja v vsakem trenutku naključna spremenljivka, je posledično naključno tudi prehajanje stanj avtomata in nadaljnje krmiljenje okolja. Tedaj je:

$$\begin{aligned} c &= P(\beta(n) = 1) \\ d = 1 - c &= P(\beta(n) = 0)\,, \end{aligned}$$

kjer je c verjetnost kazni in d verjetnost nagrade, in posledično:

$$\begin{aligned} \tilde{f}_{ij} &= P(\phi(n+1) = \phi_j | \phi(n) = \phi_i, \beta(n) = 0) \cdot P(\beta(n) = 0 | \phi(n) = \phi_i) \\ &+ P(\phi(n+1) = \phi_j | \phi(n) = \phi_i, \beta(n) = 1) \cdot P(\beta(n) = 1 | \phi(n) = \phi_i) \\ &= f^0_{ij} \cdot (1 - c) + f^1_{ij} \cdot c\,, \end{aligned}$$

kjer simbol $\tilde{}$ pomeni, da gre za verjetnostni operator prehajanja stanj, ki je posledica naključnega okolja, in sta f^0_{ij} in f^1_{ij} elementa matrik $\mathbf{F}(0)$ in $\mathbf{F}(1)$ deterističnega avtomata. Avtomat s stacionarno naključno sekvenco 0 in 1 na vhodu lahko torej opišemo z eno samo verjetnostno (stohastično) matriko prehodov $\tilde{\mathbf{F}}$, $\tilde{f}_{ij} \in \tilde{\mathbf{F}}$. Od tod sledi pomembna ugotovitev: *deterministični avtomat v naključnem okolju se obnaša kot stohastični avtomat s konstantnim vhodom.*

Če sedaj zamenjamo deterministični avtomat s stohastičnim avtomatom s fiksno strukturo, dobimo podoben rezultat, le da sta sedaj f^0_{ij} in f^1_{ij} konstanti na intervalu [0, 1], torej verjetnosti. Sledi naslednja pomembna ugotovitev: *stohastični avtomat s fiksno strukturo in stacionarnim naključnim binarnim vhodom je ekvivalenten nekemu drugemu stohastičnemu avtomatu s fiksno strukturo in konstantnim vhodom.*

V primeru, da stohastični avtomat s fiksno strukturo (SAFS) zamenjamo s stohastičnim avtomatom s spremenljivo strukturo (SASS), pa dobimo:

$$\tilde{f}_{ij}(n) = P(\phi(n+1) = \phi_j | \phi(n) = \phi_i) = f^0_{ij}(n) \cdot (1-c) + f^1_{ij}(n) \cdot c\,.$$

Sedaj so vse verjetnosti prehodov $\tilde{f}_{ij}$, f^0_{ij} in f^1_{ij} odvisne od diskretnega časa n. Ker se $f^\beta_{ij}(n)$ ažurirajo v vsakem koraku glede na vhodno sekvenco in ker se vsaka vhodna sekvenca pojavi z določeno verjetnostjo, je $f^\beta_{ij}(n)$ naključna spremenljivka. Zato je tudi $\tilde{f}_{ij}(n)$ naključna spremenljivka, iz česar sledi, da je vektor verjetnosti stanj $\pi(n)$ naključni vektor.

2.2.1 Markovski procesi in verige

Kadar imamo opravka s stohastičnimi procesi, le-te pogosto opisujemo s pomočjo Markovskih procesov oz. verig. Tedaj nam predstavlja stohastični proces družino naključnih spremenljivk $X(t)$, $t \in \mathbf{T}$, definiranih v prostoru stanj Ω, kjer je $\mathbf{T}$ časovni (indeksni) niz procesa, $X(t)$ pa je stanje stohastične spremenljivke X v času t. Glede na naravo niza $\mathbf{T}$ govorimo o diskretnih ali zveznih sistemih, glede na naravo prostora stanj Ω pa o stohastičnih verigah (diskreten prostor stanj) ali stohastičnih procesih (zvezen prostor stanj). Stohastične procese lahko torej klasificiramo v štiri skupine: diskretne verige, diskretne procese, zvezne verige in zvezne procese. V kontekstu učečih avtomatov bomo imeli opravka z diskretnimi verigami, saj sta v tem primeru časovni prostor in prostor stanj diskretna.

Vzemimo, da stanja $\phi_i (i = 1, 2, ...)$ diskretne Markovske verige $X(n) = X_n$ ustrezajo stanjem avtomata v stohastičnem okolju. Označimo verjetnost, da smo v času m v stanju j, z izrazom:

$$p_j(m) = P(X_m = \phi_j)$$

in pogojno verjetnost, da smo v času n v stanju ϕ_k, če smo bili v času $m < n$ v stanju ϕ_j, z izrazom:

$$p_{jk}(m, n) = P(X_n = \phi_k | X_m = \phi_j)\,.$$

Izraz $p_{jk}(m, n)$ je prenosna verjetnostna funkcija Markovske verige. V primeru q diskretnih časov $n_1, n_2, ..., n_q$ in diskretnih stanj $\phi_1, \phi_2, ..., \phi_q$, je vezana verjetnost enaka:

$$P(X_{n1} = \phi_1,\ ...\ , X_{nq} = \phi_q) = p_1(n_1) p_{12}(n_1, n_2)\ ...\ p_{q-1,q}(n_{q-1}, n_q)\,.$$

To je verjetnost sekvence stanj.

Markovska veriga je homogena v času, če je izraz $p_{jk}(m,n)$ odvisen le od $n-m$. Tedaj je prenosna verjetnostna funkcija za n korakov naprej v Markovski verigi podana z:

$$p_{jk}{}^{(n)} = P(X_{n+m} = \phi_k | X_m = \phi_j), \qquad \text{za } m \geq 0$$

Torej izraz $p_{jk}{}^{(n)}$ označuje pogojno verjetnost, da bo Markovska veriga iz stanja ϕ_j prišla v stanje ϕ_k po n korakih. V primeru $n = 1$ pišemo $p_{jk}^{(1)} = p_{jk}$. Vse možne prehode označuje matrika P, katere splošni člen je p_{ij}. Tedaj lahko pišemo:

$$\boldsymbol{\pi}(1)^T = \boldsymbol{\pi}(0)^T \cdot \mathbf{P}\,,$$

oz. splošno:

$$\boldsymbol{\pi}(n)^T = \boldsymbol{\pi}(0)^T \cdot \mathbf{P}^{(n)}\,.$$

Če je P prenosna matrika Markovske verige in če obstaja $\lim_{n\to\infty} p_{ij}{}^{(n)} = p_j^*$ za vse j, neodvisno od i, in če je $\sum_{j=1}^{\infty} p_j^* = 1$, potem je takšna Markovska veriga ergodična.

Povedano z besedami, Markovska veriga je ergodična, če je *nereducibilna* (obstaja pozitivno celo število k tako, da je $P^{(k)}$ matrika, ki nima elementov 0) in če je aperiodična (vsaj en diagonalni element v P je različen od 0, kar pomeni da je perioda 1). Drugačna definicija pa pravi, da je stohastični sistem ergodičen, če je njegovo časovno povprečje enako statističnemu, oz. če je sistem aktiven le v enem režimu delovanja.

Sekvenca stanj avtomata s fiksno strukturo s stacionarnimi naključnimi vhodi je torej Markovska veriga, katere stanja ustrezajo stanjem avtomata. Ker je tedaj matrika verjetnosti prehodov konstantna, sekvenca stanj ustreza homogeni Markovski verigi. V primeru stohastičnega avtomata s spremenljivo strukturo pa sekvenca stanj ustreza nehomogeni Markovski verigi.

2.2.2 Povezava avtomata z okoljem

Do sedaj sta bila oba sistema, avtomat in okolje obravnavana ločeno. Sedaj pa si oglejmo, kako delujeta skupaj, kot zaprt sistem. Avtomat bo s svojimi akcijami vplival okolje, to pa bo s svojimi odgovori oz. ocenami generiralo vhode v avtomat. V nekem začetnem stanju $\phi(0)$ bo avtomat generiral akcijo $\alpha(0)$. Odziv okolja na to akcijo je $\beta(0)$, ki spremeni stanje avtomata v $\phi(1)$. Postopek

se nato ponavlja, kar pripelje do zaporedja stanj, akcij in odgovorov okolja. V primeru stohastičnega avtomata s spremenljivo strukturo se verjetnostni vektor $p(n)$ ali matrika prehajanja stanj $F(\beta)$ ažurirata v vsakem časovnem koraku n. Avtomat, ki na ta način deluje v neznanem in stohastičnem okolju tako, da v nekem smislu izboljša svoje delovanje, je *učeči avtomat*.

Delovanje avtomata v okolju je mogoče ilustrirati s parom študent učitelj. Študent ustreza učečemu avtomatu, učitelj pa okolju. Vzemimo, da študent dobi vprašanje, na katerega je na razpolago končno število odgovorov (test). Študent izbere enega, učitelj pa odgovori v binarnem smislu, glede na to, ali je odgovor pravilen ali ne. Toda učitelj je nezanesljiv (zmotljiv), kar pomeni, da ima vsak odgovor verjetnost kaznovanja med 0 in 1, vendar je verjetnost kaznovanja najmanjša v primeru pozitivnega odgovora. Pod takšnimi pogoji je zanimivo poiskati način, kako naj se obnaša študent, da bo spoznal, kateri odgovor je pravilen. Cilj 'učenja' je torej spoznati optimalno akcijo (izbiro), ki ustreza pravilnemu odgovoru. Verjetnostni učitelj igra nekakšno vlogo kritika, zato se takšno učenje imenuje učenje s kritiko. V kontekstu umetnih nevronskih mrež pa se takšno učenje imenuje spodbujevano učenje (angl. reinforcement learning).

2.2.3 Norme obnašanja

Cilj učenja učečih avtomatov je torej izboljšati izbiro svojih akcij tako, da bo od okolja dobival vedno več nagrad oz. vedno manj kazni. Za objektivno ocenjevanje postopka učenja potrebujemo kvantitativne norme obnašanja učečih avtomatov v ustreznem okolju. Najprej bomo predpostavljali najpreprostejši model okolja, tak, kot je bil že omenjen, in ki odgovarja na akcije učečega avtomata le z binarnimi znaki (0 = nagrada, 1 = kazen). Takšen model okolja bomo označevali z **B** (binarno okolje).

Najpreprostejša ocena uspešnosti učenja je, če rezultat učenja primerjamo z naključnim (angl. pure chance) izborom, kjer je verjetnost vsake akcije enaka:

$$p_i(n) = 1/r, \quad i = 1, 2, ..., r\,.$$

Takšen avtomat bo označen s PCA (angl. Pure Chance Automaton). Vsak avtomat, ki izkazuje lastnost učenja, mora biti boljši od PCA.

Naslednja količina, s pomočjo katere bomo vrednotili uspešnost učenja, je

povprečna kazen iz okolja, podanega s $\underline{c}$, ki jo določa izraz:

$$\begin{aligned} M(n) &= E[\beta(n)|p(n)] = P(\beta(n)=1|p(n)) = \\ &= \sum_{i=1}^{r} P(\beta(n)=1|\alpha(n)=\alpha_i)\cdot P(\alpha(n)=\alpha_i) = \\ &= \sum_{i=1}^{r} c_i \cdot p_i(n)\,, \end{aligned}$$

kjer je E[ ] oznaka za pričakovano oz. srednjo vrednost. V primeru PCA je $M(n)$ konstantna in označena z M_0:

$$M_0 = \frac{1}{r}\sum_{i=1}^{r} c_i\,.$$

Vsak avtomat, ki se obnaša boljše od PCA, mora imeti $M(n)$ manjšo kot M_0, vsaj asimptotično, ko gre $n \to \infty$. Ker so v splošnem $p(n)$, $\lim_{n\to\infty} p(n)$ in posledično tudi $M(n)$, $\lim_{n\to\infty} M(n)$ naključne spremenljivke, je potrebno primerjati $E[M(n)]$ z M_0. Ker velja:

$$E[M(n)] = E[E[\beta(n)|p(n)]] = E[\beta(n)]\,,$$

je $E[M(n)]$ enaka povprečnemu vhodu v avtomat.

Sedaj je mogoče podati norme za ocenjevanje učenja avtomata v stohastičnem in stacionarnem okolju.

A. Avtomat je **prikladen** (angl. expedient), če je boljši od PCA:

$$\lim_{n\to\infty} E[M(n)] < M_0$$

B. Najbolj je seveda zanimiv **optimalni** avtomat, katerega učenje vodi k minimalni povprečni kazni, oz.:

$$\inf M(n) = \inf_{p(n)} \left(\sum_{i=1}^{r} c_i p_i(n) \right) = \min\left(c_i\right) = c_l\,,$$

kjer je inf oznaka za infimalno operacijo in je l indeks najmanjše (angl. low) komponente po vrednosti v vektorju c. Avtomat je **optimalen**, če velja:

$$\lim_{n\to\infty} E[M(n)] = c_l = \min_i(c_i),$$

V tem primeru bo avtomat izbiral akcijo α_l, ki ji ustreza c_l z verjetnostjo, ki se bo s časom asimptotično približevala 1.

C. Avtomat je **sub-optimalen**, kadar optimalnosti ni mogoče doseči v celoti:

$$\lim_{n\to\infty} E[M(n)] = c_l + \varepsilon$$

kjer je ε poljubno majhna veličina.

D. Pomembna je še definicija **absolutno prikladnega** (angl. absolutely expedient) avtomata, do katerega je mogoče priti, kot bomo spoznali v nadaljevanju, po analitični poti:

$$E[M(n+1)] < E[M(n)],$$

Takšen avtomat izkazuje monotono padajočo funkcijo povprečne kazni iz okolja.

2.3 Avtomati v stacionarnem stohastičnem okolju

V tem poglavju želimo spoznati obnašanje različnih avtomatov v stacionarnem stohastičnem okolju (SSO) tipa **B**. Najprej nas bo zanimalo obnašanje deterministčnega avtomata (DA), nato pa še obeh tipov stohastičnih avtomatov: s fiksno strukturo (SAFS) in s spremenljivo strukturo (SASS). V vseh primerih bo vrednoteno njihovo obnašanje z normami obnašanja, ki so bile definirane v prejšnjem poglavju.

2.3.1 Obnašanje avtomatov s fiksno strukturo

Pri deterministčnih avtomatih [1] so vrednosti v matrikah operatorjev F in G le 0 in 1. Ker teh vrednosti tudi ne spreminjamo, lahko vplivamo na obnašanje deterministčnega avtomata le z ustrezno (hevristično) izbiro njegovih operatorjev. Primer takšnega deterministčnega avtomata je avtomat $L_{2,2}$.

Avtomat $L_{2,2}$

Pri avtomatu $L_{2,2}$ indeksa povesta, da ima dve stanji in dva izhoda, črka L pa poudarja, da gre za učeči avtomat (angl. Learning Automaton). Njegova

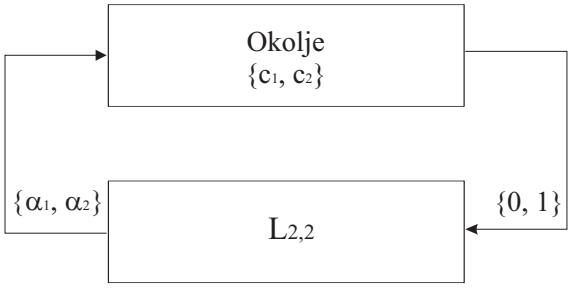
značilnost je, da spremeni stanje, če dobi na vhodu odgovor okolja 1 (kazen) in ohrani stanje, če dobi 0 (nagrado). Temu ustrezata matriki prehajanja stanj:

$$\mathbf{F}(0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{F}(1) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} . \tag{2.1}$$

Vsakemu stanju ustreza svoja akcija, kar pomeni, da je izhodna matrika kar identiteta:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} . \tag{2.2}$$

Slika 2.4 prikazuje avtomat $L_{2,2}$ v okolju.



Slika 2.4: Avtomat $L_{2,2}$ v okolju.

Preprosta (hevristična) strategija avtomata $L_{2,2}$ je, da nadaljuje s ponavljanjem akcij iz preteklosti v primeru nagrade iz okolja in da akcijo (stanje) spremeni v primeru kazni. Verjetnosti c iz okolja seveda ne poznamo, zanima pa nas, kako bo avtomat deloval skozi čas, oz. kakšna bo njegova povprečna kazen $M(n)$ in kakšne bodo limitne verjetnosti za obe akciji (statistika izbora akcij). PCA avtomat bi v istem okolju izbiral med obema akcijama z enako verjetnostjo, zato je njegova povprečna kazen

$$M_0 = (c_1 + c_2)/2 .$$

Ker $L_{2,2}$ deluje v stohastičnem in stacionarnem okolju, obnašanje celotnega sistema (učeči avtomat + okolje) ustreza stohastičnemu avtomatu s konstantnimi vhodi. Zato je njegov operator $\tilde{F}$ določen z:

$$\begin{aligned} \tilde{f}_{ij} &= P_{ij} = c_i \cdot f^1_{ij} + d_i \cdot f^0_{ij}, \\ d_i &= 1 - c_i . \end{aligned}$$

Ker delovanje avtomata $L_{2,2}$ ustreza stohastični in ergodični verigi, je matrika prehodov P podana s:

$$\mathbf{P} = \begin{bmatrix} d_1 & c_1 \\ c_2 & d_2 \end{bmatrix}, \tag{2.3}$$

katere vsebina sledi iz matrik $F(0)$ in $F(1)$ in izraza za $\tilde{f}_{ij}$, ki je splošni člen matrike P. Zaradi ergodične lastnosti delovanja avtomata $L_{2,2}$ v okolju $\mathbf{B}$, lahko končni verjetnosti obeh stanj avtomata ϕ_i $(i = 1, 2)$ in torej tudi akcij α_i $(i = 1, 2)$ dobimo iz pogoja:

$$\boldsymbol{\pi} = \mathbf{P}^T \boldsymbol{\pi}\,,$$

kjer je π oznaka za vektor verjetnosti stanja v limiti:

$$\pi_i = \lim_{n\to\infty} \pi_i(n), \qquad i = 1, 2\,.$$

V razviti obliki lahko pišemo:

$$\begin{aligned} d_1\pi_1 + c_2\pi_2 &= \pi_1 \\ c_1\pi_1 + d_2\pi_2 &= \pi_2\,. \end{aligned}$$

Ob upoštevanju $\pi_1 + \pi_2 = 1$ dobimo rešitev sistema enačb z dvema neznankama:

$$\begin{aligned} \pi_1 &= \frac{c_2}{c_1 + c_2}\,, \\ \pi_2 &= \frac{c_1}{c_1 + c_2}\,. \end{aligned}$$

Od tod lahko izračunamo še povprečno kazen $M(n)$ v limiti:

$$\lim_{n\to\infty} M(n) = \sum_{i=1}^{2} c_i\pi_i = \frac{2c_1c_2}{c_1 + c_2} = M(L_{2,2})\,.$$

Če je $c_1 \neq c_2$, potem velja

$$\frac{2c_1c_2}{c_1 + c_2} < \frac{c_1 + c_2}{2}$$

oz. je $L_{2,2}$ prikladen učeči avtomat. V limiti izbere akciji z verjetnostima, ki sta proporcionalni nasprotni verjetnosti napake, $\pi_1 \approx c_2$ in $\pi_2 \approx c_1$. Verjetnost prvega stanja oz. pripadajoče akcije je proporcionalna verjetnosti kaznovanja drugega stanja oz. pripadajoče akcije in obratno. Če je npr. $c_1 < c_2$, potem

je verjetnost izbire akcije α_1 (tj. verjetnost stanja π_1) večja kot pri akciji α_2 (oz. π_2), kar pomeni, da bo kaznovanje iz okolja omejeno na minimum glede na naravo okolja.

Pomembne značilnosti analize obnašanja deterministčnih avtomatov v stacionarnem stohastičnem okolju so torej:

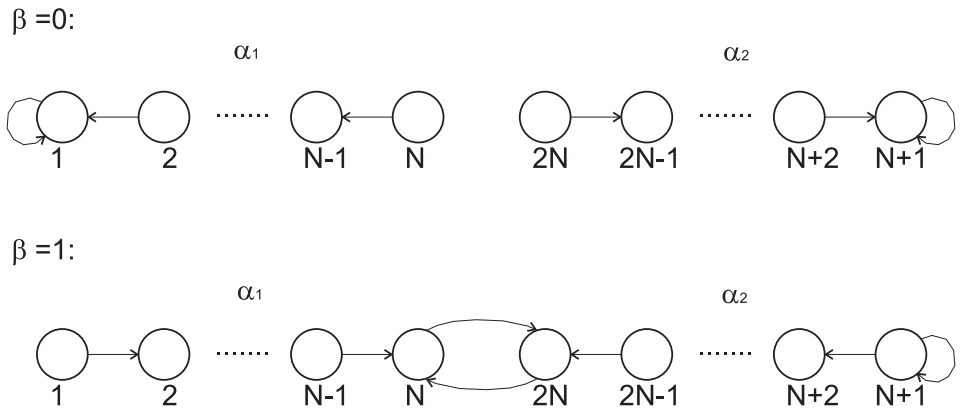
- Operatorja deterministčnega avtomata sta določena s hevristiko
- Delovanje ekvivalentnega stohastičnega avtomata opisuje homogena diskretna Markovska veriga
- Zaradi ergodične lastnosti stohastičnega avtomata je mogoče določiti stacionarne verjetnosti stanj in akcij ter posledično limitno povprečno kazen $\lim_{n\to\infty} M(n)$, ki določa prikladnost avtomata.

Avtomat $SL_{2,2}$: stohastična varianta $L_{2,2}$

Če so v matriki $F(0)$ namesto ničel uporabljene poljubno majhne vrednosti γ_1 in namesto enic 1-γ_1, v matriki $F(1)$ pa namesto ničel γ_2 in namesto enic (1 γ_2), avtomat postane stohastičen, oz. $SL_{2,2}$. S podobno analizo kot prej je mogoče ugotoviti, da s to spremembo ni mogoče bistveno izboljšali obnašanja avtomata, ki bi še vedno ostajal v najboljšem primeru le prikladen (pri $0 \leq \gamma_1 = \gamma_2 < \frac{1}{2}$), lahko pa bi postal celo neprikladen ($\frac{1}{2} \leq \gamma_1 = \gamma_2 \leq 1$).

Avtomat $L_{2N,2}$

Avtomat $L_{2,2}$ je povečan tako, da ima novi avtomat $2N$ stanj, torej ima namesto vsakega prejšnjega stanja po N novih stanj. Prvih N stanj ima eno akcijo, drugih N stanj pa drugo. S tem je v avtomat vgrajena inercija, saj avtomat vztraja dalj časa pri eni in isti akciji, preden jo spremeni. Graf prehajanja stanj avtomata $L_{2N,2}$ prikazuje Slika 2.5. Bistvo povečanja stanj je torej v globini N, ki pomeni povečan odmik od meje med dvema skupinama stanj (glede na akciji) v primeru nagrade in približevanje meji in preskok v primeru kazni. Avtomat $L_{2N,2}$ ima matriki $F(0)$ in $F(1)$ podobni kot $L_{2,2}$, le da sta (lahko) precej večji, zato je tudi reševanje sistema enačb zahtevnejše.



Slika 2.5: Diagram prehajanja stanj za avtomat $L_{2N,2}$.

N	$M(L_{2N,2})$
1	0.32
2	0.235
3	0.209
∞	0.200

Tabela 2.1: Povprečna kazen avtomata $L_{2N,2}$ glede na globino pomnilnika N.

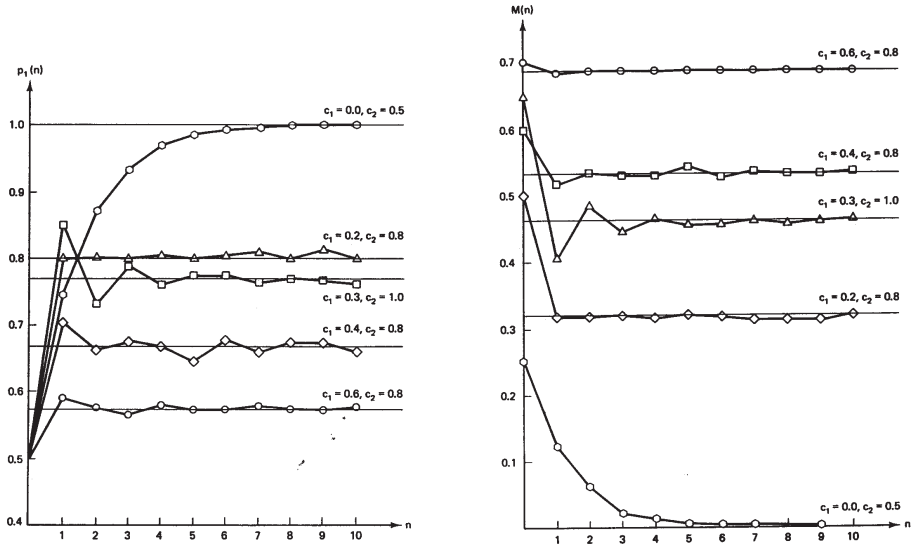
Analiza, podobna kot v primeru $L_{2,2}$, pokaže, da v primeru $N \to \infty$ povprečna kazen limitira k minimalni komponenti vektorja okolja c, ozr:

$$\lim_{N\to\infty} M(L_{2N,2}) = \min(c_1, c_2), \quad \text{če } \min(c_1, c_2) \leq \frac{1}{2}\,.$$

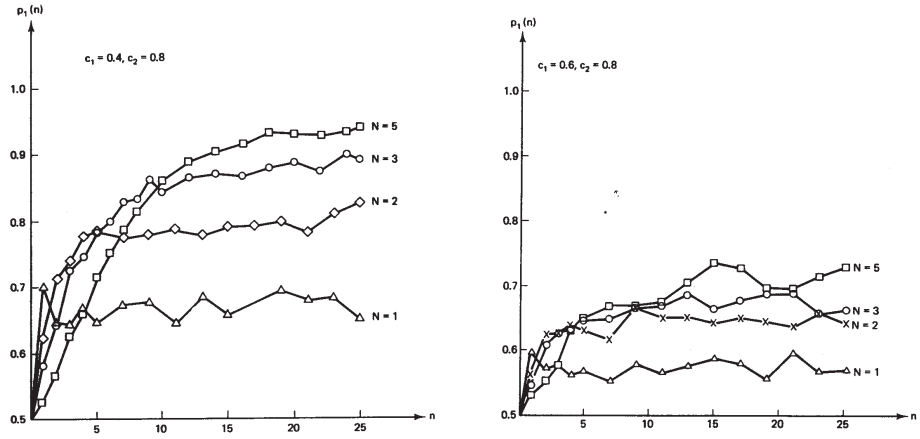
Čeprav je $L_{2N,2}$ optimalen šele pri $N \to \infty$, pa M s povečevanjem N zelo hitro konvergira k optimalni vrednosti oz. k $c_l = \min_i(c_i)$. To kaže tudi primer v Tabeli 2.1, kjer je okolje $c = (0,2; 0,8)$.

Primer
Prikaz simulacij z avtomatoma $L_{2,2}$ (Slika 2.6) in $L_{2N,2}$ (Slika 2.7) pri različnih vrednostih vektorja okolja c, $c_1 < c_2$. Verjetnosti p_i so določene eksperimentalno.

Poleg natančnosti učenja učečega avtomata je pomembna tudi hitrost. Če globina pomnilnika vpliva na natančnost tako, da z večanjem N pada povprečna kazen M, potem sledi iz teorije Markovskih verig (dokaz je mogoče najti v [2]), da na hitrost konvergence učenja vplivajo lastne vrednosti matrike preha-



Slika 2.6: $L_{2,2}$: časovna odvisnost $p_1(n)$ in $M(n)$ za slučaj $c_1 < c_2$. Vir: [2]



Slika 2.7: $L_{2N,2}$: časovna odvisnost $p_1(n)$ in $M(n)$ za slučaj $c_1 < c_2$. Vir: [2]

janja stanj Markovske verige P, oz. točneje, absolutna vrednost druge lastne vrednosti $|\lambda_2|$ je obratno sorazmerna s hitrostjo učenja. Pri $L_{2,2}$ bi dobili za $\lambda_2 = 1 - c_1 - c_2$, kar pomeni, da se hitrost poveča, če se $c_1 + c_2$ približuje 1.

2.3.2 Obnašanje avtomatov s spremenljivo strukturo

Pri avtomatih s fiksnimi strukturami (verjetnostmi v matrikah operatorjev) so kot matematično orodje služile Markovske verige. Za obnašanje teh avtomatov je značilno prikladno obnašanje, če so le bile pravilno izbrane verjetnosti prehajanja stanj avtomata in če je bilo okolje primerno (z različnimi vrednostmi verjetnosti za kaznovanje akcij).

Večjo fleksibilnost je mogoče vgraditi v modele, če se predpostavlja, da se verjetnosti prehajanja stanj ali verjetnosti izbire akcij ažurirajo po vsakem koraku, glede na neko učilno ali korekcijsko shemo (angl. learning ali reinforcement scheme) [3]. Takšni avtomati bodo opisani v tem poglavju. Tudi tukaj bo kot glavno orodje uporabljena teorija Markovskih procesov. Učilne sheme, uporabljene pri avtomatih v stacionarnem stohastičnem okolju, rezultirajo v Markovske verige, ki so ali ergodične ali pa vsebujejo absorpcijska stanja, t.j. stanja, iz katerih ne morejo pobegniti.

Stohastični avtomat s spremenljivo strukturo je v splošnem določen s peterčkom $< \phi, \alpha, \beta, A, \mathcal{G} >$, kjer so vse veličine razen A že poznane, A pa je oznaka za učilno ali korekcijsko shemo, oz. za ažurirni algoritem. Brez izgube na splošnosti (zaradi možnih prehodov med Mealy-jevim in Moore-ovim tipom avtomata), bo v nadaljevanju uporabljen Moore-ov tip, kar pomeni, da bo pozornost namenjena predvsem ažuriranju verjetnosti izbire akcije. Če pri tem vsakemu stanju avtomata ustreza posebna akcija, je operator $\mathcal{G}$ določen z matriko identitete, $G = I$. Tedaj je mogoče stohastični avtomat s spremenljivo strukturo podati kar s trojčkom $< \alpha, \beta, A >$.

Za stohastični avtomat s spremenljivo strukturo je primerna predstavitev v obliki sekvence akcijskih verjetnosti, $(\mathbf{p}(n))$, $n \geq 0$, ki je časovno diskreten Markovski proces, definiran nad ustreznim prostorom stanj.

2.3.3 Korekcijske sheme

Korekcijska shema je predstavljena ali s

$$p(n+1) = T[p(n), \alpha(n), \beta(n)],$$

v primeru verjetnosti izbire akcij, ali pa z

$$f_{ij}^{\beta}(n+1) = T^{*}[f_{ij}^{\beta}(n), \phi(n), \phi(n+1), \beta(n)],$$

v primeru verjetnosti prehodov med stanji. Pri tem sta T in T^* preslikavi. Korekcijske sheme so klasificirane glede na:

- asimptotično obnašanje avtomata oz. kateri normi obnašanja ustreza (prikladnost, optimalnost, ...),
- naravo preslikave T ali T^* (linearna, nelinearna, hibridna),
- lastnosti Markovskega procesa, ki opisuje učeči avtomat (ergodični, neergodični).

Osnovna ideja korekcijske sheme je preprosta. Če avtomat izbere akcijo α_i v času n in temu sledi nagrada iz okolja ($\beta(n) = 0$), potem se verjetnost akcije $p_i(n)$ poveča, verjetnosti ostalih akcij pa zmanjšajo. Če pa sledi iz okolja kazen ($\beta(n) = 1$), se verjetnost ustrezne akcije zmanjša, verjetnosti ostalih akcij pa se povečajo. Možen je tudi 'status quo', ko se vse verjetnosti ohranijo. Analogno velja tudi za verjetnosti prehajanja stanj, v primeru, da so uporabljene le-te.

Vzemimo stohastični avtomat s spremenljivo strukturo (SASS) z r akcijami, ki deluje v stacionarnem stohastičnem okolju tipa **B** ($\beta \in \{0, 1\}$). Za splošno korekcijsko shemo tedaj velja:

če je $\alpha(n) = \alpha_i$, $i = 1, 2, ..., r$, potem je:

$$\begin{aligned} p_j(n+1) &= p_j(n) - g_j[p(n)], \quad \text{za vse } j \neq i, \quad \text{ko je } \beta(n) = 0\,, \\ p_j(n+1) &= p_j(n) + h_j[p(n)], \quad \text{za vse } j \neq i, \quad \text{ko je } \beta(n) = 1\,. \end{aligned} \tag{2.4}$$

Da ostane vsota enaka 1, mora veljati:

$$\begin{aligned} p_i(n+1) &= p_i(n) + \sum_{j=1, j\neq i}^{r} g_j[p(n)], \qquad \text{ko je } \beta(n) = 0 \\ p_i(n+1) &= p_i(n) - \sum_{j=1, j\neq i}^{r} h_j[p(n)], \qquad \text{ko je } \beta(n) = 1\,. \end{aligned} \tag{2.5}$$

V zgornjih izrazih sta g_j in h_j korekcijski funkciji, ki morata zagotavljati, da se verjetnosti tudi po korekcijah ohranjajo znotraj intervala $[0, 1]$. Zato morata izpolnjevati naslednje pogoje ($j = 1, 2, ..., r$):

- g_j in h_j sta zvezni funkciji

- g_j in h_j sta nenegativni funkciji
- $0 < g_j(p) < p_j$
  $0 < \sum_{j=1, j\neq i}^{r} [p_j + h_j(p)] < 1$, za vse $i = 1, 2, ..., r$.

Funkcija g se uporablja v primeru nagrade, h pa v primeru kazni iz okolja, obe pa sta neodvisni od izbrane akcije α_i.

Splošna korekcijska shema v primeru ažuriranja verjetnosti prehajanja stanj pa ima obliko:

če je $\phi(n) = \phi_i$ in $\phi(n+1) = \phi_j$, potem je:

$$\begin{aligned} f_{ik}^0(n+1) &= f_{ik}^0(n) - g_{ik}[F(0,n)], \qquad \text{ko je } \beta(n) = 0 \\ f_{ik}^1(n+1) &= f_{ik}^1(n) + h_{ik}[F(1,n)], \qquad \text{ko je } \beta(n) = 1 \\ &\quad \text{za vse } k = 1, 2, ..., s \text{ in } k \neq j \end{aligned}$$

in:

$$f_{ij}^0(n+1) = f_{ij}^0(n) + \sum_{k=1, k\neq j}^{s} g_{ik}[F(0,n)], \qquad \text{ko je } \beta(n) = 0$$

$$f_{ij}^1(n+1) = f_{ij}^1(n) - \sum_{k=1, k\neq j}^{s} h_{ik}[F(1,n)], \qquad \text{ko je } \beta(n) = 1$$

ter za vse $u \neq i$ in/ali $\beta(n) \neq \beta$:

$$f_{uv}^{\beta}(n+1) = f_{uv}^{\beta}(n)\,.$$

Tudi tukaj veljajo predpostavke glede funkcij g_{ik} in h_{ik}, tako kot prej za g_j in h_j.

Linearne korekcijske sheme

Najprej bodo podane tri linearne sheme na preprostem primeru avtomata z dvema stanjema in dvema akcijama. Te sheme bodo v nadaljevanju razširjene na sheme z večjim številom stanj/akcij ter na nelinearne sheme.

A. Linearna L_{R-P} ('Reward-Penalty') shema

Funkciji g in h sta podani z linearnima enačbama:

$$\begin{aligned} g_j[p(n)] &= ap_j(n) \\ h_j[p(n)] &= b(1-p_j(n)) = b - bp_j(n), \end{aligned}$$

kjer sta a in b parametra za nagrado in kazen, $0 < a < 1$, $0 \leq b < 1$ [4]. Če zgornje izraze vstavimo v splošne korekcijske enačbe (Enačbi 2.4 in 2.5), dobimo:

če je $\alpha(n) = \alpha_1$ in $\beta(n) = 0$ (nagrada):

$$\begin{aligned} p_1(n+1) &= p_1(n) + a(1-p_1(n)) = p_1(n) + ap_2(n) \\ p_2(n+1) &= (1-a)p_2(n) = p_2(n) - ap_2(n) \end{aligned}$$

če pa je $\alpha(n) = \alpha_1$ in $\beta(n) = 1$ (kazen):

$$\begin{aligned} p_1(n+1) &= (1-b)p_1(n) = p_1(n) - b(1-p_2(n)) \\ p_2(n+1) &= p_2(n) + b(1-p_2(n))\,. \end{aligned}$$

Zgornje izraze je mogoče pregledneje pisati samo za eno verjetnost (in ob upoštevanju vseh kombinacij), npr. za p_1:

$$p_1(n+1) = p_1(n) + a(1-p_1(n)) \qquad \alpha(n) = \alpha_1 \quad \beta(n) = 0 \tag{2.6}$$
$$p_1(n+1) = (1-b)p_1(n) \qquad \alpha(n) = \alpha_1 \quad \beta(n) = 1 \tag{2.7}$$
$$p_1(n+1) = (1-a)p_1(n) \qquad \alpha(n) = \alpha_2 \quad \beta(n) = 0 \tag{2.8}$$
$$p_1(n+1) = p_1(n) + b(1-p_1(n)) \qquad \alpha(n) = \alpha_2 \quad \beta(n) = 1\,. \tag{2.9}$$

Te enačbe določajo algoritem L_{R-P}. Pri njem ažuriramo vse verjetnosti tako v primeru nagrade, kot tudi kazni. Poseben primer nastopi, ko izberemo $a = b$.

Podobno kot v primeru avtomatov s fiksno strukturo so tudi tukaj zanimive predvsem asimptotične verjetnosti akcij, $p_i(\infty)$, $i = 1, 2$. Postopek za njihovo določitev je naslednji:

- Najprej je potrebno izraziti pogojno pričakovanje za $p_1(n+1)$ pri danem $p_1(n)$ ob upoštevanju vseh štirih možnih slučajev (zaradi enostavnosti bo

privzeto $a = b$):

$$\begin{aligned} E[p_1(n+1)|p_1(n)] &= [p_1(n) + a(1 - p_1(n))][p_1(n)(1 - c_1)] \\ &+ [(1-a)p_1(n)][p_1(n)c_1] \\ &+ [(1-a)p_1(n)][(1 - p_1(n))(1 - c_2)] \\ &+ [p_1(n) + a(1 - p_1(n))][(1 - p_1(n))c_2)] \\ &= [1 - a(c_1 + c_2)]p_1(n) + ac_2 \end{aligned}$$

- Sledi povprečenje obeh strani enačbe:

$$E[p_1(n+1)] = [1 - a(c_1 + c_2)]E[p_1(n)] + ac_2$$

- Rešitev zgornje diferenčne enačbe je (postopek reševanja je izpuščen):

$$E[p_1(n)] = [1 - a(c_1 + c_2)]^n p_1(0) + [1 - (1 - a(c_1 + c_2))^n]\frac{c_2}{c_1 + c_2}$$

kar daje v limiti:

$$\lim_{n\to\infty} E[p_1(n)] = \frac{c_2}{c_1 + c_2}, \quad \text{če } |1 - a(c_1 + c_2)| < 1$$

- Kot posledica limitne vrednosti za p_1 velja nadalje:

$$\lim_{n\to\infty} E[p_2(n)] = \frac{c_1}{c_1 + c_2}\,.$$

Če je $c_2 < c_1$, je akcija α_2 izbrana asimptotično z večjo verjetnostjo kot α_1. Povprečna kazen v limiti je na osnovi zgornjih izrazov določena z:

$$\lim_{n\to\infty} E[M(n)] = c_1 \lim_{n\to\infty} E[p_1(n)] + c_2 \lim_{n\to\infty} E[p_2(n)] = \frac{2c_1c_2}{c_1 + c_2} < \frac{c_1 + c_2}{2} = M_0\,.$$

L_{R-P} je prikladna shema za vse začetne pogoje ($p(0)$) in vsa stacionarna stohastična okolja $c = \{c_1, c_2\}$, saj zgornja neenačba ni odvisna od njih. Izjema je le primer $c_1 = c_2$.

B. Linearna L_{R-I} ('Reward Inaction') shema

Za to shemo je značilno, da daje povsem drugačno asimptotično obnašanje kot L_{R-P}. Kot sledi iz njene oznake, v primeru kazni ne spreminjamo verjetnosti.

Enačbe sheme sledijo iz splošne korekcijske sheme ob predpostavki $b = 0$.

$$p_1(n+1) = p_1(n) + a(1 - p_1(n)) \qquad \alpha(n) = \alpha_1 \quad \beta(n) = 0 \tag{2.10}$$
$$p_1(n+1) = p_1(n) \qquad \alpha(n) = \alpha_1 \quad \beta(n) = 1 \tag{2.11}$$
$$p_1(n+1) = (1-a)p_1(n) \qquad \alpha(n) = \alpha_2 \quad \beta(n) = 0 \tag{2.12}$$
$$p_1(n+1) = p_1(n) \qquad \alpha(n) = \alpha_2 \quad \beta(n) = 1\,. \tag{2.13}$$

Verjetnost očitno povečujemo v prvem in zmanjšujemo v tretjem primeru. Izkaže se, da imamo tukaj dve absorpcijski stanji.

$$p(k) = e_i, \quad \text{če} \;\; p(n) = e_i, \quad i \in \{1, 2\}, \quad k \geq n\,.$$

kjer je e_i eno od dveh absorpcijskih stanj, $e_1 = (1, 0), e_2 = (0, 1)$, ki sta ravno enotina vektorja. Ker se $p_1(n)$ zmanjša le, če je izbrana akcija α_2, ki jo spremlja nagrada iz okolja, je limitno stanje odvisno od relacije med c_1 in c_2. V primeru $c_1 > c_2$ je $\lim_{n\to\infty} p(n) = e_2$, v nasprotnem primeru pa e_1. Markovski proces $p(n)$ torej konvergira k nizu absorpcijskih stanj $V_2 = (e_1, e_2)$ z verjetnostjo 1.

Označimo pričakovano spremembo p_1 s

$$\Delta p_1(n) = E[p_1(n+1)|p_1(n)] - p_1(n)\,.$$

Iz enačb za L_{R-I} sledi:

$$\Delta p_1(n) = a p_1(n)(1 - p_1(n))(c_2 - c_1)$$

ali:

$$\begin{aligned} \Delta p_1(n) &\geq 0, \qquad \text{če} \;\; c_2 > c_1 \\ &\leq 0, \qquad \text{če} \;\; c_2 < c_1 \end{aligned}$$

in:

$$\Delta p_1(n) = 0, \qquad \text{če} \;\; p_1(n) \in \{0, 1\}\,.$$

p_1 torej narašča oz. se zmanjšuje monotono z n, glede na to, ali je c_2 večji ali manjši od c_1. Verjetnost akcije, ki ustreza minimalni verjetnosti kazni, narašča monotono z n proti 1, verjetnost preostale akcije pa proti 0. L_{R-I} je zato sub-optimalna korekcijska shema.

C. Linearna $L_{R-\varepsilon P}$ ('Reward-εPenalty') shema

Ta shema, ki je kombinacija shem L_{R-P} in L_{R-I}, spreminja verjetnosti v primeru nagrade s parametrom a, v primeru kazni iz okolja pa z εb, kjer je

ε poljubno majhno število. To pomeni, da bolj spreminjamo verjetnosti v primeru nagrade kot v primeru kazni. Tedaj veljajo enačbe:

$$E[p_1(n+1)|p_1(n)] = p_1(n)+b[(1-p_1(n))^2c_2-p_1^2(n)c_1]+ap_1(n)(1-p_1(n))(c_2-c_1)$$

oz.:

$$\Delta p_1(n) = b[(1-p_1)^2c_2 - p_1^2c_1] + ap_1(1-p_1)(c_2-c_1)\,,$$

kar je nelinearna zveza glede na p_1. Krivulja funkcije $\Delta p_1(n)$ je pozitivna pri $p_1 = 0$ (v primeru $0 < b << a$), negativna pri $p_1 = 1$, ter enaka 0 nekje vmes, npr. pri p^*. Pri zelo majhnem b se p^* približuje 1 (če $c_2 > c_1$) oz. 0 v nasprotnem primeru (če $c_2 < c_1$). Zato je $L_{R-\varepsilon P}$ ε-optimalna shema, kjer je asimptotična verjetnost najugodnejše akcije lahko tako blizu 1 (oz. srednja vrednost p^* blizu ustreznemu enotinemu vektorju), kot si želimo, če le izberemo a in b zadosti majhna.

V nasprotju s tem, L_{R-P} shema z $a \neq b$ in $b \neq 0$ nima absorpcijskih stanj in je zato ergodična. $p(n)$ konvergira k naključni spremenljivki p^*, ki je neodvisna od $p(0)$.

2.3.4 Posplošitve stohastičnih avtomatov s spremenljivo strukturo

Osnovne lastnosti stohastičnih avtomatov s spremenljivo strukturo v stacionarnem stohastičnem okolju so bile podane na primerih avtomatov z dvema stanjema/akcijama. Zato je prva možna posplošitev vezana na povečanje števila stanj oz. akcij SASS.

Stohastični avtomati s spremenljivo strukturo z več akcijami

V tem primeru je potrebno ustrezno modificirati funkcije g_j in h_j iz korekcijskih shem, tako kot prikazujejo enačbe:

$$\begin{aligned} g_j[p(n)] &= ap_j(n), & 0 < a < 1 \\ h_j[p(n)] &= \frac{b}{r-1} - bp_j(n), & 0 < b < 1\,. \end{aligned}$$

S tem se ustrezno spremenijo tudi korekcijske enačbe, npr. v primeru L_{R-P} v:

$$\begin{aligned}
&\alpha(n)=\alpha_i,\ \beta(n)=0: \\
&\qquad p_i(n+1) &&= p_i(n)+a[1-p_i(n)], \quad \text{ker} \sum_{j=1,j\neq i}^{r} p_j(n) = 1-p_i(n) \\
&\qquad p_j(n+1) &&= (1-a)p_j(n), \quad \text{za vse } j\neq i \\
&\alpha(n)=\alpha_i,\ \beta(n)=1: \\
&\qquad p_i(n+1) &&= (1-b)p_i(n), \\
&\qquad p_j(n+1) &&= \frac{b}{r-1}+(1-b)p_j(n), \quad \text{za vse } j\neq i
\end{aligned}$$

Izkaže se, da je tudi v primeru korekcijske sheme L_{R-P} in avtomata z več stanji obnašanje v stacionarnem stohastičnem okolju prikladno za vse akcijske verjetnosti in v vseh stacionarnih okoljih. L_{R-I} pa je v kombinaciji z več stanji ε–optimalna shema, če je le parameter a dovolj majhen.

Stohastični avtomati s spremenljivo strukturo in nelinearne sheme

Naslednja možna posplošitev je v bolj kompliciranih korekcijskih enačbah, v nelinearnih ali pa hibridnih (kombinacije linearnih in nelinearnih členov). Ker je takšnih shem ogromno, bo podan le primer nelinearne sheme N [5], ki velja tudi za avtomate z več stanji. Določata jo funkciji g_j in h_j:

$$g_j[p(n)] = \frac{a}{r-1}p_i(n)(1-p_i(n)) = h_j[p(n)], \qquad 0<a\leq 1$$

Ta shema porazdeli nelinearni prirastek $ap(1-p)$ enakomerno med vse akcije, ki niso izbrane v času n. Zaradi enostavnosti vzemimo $r=2$. Tedaj je:

$$\Delta p_1(n) = ap_1(1-p_1)[(2c_2-1)+2(1-c_1-c_2)p_1]\,.$$

Ker je $\Delta p_1(n)+\Delta p_2(n)=0$ za vse n, izhaja iz zgornje enačbe tudi:

če: $c_1<\frac{1}{2}<c_2$, potem $\Delta p_1(n)>0$ in $\Delta p_2(n)<0$ pri $p_1\in(0,1)$

in

če: $c_2<\frac{1}{2}<c_1$, potem $\Delta p_1(n)<0$ in $\Delta p_2(n)>0$ pri $p_1\in(0,1)$

Velja še:

$$\Delta M(n) = \Delta p_1(n)(c_1 - c_2) < 0,$$

če je $c_1 < \frac{1}{2} < c_2$ ali $c_2 < \frac{1}{2} < c_1$. Zgornja nelinearna shema N je absolutno prikladna pri omejenih začetnih pogojih in omejenih okoljih.

2.3.5 Absolutno prikladne sheme

Obnašanje avtomatov v stacionarnem stohastičnem okolju je v precejšnji meri odvisno od korekcijskih enačb. Razvoj zgodnjih korekcijskih shem je potekal bolj empirično, pri čemer so bile v veliko pomoč preproste funkcije za nagrado in kazen. Intuicija je bila uspešna v primeru dveh akcij, ko je šlo v bistvu za ažuriranje le ene verjetnosti. Posplošitev na več akcij ni bila enostavna, če je sploh bila mogoča. Zato je šel nadaljnji razvoj korekcijskih shem v smeri njihove sinteze. Vprašanje je bilo, kakšni so pogoji funkcij, ki v shemah nastopajo, da zagotavljajo želeno obnašanje. Iskanje odgovora je privedlo do koncepta absolutne prikladnosti. Razred absolutno prikladnih shem [6] predstavlja edini razred shem, za katerega so poznani potrebni in zadostni pogoji snovanja korekcijskih funkcij g in h. Predstavljajo posplošitev L_{R-I} sheme. V stacionarnih okoljih so te sheme sub-optimalne.

Teorem, ki določa omenjene pogoje, pravi:

Učeči avtomat, ki uporablja splošno korekcijsko shemo, je absolutno prikladen, če in samo če funkciji $g_i(p)$ in $h_i(p)$ zadoščata naslednjim pogojem simetrije:

$$\begin{aligned} \frac{g_1(p)}{p_1} &= \frac{g_2(p)}{p_2} = \ldots = \frac{g_r(p)}{p_r} = \lambda(p) \\ \frac{h_1(p)}{p_1} &= \frac{h_2(p)}{p_2} = \ldots = \frac{h_r(p)}{p_r} = \mu(p) \end{aligned}$$

Od tod neposredno sledi najbolj splošen zapis absolutno prikladne korekcijske sheme:

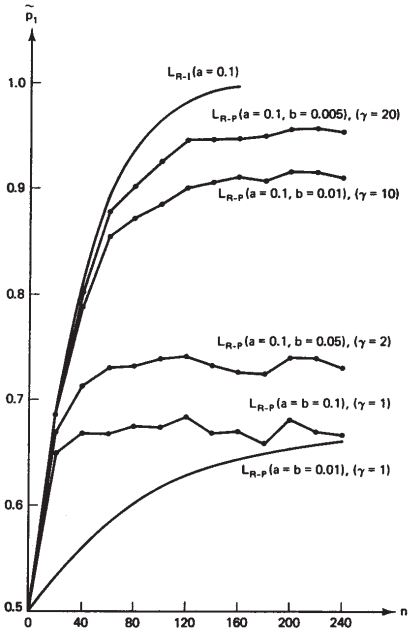
$$\begin{aligned} p_i(n+1) &= p_i(n)[1 - \lambda(p(n))], && \text{če } \alpha(n) \neq \alpha_i, \beta(n) = 0 \\ p_i(n+1) &= p_i(n)[1 + \mu(p(n))], && \text{če } \alpha(n) \neq \alpha_i, \beta(n) = 1 \\ p_i(n+1) &= p_i(n) + \lambda(p(n))(1 - p_i(n)), && \text{če } \alpha(n) = \alpha_i, \beta(n) = 0 \\ p_i(n+1) &= p_i(n) - \mu(p(n))(1 - p_i(n)), && \text{če } \alpha(n) = \alpha_i, \beta(n) = 1\,. \end{aligned}$$

Ker sedaj funkciji λ in μ vplivata na izračune verjetnosti, mora zanju veljati podobno kot prej za funkciji g in h:

- $\lambda(p)$ in $\mu(p)$ sta zvezni funkciji
- $0 < \lambda(p) < 1$ in $0 \leq \mu(p) < \min_i(\frac{p_i}{1-p_i})$.

Shemo L_{R-I} lahko dobimo z

$$\lambda(p) = a\,, \quad \mu(p) = 0\,.$$



Slika 2.8: Povprečna verjetnost akcije, ki ustreza c_{min}, za okolje 1. Vir: [2].

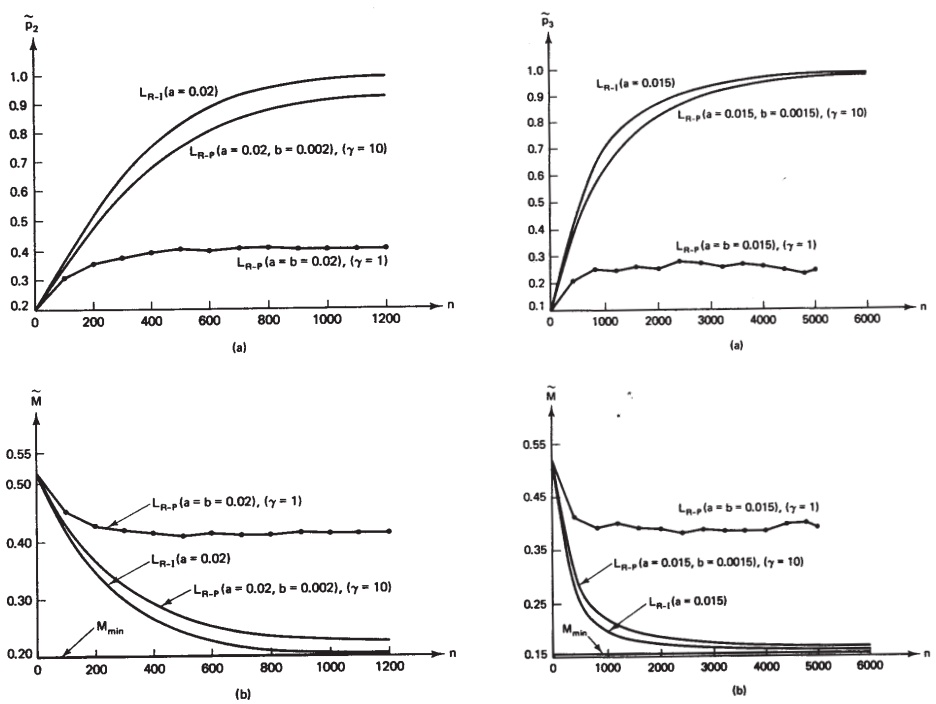
2.3.6 Primeri simulacij

Nad tremi različnimi primeri okolja bodo prikazani rezultati delovanja učečih avtomatov (SASS), ki uporabljajo različne korekcijske sheme, linearne L_{R-P}, L_{R-I}, $L_{R-\varepsilon P}$ in nelinearno N.

OKOLJE 1: $r = 2$;
$c_1 = 0{,}4$; $c_2 = 0{,}8$
OKOLJE 2: $r = 5$;
$c_1 = 0{,}65$; $c_2 = 0{,}2$; $c_3 = 0{,}5$; $c_4 = 0{,}4$; $c_5 = 0{,}8$
OKOLJE 3: $r = 10$;
$c_1 = 0{,}9$; $c_2 = 0{,}55$; $c_3 = 0{,}16$; $c_4 = 0{,}24$; $c_5 = 0{,}8$;
$c_6 = 0{,}6$; $c_7 = 0{,}4$; $c_8 = 0{,}3$; $c_9 = 0{,}5$; $c_{10} = 0{,}7$

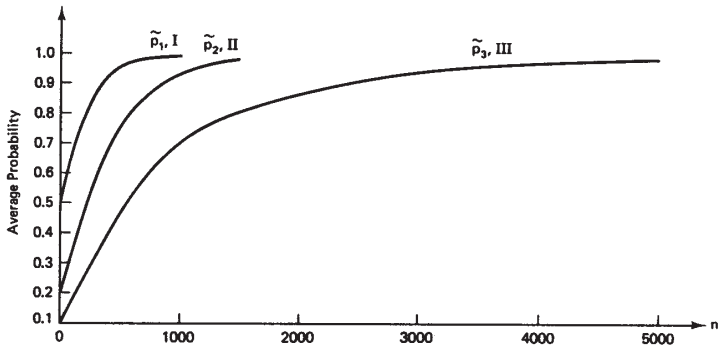
Slika 2.8 prikazuje povprečno verjetnost (preko več tekov simulacije) $\tilde{p}_1$ za primer okolja 1, v odvisnosti od diskretnih časovnih korakov n in v primerih različnih vrednosti parametrov a in b korekcijske sheme L_{R-P}.

Rezultati učenja v primeru drugega in tretjega okolja so podani na Sliki 2.9. Pri drugem okolju je podana odvisnost verjetnosti $\tilde{p}_2$ (ker je $c_2 = \min_i(c_i)$), pri tretjem pa iz istega razloga $\tilde{p}_3$. Zgoraj so podane povprečne verjetnosti, spodaj pa povprečne kazni v odvisnosti od diskretnega časa n.



Slika 2.9: Rezultati učenja v okolju 2 (levo) in okolju 3 (desno). Vir: [2]

Iz zgornjih slik izhaja, da je učenje daljše, če je število akcij večje. Slika 2.10 podaja poteke relevantnih verjetnosti akcij (tistih, ki ustrezajo c_{min}), za vse tri okoljske primere in za slučaj sheme L_{R-I} ($a = 0{,}015$).



Slika 2.10: Vpliv števila akcij na hitrost učenja pri L_{R-I} (a=0.015). Vir: [2]

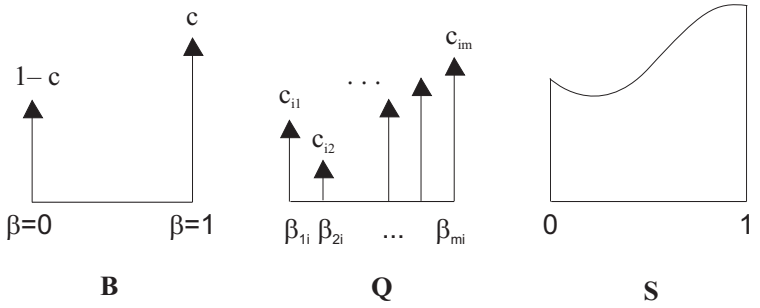
2.3.7 Posplošitve okolja

Do sedaj smo predpostavljali, da je okolje stohastično in stacionarno (SSO), ter da odgovarja na akcije s strani avtomata le binarno (okolje **B**), z nagrado ($\beta = 0$) ali z kaznijo ($\beta = 1$). V nadaljevanju bo pokazano, kako vpliva na učenje avtomatov okolje, ki odgovarja z večjim številom diskretnih vrednosti ali celo z zveznimi vrednostmi in kako se lahko uči avtomat v nestacionarnem okolju.

Modela okolja Q in S

Pri modelu **Q** okolje odgovori na akcije avtomata s končnim (diskretnim) številom odzivov v intervalu $[0, 1]$, vsakim s svojo verjetnostjo. Vsaki akciji α_i tako ustreza niz diskretnih odgovorov β_{ji}, kjer indeks j določa odgovor v primeru akcije z indeksom i. Za model **S** pa je značilno, da odgovarja z zveznimi vrednostmi na istem intervalu. V primeru teh dveh modelov okolja odgovori torej niso niti v celoti zaželeni (nagrajeni), niti nezaželeni (kaznovani).

Primerjavo med modeli **B**, **Q** in **S** lahko prikažemo tudi z verjetnostnimi funkcijami za odzive okolja na dano akcijo α_i (Slika 2.11).



Slika 2.11: Modeli **B**, **Q** in **S**.

Pri modelih okolja **Q** in **S** se nekoliko spremenijo performančni kriteriji oz. norme obnašanja. Tedaj je povprečna kazen definirana z izrazom:

$$M(n) = E[\beta(n)|p(n)] = \sum_{i=1}^{r} s_i p_i \,,$$

kjer je:

$$s_i = E[\beta(n)|\alpha(n) = \alpha_i] \,,$$

ki se imenuje moč kazni (angl. penalty strength) in igra podobno vlogo kot c_i pri modelu okolja **B**. Če pri modelu **Q** označimo:

$$c_{ij} = P(\beta(n) = \beta_{ji}|\alpha(n) = \alpha_i), \qquad i = 1, ..., r, \qquad j = 1, ..., m_i,$$

potem je:

$$s_i = E[\beta(n)|\alpha(n) = \alpha_i] = \sum_{j=1}^{m_i} \beta_{ji} c_{ij} \,.$$

Če so poznane vse verjetnosti c_{ij}, potem lahko vse s_i izračunamo. Izračun povprečne kazni je odvisen le od s_i in ne od detajlov, kot so β_{ji} in c_{ij}. Pri modelu **S**, kjer je verjetnostna funkcija gostote $f_i(\beta)$ poznana, moč kazni s_i ustreza pričakovani (povprečni) vrednosti izhoda.

Performančni kriteriji oz. norme so zato v primeru okolij **Q** in **S** enaki:

1. Avtomat PCA je podan z:

$$M_0 = \sum_{i=1}^{r} \frac{s_i}{r}$$

2. Učeči avtomat je prikladen, če:

$$\lim_{n\to\infty} E[M(n)] < \sum_{i=1}^{r} \frac{s_i}{r} = M_0,$$

3. Učeči avtomat je optimalen, če:

$$\lim_{n\to\infty} E[M(n)] = s_l, \qquad l \equiv 'low'$$

4. Učeči avtomat je sub-optimalen, če:

$$\lim_{n\to\infty} E[M(n)] < s_l + \varepsilon, \qquad l \equiv 'low'$$

5. Učeči avtomat je absolutno prikladen, če:

$$E[M(n+1) - M(n)] < 0\,.$$

Zaradi omenjenega se spremenijo tudi korekcijske enačbe. Ker je odziv okolja sedaj hkrati nagrada in kazen, imamo le dve enačbi:

$$\begin{aligned} p_i(n+1) &= p_i(n) - (1-\beta(n))g_i[p(n)] + \beta(n)h_i[p(n)], \qquad \text{če}\,\alpha(n) \neq \alpha_i \\ p_i(n+1) &= p_i(n) + (1-\beta(n))\sum_{j\neq i} g_j[p(n)] - \beta(n)\sum_{j\neq i} h_j[p(n)], \quad \text{če}\,\alpha(n) = \alpha_i\,. \end{aligned}$$

V zgornjih enačbah sta funkciji g in h zopet vezani na nagrado in kazen, $(1-\beta)$ pa je indikator, kako daleč je β od maksimalne vrednosti (kazni).

Primer
Linearno shemo v okolju **Q** ali **S** označujemo s SL_{R-P}. V primeru r akcij avtomata so funkcije g in h določene z izrazi:

$$\begin{aligned} g_i(p) &= ap_i(n) \\ h_i(n) &= \frac{a}{r-1} - ap_i(n) \end{aligned}$$

Zgornja shema je prikladna za vsa naključna stacionarna okolja, limitno pričakovanje akcijskih verjetnosti pa je inverzno proporcionalno ustreznim močem kazni s_i.

2.4 Nestacionarna okolja

Dejanska potreba po učenju oz. adaptaciji nastopa predvsem tam, kjer se okolje spreminja s časom. Če v takšnem okolju uporabimo učeči avtomat s fiksno strategijo, postane s časom lahko le manj prikladen oz. celo neprikladen.

Okolje je nestacionarno, če se verjetnosti kazni c_i, $i = 1, ..., r$, ki ustrezajo posameznim akcijam, spreminjajo s časom. Če so spremembe hitre, med njimi pa daljši čas, ko ni sprememb, lahko govorimo o etapno različnih naključnih stacionarnih okoljih. Tedaj je okolje sestavljeno iz niza različnih stacionarnih okolij. Če avtomat ve, v katerem okolju se nahaja, potem lahko uporabi sheme, ki glasijo na ustrezne stacionarne razmere. Takšne razmere opisujemo z avtomati A_i, ki delujejo v okoljih E_i.

Dva razreda nestacionarnih okolij sta bila podrobno analizirana: *Markovska preklopna okolja* (MPO) in *Stanjsko odvisna okolja* (SOO). Tretji razred (*Dinamična okolja*) pa je razširitev drugega.

2.4.1 Markovska preklopna okolja

V tem primeru so etapna stacionarna okolja E_i, $i = 1, ..., d$, stanja Markovske verige. Če je veriga ergodična, bo avtomat v vsakem stanju verige z verjetnostjo, ki pripada asimptotični verjetnostni porazdelitvi verige. Celoten sistem (učeči avtomat in okolje) je v primeru MPO ekvivalenten homogeni Markovski verigi.

Vzemimo, da učeči avtomat deluje v MPO z r akcijami. Če je prostor stanj učečega avtomata označen z S, potem je prostor stanj sistema (učeči avtomat, okolje) kartezijski produkt $S \times E$. Če je q_j verjetnost stanja E_j in c_{ij} verjetnost kazni za akcijo α_i v E_j, potem je povprečna kazen:

$$M(n) = \sum_{i=1}^{r} p_i(n)(\sum_{j=1}^{d} q_j(n)c_{ij}) \, .$$

V primeru:

$$\lim_{n\to\infty} E[M(n)] < \frac{1}{r} \sum_{i=1}^{r} \sum_{j=1}^{d} q_j^* c_{ij} \, ,$$

kjer so q_j^* stacionarne verjetnosti stanj MPO, je učeči avtomat prikladen.

2.4.2 Stanjsko odvisna okolja

V tem primeru gre za nov matematični model učečega avtomata, pri katerem akcije avtomata vplivajo na odzivne karakteristike okolja. Takšen primer adaptivnega procesa v nestacionarnem okolju je mogoče zaslediti v elektronskih komunikacijskih sistemih, kjer učeči avtomat ustreza komunikacijskemu sistemu , okolje pa prometu. V odvisnosti od izbire smeri za določeno zahtevo (klic), se prometni režim spremeni tako, da se obremenitve prometa porazdelijo po vseh možnih komunikacijskih poteh.

Ogledali si bomo tak model, pri katerem se ob akciji $\alpha(n) = \alpha_i$ ustrezna verjetnost kazni c_i iz okolja poveča, ostale c_j, $j \neq i$, pa zmanjšajo. To velja za vse n, kar pomeni, da se izvedene akcije slabšajo, neizvedene pa izboljšujejo.

Matematično lahko takšen model okolja opišemo z:

če:

$$c_i(n) = P(\beta(n) = 1 | \alpha(n) = \alpha_i), \qquad i = 1, ..., r,$$

potem:

$$c_i(n+1) = c_i(n) + \vartheta_i(n)$$

$$c_j(n+1) = c_j(n) - \rho_j(n), \qquad j \neq i\,,$$

kjer sta $\vartheta_i(n)$ in $\rho_j(n)$, $i, j = 1, ..., r$, nenegativni funkciji diskretnega časa n in v splošnem odvisni tudi od $p(n)$ in $c_i(n)$. V najenostavnejšem primeru pa sta lahko kar konstanti, npr.:

$$\vartheta_i(n) = \begin{cases} \vartheta_i, & \text{če } c_i(n) + \vartheta_i \leq 1, \\ 1 - c_i(n), & \text{sicer.} \end{cases}$$

in:

$$\rho_i(n) = \begin{cases} \rho_i, & \text{če } c_i(n) - \rho_i \geq 0, \\ c_i(n), & \text{sicer.} \end{cases}$$

2.4.3 Dinamična okolja

Še bolj splošen primer kot SOO je model dinamičnega okolja. Opisuje ga enačba:

$$c(n+1) = Q[c(n), p(n)]\,,$$

ki skupaj z učilnim algoritmom:

$$p(n+1) = T[p(n), \alpha(n), \beta(n)]$$

tvori enačbe stanj celotnega sistema. Če za splošni primer SOO velja, da so funkcije verjetnosti kazni iz okolja odvisne od verjetnosti izbire akcij, torej $c_i(p)$, potem za proces $p(n)$, definiran z L_{R-P}, velja, da je Markovski in ergodičen. Tedaj $p(n)$ konvergira k p^*, ko gre $n \to \infty$, kjer je p^* naključna spremenljivka, neodvisna od $p(0)$.

2.5 Posplošitve učečih avtomatov

Učeči avtomati, ki smo jih obravnavali do sedaj, so izbirali akcije izključno na osnovi lastnih parametrov oz. vektorja verjetnosti za izbiro akcij $p(n)$. Največkrat pa na akcije avtomatov vplivajo zunanji dejavniki, npr. vhodni vektor $\mathbf{x}(n) = (x_1(n), ..., x_H(n))$. Če v učeči avtomat vstopa poleg ocene iz okolja še omenjeni vhodni ali kontekstni vektor $\mathbf{x}$, potem se naloga učečega avtomata spremeni. Cilj učenja je tedaj priti do čim ugodnejše učilne strategije z vsakim kontekstnim vektorjem. Verjetnost izbire akcije je v tem primeru zamenjana z $p(\mathbf{x}(n))$, kar pomeni, da v procesu učenja učeči avtomat asociira (povezuje) akcije s vhodnim kontekstom s ciljem podobno kot prej, izboljšati povprečen odziv iz okolja. Če se kontekstni prostor deli na različna področja, je mogoča uporaba različnih avtomatov v različnih kontekstnih področjih, podobno kot v primeru različnih stacionarnih okolij.

Možno pa je uporabiti en sam učeči avtomat, ki pa potrebuje vektor dodatnih parametrov θ, ki ga je potrebno ažurirati po vsakem koraku procesiranja. Tedaj je izbira akcije učečega avtomata odvisna od $\mathbf{x}$ in θ ($p(\mathbf{x}, \theta)$), odziv okolja pa vpliva na spremembo θ. Takšno učenje učečih avtomatov ima precej podobnosti z učenjem pri umetnih nevronskih mrežah, kar bo podrobneje razloženo v naslednjem poglavju.

Poglavje 3

Umetne nevronske mreže

3.1 Uvod

Umetna nevronska mreža predstavlja model procesiranja podatkov, ki se bistveno razlikuje od procesiranja klasičnega von Neumann-ovega digitalnega računalnika. Če za slednjega velja, da sledi programu, oz. nizu ukazov tako, da izvaja aritmetične, logične, testne in pomnilniške operacije nad podatki, potem je značilnost umetne nevronske mreže, da oponaša delovanje živčnih celic ali nevronov. Zelo pomembna lastnost umetne nevronske mreže je tudi v tem, da predstavlja porazdeljen sistem specialnih (nevronskih) procesorjev, ki delujejo vzporedno. Pri tem se procesiranje (vezano na model nevrona) največkrat izvaja tako, da je najprej na vrsti učenje na znanih (učnih) podatkih oz. učenje preslikav (med pari vhodnih in izhodnih vrednosti), temu pa sledi računanje odzivov na nove vhodne kombinacije na osnovi z učenjem pridobljenega znanja. Nevronske mreže torej na osnovi znanih podatkov v postopku učenja pridobivajo znanje, ki ga nato uporabljajo na novih (do tedaj) neznanih vhodnih primerih.

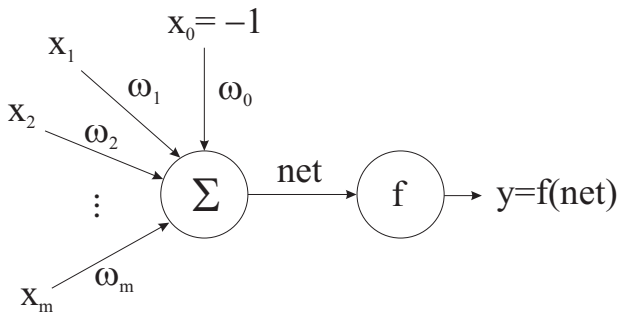
Nevronske mreže izkazujejo visoko stopnjo robustnosti oz. neodvisnosti od napak, ki so posledica realnih meritev vhodno-izhodnih podatkov, kjer je običajno prisoten tudi šum. Za nevronske mreže je torej značilno posploševanje in tolerantnost na možne napake na vhodih. Zaradi porazdeljene strukture nevronske mreže je tudi pridobljeno znanje porazdeljeno (vrednosti uteži v mreži), kar ji daje določeno stopnjo neobčutljivosti od delovanja posameznega nevrona.

Zaradi takšne narave procesiranja so za nevronske mreže primerni predvsem problemi, ki procesirajo t.i. senzorske podatke, oz. podatke, ki so rezultat meritev na vhodu in izhodu neznanega sistema. Takšne probleme običajno imenujemo za klasifikacijske, za probleme razpoznavanja vzorcev, regresijske oz. aproksimacijske probleme in za probleme krmiljenja oz. regulacije.

Prvi model umetnega nevrona je nastal leta 1943. Avtorja Warren McCulloch in Walter Pitts [7] sta tedaj definirala pragovni element, ki je zelo poenostavljeno opisoval delovanje živčne celice ali nevrona. V bistvu je predstavljal analogno-digitalno transformacijo tako, da je vseboval uteženo vsoto vhodnih spremenljivk, ki ji je sledila preprosta pragovna (stopničasta) logična funkcija, ki je odločala o binarni vrednosti na izhodu v odvisnosti od vrednosti utežene vsote:

$$y = f(\sum_i x_i\omega_i) = \begin{cases} 1, & \text{če } \sum_i x_i\omega_i > 0 \\ 0, & \text{sicer} \end{cases}$$

Primer pragovnega elementa oz. prvega matematičnega modela umetnega nevrona prikazuje Slika 3.1.



Slika 3.1: Model McCulloch & Pitts.

Prikazani model je bil deležen precejšnje pozornosti, z manjšimi modifikacijami je en nivo (enonivojska nevronska mreža) podobnih pragovnih elementov dobil ime Perceptron [8], ki je v tistih časih veljal za enega najbolj perspektivnih elementov bodočih adaptivnih oz. inteligentnih sistemov. Žal je v istem času članek avtorjev M. Minsky in S. Papert [9] korektno dokazal, da z zgornjim modelom nevrona ni mogoče rešiti niti enostavne logične funkcije EX-OR in torej tudi ne nobene druge logične funkcije, ki ni linearno ločljiva, oz. pri kateri ni mogoče linearno ločiti vhodnih vektorjev, ki se preslikajo v binarno vrednost 0 od tistih, ki se preslikajo v vrednost 1 na izhodu.

Ponoven zagon je področje nevronskih mrež doživelo z delom avtorjev Rumelhart, Hinton in Williams [10], ki so pokazali, da je mogoče vsako funkcijo realizirati do poljubne natančnosti s pomočjo dovolj velike nevronske mreže, ki ima za vhodno plastjo vsaj dve plasti nevronov: eno skrito plast, ki ni povezana z izhodi, in eno izhodno plast nevronov. Za takšno nevronsko mrežo so izdelali posebni vzvratni (angl. backpropagation) učilni algoritem, ki omogoča iterativno učenje vseh uteži v mreži od izhodne plasti proti vhodu. Zaradi gradientne narave učilnega postopka, ki išče optimalne uteži tako, da je vsota kvadratov napak na izhodu nevronske mreže (napaka je razlika med želeno in dejansko vrednostjo) za vse učne primere minimalna, je bila za izhodno oz. aktivacijsko funkcijo izbrana t.i. sigmoidna funkcija. Le-ta se od pragovne (stopničaste) funkcije razlikuje po tem, da je zglajena in s tem zvezna ter zvezno odvedljiva. Sigmoidno funkcijo predstavlja enačba:

$$y = f(v) = \frac{1}{1 + e^{-v}} \,.$$

Argument sigmoidne funkcije je utežena vsota:

$$v = \sum_{i=1}^{m} \omega_i x_i - \omega_0 = \sum_{i=0}^{m} \omega_i x_i \,.$$

Utež ω_0 (pogosto tudi θ) se imenuje prag (angl. threshold) ali odmik (angl. bias).

Od tedaj dalje so se raziskave umetnih nevronskih mrež nezadržno nadaljevale, tako da danes velja prepričanje, da ni več daleč čas, ko bo s pomočjo razvoja različnih tehnologij mogoče veliko bolje razumeti delovanje človekovih možganov. Razvoj je šel naprej predvsem v smereh novih modelov nevronov, novih topologij nevronskih mrež in novih učilnih algoritmov. Izmed novih modelov nevronov velja omeniti stohastični model, pri katerem dobi nevron na izhodu eno vrednost (npr. $+1$) z verjetnostjo $p(v)$ in drugo vrednost (npr. -1) z alternativno verjetnostjo $1 - p(v)$. Pri tem je $p(v)$ običajno zopet sigmoidna funkcija:

$$p(v) = \frac{1}{1 + e^{-v/T}}$$

kjer je T parameter (temperatura), ki določa stohastično naravo nevrona. V primeru, ko gre T proti 0, ta model prehaja v model McCulloch & Pitts [7]. V zadnjem času je precejšnje pozornosti deležen tudi impulzni model (angl. spiking neuron), ki na izhodu generira verige impulzov različnih frekvenc, glede

na vzbujanje na vhodu. Izmed različnih topologij nevronskih mrež obstajajo tri velike skupine: naprej povezane (angl. feedforward) mreže za kombinatorične ali statične probleme, rekurentne (angl. recurrent) mreže, ki se običajno uporabljajo za dinamične probleme, in specialne nevronske mreže. Med mrežami prve skupine so najbolj znane večnivojski perceptron, nevronski mreži RBF in LVQ, v drugi skupini so rekurentna nevronska mreža, nevronska mreža GARNN, Hopfield-ova mreža, v specialno skupino pa največkrat uvrščamo mreže SOM, Alopex, CRBP itd.

Učilne algoritme nevronskih mrež pa razvrščamo na nadzorovane (angl. supervised), nenadzorovane (angl. unsupervised) in na spodbujevane algoritme (angl. reinforcement). Različne paradigme učenja določajo naravo učenja glede na podatke, ki so na voljo. V kolikor poznamo vhodno-izhodne pare, oz. želene odzive na določene vhodne kombinacije, ki tvorijo učno množico, govorimo o nadzorovanem učenju. Če želenih izhodnih vrednosti ne poznamo, oz. če želimo z učenjem doseči maksimalno prilagoditev na vhodne podatke, govorimo o nenadzorovanem oz. samo-organizirajočem učenju. Če pa lahko za posamezne vhodne kombinacije in odziv mreže nanje podamo le oceno odziva, ki je ali pozitivna (nagrada) ali negativna (kazen), pa govorimo o spodbujevanem učenju.

V nadaljevanju bodo po nekaterih teoretičnih osnovah predstavljene predvsem najzanimivejše vrste nevronskih mrež in ustrezni učilni algoritmi. Opisane bodo naslednje nevronske mreže: večnivojski perceptron, nevronske mreže RBF, LVQ in SOM kot predstavnice kombinatoričnih mrež, Hopfield-ova, rekurentna, GARNN in CRBP iz skupine rekurentnih nevronskih mrež, nato pa še nevronska mreža s spodbujevanim učenjem na osnovi algoritma učečih avtomatov. Sledil bo opis postopka ekstrakcije znanja iz naučene nevronske mreže in vpliv učenja na parametre, ki se uporabljajo pri analizi kompleksnih sistemov. Na koncu bo podan še spisek tipičnih aplikacij, skupaj z nekaterimi ilustracijami.

3.2 Teoretične osnove umetnih nevronskih mrež

3.2.1 Gradientno pravilo učenja LMS

Nezadržen razvoj umetnih nevronskih mrež se je pravzaprav začel z odkritjem vzvratnega pravila učenja (angl. backpropagation), ki omogoča določitev uteži v večnivojski nevronski mreži (angl. multilayer neural network) z iterativnim gradientnim postopkom, ki minimizira srednjo kvadratično napako za vse podatke iz t.i. učne množice, ki predstavlja poznane vhodno-izhodne pare, ki so osnova za učenje. Zato bo najprej podana osnovna ideja, ki je pripeljala do tega pomembnega odkritja in s tem sprožila številne nove raziskave predvsem na področjih modeliranja nevronov, novih topologij, kriterijskih funkcij in učilnih algoritmov, povezovanja umetnih nevronskih mrež s kognitivnimi funkcijami živih organizmov in v zvezi z novimi aplikacijami.

Utežena vsota v osnovnem modelu nevrona je enaka enačbi prostorskega filtra:

$$y = \sum_{i=1}^{m} \omega_i x_i \,.$$

Pri tem je cenilna funkcija, ki spremlja učenje nevronske mreže, običajno podana z:

$$J = \frac{1}{2} E[e^2] = \frac{1}{2} E[(d-y)^2] \,,$$

kjer je E oznaka za statistični operator pričakovanja (srednja vrednost), e oznaka za napako na izhodu mreže za posamezni vhodni primer, d je želena (angl. desired) izhodna vrednost, y pa dejanska (angl. actual) izhodna vrednost nevronske mreže. Če želimo rešitev prostorskega filtra za določen nabor p vhodnih vektorjev $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, .., \mathbf{x}_p)$, kjer je $\mathbf{x}_i = (\mathbf{x}_{i1}, \mathbf{x}_{i2}, ..., \mathbf{x}_{im})$, potem rešitev določa Wiener-jev filter, do katerega pridemo na naslednji način. Če izraz za uteženo vsoto y in za napako e vstavimo v J, dobimo:

$$\begin{aligned} J &= \frac{1}{2} E[d^2] - E[\sum_{i=1}^{m} \omega_i x_i d] + \frac{1}{2} E[\sum_{j=1}^{m} \sum_{i=1}^{m} \omega_j \omega_i x_j x_i] \\ &= \frac{1}{2} E[d^2] - \sum_{i=1}^{m} \omega_i E[x_i d] + \frac{1}{2} \sum_{j=1}^{m} \sum_{i=1}^{m} \omega_j \omega_i E[x_j x_i] \,, \end{aligned}$$

kjer je:

$E[d^2] = r_d$ ... srednji kvadrat želenega odziva d

$E[dx_i] = r_{dx}(i)$ ... križna korelacija med želenim odzivom in i-tim vhodom

$E[x_j x_i] = r_x(j,i)$ ... avtokorelacija vhodov

Z uporabo novih oznak je cenilna funkcija J enaka:

$$J = \frac{1}{2} r_d - \sum_{i=1}^{m} \omega_i r_{dx}(i) + \frac{1}{2} \sum_{j=1}^{m} \sum_{i=1}^{m} \omega_j \omega_i r_x(j,i) \,.$$

Njen odvod po poljubni uteži ω_i je:

$$\frac{\partial J}{\partial \omega_i} = -r_{dx}(i) + \sum_{j=1}^{m} \omega_j r_x(j,i) \qquad i = 1, 2, ..., m \,.$$

Če iščemo uteži pri minimalni vrednosti cenilne funkcije, potem vrednosti uteži izhajajo iz enačbe, v kateri zgornji odvod izenačimo z 0:

$$\frac{\partial J}{\partial \omega_i} = 0 \quad \rightarrow \quad \sum_{j=1}^{m} \omega_{oj} r_x(j,i) = r_{dx}(i) \qquad i = 1, 2, ..., m \,.$$

Slednja enačba se imenuje Wiener-Hopf-ova enačba in predstavlja rešitev enačbe prostorskega filtra. Oznaka uteži ω_{oj} pomeni optimalno vrednost za utež ω_j. Rešitev za uteži ω_j, $j = 1, 2, ..., m$, zahteva računanje inverzne matrike avtokorelacijskih funkcij r_x, kar je v primeru velikih matrik problem, tudi zato, ker se zahtevajo vsi učni podatki naenkrat. Zato je mogoče namesto natančne matematične rešitve uporabiti približno rešitev, ki temelji na metodi najbolj strmega sestopa (angl. steepest descent). Po tej metodi se uteži ažurirajo iterativno. Omenjena metoda zagotavlja najmanjši srednji kvadrat napake nad učno množico, zato se imenuje LMS (angl. least mean squares). Namesto avtokorelacijskih in križnih korelacijskih funkcij uporablja približke:

$$\begin{aligned} \hat{r}_x(j,i;n) &= x_j(n) x_i(n) \,, \\ \hat{r}_{dx}(i;n) &= x_i(n) d(n) \,, \end{aligned}$$

kjer je n oznaka za diskretni čas, znak $\hat{}$ pa pomeni oceno oz. približno vrednost.

Ponovimo v tem trenutku na kratko, kaj je to gradient. Gradient skalarne funkcije je vektor, ki kaže v smeri najbolj strmega vzpona funkcije. Gradient skalarne funkcije J parametrov ω_1, ω_2, ...

$$\nabla J = (\frac{\partial J}{\partial \omega_1}, \frac{\partial J}{\partial \omega_2}, ...)$$

je vektor, katerega komponente so delni oz. parcialni odvodi funkcije J po njenih parametrih. Negativni gradient pa kaže v smeri najbolj strmega sestopa funkcije J:

$$-\nabla J = (-\frac{\partial J}{\partial \omega_1}, -\frac{\partial J}{\partial \omega_2}, ...)\,.$$

Če želimo pri danih parametrih ω_i le-te spremeniti za nek majhen korak tako, da se bo cenilna funkcija J čimbolj zmanjšala, jih moramo spremeniti v smeri negativnega gradienta:

$$\Delta \boldsymbol{\omega} = -\eta \nabla J\,,$$

pri čemer je $\boldsymbol{\omega}$ vektor parametrov ω_i, η pa je majhna vrednost, ki določa za kolikšen delež gradienta se spremeni utež. Potrebna sprememba posameznega parametra je torej

$$\Delta \omega_i = -\eta \frac{\partial J}{\partial \omega_i}\,, \qquad \text{za vse } i = 1, 2...$$

Posamezna komponenta $\frac{\partial J}{\partial \omega_i}$ gradienta se označuje tudi z $\nabla_{\omega_i} J$.

Sprememba uteži pri metodi LMS je tako enaka negativnemu odvodu cenilke:

$$\Delta \omega_i(n) = -\eta \nabla_{\omega_i} J(n) = -\eta \frac{\partial J(n)}{\partial \omega_i(n)}\,, \qquad i = 1, ..., m\,.$$

kjer je η učilni parameter. Nova vrednost uteži ω_i v času $n+1$ je tako enaka:

$$\begin{aligned} \omega_i(n+1) &= \omega_i(n) + \Delta\omega_i(n) \\ &= \omega_i(n) - \eta \frac{\partial J(n)}{\partial \omega_i(n)} \\ &= \omega_i(n) + \eta \Big[r_{dx}(i) - \sum_j \omega_j(n) r_x(j,i) \Big], \qquad i = 1, ..., m, \end{aligned}$$

oz. s približno oceno (LMS):

$$\begin{aligned}\hat{\omega}_i(n+1) &= \hat{\omega}_i(n) + \eta\Big[x_i(n)d(n) - \sum_{j=1}^{m}\hat{\omega}_j(n)x_j(n)x_i(n)\Big] \\ &= \hat{\omega}_i(n) + \eta\Big[d(n) - \sum_{j=1}^{m}\hat{\omega}_j(n)x_j(n)\Big]x_i(n) \\ &= \hat{\omega}_i(n) + \eta[d(n) - y(n)]x_i(n), \qquad i = 1, ..., m\,,\end{aligned}$$

ki jo imenujemo pravilo delta zaradi razlike v oglatem oklepaju, ali tudi Widrow-Hoff-ovo pravilo učenja. To pravilo je osnova za številna izpeljana pravila, kar bomo spoznali v nadaljevanju.

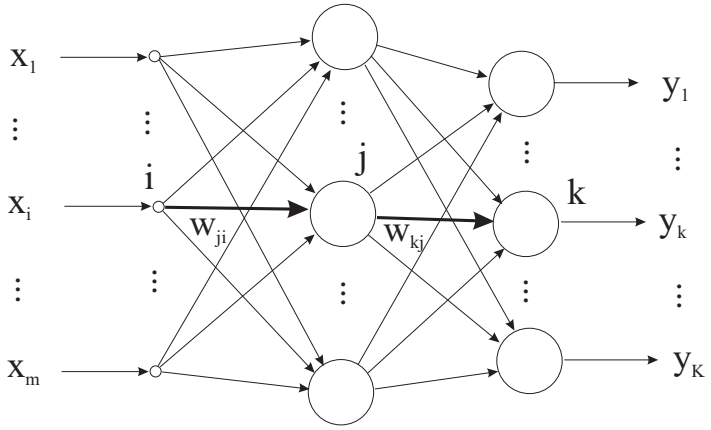
3.2.2 Večnivojski perceptron in pravilo vzvratnega učenja

Ena najpogosteje uporabljenih nevronskih mrež je večnivojski perceptron ali MLP (angl. multilayer perceptron). Zanj je bilo dokazano [11], da je z njim mogoče realizirati poljubno vhodno-izhodno preslikavo, če je le število nivojev dva ali več in če je v vsakem nivoju dovolj nevronov. Posplošitev Widrow-Hoff-ovega pravila na primer nevronske mreže MLP je nadzorovano (angl. supervised) vzvratno učilno pravilo BPG (angl. backpropagation learning algorithm), ki omogoča, da na osnovi učne množice vrednosti vhodno-izhodnih spremenljivk, določimo vse uteži v mreži tako, da je srednji kvadrat napake na izhodu mreže za uporabljeno učno množico minimalen.

O konvergenci procesa učenja v zvezi z velikostjo učne množice govori Vapnik-Červonenkis-ova (VC) dimenzija [11], ki pravi, da za uspešno učenje z natančnostjo $(1-\varepsilon)\cdot 100\%$ potrebujemo $N_{min} \approx W/\varepsilon$ vhodno-izhodnih parov v učni množici, kjer je W celotno število uteži v mreži.

Vzemimo dvonivojsko naprej povezano nevronsko mrežo oz. MLP, ki jo prikazuje Slika 3.2. Na vhodu je m vhodnih spremenljivk $x = (x_1, x_2, ..., x_m)$, ki so vse povezane na vse nevrone skritega nivoja (angl. hidden layer) $y^h = (y_1, y_2, ..., y_L)$, nevroni skritega nivoja pa so vsi povezani na vse nevrone izhodnega nivoja (angl. output layer) $y^o = (y_1, y_2, ..., y_K)$.

Vsako utež v večnivojski nevronski mreži MLP v vsakem koraku učenja spre-



Slika 3.2: Topologija dvonivojske nevronske mreže MLP.

menimo v skladu z enačbo:

$$\omega(n+1) = \omega(n) + \Delta\omega(n)\,.$$

Na ta način ne minimiziramo natančnega gradienta napake na učni množici, ampak njegov približek. Ta metoda se imenuje sprotno učenje (angl. on-line learning) in je malo enostavnejša od paketnega učenja (angl. batch learning), pri katerem minimiziramo natančen gradient.

Najprej spremenimo uteži nevronov izhodne plasti, temu sledijo spremembe skritih plasti. Če je skritih plasti več, potem so najprej na vrsti nevroni v skriti plasti, ki meji na izhodno plast, nato nevroni v naslednji skriti plasti proti vhodu, itd. Ker se spreminjanje uteži izvaja od izhodne plasti proti vhodu v mrežo, se postopek imenuje vzvratni učilni algoritem (angl. backpropagation) oz. kratko BPG.

V primeru uteži, ki vstopajo v izhodno plast, je sprememba enaka:

$$\Delta\omega(n) = -\eta\frac{\partial E(n)}{\partial\omega(n)}$$

kjer je $E(n)$ kvadratna napaka na n-tem vzorcu učne množice:

$$E(n) = \frac{1}{2}\sum_{k=1}^{K} e_k^2(n) = \frac{1}{2}\sum_{k=1}^{K}(d_k(n) - y_k(n))^2\,.$$

Pri tem je e_k napaka oz. razlika med želeno (d_k) in dejansko (y_k) vrednostjo k-tega izhodnega nevrona pri n-ti preslikavi v učni množici (angl. learning set, training set, teaching set). Faktor 1/2 nastopa zaradi kasnejše nevtralizacije konstante 2, ki izhaja iz operacije odvajanja izraza E. Skupna napaka preko celotne učne množice je:

$$E = \sum_{p=1}^{P} E(n)\,,$$

kjer je P število vzorcev učne množice.

Ob uporabi verižnega pravila odvajanja dobimo za poljubno utež, ki vstopa v izhodno plast nevronov, končni izraz:

$$\Delta\omega_{kj}(n) = -\eta e_k(n)(-1)f'(v_k(n))y_j(n) = \eta\delta_k(n)y_j(n)\,,$$

kjer indeks uteži $_{kj}$ pomeni, da gre za poljubno utež, ki povezuje nevron v skriti plasti y_j z izhodnim nevronom y_k , f' pa je odvod sigmoidne funkcije

$$y = f(v) = \frac{1}{1 + e^{-v}}$$

in je enak:

$$f'(v) = y(n)(1 - y(n))\,.$$

$\delta_k(n)$ je nadomestni izraz (delta) za:

$$\delta_k(n) = e_k(n)f'(v_k(n)) = (d_k(n) - y_k(n))y_k(n)(1 - y_k(n))\,.$$

Za uteži, ki vstopajo v katerikoli skriti nivo nevronov, pa velja naslednje. Vpliv izhodne napake se prenaša preko vseh uteži izhodnega nivoja, zato je za slučaj

nevronov skrite plasti pred izhodno spremememba uteži enaka:

$$\begin{aligned}
\Delta\omega_{ji}(n) &= -\eta\frac{\partial E(n)}{\partial\omega_{ji}(n)} \\
&= -\eta\frac{\partial E(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial\omega_{ji}(n)} \\
&= -\eta\left(\sum_k e_k(n)\frac{\partial e_k(n)}{\partial y_j(n)}\right)\frac{\partial y_j(n)}{\partial v_j(n)}\frac{\partial v_j(n)}{\partial\omega_{ji}(n)} \\
&= -\eta\left(\sum_k e_k(n)(-1)f'(v_k(n))\omega_{kj}(n)\right)f'(v_j(n))y_i(n) \\
&= \eta\left(\sum_k \delta_k(n)\omega_{kj}(n)\right)f'(v_j(n))y_i(n) \\
&= \eta\delta_j(n)y_i(n)\,,
\end{aligned}$$

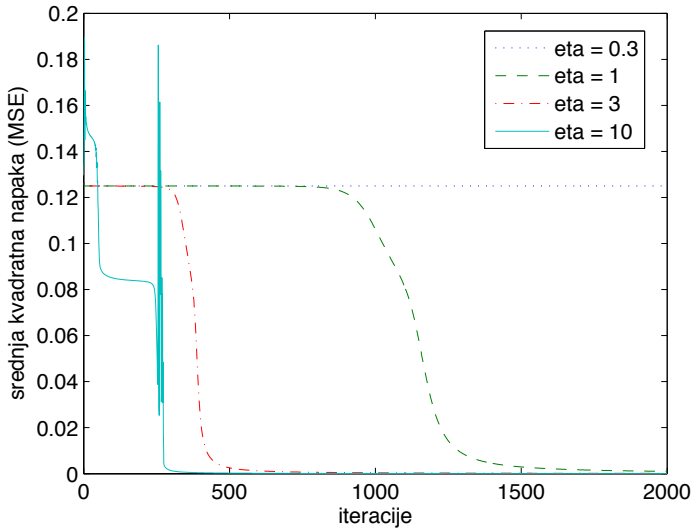
kjer je sedaj funkcija delta vsota preko vseh izhodnih nevronov:

$$\begin{aligned}
\delta_j(n) &= f'(v_j(n))\left(\sum_k \delta_k(n)\omega_{kj}(n)\right) \\
&= y_j(n)(1-y_j(n))\left(\sum_k \delta_k(n)\omega_{kj}(n)\right)\,.
\end{aligned}$$

V primeru, da je več skritih plasti, se spreminjanje uteži njihovih nevronov izvaja na podoben način kot zgoraj, le namesto izhodnih nevronov nastopajo nevroni predhodne (od zadaj) skrite plasti, namesto napake pa funkcije delta (δ) posameznih nevronov te plasti.

Primer: Učenje večnivojskega perceptrona z algoritmom vzvratnega učenja

Poglejmo za ilustracijo, kako se večnivojski perceptron nauči na logično funkcijo EX-OR dveh spremenljivk. Hitrost konvergence je odvisna od izbire učilnega parametra η. Slika 3.3 prikazuje potek srednje kvadratne napake (MSE) pri različnih vrednostih učilnega parametra η: 0.3, 1.0, 3.0 in 10.0. Učilni parameter seveda vpliva na hitrost konvergence. Iz Tabele 3.1 lahko vidimo, kako se naučena mreža odziva na posamezne učne vzorce. Uteži naučene mreže prikazuje Slika 3.4.



Slika 3.3: Potek srednje kvadratne napake (MSE) pri različnih vrednostih η.

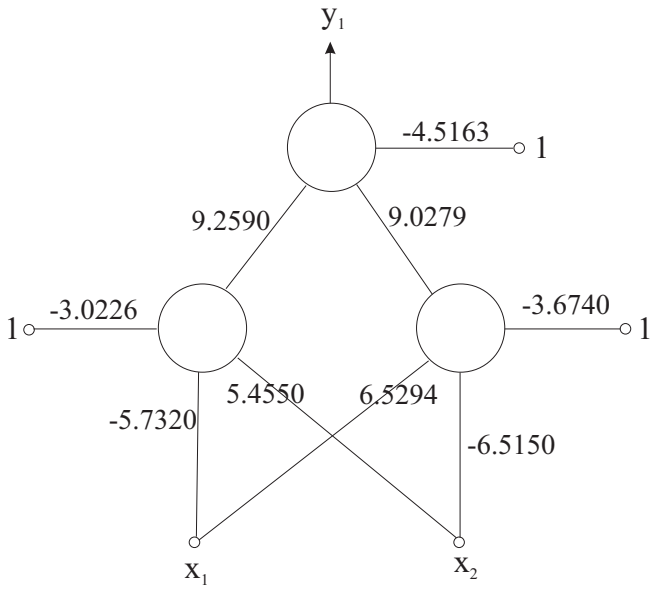
x_1	x_2	d_1	y_1
0	0	0	0.0206
0	1	1	0.9819
1	0	1	0.9824
1	1	0	0.0187

Tabela 3.1: Želeni in dejanski izhodi večnivojskega perceptrona, naučenega na logično funkcijo EX-OR, pri $\eta = 3.0$.

3.2.3 Nevronska mreža RBF

Učenje nevronske mreže MLP v prejšnjem razdelku je mogoče predstaviti kot rekurzivno metodo ali stohastično aproksimacijo. V primeru mreže RBF [14] pa lahko na razvoj (učenje) mreže gledamo kot na problem funkcijskega prileganja ali aproksimacije v večdimenzionalnem prostoru. Nevroni skritega nivoja v primeru RBF (angl. radial basis functions) predstavljajo niz baznih funkcij, ki tvorijo bazo (angl. basis) oz. bazni prostor za vhodni vektor ali vzorec, ko se le-ta preslika v skriti nivo. Zaradi simetrične oblike se funkcije skritega nivoja imenujejo radialne bazne funkcije.

V osnovi ima mreža RBF tri plasti ali nivoje. Vhodna plast vsebuje senzorske

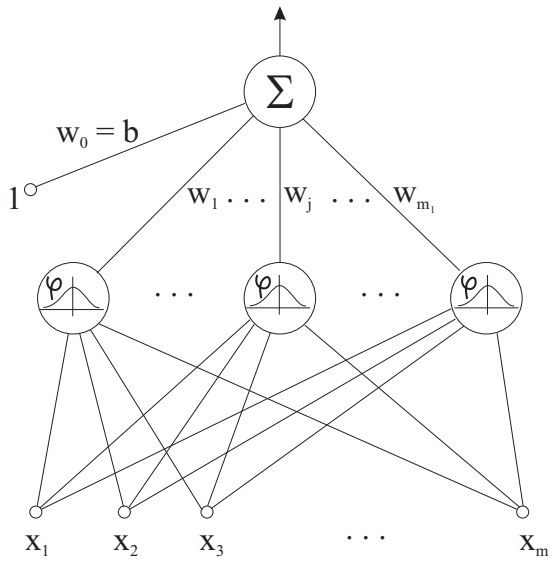


Slika 3.4: Uteži mreže, naučene na EX-OR.

(vhodne) spremenljivke, ki povezujejo mrežo z okoljem. Druga plast oz. skriti nivo mreže, je odgovorna za nelinearno transformacijo iz vhoda v skrito plast nevronov in je običajno visoko-dimenzionalna. Tretja ali izhodna plast predstavlja linearno transformacijo nad funkcijami skritega nivoja in je odgovorna za odziv mreže na vhodni vzorec. Matematična razlaga za takšno strukturo RBF mreže leži v t.i. Cover-jevem teoremu, ki dokazuje, da verjetnost obstoja linearne ločljivosti vzorcev raste z dimenzionalnostjo vzorčnega prostora.

Slika 3.5 prikazuje strukturo oz. topologijo nevronske mreže RBF. Obstaja več različnih strategij učenja RBF mrež. Pri vseh pa se linearne uteži izhodnega nivoja v splošnem spreminjajo z različno časovno dinamiko kot nelinearne funkcije skritega nivoja. Medtem ko se slednje običajno razvijajo počasi in skladno z določeno nelinearno optimizacijsko strategijo, pa se izhodne uteži lahko spreminjajo hitro v okviru linearne strategije. Različni nivoji torej izvajajo različne naloge, zato je razumljivo, da so tudi strategije in časovne skale sprememb največkrat različne.

Ogledali si bomo tri strategije učenja: strategijo z naključnim izborom fiksnih centrov baznih funkcij, strategijo s samo-organiziranim izborom centrov in strategijo nadzorovanega izbora centrov.



Slika 3.5: Topologija nevronske mreže RBF.

A. NAKLJUČNI IZBOR FIKSNIH CENTROV

Najenostavnejši pristop k učenju je v predpostavki, da so bazne funkcije fiksne, tako da je predmet učenja le naključna izbira centrov za bazne funkcije, kar je opravičljivo, kadar predpostavimo reprezentativno porazdeljeno učno množico v vhodni domeni [12]. Za radialne bazne funkcije so običajno izbrane normalne ali Gauss-ove porazdelitve (G), s fiksnimi standardnimi deviacijami in centri v točkah $\mathbf{t}_i$:

$$G(\mathbf{x}, \mathbf{t}_i) = e^{-\frac{m_1}{d_{max}^2}||\mathbf{x}-\mathbf{t}_i||^2}, \qquad i = 1, 2, ..., m_1\,,$$

kjer je m_1 število centrov (parameter učenja), d_{max} maksimalna razdalja med izbranimi centri, $\mathbf{x}$ vhodni vektor in $\mathbf{t}_i$ center i-te bazne funkcije. Pri tem je standardna deviacija (širina porazdelitve) za vse normalne bazne funkcije konstantna in enaka:

$$\sigma = \frac{d_{max}}{\sqrt{2m_1}}\,.$$

Zgornji izraz za standardno deviacijo zagotavlja, da posamezne bazne funkcije niso preveč ozke ali preveč široke. Ta izraz je mogoče za posebne primere prilagoditi tako, da izberemo pri redkih podatkih širšo obliko, pri bolj gostih pa ožjo.

Edini parametri, ki so podvrženi učenju v tej strategiji, so linearne uteži v

izhodni plasti nevronov. Direktni postopek uporablja psevdoinverzno metodo:

$$\mathbf{\Omega} = \mathbf{G}^{+}\mathbf{d}\,,$$

kjer je $\mathbf{d}$ želeni izhodni vektor v učnem nizu, $\mathbf{\Omega}$ matrika uteži v izhodni plasti in $\mathbf{G}^{+}$ psevdoinverz matrike $\mathbf{G}$, katere splošni člen je enak:

$$g_{ji} = e^{-\frac{m_1}{d_{max}^2}||\mathbf{x}_j - \mathbf{t}_i||^2}, \qquad j = 1, 2, ..., N; \quad i = 1, 2, ..., m_1\,,$$

in kjer je $\mathbf{x}_j$ j-ti vhodni vektor učnega vzorca. Računanje matrike $\mathbf{G}^{+}$ zahteva postopek SVD (angl. singular value decomposition), po katerem je:

$$\begin{aligned}
\mathbf{G}^{+} &= \mathbf{V}\mathbf{\Sigma}^{+}\mathbf{U}^{T}\,, \\
\mathbf{U} &= \{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_N\} \\
\mathbf{V} &= \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_M\} \\
\mathbf{U}^{T}\mathbf{G}\mathbf{V} &= \mathrm{diag}(\sigma_1, \sigma_2, ..., \sigma_K), \qquad K = \min(M, N)\,,
\end{aligned}$$

kjer so stolpci matrike $\mathbf{U}$ levi singularni vektorji matrike $\mathbf{G}$ in stolpci matrike $\mathbf{V}$ desni singularni vektorji matrike $\mathbf{G}$; $\sigma_1, \sigma_2, ..., \sigma_K$ so singularne vrednosti matrike $\mathbf{G}$, matrika $\mathbf{\Sigma}^{+}$ reda $N \times N$ pa je definirana z:

$$\mathbf{\Sigma}^{+} = \mathrm{diag}(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, ..., \frac{1}{\sigma_K}, 0, ..., 0)\,.$$

Detajlnejši opis postopka SVD je podan npr.v [13].

B. SAMO-ORGANIZIRAJOČA IZBIRA CENTROV BAZNIH FUNKCIJ

Problem metode s fiksnimi centri v prejšnji strategiji je v tem, da lahko zahteva obsežen učni niz za dosego zadovoljivih rezultatov. Slednje prepreči hibridno učenje z dvema stopnjama [14]:

- Samo-organizirajoča stopnja učenja, katere namen je oceniti ustrezne lokacije za centre baznih funkcij v skritem nivoju nevronov.
- Nadzorovana stopnja učenja, ki oceni linearne uteži izhodnega nivoja nevronov in s tem zaključi proces učenja.

Takšno učenje v dveh korakih je mogoče izvesti naenkrat (angl. batch processing approach) ali pa iterativno (angl. adaptive approach).

Za samo-organizirajoč proces učenja je potreben algoritem razvrščanja (angl. clustering algorithm), ki razdeli množico podatkov na skupine, od katerih naj

bo vsaka čim bolj homogena. Eden od bolj znanih algoritmov je *razvrščanje k-tih povprečij* (angl. k-means clustering) [16], ki postavi centre radialnih baznih funkcij v tista področja vhodne domene, kjer se nahajajo pomembni podatki. Določitev m_1 baznih funkcij zato zahteva naslednje iskanje oz. eksperimentiranje:

1. *Inicializacija*: Določitev naključnih vrednosti za začetne centre $\mathbf{t}_k(0)$, $k = 1, 2, ..., m_1$. Edina omejitev pri tem je, da so začetne vrednosti različne in da so Evklidske razdalje med centri majhne.

2. *Vzorčenje*: Izbira vzorčnega vektorja iz vhodne domene z določeno verjetnostjo. Vektor $\mathbf{x}$ je vhod v algoritem pri n-ti iteraciji.

3. *Primerjava ujemanja*: Naj bo $k(\mathbf{x})$ indeks najboljšega (zmagovalnega) centra pri ujemanju z vhodnim vektorjem $\mathbf{x}$. Določitev $k(\mathbf{x})$ pri iteraciji n z uporabo kriterija minimalne Evklidske razdalje:

$$k(\mathbf{x}) = \text{argmin}_k(||\mathbf{x}(n) - \mathbf{t}_k(n)||)\,, \qquad k = 1, 2, ..., m_1\,,$$

   kjer je $\mathbf{t}_k(n)$ center k-te bazne funkcije v iteraciji n.

4. *Ažuriranje*: Sprememba centrov baznih funkcij na osnovi izraza:

$$\mathbf{t}_k(n+1) = \begin{cases} \mathbf{t}_k(n) + \eta[\mathbf{x}(n) - \mathbf{t}_k(n)]\,, & k = k(\mathbf{x}) \\ \mathbf{t}_k(n)\,, & \text{sicer} \end{cases}$$

5. *Nadaljevanje*: Povečanje n za 1 in nadaljevanje pri koraku 2, dokler ni več zaznavnih sprememb centrov baznih funkcij skritega nivoja RBF.

Opisani algoritem razvrščanja *k-means* je poseben primer tekmovalnega (angl. winner-take-all) učnega procesa, ki bo podan v nadaljevanju (LVQ, SOM). Omejitev opisanega algoritma razvrščanja je v tem, da lahko doseže le lokalno optimalno rešitev, ki je odvisna od začetne izbire centrov. Slednje lahko vodi k nepotrebno veliki mreži. Rešitev predstavlja izboljšani algoritem razvrščanja *enhanced k-means clustering* [17], ki temelji na spremenljivi utežni meri (angl. variation-weighted measure) za skupine, kar omogoča algoritmu, da konvergira k sub-optimalni konfiguraciji, neodvisno od začetno izbranih baznih centrov.

Naslednji korak je ocena uteži izhodne plasti. Preprosta metoda za takšno oceno je algoritem LMS, ki je bil opisan v prejšnjem poglavju. Pri tem vektor skritega nivoja predstavlja vhodni vektor za algoritem LMS. Oba algoritma,

ki določata spremembe v skriti in izhodni plasti, lahko delujeta sočasno, kar pospeši učilni proces.

C. NADZOROVANA IZBIRA CENTROV

Tukaj so tako centri baznih funkcij kot izhodne uteži predmet nadzorovanega učnega procesa, zato je to najsplošnejša strategija učenja RBF nevronske mreže. Naravni kandidat za takšen proces je učenje s korekcijo napak, ki je lahko izveden z gradientno metodo LMS, tako kot pri MLP mrežah [15].

Prvi korak je zato določitev cenilne funkcije (vsote napak):

$$E = \frac{1}{2}\sum_{j=1}^{N} e_j^2 \,,$$

kjer je N velikost učne množice in e_j zopet napaka na izhodu mreže pri j-tem vhodnem vzorcu:

$$e_j = d_j - \sum_{i=1}^{M} \omega_i G(||\mathbf{x}_j - \mathbf{t}_i||_{\mathbf{C}_i}) \,.$$

Pri tem je vrednost Gaussove RBF enaka:

$$\begin{aligned} G(||\mathbf{x}_j - \mathbf{t}_i||_{\mathbf{C}_i}) &= e^{-(\mathbf{x}_j-\mathbf{t}_i)^T \mathbf{C}_i^T \mathbf{C}_i (\mathbf{x}_j-\mathbf{t}_i)} \\ &= e^{-\frac{1}{2}(\mathbf{x}_j-\mathbf{t}_i)^T \mathbf{\Sigma}_i^{-1} (\mathbf{x}_j-\mathbf{t}_i)} \,, \end{aligned}$$

kjer velja identiteta:

$$\frac{1}{2}\mathbf{\Sigma}_i^{-1} = \mathbf{C}_i^T \mathbf{C}_i \,.$$

Cilj učenja je poiskati proste parametre ω_i, $\mathbf{t}_i$ in $\mathbf{\Sigma}^{-1}$ tako, da postane napaka E minimalna. Z uporabo metode LMS dobimo naslednje rezultate (spremembe) za:

1. Linearne uteži v izhodni plasti: $\omega_i(n+1) = \omega_i(n) - \eta_1 \frac{\partial E(n)}{\partial \omega_i(n)}$

$$\frac{\partial E(n)}{\partial \omega_i(n)} = \sum_{j=1}^{N} e_j(n) G(||\mathbf{x}_j - \mathbf{t}_i(n)||_{C_i})$$

2. Pozicije centrov v skriti plasti: $\mathbf{t}_i(n+1) = \mathbf{t}_i(n) - \eta_2 \frac{\partial E(n)}{\partial \mathbf{t}_i(n)}$

$$\frac{\partial E(n)}{\partial \mathbf{t}_i(n)} = 2\omega_i(n) \sum_{j=1}^{N} e_j(n) G'(||\mathbf{x}_j - \mathbf{t}_i(n)||_{C_i}) \mathbf{\Sigma}_i^{-1} [\mathbf{x}_j - \mathbf{t}_i(n)]$$

3. Širino baznih funkcij: $\mathbf{\Sigma}_i^{-1}(n+1) = \mathbf{\Sigma}_i^{-1}(n) - \eta_3 \frac{\partial E(n)}{\partial \mathbf{\Sigma}_i^{-1}(n)}$

$$\begin{aligned} \frac{\partial E(n)}{\partial \mathbf{\Sigma}_i^{-1}(n)} &= -\omega_i(n) \sum_{j=1}^{N} e_j(n) G'(||\mathbf{x}_j - \mathbf{t}_i(n)||_{C_i}) \mathbf{Q}_{ji}(n) \\ \mathbf{Q}_{ji}(n) &= [\mathbf{x}_j - \mathbf{t}_i(n)][\mathbf{x}_j - \mathbf{t}_i(n)]^T \end{aligned}$$

V zgornjih treh korekcijskih enačbah so η_i , $i = 1, 2, 3$, učilni parametri.

3.2.4 Hebb-ovo učenje

Osnovna značilnost tega učenja je, da je *lokalno*, kar pomeni, da nevroni črpajo vso informacijo samo od sosednjih nevronov. Ta tip učenja je bil eksperimentalno potrjen, kar pomeni, da je *biološko utemeljen.*

Izhod y nevrona je utežena vsota vhodov

$$y(t) = \sum_{j=1}^{m} \omega_j x_j(t) \,. \tag{3.1}$$

Nevron je prikazan na Sliki 3.6. Pri Hebb-ovem učenju se uteži spremenijo sorazmerno s produktom ustreznega vhoda in izhoda:

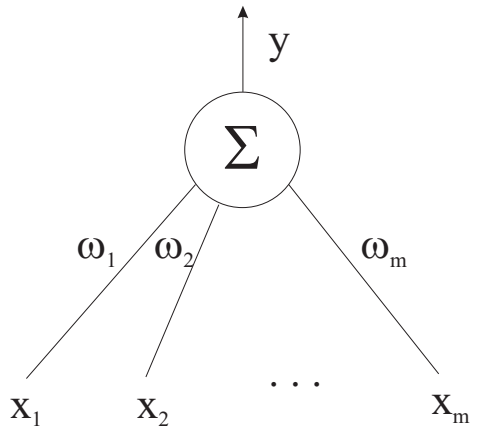
$$\omega_j(t+1) = \omega_j(t) + \rho y(t) x_j(t) \,, \qquad j = 1, ..., m \,.$$

Vektorsko lahko zapišemo Hebb-ovo učno pravilo takole:

$$\boldsymbol{\omega}(t+1) = \boldsymbol{\omega}(t) + \rho y(t) \mathbf{x}(t) \,.$$

Takšno učenje bi lahko vodilo k neskončno velikim utežem, zato obstajajo variante, ki uteži normirajo po vsakem koraku učenja:

$$\boldsymbol{\omega}(t+1) = \frac{\boldsymbol{\omega}(t) + \rho y(t) \mathbf{x}(t)}{\|\boldsymbol{\omega}(t) + \rho y(t) \mathbf{x}(t)\|} \,.$$



Slika 3.6: Nevron pri Hebb-ovem učenju.

Alternativno varianto Hebb-ovega učenja je s ciljem stabiliziranja postopka predlagal Oja. Če je ρ zelo majhen, $|\rho| \ll 1$, lahko zgornji izraz razvijemo v Taylor-jevo vrsto:

$$\begin{aligned}
\omega_j(t+1) &= \omega_j(t) + \rho y(t) x_j(t) - \rho y(t)^2 \omega_j(t) + O(\rho^2) \\
&= \omega_j(t) + \rho y(t) \Big[x_j(t) - y(t)\omega_j(t) \Big] + O(\rho^2) \\
&\doteq \omega_j(t) + \rho y(t) \Big[x_j(t) - y(t)\omega_j(t) \Big] \\
&= \omega_j(t) + \rho y(t) x'_j(t) \, .
\end{aligned}$$

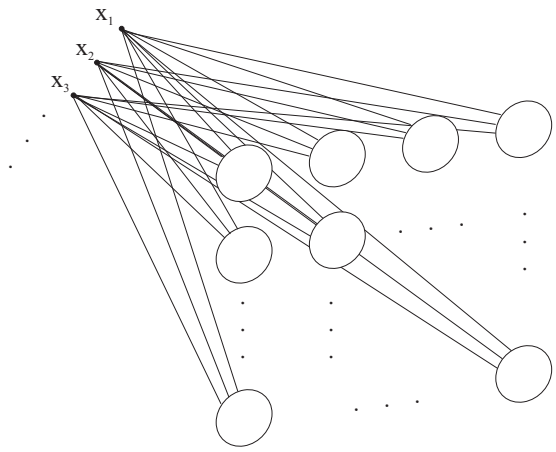
Vidimo, da člen x'_j nadomesti originalni člen x_j.

Rezultat takšnega postopka učenja je glavni lastni vektor (angl. principal eigenvector) - vektor z največjo lastno vrednostjo - avtokorelacijske matrike $\mathbf{C}$, ki jo določa izraz:

$$\mathbf{C} = < \mathbf{x}\mathbf{x}^T > \, .$$

3.2.5 Mreža samo-organizirajoče preslikave - SOM

Osnovni cilj nevronske mreže SOM (angl. self-organizing map) je v transformaciji vhodnega vzorca poljubne dimenzije v eno ali dvo-dimenzionalno preslikavo (angl. map), izvedeni na adaptiven in topološko urejen način [18]. Slika 3.7



Slika 3.7: Dvodimenzionalna mreža SOM.

prikazuje dvo-dimenzionalno mrežo nevronov, ki se običajno uporablja v omenjeni preslikavi. Vsak nevron v mreži je polno povezan z vsemi vhodnimi spremenljivkami. Zato ta mreža predstavlja vnaprej povezano strukturo z eno plastjo nevronov, urejeno v obliki matrike. Poseben primer je eno-dimenzionalna SOM mreža, kjer je plast nevronov reducirana na vrstico ali stolpec nevronov.

Algoritem samo-organizirajoče preslikave najprej inicializira uteži v mreži, kar pomeni, da se jim priredijo poljubne majhne vrednosti. Sledijo naslednji trije koraki procesiranja: tekmovalni, kooperativni in adaptivni proces.

Tekmovalni proces: za vsak vhodni vzorec ali vektor, nevroni v mreži izračunajo vrednosti ustreznih diskriminantnih funkcij, ki predstavljajo osnovo za tekmovanje med nevroni. Nevron z največjo vrednostjo diskriminantne funkcije je zmagovalec tekmovanja. V primeru m vhodnih spremenljivk in ustrezno m utežnih komponent vsakega nevrona, je ena od najpogostejših diskriminantnih funkcij kar skalarni produkt:

$$\mathbf{w}_j^T \mathbf{x}\,,$$

kjer je $\mathbf{w}_j = (w_{j1}, w_{j2}, ..., w_{jm})^T$ utežni vektor j-tega nevrona in $\mathbf{x} = (x_1, x_2, ..., x_m)^T$ vhodni vektor. Ker velja, da je maksimalni skalarni produkt ekvivalenten minimalni Evklidski razdalji med utežnim in vhodnim vektorjem, slednje pomeni, da ima zmagovalni nevron utežni vektor, ki je najbližji vhodnemu vektorju. Za zmagoviti nevron torej velja:

$$i(\mathbf{x}) = \text{argmin}_j(||\mathbf{x} - \mathbf{w}_j||)\,, \qquad j = 1, 2, ..., N,$$

kjer je N število nevronov v mreži.

Kooperativni proces: zmagovalni nevron določa center topološke sosednosti kooperativnih nevronov. Pri tem topološko sosednost določa povezava bližnjih nevronov v okolici zmagovitega nevrona. Tipična funkcija topološke sosednosti je normalna porazdelitev. Če je i-ti nevron zmagovalec tekmovanja in so kooperativni nevroni označeni z j, potem je funkcija normalne topološke sosednosti podana z izrazom:

$$h_{j,i(\mathbf{x})} = e^{-\frac{d_{j,i}^2}{2\sigma^2}}\,.$$

V zgornjem izrazu je σ širina topološke sosednosti in določa mejo, do katere nevroni v bližini zmagovitega nevrona sodelujejo v procesu učenja (kar močno vpliva na hitrejšo konvergenco učenja) in $d_{j,i}$ razdalja med zmagovitim nevronom i in nevronom j v topološki soseščini, torej razdalja v izhodnem prostoru, ki je definirana kot:

$$d_{j,i}^2 = ||\mathbf{r}_j - \mathbf{r}_i||^2\,,$$

kjer je $\mathbf{r}_j$ diskretni vektor, ki definira pozicijo nevrona j, in $\mathbf{r}_i$ vektor, ki določa diskretno pozicijo zmagovitega nevrona i, obe merjeni v diskretnem izhodnem prostoru. Pomembna lastnost algoritma SOM je tudi v tem, da se širina topološke soseščine σ zmanjšuje s časom (n), oz.:

$$\sigma(n) = \sigma_0 e^{-\frac{n}{\tau_1}}\,, \qquad n = 0, 1, 2, ...$$

kjer je σ_0 vrednost ob inicializaciji algoritma in τ_1 časovna konstanta. Ustrezno se spreminja s časom tudi funkcija topološke sosednosti:

$$h_{j,i(\mathbf{x})}(n) = e^{-\frac{d_{j,i}^2}{2\sigma^2(n)}}\,, \qquad n = 0, 1, 2, ...$$

Adaptivni proces: V primeru samo-organizacije se uteži spreminjajo skladno z vhodnim vektorjem, saj ni nadzornika (angl. supervisor) ali učitelja, ki bi nadziral proces učenja. Zato je potrebno Hebb-ovemu učilnemu pravilu, ki je primerno za asociativno učenje in ki povečuje vrednost uteži pri hkratno aktivnih nevronih na vhodu in izhodu uteži (kar pomeni spremembo uteži le v eno smer in posledično vodi v zasičenje), dodati člen pozabljivosti (angl. forgetting term) $g(y_j)\mathbf{w}_j$, kjer je $\mathbf{w}_j$ utežni vektor nevrona j in je $g(y_j)$ določena

pozitivna skalarna funkcija odziva y_j, z zahtevo, da je $g(y_j) = 0$, če je $y_j = 0$. Tedaj je sprememba uteži enaka:

$$\Delta \mathbf{w}_j = \eta y_j \mathbf{x} - g(y_j)\mathbf{w}_j \,,$$

kjer je η zopet učilni parameter oz. korak učenja. Prvi izraz na desni je Hebbov člen, drugi pa člen pozabljivosti. Da zadostimo zahtevam glede funkcije $g(y_j)$, izberemo linearno funkcijo za $g(y_j)$:

$$g(y_j) = \eta y_j \,.$$

Če nadalje upoštevamo:

$$y_j = h_{j,i(\mathbf{x})} \,,$$

pridemo do drugačne oblike za enačbo spremembe uteži:

$$\Delta \mathbf{w}_j = \eta h_{j,i(\mathbf{x})}(\mathbf{x} - \mathbf{w}_j) \,.$$

Končno je enačba za spremembo vrednosti uteži v času $n + 1$ enaka:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(\mathbf{x})}(n)(\mathbf{x} - \mathbf{w}_j(n)) \,,$$

kjer se učilni parameter spreminja s časom podobno kot širina topološke sosednosti:

$$\eta(n) = \eta_0 e^{-\frac{n}{\tau_2}} \,, \qquad n = 0, 1, 2, ...,$$

in se uporablja za vse uteži nevronov, ki ležijo znotraj topološke soseščine. Efekt enačbe učenja se kaže v pomikanju utežnega vektorja $\mathbf{w}_i$ zmagovitega nevrona proti vhodnemu vektorju $\mathbf{x}$. Algoritem zato vodi k topološki urejenosti preslikave iz vhoda na izhod na način, da sosednji nevroni v mreži težijo k podobnim utežnim vektorjem.

Proces adaptacije uteži v mreži SOM je mogoče razdeliti v dve fazi:

1. *Samo-organizirajoča faza ali faza urejanja*: v tej fazi se izvede topološko urejanje utežnih vektorjev. Traja okoli 1000 iteracij ali več. Pozorno je potrebno izbrati učilni parameter in sosednostno funkcijo, npr. za učilni parameter:

   $$\begin{aligned} \eta_0 &= 0.1 \\ \tau_2 &= 1000 \end{aligned}$$

   in za sosednostno funkcijo:

   $$\tau_2 = \frac{1000}{\log \sigma_0} \,,$$

   kjer je σ_0 radij mreže.

2. *Konvergenčna faza*: je potrebna za fino nastavitev (angl.fine tuning) preslikave. V splošnem je priporočljivo število iteracij v tej fazi najmanj 500 × število nevronov. Učni parameter v tej fazi naj bo reda 0.01, sosednostna funkcija pa naj vsebuje le najbližje sosede zmagovitega nevrona.

Lastnosti mreže SOM

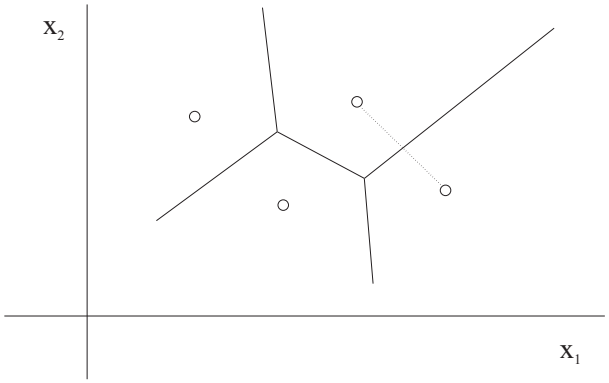
Preslikava Φ nevronske mreže SOM, ki jo določa niz utežnih vektorjev v izhodni (in hkrati edini) plasti nevronov, ima naslednje lastnosti:

- dobro aproksimira vhodni prostor
- na izhodu zagotavlja topološko urejenost, skladno z vhodnim prostorom, kar pomeni, da prostorska lokacija nevrona ustreza določeni domeni ali lastnosti vhodnih vzorcev
- preslikava Φ preslika statistične lastnosti vhodne domene v analogne statistične lastnosti izhodnega prostora: področja na vhodu z visoko verjetnostjo izbire se transformirajo v večjo domeno na izhodu oz. v večje število nevronov, kot področja na vhodu z nizko verjetnostjo izbire
- podatke na vhodu z nelinearno porazdelitvijo transformira topološka preslikava tako, da je mogoča izbira niza najboljših lastnosti za njeno aproksimacijo

3.2.6 Nevronska mreža za učečo vektorsko kvantizacijo - LVQ

Vektorska kvantizacija je posebna tehnika, ki opisuje vhodne vektorje na poenostavljen način s ciljem podatkovne kompresije. Vhodni prostor razdeli na številna področja in za vsako definira poseben predstavniški vektor. Ko se pojavi na vhodu nov vektor, se najprej določi področje, v katerem leži nov vektor, nato pa se ga predstavi s predstavniškim vektorjem (prototipom) pripadajočega področja. S tem je dosežen precejšen prihranek na pomnilniku, saj namesto številnih vhodnih vektorjev predstavljamo vhodno domeno le s precej manjšim številom predstavniških vektorjev. Niz predstavniških vektorjev se imenuje kodna knjiga (angl. code book), njeni elementi pa kodne besede.

Posamezen prototip predstavlja oz. pokriva v prostoru vse točke, katerim je sam najbližji izmed vseh prototipov. Vhodni prostor lahko za primer dveh dimenzij predstavimo kot t.i. Voronoi-ev diagram (Slika 3.8), posamezna področja pa se imenujejo Voronoi-eve celice.

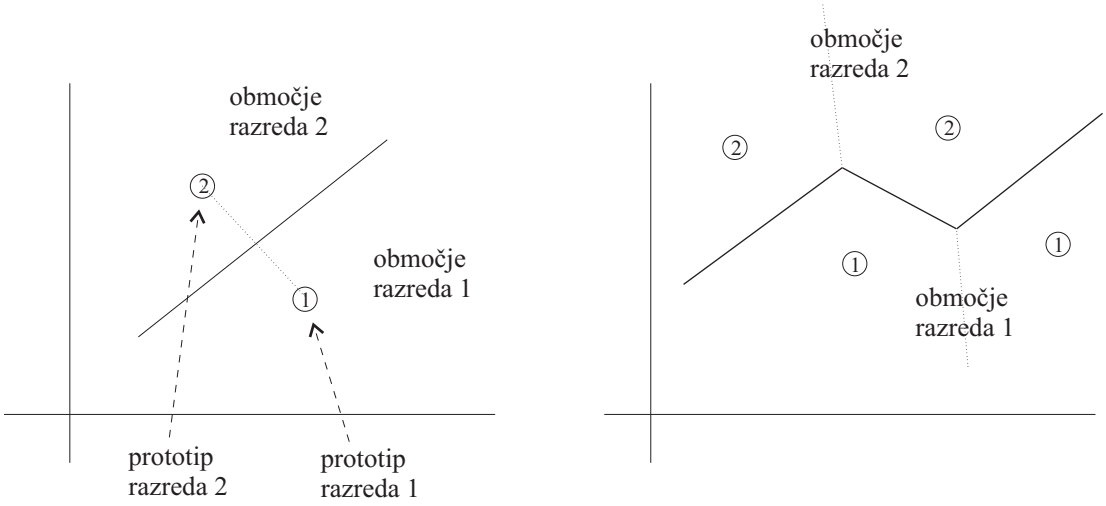


Slika 3.8: Voronoi-ev diagram. Meje med področji prototipov oz. Voronoi-evimi celicami potekajo po simetralah med pari prototipov.

Nevronska mreža LVQ [18] se imenuje po algoritmu učeče vektorske kvantizacije LVQ (angl. learning vector quantization), ki je nadzorovani algoritem učenja in uporablja informacije o področjih ali razredih (nazivi ali labele) tako, da izboljšuje ločevanje med razredi. Ker se LVQ uporablja za razvrščanje oz. klasifikacijo vzorcev, vsak prototip pripada enemu izmed razredov vzorcev; zato ima vsak prototip labelo razreda in vse točke, katerim je ta prototip najbližji, so razvrščene v njegov razred. Najpreprostejši primer je prikazan na Sliki 3.9 levo, kjer vsak od dveh prototipov predstavlja svoj razred. V takem primeru dobimo linearno mejo med obema razredoma. Če pa za vsak razred vzamemo več prototipov, že dobimo nelinearno (natančneje: odsekoma linearno) mejo (Slika 3.9) desno.

Algoritem LVQ sestoji iz ponavljanja sledečega postopka:

1. naključno izberemo vektor $\mathbf{x}_i$ med vzorci
2. poiščemo najbližji predstavniški vektor oz. prototip $\mathbf{w}_r$ vhodnemu vektorju $\mathbf{x}_i$.
$$||\mathbf{w}_r - \mathbf{x}_i|| \leq ||\mathbf{w}_j - \mathbf{x}_i||, \quad j = 1, ..., N, \tag{3.2}$$
kjer je N število vzorcev, $||.||$ evklidska norma, $d = ||\mathbf{a} - \mathbf{b}||$ pa evklidska razdalja



Slika 3.9: Meja med dvema razredoma v primeru enega (levo) oz. dveh prototipov na posamezen razred.

3. Če z R označimo razred (labelo) vektorja, potem v primeru $R_{w_r} = R_{x_i}$ sledi modifikacija vektorja $\mathbf{w}_r$ na naslednji način:

$$\mathbf{w}_r(n+1) = \mathbf{w}_r(n) + \alpha(n)[\mathbf{x}_i - \mathbf{w}_r(n)]\,, \tag{3.3}$$

$0 < \alpha_n < 1$. Če pa velja $R_{wr} \neq R_{xi}$, potem je modifikacija enaka:

$$\mathbf{w}_r(n+1) = \mathbf{w}_r(n) - \alpha(n)[\mathbf{x}_i - \mathbf{w}_r(n)]\,. \tag{3.4}$$

Ostali predstavniški vektorji ostanejo nespremenjeni.

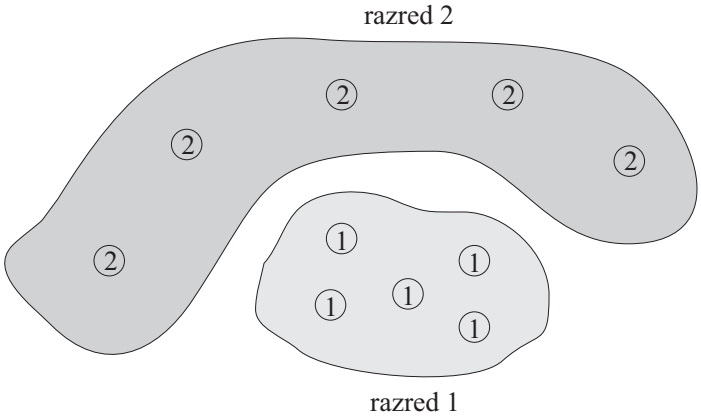
V zgornjih izrazih je $\alpha(n)$ učilni parameter, ki se monotono zmanjšuje z naraščajočim diskretnim časom n. Tipično je na začetku enak okoli 0.1-0.2, nato pa se monotono zmanjšuje s časom (eksponentno ali linearno).

Po LVQ učenju prototipi približno ustrezajo verjetnostni porazdelitvi vzorcev:

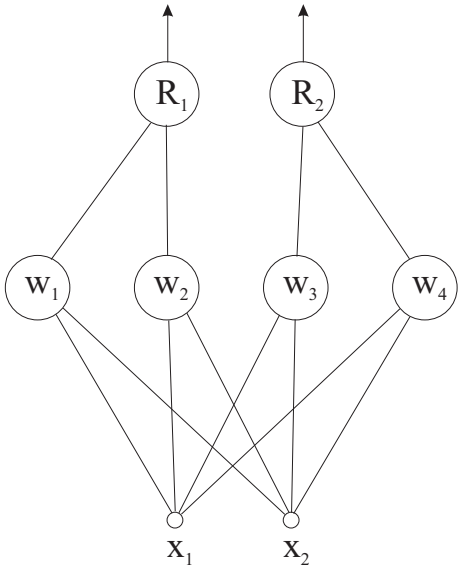
Postopek LVQ morda ne izgleda kot tipična nevronska mreža, pa vendar LVQ lahko obravnavamo kot nevronsko mrežo, kot prikazuje Slika 3.11.

3.2.7 Hopfield-ova nevronska mreža

Vsaka rekurentna nevronska mreža (RNM) ima zaradi povratnih povezav sposobnost pomnenja in zato lahko RNM v vsakem trenutku pripišemo določeno



Slika 3.10: Prototipi pri LVQ približno ustrezajo verjetnostni porazdelitvi vzorcev.

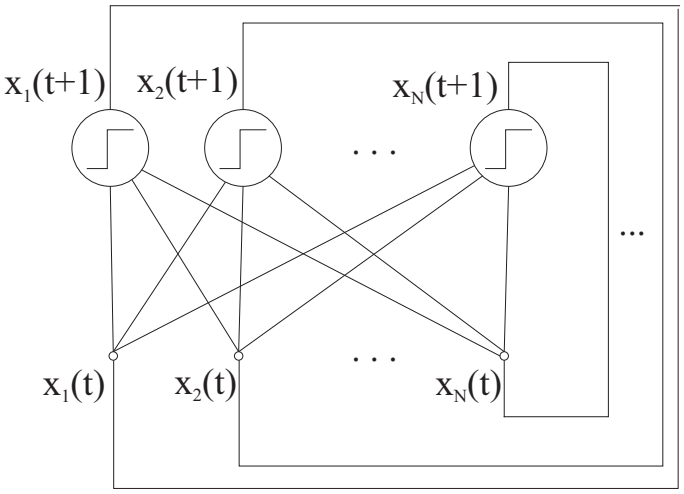


Slika 3.11: LVQ kot nevronska mreža.

stanje, ki ga tvorijo izhodi nevronov. Večinoma se RNM uporabljajo za modeliranje dinamičnih sistemov. Obstaja pa poseben razred RNM, ki se uporabljajo v povsem drug namen. To so asociativne RNM. Cilj asociativne RNM je poljubno začetno stanje mreže v določenem številu korakov pripeljati v stabilno oz. fiksno stanje. Na ta način mreža poljuben vhodni vzorec, ki predstavlja začetno stanje mreže, 'pripelje' v stabilno stanje, ki ustreza nekemu vzorcu,

ki je shranjen v mreži - implicitno, preko ustreznih uteži. V mrežo je mogoče shraniti večje število vzorcev oz. prototipov, ki torej predstavljajo stabilna stanja mreže. Vhodni vzorec si lahko predstavljamo kot nek nepopoln oz. delno zašumljen vzorec, odgovor mreže pa je eden od shranjenih prototipov, na katerega vhodni vzorec mrežo najbolj 'asociira'.

Hopfield-ova nevronska mreža je tipičen predstavnik asociativnih RNM. Sestavlja jo N nevronov, katerih izhodi so povratno vezani na vhode vseh nevronov (polno povezana RNM), kot kaže Slika 3.12. Posamezen nevron Hopfieldove



Slika 3.12: Hopfield-ova nevronska mreža.

mreže je lahko v enem od dveh stanj: 1 (prižgan) in -1 (ugasnjen); nevron je torej binaren - hrani 1 bit informacije. Stanje Hopfieldove mreže je vektor izhodov posameznih nevronov:

$$\mathbf{x} = [x_1, x_2, ..., x_N]^T . \tag{3.5}$$

Vseh možnih stanj je seveda 2^N. Izhod i-tega nevrona je določen takole:

$$x_i(t+1) = \begin{cases} 1, & \text{če} \sum_{i=1}^{N} w_{ij} x_j(t) > \Theta_i \\ x_i(t), & \text{če} \sum_{i=1}^{N} w_{ij} x_j(t) = \Theta_i \\ -1, & \text{če} \sum_{i=1}^{N} w_{ij} x_j(t) < \Theta_i \end{cases} \tag{3.6}$$

Na vhodu nevrona je utežena vsota vhodov, tj. povratno vezanih izhodov. Uteži so realne vrednosti. Aktivacijska funkcija nevrona pa je pragovna funkcija

s pragom Θ. Torej, nevron je prižgan, kadar utežena vsota preseže prag, oz. je ugasnjen, kadar utežena vsota ne doseže praga. V primeru enakosti ostane stanje nevrona nespremenjeno. V razširjeni varianti Hopfieldove mreže, ki je tukaj ne bomo obravnavali, je aktivacijska funkcija bolj splošna; lahko tudi zvezna.

Hopfield-ova mreža deluje asinhrono, tj. nevroni procesirajo eden za drugim; ni potrebno, da po vrsti, pomembno je le, da procesirajo vsi nevroni približno enako pogosto. Model, pri katerem nevroni procesirajo sinhrono oz. paralelno, pa se imenuje Little-ov model in ima malo drugačne konvergenčne lastnosti.

Značilnost Hopfield-ove mreže je, da lahko vanjo shranimo prototipe tudi brez učenja. V tem primeru mora matrika uteži ustrezati dvema dodatnima pogojema oz. omejitvama:

- $w_{ij} = w_{ji}$ za vse $i = 1, 2, ..., N$, in $j = 1, 2, ..., N$, in
- $w_{ii} = 0$ za vse $i = 1, 2, ..., N$.

Matrika uteži $\mathbf{W}$, ki je dimenzije $N \times N$, je torej simetrična ($\mathbf{W}^T = \mathbf{W}$) in ima na diagonali ničle. Pragovom lahko dodelimo vrednosti 0.

Obstaja preprost način določevanja ustreznih uteži, s katerim lahko v mrežo 'shranimo' P N-dimenzionalnih vektorjev - prototipov: $\{\boldsymbol{\xi}_p | p = 1, 2, ..., P\}$:

$$w_{ji} = \frac{1}{N} \sum_{p=1}^{P} \xi_{pj} \xi_{pi} \,,$$

oz. v vektorskem zapisu:

$$\mathbf{W} = \frac{1}{N} \sum_{p=1}^{P} \boldsymbol{\xi}_p \boldsymbol{\xi}_p^T - \frac{P}{N} \mathbf{I} \,,$$

kjer je $\boldsymbol{\xi}_p \boldsymbol{\xi}_p^T$ vektorski produkt vektorja vzorcev $\boldsymbol{\xi}_p$ s samim seboj. Na ta način smo uteži določili tako, da bodo stanjski vektorji $\boldsymbol{\xi}_p$ stabilna stanja. Ta stanja so (implicitno) shranjena v mreži.

Za dokazovanje stabilnosti dinamičnih sistemov se pogosto uporablja t.i. funkcija Ljapunov-a oz. energijska funkcija. Če nam za nek dinamični sistem uspe

poiskati funkcijo Ljapunov-a, je sistem stabilen. Če je ne najdemo, ni rečeno, da ne obstaja, oz. da sistem ni stabilen.

Tudi za Hopfield-ovo mrežo obstaja ustrezna funkcija Ljapunova $E(\mathbf{x})$, ki je določena tako, da s časom monotono pada. To pomeni, da se po spremembi stanja kateregakoli nevrona, kar ustreza prehodu v novo stanje, E zmanjša ali pa kvečjemu ostane enaka. Za Hopfield-ovo mrežo se funkcija Ljapunov-a lahko glasi

$$E = -\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ji}x_i x_j \,. \tag{3.7}$$

Sprememba funkcije Ljapunov-a zaradi spremembe posameznega (k-tega) nevrona v času t je

$$\begin{aligned}
\Delta E &= E(\mathbf{x}(t+1)) - E(\mathbf{x}(t)) && (3.8)\\
&= -\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}x_i(t+1)x_j(t+1) && (3.9)\\
&\quad +\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}x_i(t)x_j(t) && (3.10)\\
&= -2x_k(t+1)\sum_{j=1}^{N} w_{kj}x_j(t+1) + 2x_k(t)\sum_{j=1}^{N} w_{kj}x_j(t)\,. && (3.11)
\end{aligned}$$

V Enačbi 3.11 smo upoštevali, da se lahko spremenijo samo členi, ki vsebujejo x_k, ostali se torej krajšajo. Obe vsoti v Enačbi 3.11 sta enaki, kajti sprememba je lahko samo v k-tem nevronu, utež w_{kk} pa je 0. Zato lahko vsoto izpostavimo:

$$\Delta E = 2(x_k(t) - x_k(t+1))\left[\sum_{j=1}^{N} w_{kj}x_j(t)\right]\,. \tag{3.12}$$

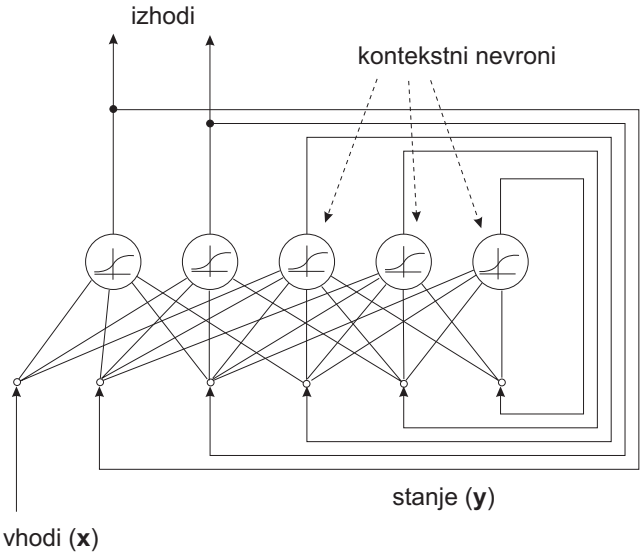
Če je

$$\sum_{j=1}^{N} w_{kj}x_j(t) < 0\,, \tag{3.13}$$

gre k-ti nevron v stanje -1, tj. $x_k(t+1) = -1$, zato bo v vsakem primeru $x_k(t) - x_k(t+1) \geq 0$, če pa je vsota (3.13) večja od 0, gre k-ti nevron v stanje 1, tj. $x_k(t+1) = 1$, zato bo $x_k(t) - x_k(t+1) \leq 0$. V obeh primerih je torej $\Delta E \leq 0$. Zato je funkcija Ljapunov-a res monotono padajoča.

3.2.8 Rekurentna nevronska mreža

V primeru, da rešujemo dinamične probleme, kjer so preslikave od vhoda na izhod odvisne od časa, so potrebne nevronske mreže s povratnimi povezavami ali rekurentne nevronske mreže - RNM (angl. recurrent neural networks). Primer polno povezane rekurentne mreže prikazuje Slika 3.13.



Slika 3.13: Polno povezana rekurentna mreža RNM.

Učilni algoritem mreže RNM se imenuje RTRL (angl. real-time recurrent learning). Njegova značilnost je, da izvaja spremembe uteži v mreži RNM v realnem času, oz. medtem ko mreža procesira vhodne podatke [19]. Mreža RNM ima eno plast nevronov, katere množica nevronov M je razdeljeno v dve množici, v množico O, v kateri so indeksi nevronov, ki so povezani z izhodom, in v množico H, v kateri so indeksi kontekstnih nevronov, ki so vezani le povratno na vhod. Vsi nevroni iz množice M so povratno vezani na vhode vseh nevronov, skupaj z množico I vhodnih spremenljivk. Kriterijska funkcija v procesu učenja je vsota kvadratov napak na izhodnih nevronih, ozr:

$$E(t) = \frac{1}{2} \sum_{k \in O} e_k^2(t) \,,$$

kjer je :

$$e_k(t) = \begin{cases} d_k(t) - y_k(t), & \text{če } k \in O \\ 0, & \text{sicer}\,. \end{cases} \tag{3.14}$$

Pri tem t označuje diskretni čas.

Če z $\omega_{ij}(t)$ označimo poljubno utež v RNM, ki vstopa v i-ti nevron iz j-te spremenljivke, ki je lahko ali vhodna ali notranja (povratna), potem je sprememba te uteži enaka:

$$\begin{aligned} \Delta\omega_{ij}(t) &= -\eta \frac{\partial E(t)}{\partial \omega_{ij}(t)} \\ \frac{\partial E(t)}{\partial \omega_{ij}(t)} &= -\sum_{k \in O} e_k(t) \frac{\partial y_k(t)}{\partial \omega_{ij}(t)}\,, \end{aligned}$$

kjer je zopet $y_k(t+1) = f(v_k(t))$ in je f sigmoidna funkcija, v pa utežena vsota:

$$v_k(t) = \sum_{l \in I \cup M} \omega_{kl}(t) z_l(t)$$

in je z_l splošna oznaka za l-to spremenljivko (vhodno ali notranjo). Tedaj je:

$$\frac{\partial y_k(t+1)}{\partial \omega_{ij}(t)} = \frac{\partial y_k(t+1)}{\partial v_k(t)} \frac{\partial v_k(t)}{\partial \omega_{ij}(t)} = f'(v_k(t)) \frac{\partial v_k(t)}{\partial \omega_{ij}(t)}$$

$$\frac{\partial v_k(t)}{\partial \omega_{ij}(t)} = \sum_{l \in I \cup M} \frac{\partial(\omega_{kl}(t) z_l(t))}{\partial \omega_{ij}(t)} = \sum_{l \in I \cup M} [\omega_{kl}(t) \frac{\partial z_l(t)}{\partial \omega_{ij}(t)} + \frac{\partial \omega_{kl}(t)}{\partial \omega_{ij}(t)} z_l(t)]$$

V zadnjem izrazu na desni velja:

$$\frac{\partial \omega_{kl}(t)}{\partial \omega_{ij}(t)} = 1\,, \quad \text{če } k = i \text{ in } l = j\,,$$

zaradi česar lahko pišemo:

$$\frac{\partial v_k(t)}{\partial \omega_{ij}(t)} = \sum_{l \in I \cup M} [\omega_{kl}(t) \frac{\partial z_l(t)}{\partial \omega_{ij}(t)} + \delta_{ki} z_j(t)]\,,$$

kjer je:

$$\frac{\partial z_l(t)}{\partial \omega_{ij}(t)} = \begin{cases} 0\,, & \text{če } l \in I \\ 1\,, & \text{sicer}\,, \end{cases}$$

in je δ_{ki} Dirac-ova funkcija delta. Sedaj lahko delne rezultate ustrezno združimo in dobimo:

$$\frac{\partial y_k(t+1)}{\partial \omega_{ij}(t)} = f'(v_k(t))[\sum_{l \in M} \omega_{kl}(t) \frac{\partial y_l(t)}{\partial \omega_{ij}(t)} + \delta_{ki} z_j(t)] \,.$$

Če vpeljemo substitucijo:

$$p_{ij}^k(t) = \frac{\partial y_k(t)}{\partial \omega_{ij}(t)} \,,$$

potem je končno:

$$p_{ij}^k(t+1) = f'(v_k(t))[\sum_{l \in M} \omega_{kl}(t) p_{ij}^l(t) + \delta_{ki} z_j(t)] \,.$$

Odvod sigmoidne funkcije je enak:

$$f'(v_k(t)) = y_k(t+1)[1 - y_k(t+1)] \,,$$

začetna vrednost odvoda pa je enaka 0:

$$p_{ij}^k(0) = 0 \,.$$

V vsaki iteraciji učenja je torej sprememba vsake uteži v mreži:

$$\Delta \omega_{ij}(t) = \eta \sum_{k \in O} e_k(t) p_{ij}^k(t) \,,$$

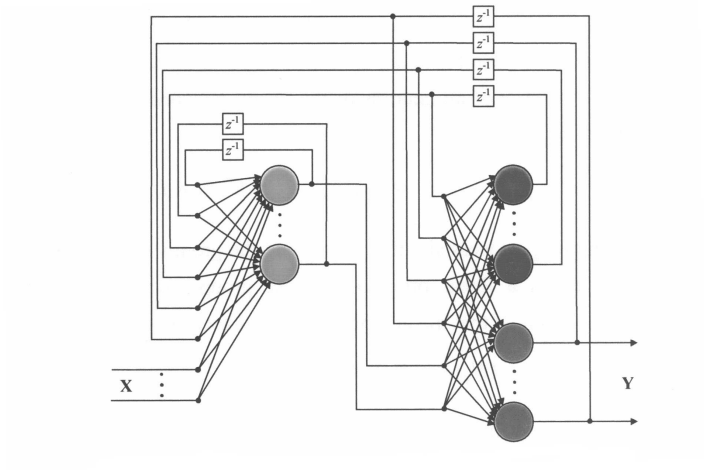
utež pa se spremeni na naslednji način:

$$\omega_{ij}(t+1) = \omega_{ij}(t) + \Delta \omega_{ij}(t) \,.$$

3.2.9 Posplošena rekurentna mreža GARNN

Nevronska mreža RNM ima zaradi le ene plasti nevronov probleme pri učenju zahtevnejših dinamičnih problemov. Ustrezno izboljšavo zato predstavlja posplošena nevronska mreža GARNN (angl. generalized artificial recurrent neural network) [20], katere dvonivojsko topologijo prikazuje Slika 3.14.

Zaradi dvonivojske topologije omogoča GARNN učenje zahtevnejših časovnih problemov, saj je v tem primeru časovna funkcija prehajanja stanj stanj dinamičnega sistema lahko poljubno zahtevna. Zaradi spremenjene topologije



Slika 3.14: Struktura posplošene rekurentne mreže GARNN.

se ustrezno modificira učilni algoritem glede na RTRL v prejšnjem poglavju. Vendar pa se konvergenca učenja bistveno izboljša, če sprememba uteži sledi enačbi Kalman-ovega filtra DEKF (angl. decoupled extended Kalman filter) [20] in če se v postopku učenja upošteva še vsiljeni sistem povratnih vezav (angl. teacher forcing technique). Oceno za uteži i-tega nevrona v času $t+1$ podaja enačba:

$$\hat{\mathbf{w}}_i(t+1|t) = \hat{\mathbf{w}}_i(t|t-1) + \mathbf{G}_i(t)\boldsymbol{\alpha}(t)\,,$$

kjer je $\mathbf{G}_i$ t.i. Kalman-ovo ojačanje, ki se v postopku učenja spreminja skladno z enačbami:

$$\begin{aligned}
\mathbf{G}_i &= \mathbf{K}_i(t,t-1)\mathbf{C}_i^T(t)\boldsymbol{\Gamma}(t) \\
\boldsymbol{\Gamma}(t) &= \left[\sum_{i=1}^{N}\mathbf{C}_i(t)\mathbf{K}_i(t,t-1)\mathbf{C}_i^T(t) + \mathbf{R}(t)\right]^{-1} \\
\mathbf{K}_i(t+1,t) &= \mathbf{K}_i(t,t-1) - \mathbf{G}_i(t)\mathbf{C}_i(t)\mathbf{K}_i(t,t-1) + \mathbf{Q}_i(t) \\
\boldsymbol{\alpha}(t) &= \mathbf{d}(t) - \mathbf{y}(t)
\end{aligned}$$

V zgornjih enačbah je $\mathbf{K}_i$ kovariančna matrika napake i-tega nevrona, $\boldsymbol{\Gamma}$ konverzijska matrika, $\boldsymbol{\alpha}$ inovacijski vektor (razlika med želeno in dejansko vrednostjo na izhodu), $\mathbf{C}_i$ merska matrika linearizirane dinamskega modela ($\Delta\mathbf{w}_i = 0$) nevrona i, katere splošni element je enak:

$$\frac{\partial y_k(t)}{\partial \omega_{ij}}$$

in ki ga izračunamo ne enak način kot pri RTRL, $\mathbf{Q}_i$ je diagonalna kovariančna matrika, ki opisuje procesni šum, $\mathbf{R}$ diagonalna kovariančna matrika merskega šuma in N število nevronov.

3.2.10 Komplementarni spodbujevani vzvratni učilni algoritem

Ta postopek učenja sodi v skupino spodbujevalnih algoritmov, ki se razlikujejo od nadzorovanih in nenadzorovanih ali samo-organizirajočih algoritmov po tem, da imajo na razpolago le oceno delovanja mreže, ki je ali pozitivna ali negativna. Zanimiv je tudi zato, ker je uporaben tako pri statičnih (vnaprej povezanih), kot tudi dinamičnih (rekurentnih) mrežah. Pri slednjih je potrebno namesto BPG (backpropagation) algoritma uporabiti RTRL (real-time recurrent learning). Postopek učenja je naslednji:

1. Na vhod nevronske mreže pripeljemo vhodni vzorec
2. Izračuna se ustrezni odziv na izhodu mreže (R)
3. Izhodi se interpretirajo kot verjetnosti, da so izhodi 1
4. Določi se verjetnostno binarno vrednost izhoda (B)
5. Izvede se akcija na izhodu, ki ustreza S
6. Akciji v odgovor sledi spodbuda okolja
7. Na osnovi spodbude se izračuna napaka:
   - če je nagrada ($r = +1$), potem je $E = (B - R)$
   - če je kazen ($r = -1$), potem je $E = ((1 - B) - R)$
   - če ni spodbude ($r = 0$), potem je $E = 0$
8. Če $E \neq 0$, se izvede vzvratno učenje, BPG ali RTRL.

3.2.11 Spodbujevano učenje z algoritmom učečih avtomatov

V poglavju o učečih avtomatih so bile podane t.i. korekcijske enačbe, s katerimi smo spreminjali parametre avtomata glede na odzive iz okolja na izbrane

akcije avtomata. Podobne enačbe je mogoče uporabiti pri spreminjanju uteži nevronskih mrež, tako vnaprej povezanih (npr. MLP), kot tudi rekurentnih nevronskih mrež (npr. RNN).

Predpostavimo, da so vse uteži v mreži zajete v množici:

$$W = \{\omega_1, \omega_2, ..., \omega_N\}$$

in da imamo na razpolago dvakrat toliko akcij (učečega avtomata), kot je vseh uteži v mreži, torej $2N$:

$$\alpha = \{\alpha_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, ..., \alpha_{N1}, \alpha_{N2}\}\,.$$

Vsako utež, npr. ω_i, lahko tako spremenimo na dva načina (z dvema akcijama):

$$\begin{aligned} \alpha_{i1} : \omega_i(t+1) &= \omega_i(t) + \Delta \\ \alpha_{i2} : \omega_i(t+1) &= \omega_i(t) - \Delta \end{aligned}$$

Postopek učenja poteka na naslednji način. Mreža določi odziv na vhodni vzorec in shrani oceno okolja. Nato iz trenutne verjetnostne porazdelitve akcij izbere eno, začasno izvede ustrezno modifikacijo uteži, ponovno določi odziv na isti vhodni vzorec in pridobi novo oceno okolja. Če se je ocena zaradi spremembe uteži izboljšala (kazen je zamenjala nagrada ali pa se je izhod približal želeni vrednosti), potem se skladno s korekcijsko enačbo UA (učečega avtomata), spremenijo verjetnosti akcij tako, da se poveča verjetnost uspešne akcije, ostale verjetnosti pa se ustrezno zmanjšajo. V nasprotnem primeru pa se verjetnost izbrane akcije zmanjša, verjetnosti ostalih akcij pa se skladno poveča:

$\alpha(n) = \alpha_i$, $\beta(n) = 0$ (nagrada):

$$\begin{aligned} p_j(n+1) &= p_j(n) - g_j[p(n)]\,, \qquad j \neq i \\ p_i(n+1) &= p_i(n) + \sum_{j=1, j\neq i}^{2N} g_j[p(n)] \end{aligned}$$

$\alpha(n) = \alpha_i$, $\beta(n) = 1$ (kazen):

$$\begin{aligned} p_i(n+1) &= p_i(n) - \sum_{j=1, j\neq i}^{2N} h_j[p(n)] \\ p_j(n+1) &= p_j(n) + h_j[p(n)]\,. \end{aligned}$$

V primeru pozitivnega rezultata spremembo uteži ohranimo, v nasprotnem primeru pa utež vrnemo na prvotno vrednost.

3.3 Analiza procesa učenja v nevronskih mrežah

Umetne nevronske mreže so bile dolgo časa predmet številnih kritik. Slednje so se nanašale predvsem na danes že zmotno prepričanje, da naučene nevronske mreže ne dajejo ustrezne razlage za svoje delovanje. Predvsem so se delale primerjave z metodami umetne inteligence, kjer se empirično znanje kot posledica učenja z metodami umetne (strojne) inteligence, pogosto predstavlja z odločitvenimi drevesi, ki ponujajo razlago za delovanje ne le za učne primere, temveč tudi za testne oz. od učenja neodvisne primere.

V nadaljevanju bo zato najprej opisana metoda ekstrakcije znanja, ki se nahaja v utežeh naučene nevronske mreže, temu pa bo sledila še analiza procesa učenja z vidika parametrov kompleksnih sistemov, kamor brez dvoma sodijo tudi umetne nevronske mreže, ki modelirajo danes prav gotovo enega najkompleksnejših znanih sistemov, to je možganski živčni sistem živih organizmov.

3.3.1 Ekstrakcija znanja iz naučene nevronske mreže

Nevronska mreža spreminja v procesu učenja svoje notranje parametre oz. uteži toliko časa, dokler se njeni rezultati procesiranja na izhodu mreže ne ujemajo dovolj dobro z želenimi vrednostmi. Rezultat učenja je torej porazdeljena množica uteži, ki imajo praviloma realne pozitivne ali negativne uteži. Na prvi pogled je zato z njimi težko razložiti vzroke za pravilno delovanje. Vendar pa obstajajo načini, ki pridobljeno porazdeljeno znanje prevedejo v ustrezne logične zapise, s katerimi je mogoče podati splošno razlago za delovanje mreže za vse možne primere na vhodu. Ogledali si bomo dve metodi ekstrakcije znanja iz naučene nevronske mreže, ki obe dajeta kot rezultat univerzalno logično enačbo v obliki disjunktivne normalne oblike (DNO), t.j. oblike, ki z supremalno logično operacijo (disjunkcijo) povezuje infimalne izraze (konjunkcije vhodnih spremenljivk), ki je logična alternativa kaskade odločitvenih konstruktov IF-THEN-ELSE. Prva metoda je *osnovna*, ki z zaporednimi substitucijami od vhoda proti izhodu postopoma gradi končno logično enačbo. Ker je računska kompleksnost te metode eksponentna, je podana še druga, *polinomska* metoda, ki s polinomsko kompleksnostjo zagotovi določitev DNO [21].

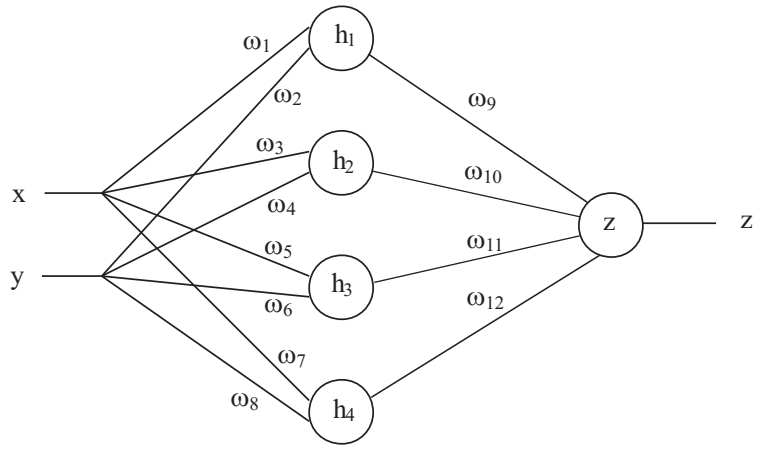
Osnovna metoda ekstrakcije znanja

Ta metoda je precej preprosta, zato je nenavadno, da je kar nekaj časa veljalo prepričanje, da naučene nevronske mreže ne dajejo razlage za svoje delovanje. Temelji na aproksimaciji nevronske funkcije z Boole-ovo (preklopno) na naslednji način. Če je f_i izhodna vrednost nevrona pri i-ti vhodni kombinaciji spremenljivk, potem je ustrezna Boole-ova funkcija g_i določena z odločitvijo:

$$g_i = \begin{cases} 1, & \text{če } f_i \geq 0.5 \\ 0, & \text{če } f_i < 0.5 \end{cases}$$

Pri naučeni nevronski mreži poznamo končne vrednosti uteži. Najprej s pomočjo vrednosti uteži, ki vstopajo v (prvi) skriti nivo in zgornje aproksimacije, določimo Boole-ove funkcije za nevrone (prvega) skritega nivoja. S pomočjo vrednosti teh funkcij za posamezne vhodne kombinacije določimo Boole-ove funkcije naslednjega nivoja (drugega skritega ali izhodnega), v katerih končno nadomestimo spremenljivke z Boole-ovimi funkcijami skritih nevronov, kar daje končno logično funkcijo izhodnih nevronov v odvisnosti od vhodnih spremenljivk.

Primer
Podana je naučena nevronska mreža (Slika 3.15).



Slika 3.15: Mreža MLP, naučena na problem EX-OR.

Izhode nevronov v podani mreži podajajo izrazi:

$$\begin{aligned} h_1 &= f(\omega_1 x + \omega_2 y + t_1) \\ h_2 &= f(\omega_3 x + \omega_4 y + t_2) \\ h_3 &= f(\omega_5 x + \omega_6 y + t_3) \\ h_4 &= f(\omega_7 x + \omega_8 y + t_4) \\ z &= f(\omega_9 h_1 + \omega_{10} h_2 + \omega_{11} h_3 + \omega_{12} h_4 + t_5) \end{aligned}$$

Vzemimo, da smo mrežo učili preslikave, ki ustreza logični funkciji EX-OR (npr. z algoritmom BP):

x	y	$x \bigtriangledown y$
0	0	0
0	1	1
1	0	1
1	1	0

Rezultat učenja po 1000 ponovitvah (z uporabo algoritma BP - 'BackPropagation', ki išče minimalno napako s pomočjo gradientov) so vrednosti parametrov:

$$\begin{aligned} \omega_1 &= 2.51, \quad \omega_2 = -4.80, \quad \omega_3 = -4.90 \\ \omega_4 &= 2.83, \quad \omega_5 = -4.43, \quad \omega_6 = -4.32 \\ \omega_7 &= -0.70, \quad \omega_8 = -0.62, \quad \omega_9 = 5.22 \\ \omega_{10} &= 5.24, \quad \omega_{11} = -4.88, \quad \omega_{12} = 0.31 \\ t_1 &= -0.83, \quad t_2 = -1.12, \quad t_3 = 0.93 \\ t_4 &= -0.85, \quad t_5 = -2.19 \end{aligned}$$

Za vse nevrone v mreži lahko sedaj uporabimo osnovno metodo, npr. za nevron h_1:

$$\begin{aligned} h_1(0,0) &= f(2.51{\cdot}0 - 4.80{\cdot}0 - 0.83) = f(-0.83) \approx 0 \\ h_1(0,1) &= f(2.51{\cdot}0 - 4.80{\cdot}1 - 0.83) = f(-5.63) \approx 0 \\ h_1(1,0) &= f(2.51{\cdot}1 - 4.80{\cdot}0 - 0.83) = f(1.68) \approx 1 \\ h_1(1,1) &= f(2.51{\cdot}1 - 4.80{\cdot}1 - 0.83) = f(-3.12) \approx 0 \end{aligned}$$

Tem vrednostim ustreza logična funkcija: $x \cdot \overline{y}$.

Podobno bi dobili tudi za ostale nevrone:

$$\begin{aligned} h_2 &= \overline{x}\cdot y,\ h_3 = \overline{x}\cdot\overline{y},\ h_4 = 0 \\ z &= h_1\cdot\overline{h_3} \vee h_1\cdot h_2 \vee h_2\cdot\overline{h_3} \end{aligned}$$

S substitucijo funkcij nevronov skritega nivoja v nevron izhodnega nivoja bi dobili:

$$z = \overline{x}\cdot y \vee x\cdot\overline{y}$$

kar je ravno disjunktivna normalna oblika EX-OR logične funkcije.

S pomočjo osnovne metode smo z aproksimacijo nevronske mreže, ki je bila naučena na funkcijo EX-OR, dobili njeno razlago v obliki DNO.

Polinomska metoda ekstrakcije znanja

Osnovna ideja pri tej metodi je, da vključuje logične produkte (konjunkcije) v iskano DNO, od najmanjših (z najmanjšim številom spremenljivk) proti največjim, kar sproti eliminira precejšnje število produktov, saj velja, da so večji produkti vsebovani v manjših. Poleg tega je mogoče določene logične produkte, ki so vezani na zelo majhne naučene uteži mreže, enostavno izpustiti, kar lahko precej poenostavi končno DNO, ne da bi to pomembno vplivalo na verodostojnost rezultata. Postopek polinomske metode je naslednji:

1. Ob upoštevanju aproksimacije nevronske funkcije z ortonormalno obliko:

   $$f(\omega_1 x_1 + \omega_2 x_2 + ... + \omega_n x_n + \omega_{n+1}) = f_1 x_1 ... x_n \vee f_2 x_1 ... \overline{x_n} \vee ... \vee f_{2^n}\overline{x_1}...\overline{x_n}\,,$$

   ugotovi eksistenco posameznih logičnih produktov od najnižjega reda proti najvišjemu. Posamezne vrednosti za f_i je sicer mogoče dobiti enostavno, če v enačbo vstavimo ustrezno vhodno kombinacijo, npr. za f_1 kombinacijo (1,...,1), za f_2 kombinacijo (1,...,0) in za f_{2^n} vhodno kombinacijo (0,...,0), nato pa pripisati za f_i vrednost 1, če je $f(.) \geq 0.5$, in 0 sicer. Eksistenco splošnega produkta x_{i1} ... x_{ik} $\overline{x_{i,k+1}}$ ... $\overline{x_{il}}$ v Booleovi funkciji poljubnega nevrona je mogoče v polinomskem času določiti s pomočjo izraza:

   $$f\Big(\sum_{j=i_1}^{i_k}\omega_j + \omega_{n+1} + \sum_{\substack{1 \leq j \leq n,\\ j \neq i_1...i_l,\ \omega_j \leq 0}} \omega_j\Big) \geq 0.5$$

2. Produkte, ki eksistirajo v nevronski enačbi, poveži v DNO. Postopek ponovi za vse nevrone v mreži.

3. Po vrsti, od izhoda proti vhodu nadomeščaj spremenljivke z ustreznimi Boole-ovimi aproksimacijami (DNO) posameznih nevronskih funkcij.

Primer
Določimo logično funkcijo na osnovi polinomske metode za nevron, ki ga podaja izraz:

$$f(0.65x_1 + 0.23x_2 + 0.15x_3 + 0.20x_4 + 0.02x_5 - 0.5)$$

Za x_i $(i = 1, 2, 3, 4, 5)$ velja:

$$f(\omega_1 + \omega_6) \geq 0.5,$$

kar pomeni, da člen x_1 eksistira.

Za $x_i x_j$ $(i, j = 2, 3, 4, 5)$ imamo:

$$f(\omega_i + \omega_j + \omega_6) < 0.5,$$

kar pomeni, da produktni členi drugega reda ne eksistirajo.

Za $x_i x_j x_k$ $(i, j, k = 2, 3, 4, 5)$ je:

$$f(\omega_2 + \omega_3 + \omega_4 + \omega_6) \geq 0.5,$$

in torej $x_2 x_3 x_4$ eksistira. S tem smo vse možne izraze izločili, kar pomeni, da je logična funkcija nevrona enaka:

$$g = x_1 \vee x_2 x_3 x_4 \,.$$

Cilj v postopku ekstrakcije znanja iz naučene nevronske mreže je seveda izluščiti *natančen* in *enostaven* logični približek k izhodni nevronski funkciji.

Natančnost zagotavljamo z upoštevanjem ustrezne cenilne funkcije C, ki običajno primerja izhodne nevronske funkcije z Boole-ovimi aproksimacijami, n.pr:

$$\begin{aligned} C &= \max_{i \in N}(\max_{j \in M} e_{ij}) \\ e_{ij} &= |g_{ij} - f_{ij}| \,, \end{aligned}$$

kjer je e_{ij} napaka oz. absolutna razlika med Boole-ovo aproksimacijo in nevronsko funkcijo na mestu i-tega nevrona pri j-tem podatku. N je število nevronov, M pa število učnih podatkov. Premajhna natančnost lahko zahteva dodatno učenje, ali pa celo ponovno učenje z novimi začetnimi parametri oz. utežmi.

Enostavnost logičnega zapisa pa lahko povečamo tako, da v procesu substitucije eliminiramo tiste spremenljivke (oz. ustrezne preklopne funkcije), ki so povezane z zelo majhnimi vrednostmi uteži. Tedaj je potrebno poenostavljanje Boole-ove funkcije regulirati s pomočjo funkcije občutljivosti, ki jo lahko izračunamo s pomočjo njenega spektra, kot bomo spoznali v nadaljevanju.

Občutljivost preklopne funkcije

Vsaki preklopni funkciji je mogoče poiskati t.i. R-W (Rademacher-Walsh) spekter. Dobimo ga tako, da najprej preklopno funkcijo $f(x),\ x \in \{0,1\}$, zapišemo kot $f(z),\ z \in \{+1,-1\}$, kjer velja relacija $z = -(2x-1)$. Množenje $f(z)$ z matriko $\mathbf{T}$ pa daje spekter logične funkcije S:

$$S = \mathbf{T} \cdot f(z) = \mathbf{T} \cdot F\,,$$

kjer je $\mathbf{T}$ R-W matrika, ki na poseben način podaja vse možne kombinacije preklopnih spremenljivk. Za slučaj treh preklopnih spremenljivk ima matrika $\mathbf{T}$ naslednjo obliko:

$$\begin{array}{l} R_0 \\ R_1 \\ R_2 \\ R_3 \\ R_1R_2 \\ R_1R_3 \\ R_2R_3 \\ R_1R_2R_3 \end{array} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

Izrazi na desni strani po vrsti od zgoraj navzdol opisujejo konstanto (R_0), spremenljivke (R_i) in operacije EX-OR nad dvema oz. tremi spremenljivkami. Analogno pravilo se uporabi tudi pri matrikah $\mathbf{T}$ za drugačno število spremenljivk.

Primer
Določite spekter preklopne funkcije $f_{DNO}(x_1, x_2, x_3) = V(2,3,4,5,7)$.

Iz enačbe $S = \mathbf{T} \cdot F$ sledi rezultat: $S^T = [-2, 2, 2, 2, 6, -2, -2, 2]$, oz.:

v	x_1	x_2	x_3	$f(x)$	$f(z) = F$	r_v	S	V
0	0	0	0	0	1	r_0	-2	0
1	0	0	1	0	1	r_1	2	1
2	0	1	0	1	-1	r_2	2	1
3	0	1	1	1	-1	r_3	2	2
4	1	0	0	1	-1	r_{12}	6	1
5	1	0	1	1	-1	r_{13}	-2	2
6	1	1	0	0	1	r_{23}	-2	2
7	1	1	1	1	-1	r_{123}	2	3

Poleg vrstnega reda (v) in spektra S je podan še V, ki podaja število indeksov pri posameznih komponentah spektra S.

Ustrezne spektralne komponente imajo naslednji pomen:

- r_0 = (število ničel v $f(x)$) − (število enic v $f(x)$)
- r_i = (število soglasij med $f(x)$ in x_i) − (število nesoglasij med $f(x)$ in x_i)
- r_j = (število soglasij med $f(x)$ in ustrezno EX-OR kombinacijo) − (število nesoglasij med $f(x)$ in ustrezno EX-OR kombinacijo), j = 12, 13, ..., 12..n

Funkcija občutljivosti Boole-ove ali preklopne funkcije kot približka k nevronski funkciji je sedaj definirana z izrazom:

$$\Psi(f) = \frac{\sum_v V \cdot r_v^2}{(2^n)^2},$$

kjer vektor V podaja število indeksov v komponentah r_v in je n število preklopnih spremenljivk. Funkcija občutljivosti $\Psi(f)$ določa povprečno število sosednih vhodnih kombinacij preklopne funkcije u' glede na kombinacijo u tako, da je $f(u) \neq f(u')$, kjer je $u, u' \in \{0, 1\}^n$. Njena vrednost se v procesu poenostavljanja Boole-ove funkcije kot približka nevronske funkcije ne sme bistveno zmanjšati.

Za prejšnji primer je funkcija občutljivosti enaka:

$$\Psi(f) = \frac{112}{(2^3)^2} = \frac{112}{64} = 1.7\,.$$

3.3.2 Vpliv učenja nevronskih mrež na strukturne parametre

Umetna nevronska mreža je kompleksen sistem in jo zato lahko tako tudi obravnavamo. Za splošne kompleksne sisteme velja, da jih lahko obravnavamo kot omrežja (grafe), kjer povezave vozlišč omogočajo njihove klasifikacije s pomočjo posebnih parametrov [22]. Najprej bomo definirali nekaj značilnih omrežij in spoznali relevantne strukturne parametre, s katerimi opisujemo kompleksne sisteme, nato pa si bomo ogledali, kako proces učenja nevronske mreže vpliva na strukturne parametre mreži ustreznega grafa.

Naključna omrežja RN (Random Networks)

Pri naključnih omrežjih je število vozlišč n fiksno. Poljubni dve vozlišči sta povezani z verjetnostjo p. V povprečju ima RN naslednje število povezav e:

$$e = pn(n-1)/2\,.$$

Porazdelitev števila povezav na vozlišče k je binomska:

$$P(k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}\,,$$

kar pomeni, da je povprečna stopnja (angl. average degree) enaka:

$$\overline{k} = p(n-1) \approx pn\,.$$

Za velike vrednosti n zavzame $P(k)$ Poissonovo obliko kot aproksimacijo za binomsko porazdelitev. Ocena za povprečno najkrajšo pot $L(p)$ (število vozlišč na poti med dvema vozliščema), ki je globalna lastnost, izhaja iz izraza:

$$\begin{aligned} (\overline{k})^L &= n \\ L(p) &= \frac{\ln n}{\ln \overline{k}} \approx \frac{\ln n}{\ln pn}\,. \end{aligned}$$

Povprečna najkrajša pot je v primeru RN nizka, kar je tipično za efekt 'majhnega sveta' SW (angl. small world).

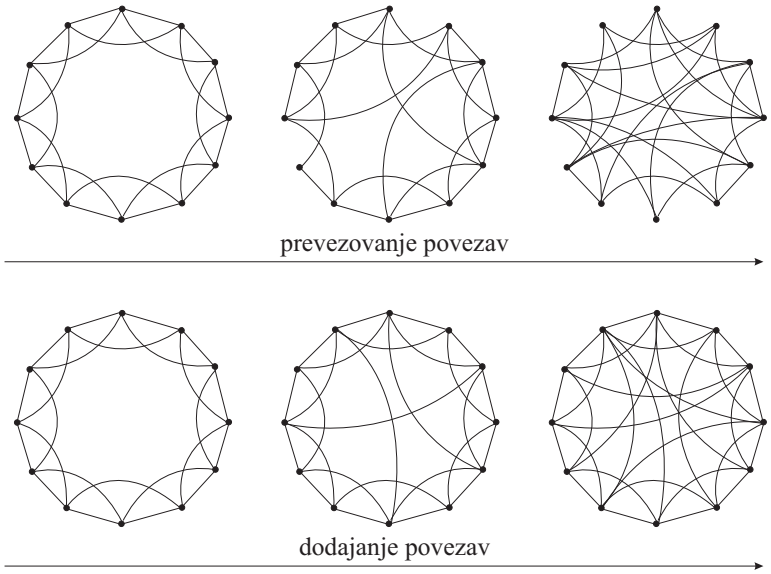
Koeficient grupiranja (angl. clustering coefficient) je pri naključnih omrežjih enak

$$C(p) = p \approx \frac{\overline{k}}{n}\,,$$

saj so povezave (angl. edges) tukaj porazdeljene naključno. Ta koeficient je precej manjši, kot bomo videli, kot pri SW omrežjih, pri istem številu vozlišč in povezav. Predstavlja lokalno lastnost, ki pomeni povprečni delež povezav sosednjih vozlišč.

Omrežja majhnega sveta SW (Small-World)

Pri tej skupini omrežij je povprečna najkrajša pot med vozlišči tudi majhna, koeficient razvrščanja (angl. clustering coefficient) pa precej večji kot pri RN. Do omrežja SW pridemo iz regularno povezanega omrežja, če z verjetnostjo $p \approx 0.1\text{-}0.3$ spremenimo regularne povezave v naključne, kot kaže Slika 3.16.



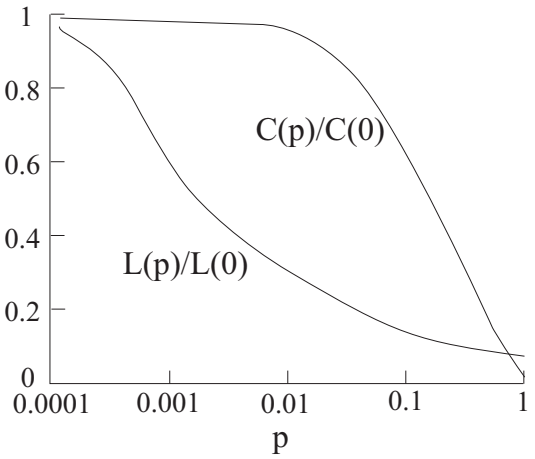
Slika 3.16: Prehod od regularnega omrežja k RN preko SW.

Oblika porazdelitve $P(k)$ je podobna kot pri RN, z vrhom pri $\overline{k}$. Veliko realnih omrežij ima lastnosti SW, kar je razvidno iz Tabele 3.2, kjer so podani parametri C za različna omrežja, skupaj s parametri primerljivega RN.

Tipične vrednosti parametrov L in C, relativno glede na izhodišče ($p = 0$), za primer omrežja SW, prikazuje Slika 3.17.

			koeficient grupiranja C	
mreža	n	z	merjeni	naključni graf
Internet	6 374	3.8	0.24	0.00060
Svetovni splet - WWW	153 127	35.2	0.11	0.00023
močnostno omrežje	4 941	2.7	0.080	0.00054
sodelovanja biologov	1 520 251	15.5	0.081	0.000010
sodelovanja matematikov	253 339	3.9	0.15	0.000015
sodelovanja filmskih igralcev	449 913	113.4	0.20	0.00025
direktorji družb	7 673	14.4	0.59	0.0019
so-pojavljanje besed	460 902	70.1	0.44	0.00015
nevronska mreža (C-elegans)	282	14.0	0.28	0.049
metabolična mreža	315	28.3	0.59	0.090
prehranski splet	134	8.7	0.22	0.065

Tabela 3.2: Parametri C za različna realna omrežja. Vir: [22]



Slika 3.17: Odvisnost L in C od verjetnosti p za slučaj omrežja SW.

Omrežje SF (Scale-Free)

Za večino obsežnih omrežij je porazdelitev povezav (angl. degree distribution) precej različna od Poisson-ove porazdelitve. Npr. WWW in Internet imata potenčno porazdelitev (angl. power-law distribution):

$$P(k) \sim k^{-\gamma}\,.$$

Do takšnih (SF) omrežij pridemo, če v naključnem omrežju število vozlišč narašča tako, da se vsako novo vozlišče poveže s starim vozliščem na osnovi

verjetnosti, ki je proporcionalna številu povezav obstoječih vozlišč. To pomeni, da se nova vozlišča povezujejo z večjo verjetnostjo s tistimi, ki imajo večje število povezav. Temu načinu pravimo prednostni dostop (angl. preferential attachment). Vsako novo vozlišče i se želi povezati z m_0 (parameter povezave) obstoječimi vozlišči v omrežju. Verjetnost $\Pi(j)$, da obstoječe vozlišče j dobi eno od novih povezav, je enaka:

$$\Pi(j) = \frac{k_t(j)}{2m_t},$$

kjer je m_t število povezav v omrežju, $k_t(j)$ pa trenutno število povezav vozlišča j v času t.

Transformacija nevronske mreže v graf

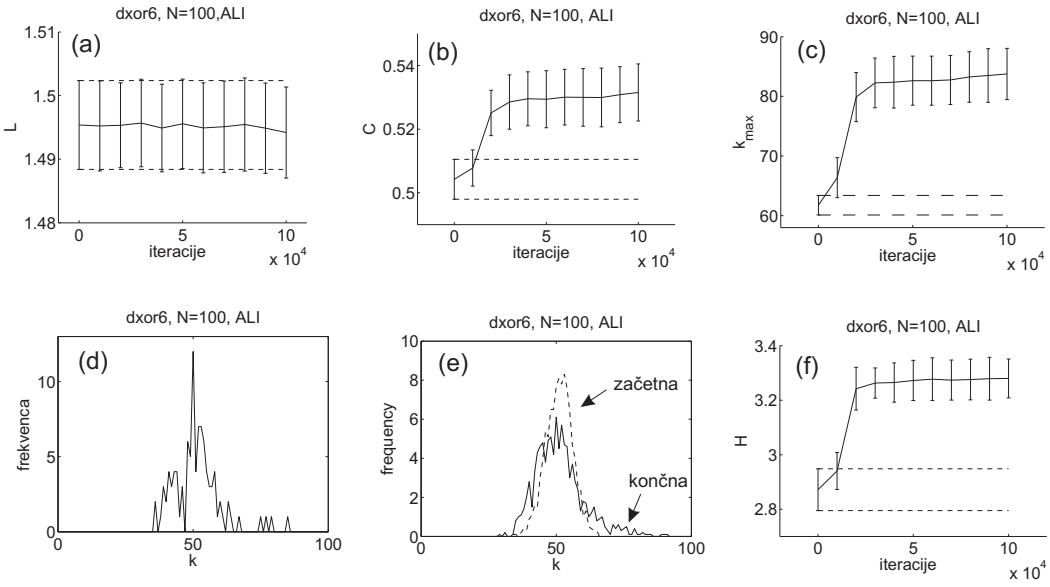
Zaradi kompatibilnosti s kompleksnimi sistemi bomo transformirali univerzalno povezano nevronsko mrežo RNN v neusmerjen graf. Transformacija je izvedena na naslednji način. Vsak nevron v mreži ustreza enemu vozlišču v grafu. Nevronski vozlišči i in j sta povezani, če velja $|\omega_{ij}| \geq \Theta \vee |\omega_{ji}| \geq \Theta$, kjer je Θ pragovna vrednost, ki je parameter v postopku, in $\vee$ logična operacija ALI oz. disjunkcija. Namesto disjunkcije je mogoče uporabiti tudi logično operacijo IN oz. konjunkcijo (&), vendar to ne vpliva bistveno na rezultate. Prag Θ je določen iz RNN mreže tako, da odstranitev vseh uteži v mreži, katerih absolutna vrednost je manjša kot Θ, ne vpliva na obnašanje mreže pri reševanju aktualnega problema. Zaradi boljše primerjave z neusmerjenim grafom so uteži, ki povezujejo izhod nevrona s svojim vhodom prepovedane že v postopku učenja, kar problem poveča in podaljša učenje. Tudi pragovne uteži (angl. bias) so prepovedane iz istega naslova in s podobnimi posledicami, vendar vse to ne vpliva pomembno na postopek učenja.

Pri uporabi pragovnega parametra v postopku transformacije RNN mreže v ustrezni graf je potrebno rešiti naslednji problem. Začetne uteži v mreži so običajno poljubno majhne (med -0.5 in $+0.5$). Če je tudi prag Θ istega velikostnega razreda, potem ustrezni graf praktično nima povezav. Zato moramo najprej mrežo (na)učiti in iz nje določiti povezljivost oz. verjetnost povezave dveh poljubnih vozlišč (pri izbranem pragu):

$$P = \frac{e}{N(N-1)},$$

kjer je e število povezav v mreži in N število nevronov oz. vozlišč. Število povezav mora biti torej konstantno in enako številu, ki smo ga določili iz (na)učene mreže. Če nas zanima graf mreže pred zaključkom učenja, moramo torej prag Θ prilagoditi tako, da je število povezav e ohranjeno. Le tako izračunane vrednosti parametrov L, C in P kažejo, kako vpliva učenje na strukturo povezav.

Zaradi ilustracije vpliva učenja nevronske mreže na omenjene strukturne parametre je bil izveden eksperiment [23], pri katerem je bila mreža RNN s 100 nevroni naučena na diskretni dinamični problem EX-OR(d), ki pomeni izvedbo operacije EX-OR nad elementoma $t-d$ in $t-d-1$, ki se nahajata v časovni vrsti binarnih znakov, $d = 6$. Slika 3.18 podaja funkcije parametrov v odvisnosti od korakov učenja.



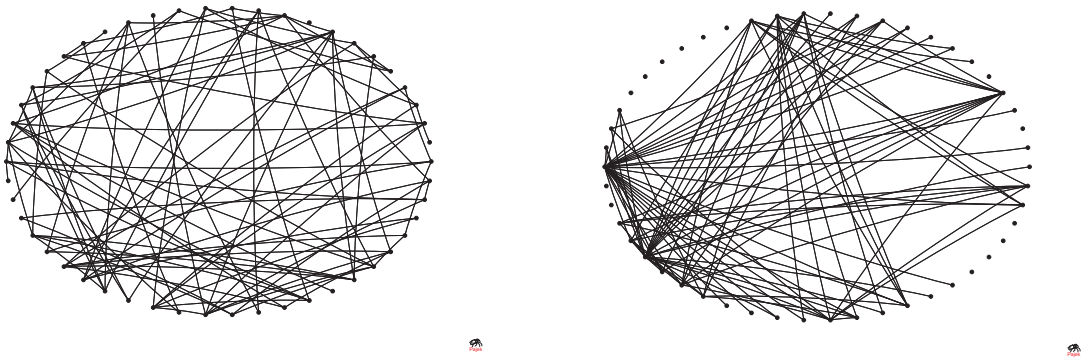
Slika 3.18: Funkcije parametrov L, C, P in H v odvisnosti od korakov učenja.

Poleg omenjenih parametrov je zaradi neizrazite funkcije parametra verjetnostne porazdelitve števila povezav P dodana še funkcija entropije H v odvisnosti od korakov učenja, ki jo določa enačba:

$$H(p_k) = -\sum_k p_k \log p_k \,,$$

kjer je p_k verjetnost, da ima vozlišče grafa ravno k povezav. Omenjene funkcije nazorno kažejo, da se parametra C in H izrazito povečujeta s koraki učenja, da je L približno konstanten, P pa v grobem ohranja začetno porazdelitev z dodanimi manjšimi vrhovi v oddaljenosti od srednje vrednosti porazdelitve.

Slika 3.19 prikazuje grafa nenaučene (levo) in naučene (desno) nevronske mreže. Grafa sta narisana s pomočjo programa Pajek [24].



Slika 3.19: Graf nenaučene (levo) in naučene (desno) nevronske mreže.

Poglavje 4

Evolucijsko računanje

4.1 Uvod

Narava je bila že od nekdaj navdih številnim znanstvenikom v okviru najrazličnejših disciplin. Za področje računalništva so bili in so še vedno zanimivi predvsem naravni postopki reševanja problemov, kjer sta se s časom izdvojili dve ciljni področji:

- Človeški možgani
- Evolucijski proces

S prvim se ukvarja področje umetnih nevronskih mrež, ki smo jih v omejenem obsegu spoznali v prejšnjem poglavju. Drugi pa predstavlja osnovo za t.i. evolucijsko računanje, ki je predmet tega poglavja [25].

Evolucijsko računanje (ER) je danes pomembna raziskovalna disciplina v okviru področja računalniških znanosti. Predstavlja poseben način računanja, ki črpa ideje iz procesa naravne evolucije v kombinaciji s posebnim načinom reševanja problemov na osnovi napak (angl. trial-and-error). V najosnovnejši obliki si pod naravno evolucijo predstavljamo naslednje: v danem okolju je populacija posameznikov (angl. individuals), ki se borijo za obstanek in se ohranjajo z reprodukcijo. Okolje določa stopnjo prilagajanja posameznikov pri doseganju njihovega osnovnega cilja. Od tega je odvisna njihova sposobnost preživetja. V kontekstu reševanja problemov veljajo naslednje analogije:

EVOLUCIJA	REŠEVANJE PROBLEMOV
Okolje	Problem
Posameznik	Možna rešitev (kandidat)
Prilagajanje	Kvaliteta

Seveda je bila Darwin-ova teorija tista, ki je ponudila razlago za razvoj življenja na zemlji, kjer igra **naravni izbor** osrednjo vlogo. V okolju, kjer je mogoča omejena eksistenca števila posameznikov z osnovnim instinktom reprodukcije, je selekcija neizbežna, da se populacija ne povečuje eksponentno. Selekcija, ki jo povzroča tekmovanje v prilagajanju okolju (naravi) med posamezniki, je naravna in predstavlja boj za preživetje. Druga pomembna značilnost Darwinove teorije je v eksistenci različnih **fenotipov**, t.j. obnašanj posameznikov v okolju. Vsak posameznik predstavlja edinstveno kombinacijo obnašanja (angl. phenotypic traits), ki ga vrednoti njegovo okolje. Če je ocena posameznika ugodna, potem se njegovo obnašanje prenaša na potomce, sicer brez potomcev zamre. V zvezi s tem je Darwin verjel, da se pri prenosu na potomce pojavljajo manjše fenotipske modifikacije oz. **mutacije** (angl. mutation), zaradi česar prihaja do sprememb v obnašanju posameznikov. Posamezniki v populaciji so torej elementi selekcije, katerih reprodukcija (in s tem njihovo obnašanje v okolju) je odvisna od njihove prilagoditve okolju. Ker se bolj prilagodljivi posamezniki reproducirajo, njihovi mutirani potomci predstavljajo dodatno možnost za še boljše prilagajanje okolju.

Pri izvajanju procesa naravne evolucije igra pomembno vlogo molekularna genetika, saj razjasnjuje vzroke za različne fenotipske lastnosti in njihovo dedovanje. Posamezniku pripisuje dualne entitete: zunanje fenotipske lastnosti in notranje genotipske strukturne značilnosti. Povedano drugače, **genotip** posameznika predstavlja kodni zapis za njegov fenotip. Pri tem so **geni** funkcijske enote genotipov. Kombinacija lastnosti dveh posameznikov pri potomcih se imenuje **križanje** (angl. crossover), naključna sprememba posameznika pa njegova **mutacija**. Čeprav se zdi, da križanje ni bistveno pri naravni evoluciji, pa je bilo dokazano, da bistveno vpliva na konvergenco želenih lastnosti obnašanja, s tem pa seveda igra pomembno vlogo v hitro se spreminjajočih okoljih, kakršnega npr. predstavljajo tudi pogoji življenja na zemlji.

Poleg motivacije za uporabo evolucijskega računanja, omenjenega na začetku, obstajajo še drugi razlogi za njeno afirmacijo. Enega predstavlja poenoten način reševanja številnih različnih problemov, drugi pa se skriva v človeški radovednosti, saj je s pomočjo evolucijskega računanja mogoče izvajati eksper-

imente, ki nimajo podlage v tradicionalni biologiji. Tipičen primer predstavlja t.i. Lamarck-ov model evolucije, ki predpostavlja, da se pridobljene izkušnje prenašajo na potomce. Čeprav se ne ujema z lastnostmi naravne evolucije, pa ga je mogoče uporabiti pri reševanju različnih tehničnih problemov.

Danes velja prepričanje, da je prednost evolucijskega računanja pri reševanju problemov iz najrazličnejših področij v tem, da je v mnogih ozirih univerzalno in torej neodvisno od narave problema. Vzemimo splošen model sistema z vhodnimi spremenljivkami, modelskimi enačbami in izhodnimi spremenljivkami tako, da je pri znanih modelskih enačbah sistema in znanih vrednostih vhodnih spremenljivk, mogoče določiti vrednosti izhodnih spremenljivk. Tako lahko evidentiramo tri osnovne tipe problemov, ki so primerni za evolucijsko reševanje, glede na to, česa v sistemu ne poznamo:

- *Optimizacijski problem* imamo takrat, kadar poznamo modelske enačbe in želene izhodne vrednosti, iščemo pa optimalne vhodne vrednosti.
- *Identifikacijski problem* išče modelske enačbe pri znanih parih vhodnih in izhodnih spremenljivk.
- *Simulacijski problem* pa išče vrednosti izhodnih spremenljivk, če poznamo vrednosti vhodnih spremenljivk in modelske enačbe.

Evolucijsko računanje sledi evolucijskemu algoritmu (EA). Ker obstaja več različnih vrst takšnih algoritmov, bo najprej podana splošna shema evolucijskega algoritma, ki je skupna vsem njenim izpeljankam. Nato pa bodo v naslednjih poglavjih podrobneje opisani genetski algoritem (GA), kjer so posamezniki (kandidati za rešitev problema) predstavljeni s serijo znakov iz končne (običajno binarne) abecede, evolucijske strategije (ES), pri katerih so kandidati največkrat vektorji z realnimi komponentami, evolucijsko programiranje (EP), ki uporablja pri reševanju problemov končne avtomate in genetsko programiranje (GP), kjer gre za avtomatizirano kodiranje na osnovi drevesnih programskih segmentov.

Splošno shemo evolucijskega algoritma podaja naslednja psevdokoda:

```
BEGIN
    INICIALIZACIJA populacije z naključno izbiro kandidatov
    OCENA vseh kandidatov
    REPEAT UNTIL (ZAKLJUČNI POGOJ NI IZPOLNJEN) DO
```

```
        1 IZBERI pare staršev
        2 KRIŽAJ pare staršev
        3 MUTIRAJ dobljene potomce
        4 OCENI nove kandidate
        5 IZBERI kandidate za novo generacijo
    END DO
END
```

4.2 Genetski algoritmi

Genetski algoritem je v obliki, kot bo predstavljen, prvi objavil Holland [26]. Zanj je značilna binarna predstavitev kandidatov (genotipov) za rešitev problema, selekcija na osnovi proporcionalnega prilagajanja okolju (angl. fitness function), nizka verjetnost mutacije in križanja kot bistvena genetska operatorja za generiranje novih posameznikov ali rešitev za obravnavani problem.

4.2.1 Predstavitev posameznikov

Pri predstavitvi kandidatov za rešitev problema je zelo pomembno izbrati 'pravilno' predstavitev, to je takšno strukturo podatkov, ki se najbolj prilega danemu problemu. Zato je to tudi eden najtežjih delov pri razvoju evolucijskega algoritma. Pogosto je odločitev posledica izkušenj in dobrega poznavanja problemske domene. V nadaljevanju bomo podrobneje spoznali nekaj najpogosteje uporabljenih predstavitev in genetske operatorje, ki posameznim predstavitvam najbolje ustrezajo. To pa ne pomeni, da bo izmed opisanih predstavitev mogoče najti najboljšo za naš problem. Velikokrat je tudi kombinacija operatorjev, ki bodo podani v nadaljevanju, najbolj naravna ali najprimernejša rešitev za konkretni problem in ne posamezne predlagane variante operatorjev.

Binarna predstavitev

To je ena prvih predstavitev, ki se je v preteklosti zmotno uporabljala neodvisno od problema, ki ga je reševala. Genotip je pri binarni predstavitvi seveda

enostavno kar niz (angl. string) binarnih znakov.

Dejansko je za vsak konkretni problem potrebno določiti podatkovno strukturo genotipa in način, kako bo genotip interpretiran kot fenotip. Za določene probleme je preslikava genotip/fenotip naravna (npr. problemi z Boolovimi ali preklopnimi odločitvami), pogosto pa binarni niz pomeni le kodiranje nebinarne informacije.

Pogost problem pri binarnem kodiranju je, da imajo različni biti različen pomen. Tedaj je lahko rešitev Gray-eva koda, pri kateri je Hamming-ova razdalja med kodama dveh zaporednih celih števil, enaka 1.

Celoštevilčna predstavitev

Tipičen primer, kjer je celoštevilčna predstavitev bolj primerna kot binarna, je, kadar imamo opravka s problemom, kjer iščemo optimalne vrednosti za niz spremenljivk, ki zavzemajo le celoštevilčne vrednosti. Tedaj so vrednosti lahko omejene ali pa ne. Pri snovanju kodiranja in genetskih operatorjev je pomembno upoštevati evidentirane relacije med posameznimi vrednostmi, npr. med vrstilnimi števniki, kjer je bližnje kodiranje zaporednih vrednosti naravno kodiranje.

Realna predstavitev

Pogosto je najbolj smiselna odločitev za predstavitev genotipa, če vzamemo niz realnih vrednosti. Takšen primer nastopi, kadar vrednost gena ustreza zvezni porazdelitvi. Ker je znotraj računalnika realna vrednost omejena (kvantizirana) s širino računalnikove besede, je tedaj dejanska predstavitev izvedena v formatu plavajoče vejice.

Permutacijska predstavitev

Pri mnogih problemih je odločitev vezana na vrstni red dogodkov. Tedaj je naravna predstavitev permutacija niza celih števil, kar pomeni, da se posamezen znak lahko pojavi le enkrat v seriji. Posledica tega je, da potrebujemo operator

spreminjanja predstavitve, ki ohranja omenjeno lastnost permutacije. Obstajata dva načina za kodiranje permutacij. V prvem, ki se pogosteje uporablja, i-ti element permutacije predstavitve označuje dogodek, ki se zgodi na tem mestu osnovne sekvence. V drugem pa vrednost i-tega elementa permutacije označuje pozicijo v sekvenci, ki jo določa prvi način. Npr. pri predstavitvi (A,B,C,D) in permutaciji (3,1,2,4), prvi način določa serijo oz. predstavitev (C,A,B,D), drugi pa (B,C,A,D).

4.2.2 Mutacija

Mutacija je generično ime za operator, ki spreminja genotip samo na osnovi enega prednika z naključno spremembo njegovega genotipa. Izvedba mutacije je odvisna od kodiranja oz. predstavitve genotipa. Ne da bi se spuščali v sicer izredno pomembno izbiro parametrov (ker je odvisna od problema), ki tvorijo genotip, si bomo ogledali različne izvedbe mutacije glede na predstavitev genotipa.

Mutacija binarne predstavitve

Najpogosteje uporabljena mutacija binarnih serij deluje nad vsakim genom posebej tako, da se z neko (majhno) verjetnostjo p_m (ali p_{mut}) spremenijo posamezni biti, ki predstavljajo gen. S tem se dejansko število spremenjenih bitov praktično menja v vsakem koraku mutacije, saj je odvisno od naključnega generatorja, s katerim jih verjetnostno spreminjamo. Izbira p_m zavisi od tega, kakšen rezultat pričakujemo, ali celotno populacijo ustreznih posameznikov, ali pa je dovolj en sam zadovoljiv posameznik. Izkustveno pravilo pravi, da je dobro izbrati takšno verjetnost za p_m, s katero je zagotovljena v povprečju ena sprememba gena na populacijo staršev oz. potomcev (manjšo od obeh).

Mutacija celoštevilčne predstavitve

Obstajata dve glavni obliki mutacije, pri obeh se vsak gen posebej mutira z verjetnostjo p_m.

A. Naključno resetiranje: je podobno binarni mutaciji, le da se tukaj z ver-

jetnostjo p_m izbere z enako verjetnostjo eno izmed možnih vrednosti gena.

B. Drseča mutacija: pomeni spremembo gena z verjetnostjo p_m tako, da se trenutni vrednosti gena doda ali odvzame majhno vrednost. Te majhne vrednosti se običajno izberejo iz takšne porazdelitve, ki je simetrična okoli 0 in ki z večjo verjetnostjo izbere manjše cele vrednosti kot večje.

Mutacija predstavitve s plavajočo vejico

Za vsak realni gen se predpostavlja, da so poznane njegova minimalna in maksimalna možna vrednost, npr. (L_i, U_i) za gen g_i. Mutacija je tedaj lahko ali uniformna, kar pomeni, da je nova vrednost gena dobljena iz homogene porazdelitve v določenih mejah gena, ali pa s spreminjanjem vrednosti gena na osnovi majhne vrednosti iz naravne zvezne porazdelitve v okolici 0. Alternativa Gauss-ovi porazdelitvi je lahko Cauchy-jeva, ki ima debelejši rep, kar pomeni večjo verjetnost za izbiro večjih modifikacij.

Mutacija permutacijske predstavitve

V tem primeru genov ni več mogoče obravnavati posebej. Zato parameter p_m pomeni verjetnost zamenjave celotnega niza namesto le posameznega gena. Najbolj znani so trije primeri mutacije:

A. Zamenjava (angl. swap): pomeni naključni izbor dveh pozicij v nizu in njuna zamenjava

B. Vstavitev (angl. insertion): zahteva naključno izbiro dveh vrednosti in premik ene na sosednje mesto k drugi tako, da se ostale vrednosti ustrezno razmaknejo. Spodnja slika ilustrira omenjeno mutacijo:

$$1\ \underline{2}\ 3\ 4\ \underline{5}\ 6\ 7\ 8\ 9 \rightarrow 1\ 2\ 5\ 3\ 4\ 6\ 7\ 8\ 9$$

C. Inverzija (angl. inversion): tudi ta mutacija predpostavlja naključno izbiro dveh pozicij, nato pa se zamenja vrstni red pozicij med njima:

$$\underline{1}\ 2\ 3\ 4\ 5\ \underline{6}\ 7\ 8\ 9 \rightarrow 1\ 5\ 4\ 3\ 2\ 6\ 7\ 8\ 9$$

4.2.3 Križanje

Križanje (angl. crossover, recombination) je proces, v katerem nastane nov posameznik (genotip) na osnovi informacij, vsebovanih v dveh (ali več) posameznikih (prednikih). Ta operator je gotovo ena od lastnosti, ki najbolj razlikuje genetske algoritme ali ostale evolucijske algoritme od drugih globalnih optimizacijskih algoritmov.

Operator križanja se izvaja v odvisnosti od parametra verjetnosti p_c (ali p_{cross}), katere vrednost je tipično v območju $[0.5, 1.0]$. Običajno sta izbrana dva prednika ali starša (angl. predecessors, parents). Če je nato naključna vrednost med 0 in 1 manjša od p_c, potem se nad njima izvede križanje, ki generira dva potomca, sicer se enostavno oba prednika prekopirata v potomca.

Križanje binarnih predstavitev

Običajno se križanje binarnih predstavitev izvaja nad dvema prednikoma oz. njunima genotipoma. Največ se uporabljajo trije tipi križanj: enotočkovno, N-točkovno in uniformno ali homogeno križanje.

A. *Enotočkovno križanje*: zahteva naključno izbiro enega števila iz področja $[0,\ l-1]$, kjer je l dolžina binarne kode, ki predstavlja genotip. Ta točka razdeli genotipa prednikov na dva dela. Križanje ustvari dva potomca tako, da se dela genotipov, desno od izbrane točke zamenjata, levo od nje pa ohranita:

$$\begin{array}{cccc|ccccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ & & & & & & & & \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array} \quad \rightarrow \quad \begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ & & & & & & & & \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{array}$$

B. *N-točkovno križanje*: predpostavlja generiranje N naključnih števil iz področja $[0,\ l-1]$, nato pa ustvari dva potomca tako, da alternativno vzame segmente dveh prednikov med izbranimi točkami. Za slučaj $N = 2$ (ki je tudi najpogostejši) to pomeni:

$$\begin{array}{cccc|cc|ccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ & & & & & & & & \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array} \quad \rightarrow \quad \begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & & & & & & \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}$$

C. *Uniformno križanje*: obravnava gene neodvisno tako, da pri vsakem genu z naključnim generatorjem določi, kako se bosta gena prednikov prenašala na potomca. Če je naključna vrednost < 0.5, se prenese gen prvega prednika k prvemu potomcu, če pa je ≥ 0.5, se prenese gen drugega prednika k prvemu potomcu. Drugi potomec je nato rezultat inverzne preslikave. Na spodnji sliki podčrtani geni določajo prenos od prvega prednika, nepodčrtani pa od drugega:

$$\begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ & & & & & & & & \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array} \quad \rightarrow \quad \begin{array}{ccccccccc} \underline{0} & 1 & \underline{0} & \underline{0} & 0 & 0 & \underline{0} & 0 & \underline{0} \\ & & & & & & & & \\ 1 & \underline{0} & 0 & 1 & \underline{1} & \underline{0} & 0 & \underline{0} & 1 \end{array}$$

Križanje celoštevilčnih predstavitev

Ker križanje ne spreminja genov, je križanje celoštevilčnih predstavitev analogno križanju binarnih predstavitev.

Križanje predstavitev s plavajočimi vejicami

V tem primeru sta možni dve varianti križanja. Po prvi, ki se imenuje *diskretna rekombinacija*, se vsako število v plavajoči vejici obravnava kot bit pri binarni predstavitvi, postopek križanja pa je potem enak kot pri križanju binarnih predstavitev. To ima slabo stran, da le mutacija vnaša nove vrednosti v potomce. Po drugi varianti, imenovani *aritmetična rekombinacija*, pa se najprej določi naključno točko križanja, nato pa se za oba potomca določi za vse gene desno od točke križanja srednja vrednost glede na vrednosti genov obeh prednikov:

$$z_i = \alpha x_i + (1 - \alpha) y_i \,,$$

kjer je $\alpha \in [0, 1]$. Za slučaj točke križanja $k = 6$ in $\alpha = 0.5$ takšno križanje vodi k naslednjim potomcem:

$$\begin{array}{l|l} 0.1\ 0.2\ 0.3\ 0.4\ 0.5\ 0.6 & 0.7\ 0.8\ 0.9 \\ 0.3\ 0.2\ 0.3\ 0.2\ 0.3\ 0.2 & 0.3\ 0.2\ 0.3 \end{array} \rightarrow \begin{array}{l|l} 0.1\ 0.2\ 0.3\ 0.4\ 0.5\ 0.6 & 0.5\ 0.5\ 0.6 \\ 0.3\ 0.2\ 0.3\ 0.2\ 0.3\ 0.2 & 0.5\ 0.5\ 0.6 \end{array}$$

4.2.4 Modeli populacij

Pri genetskih algoritmih se v zvezi s populacijami uporabljata dva modela: generacijski (angl. generational) in stabilni (angl. steady state) model populacij.

Pri generacijskem modelu populacij pričnemo s populacijo velikosti μ, iz katere izbiramo populacijo staršev. Nato se določi s pomočjo križanja in mutacije populacija potomcev velikosti $\lambda\,(=\mu)$, sledi pa zamenjava generacij, oz. potomci tvorijo naslednjo generacijo v procesu evolucije.

V stabilnem modelu populacij celotna populacija ni zamenjana naenkrat, temveč le njen del, oz. $\lambda < \mu$. Procent zamenjane populacije se imenuje generacijska luknja (angl. generational gap) in je določena z λ/μ.

Populacija igra pomembno vlogo v dveh korakih evolucijske zanke. Prvič, pri izbiri prednikov, ki so potrebni v procesu križanja in drugič, pri selekciji posameznikov za preživetje oz. prenos v naslednjo generacijo.

Izbira prednikov

Najobičajnejša izbira prednikov izhaja iz cenilne funkcije (angl. fitness function). Če z f_i označimo absolutno vrednost cenilne funkcije za i-ti genotip g_i, potem je verjetnost, da izberemo g_i za prednika v procesu križanja, podana z enačbo:

$$p_i = \frac{f_i}{\sum_{j=1}^{\mu} f_j}\,.$$

Vendar takšen način izbire povzroča določene težave. Prvič, izjemni posamezniki hitro prevzamejo celotno populacijo (angl. premature convergence), in drugič, če so cenilne funkcije zelo blizu skupaj, potem praktično ni efekta selekcije, ker je izbira tako rekoč naključna in tretjič, process evolucije se obnaša različno pri transpoziciji iste cenilne funkcije ($f \rightarrow f + K$), ker ta spreminja

verjetnosti izbire posameznih kandidatov. Naslednje tri izbire delno odpravljajo zgornje pomanjkljivosti.

A. Izbira z rangiranjem: pri tem je preslikava od rangiranja posameznikov do verjetnosti njihove izbire poljubna, največkrat se uporablja linearna ali eksponentna monotono padajoča funkcija.

B. Ruletni postopek: temelji na ruletnem kolesu, katerega površine izsekov določajo verjetnosti izbire posameznikov. Pri izbiri λ posameznikov iz niza μ prednikov za proces križanja to pomeni naslednje. Pri določeni urejenosti populacije (ali z rangiranjem ali naključni) od 1 do μ, izračunamo seznam vrednosti $[a_1, a_2, ..., a_\mu]$ tako, da je $a_i = \sum_{j=1}^{i} P_{sel}(j)$, kjer je $P_{sel}(j)$ definirana kot verjetnost izbire oz. selekcije, proporcionalna oceni (prileganju) ali rangiranju posameznika j. To pomeni seveda, da je $a_\mu = 1.0$, kar ustreza celotni površini ruletnega kroga. Ruletni postopek sedaj na osnovi naključnega števila (točke, v kateri se ruleta ustavi) določi posameznika glede na to, ali pripada njegovemu deležu v površini kolesa.

C. Turnirska izbira: predpostavlja, da ne poznamo vseh lastnosti posameznikov. Tedaj z zaporedno izbiro enega in naključno izbiro drugega posameznika (z ali brez vračanja) in primerjavo njunih cenilnih funkcij izberemo boljšega kot prednika v procesu križanja. Postopek ponovimo μ krat. Zaradi ocenjevanja relativnega prileganja ima turnirska izbira podobne lastnosti kot izbira na osnovi rangiranja.

Izbira preživetja

Od izbire preživetja (angl. survivor selection) zavisi nadzor procesa evolucije, kjer se v vsakem koraku število prednikov μ in potomcev λ reducira v μ posameznikov naslednje generacije. Temu procesu pravimo tudi zamenjava (angl. replacement).

Najpogosteje je v rabi izbira na osnovi cenilne funkcije (angl. fitness-based replacement), kjer sta uporabni tako izbira z verjetnostjo, proporcionalno cenilni funkciji, kot tudi turnirska izbira.

Eden od načinov izbire preživetja je tudi zamenjava najslabših predstavnikov (angl. replace worst), kjer najslabših λ predstavnikov populacije izberemo za

zamenjavo. To lahko vodi k hitremu izboljšanju glede povprečnega ocene kandidatov, lahko pa tudi k prezgodnji konvergenci in slabšemu končnemu rezultatu. Zato se omenjeni način uporablja v kombinaciji z velikimi populacijami in s pravilom prepovedi dupliciranja kandidatov.

Nasprotje prejšnjemu pravilu je pravilo elitizma, ki želi preprečiti izgubo trenutno najboljšega kandidata v populaciji. Zato se ta vedno ohrani pri prehodu v novo generacijo.

Primer: Genetski algoritem
Iskanje x, pri katerem je funkcija $f(x) = |x \sin(\sqrt{|x|})|$ na intervalu [0, 1000] maksimalna.

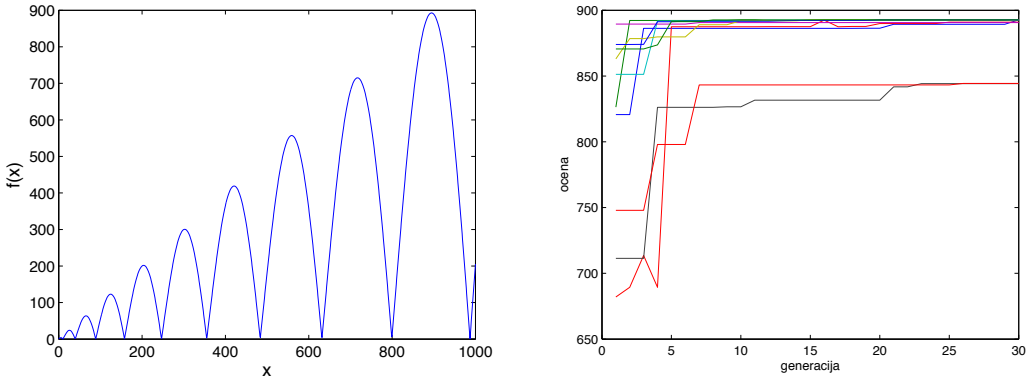
Funkcija $f(x)$ je na Sliki 4.1 levo. Njen maksimum na danem intervalu znaša $f = 892.7$ pri vrednosti $x = 894.7$.

Npr., da za genetski algoritem izberemo naslednje parametre:
- število genov v kromosomu (genotipu) ... $n_G = 20$
- število kromosomov ... 30
- število generacij ... 30

Število genov določa, kako natančno obravnavamo neodvisno spremenljivko x. Ker imamo binarno predstavitev, so možne le diskretne vrednosti x, ki jih je $2^{n_G} = 2^{20}$. Vrednost kromosoma s samimi ničlami predstavlja najnižjo vrednost x na danem intervalu, tj. 0, vrednost kromosoma s samimi enicami pa predstavlja najvišjo vrednost x na danem intervalu, tj. 1000. Ostale vrednosti kromosomov po vrsti pa predstavljajo vrednosti x, ki si sledijo s korakom $1000/(2^{20} - 1) \approx 0.00095$.

Na Sliki 4.1 desno je prikazan potek ocene najboljšega kromosoma v generaciji za 10 tekov algoritma. Genetski algoritem v večini primerov najde kromosom, ki ustreza maksimumu dane funkcije:

`11100101000100110011` .



Slika 4.1: Funkcija $f(x) = |x \sin(\sqrt{|x|})|$ na intervalu [0, 1000] (levo) in potek ocene najboljšega kromosoma v generaciji za 10 tekov algoritma (desno).

4.3 Evolucijske strategije

Evolucijske strategije (ES) so eden od naslednjih pomembnih članov družine evolucijskih algoritmov. Z njimi navadno ilustriramo pomembno lastnost evolucijskega računanja, to je **samo-adaptacijo** (angl. self-adaptation) strateških parametrov. Pri tem samo-adaptivnost pomeni, da se nekateri parametri evolucijskega algoritma med njegovim izvajanjem spreminjajo na poseben način in sicer tako, da so vključeni v genotip ali kromosom in se razvijajo skupaj z ostalimi njegovimi elementi (geni). Ta lastnost je danes zaradi izjemnih rezultatov praktično neizogibna pri vseh modernih evolucijskih postopkih računanja. Glede ostalih lastnosti, ki smo jih spoznali do sedaj, pa velja za ES naslednje:

- Predstavitev: vektorji realnih vrednosti
- Križanje: diskretno ali aritmetično
- Mutacija: dodajanje majhnih vrednosti iz Gauss-ove porazdelitve
- Izbira prednikov: naključna iz uniformne porazdelitve
- Izbira preživetja: (μ, λ) = izbira samo med potomci ali $(\mu + \lambda)$ = izbira med predniki in potomci
- Posebnost: samo-adaptacija velikosti koraka mutacije

Evolucijske strategije je prvi definiral Rechenberg v šestdesetih letih [27]. Osnovni algoritem ES za primer abstraktnega problema minimizacije n-dimenzionalne funkcije podaja psevdo-koda:

```
BEGIN
    t = 0;
    definiraj začetno točko (x_1^t, ..., x_n^t) ∈ R^n;
    REPEAT UNTIL (pogoj zaustavitve) DO
        določi z_i iz G-porazdelitve za vse i neodvisno;
        y_i^t = x_i^t + z_i za vse i ∈ {1, ..., n};
        IF f(x^t) ≤ f(y^t) THEN
            x^{t+1} = x^t;
        ELSE
            x^{t+1} = y^t;
        t = t + 1;
    END DO
END
```

Naključno število, ki se dodaja k vsaki komponenti novega vektorja x^{t+1}, določimo iz Gauss-ove porazdelitve s srednjo vrednostjo 0 in standardno deviacijo σ. Ta porazdelitev je simetrična okoli 0, standardna deviacija σ pa se s časom zmanjšuje, kar pomeni, da se zmanjšujejo tudi naključna števila, ki jih dodajamo komponentam vhodnega vektorja. Zato je σ parameter algoritma, ki določa obseg perturbacije vhodnih komponent zaradi mutacije. Zato imenujemo σ tudi *velikost koraka mutacije* (angl. mutation step size). S tem v zvezi obstajajo teoretični in eksperimentalni dokazi [27], ki potrjujejo veljavnost t.i. *1/5 pravila uspešnosti* (angl. 1/5 success rule), ki pravi, da je razmerje 1/5 med uspešnimi mutacijami in vsemi mutacijami tisto, ki odloča o spremembi velikosti koraka mutacije ali σ. Če je omenjeno razmerje večje od 1/5, potem se mora σ povečati, da se razširi prostor iskanja, če pa je manjše od 1/5, se mora σ zmanjšati, da se podrobneje razišče prostor okoli trenutne rešitve. V primeru enakosti z 1/5 se širina porazdelitve ohrani. To pravilo se izvaja v periodičnih intervalih, npr. po k korakih se spremeni σ na osnovi izrazov:

$$\sigma = \begin{cases} \sigma/c\,, & \text{če je } p_s > 1/5 \\ \sigma \cdot c\,, & \text{če je } p_s < 1/5 \\ \sigma\,, & \text{če je } p_s = 1/5 \end{cases},$$

kjer je p_s relativna frekvenca uspešnih mutacij preko določenega števila poizkusov in je c parameter v območju $0.817 \leq c \leq 1$ [28]. To pravilo torej

uporablja mehanizem spreminjanja velikosti mutacije, ki bazira na povratni informaciji s strani iskalnega procesa.

Bistvene karakteristike evolucijskih strategij so naslednje:

- Uporabljajo se tipično v primeru optimizacije zveznih parametrov
- Pri kreiranju potomcev je velik poudarek na mutaciji
- Mutacija se izvaja z dodajanjem šuma iz Gauss-ove porazdelitve
- Parametri mutacije se spreminjajo v času izvajanja algoritma.

4.3.1 Predstavitev

Ker gre pri ES za optimizacijo zveznih parametrov, je standardna predstavitev niza spremenljivk $x_1, ..., x_n$ naravno vezana na predstavitev s plavajočo vejico. Zato je prostor genotipov isti kot prostor fenotipov, kar pomeni, da kodiranje ni potrebno. Ker se danes praktično vedno uporablja pri ES strategija samo-organizacije, predstavlja vektor $\mathbf{x}$ le del genotipa. Drugi del predstavljajo strateški parametri, npr. parametri operatorja mutacije in parametri interakcije med velikostmi korakov različnih spremenljivk, kar bo podrobneje razloženo v nadaljevanju.

V splošnem imamo torej pri ES opraviti z naslednjo obliko genotipov:

$$< x_1, ..., x_n, \sigma_1, ..., \sigma_{n_\sigma}, \alpha_1, ..., \alpha_{n_\alpha} > \ .$$

4.3.2 Mutacija

Operator mutacije pri ES bazira na normalni ali Gauss-ovi porazdelitvi, ki zahteva dva parametra, srednjo vrednost ξ in standardno deviacijo σ. Mutacija se izvede tako, da se doda neko vrednost Δx_i k vsaki komponenti x_i, kjer je Δx_i izbrana naključno iz Gauss-ove porazdelitve $G(\xi, \sigma)$ z naslednjo funkcijo gostote verjetnosti (angl. probability density function, pdf):

$$p(\Delta x_i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\Delta x_i - \xi)^2}{2\sigma^2}} = G(x; \xi, \sigma)\,.$$

V praksi je srednja vrednost vedno enaka nič, $\xi = 0$, vektor $\mathbf{x}$ pa je tedaj mutiran na osnovi izrazov:

$$x_i' = x_i + G(0, \sigma)\,.$$

Majhne mutacije so tedaj bolj verjetne kot velike. Posebnost mutacije pri ES pa je, da se s časom spreminjajo tudi strateški parametri, ki so zato vključeni tudi v proces izbire oz. selekcije. Pri tem je pomembno, da koraki mutacije niso definirani s strani uporabnika, temveč so predmet evolucije, skupaj s parametri aplikacije. Pri tem je bistveno, da se najprej spremenijo strateški parametri, nato pa se mutirajo še parametri x_i z novimi vrednostmi σ. Tedaj je novi posameznik (x', σ') ocenjen dvakrat, prvič z izborom preživetja nad $f(x')$ in drugič z izborom kandidatov za proces križanja, ki se izvede indirektno: velikost koraka se ocenjuje glede na uspešnost potomca v smislu izbora preživetja. Torej predstavlja posameznik (x', σ') tako dobrega kandidata x', ki je preživel izbor, in dobro strateško kombinacijo σ', ki se je izkazala uspešna pri generiranju dobrega kandidata x' iz x.

Pri spreminjanju velikosti mutacije velja predpostavka, da v različnih okoliščinah, npr. v različnih časih ali prostorih, velikost mutacije vpliva različno na proces optimizacije. Tedaj je proces samo-adaptacije tisti, ki prilagaja strategijo mutacije okoliščinam.

4.3.3 Križanje ali rekombinacija

Osnovna shema rekombinacije pri ES vsebuje dva prednika (starša), ki ustvarita enega potomca (otroka). Da pridobimo λ potomcev, se mora rekombinacija izvesti ravno λ-krat. Obstajata dve varianti rekombinacije, glede na način križanja prednikov. Pri *diskretni rekombinaciji* (angl. discrete recombination) je element genotipa potomca izbran naključno (z enako verjetnostjo) med elementoma obeh prednikov, pri *posredni* ali *vmesni rekombinaciji* (angl. intermediary recombination) pa s povprečenjem vrednosti obeh potomcev:

$$z_i = \begin{cases} x_i \text{ ali } y_i & \text{naključen izbor (diskretno križanje)} \\ (x_i + y_i)/2 & \text{srednja vrednost (vmesno križanje)} \end{cases}.$$

Razširjava te sheme dopušča več prednikov, ki sodelujejo v procesu rekombinacije, v skrajnosti vseh μ predstavnikov populacije. Takšna več-starševska rekombinacija se imenuje tudi *globalna rekombinacija*, osnovna z dvema prednikoma pa po analogiji *lokalna rekombinacija*. Evolucijske strategije tipično

uporabljajo globalno rekombinacijo, kjer je priporočljivo uporabiti diskretni tip za aplikacijske parametre in vmesni tip za strateške parametre.

4.3.4 Izbira prednikov

Pri ES izbira prednikov ni vezana na cenilno funkcijo. Kadar operator križanja potrebuje prednike, uporabi homogeno (uniformno) porazdelitev nad populacijo in naključno izbere potrebno število prednikov. Glede prednikov obstaja razlika v terminilogiji med genetskimi algoritmi in ES. Pri ES je celotna populacija obravnavana kot množica prednikov, pri genetskem algoritmu pa izraz prednik določa člana populacije, ki je udeležen v procesu rekombinacije.

4.3.5 Izbira preživetja

Po nastanku λ potomcev in izračunu njihovih funkcij prikladnosti, je izbranih najboljših μ posameznikov deterministično, ali samo izmed potomcev, kar označujemo z izborom (μ, λ), ali pa iz unije prednikov in potomcev, oz. izborom (μ + λ). Oba izbora temeljita na rangiranju posameznikov in ne na njihovemu prileganju problemu.

ES v splošnem uporablja izbor (μ, λ), ki ima pred alternativo ($\mu + \lambda$) naslednje prednosti:

- Ker ne upošteva prednikov, je mogoč odmik od lokalnih minimumov, kar je prednost v primeru večmodalnih problemov (problemov z več izhodnimi spremenljivkami)
- Če se cenilna funkcija spreminja s časom, potem izbor (μ + λ) ohranja stare rešitve, kar preprečuje sledenje novim optimalnim vrednostim
- Za izbor (μ + λ) je mehanizem samo-adaptacije strateških parametrov ovira, ker lahko napačni strateški parametri preživijo številne generacije, to pa lahko vodi k slabšim potomcem in zaradi elitizma k ohranjanju slabih strateških parametrov.

Pomembnost izbora je pri ES velika, saj je populacija potomcev λ praviloma precej večja kot populacija prednikov μ (običajno razmerje je 1/7). Pri danem

izboru je *čas prevzema* τ^* (angl. takeover time) definiran kot število generacij, potrebnih do zapolnitve populacije s kopijami najboljših posameznikov, ob začetni populaciji z eno samo kopijo:

$$\tau^* = \frac{\ln \lambda}{\ln(\lambda/\mu)} \,.$$

Pri tipični ES z $\mu = 15$ in $\lambda = 100$, je $\tau^* \approx 2$. V primeru proporcionalnega izbora pri genetskem algoritmu to pomeni:

$$\tau^* = \lambda \ln \lambda \,,$$

oz. pri $\lambda = 100 \rightarrow \tau^* = 460$.

4.3.6 Samo-adaptacija

Eden od glavnih prispevkov evolucijskih strategij v okviru evolucijskega računanja je ravno lastnost samo-organizacije, ki se je prva uporabila pri ES. Kasneje se je njena vrednost pokazala tudi pri binarnih in celoštevilčnih predstavitvah genotipov.

Osnovna trditev v okviru ES je, da vključitev samo-organizacije v postopek evolucije prinaša boljše rezultate. Pri tem obstajajo tako eksperimentalni kot teoretični rezultati [29], ki to tezo potrjujejo. Teoretični in praktični rezultati se dopolnjujejo, če je za neko cenilno funkcijo $f : \mathbb{R}^n \rightarrow \mathbb{R}$ mogoče teoretično izračunati optimalne velikosti mutacijskih korakov. Če se torej eksperimentalni podatki ujemajo z izračunanimi, potem lahko zaključimo, da samo-organizacija dejansko v praksi deluje.

Teoretični in eksperimentalni rezultati se ujemajo v tem, da je za uspešno rešitev problema potrebno, da se vrednosti σ s časom zmanjšujejo. Intuitivna razlaga je v tem, da je na začetku potrebno raziskovati obsežen prostor, da se najde obetavno področje za rešitev problema, temu pa mora slediti podrobna raziskava lokalnega obsega, kar zahteva manjše korake iskanja ali mutacije.

Dodatno razlago za uspešnost samo-organizacije predstavlja tudi spreminjanje cenilnih funkcij (angl. fitness landscapes). V tem primeru sprememba cenilne funkcije pomeni, da evolucijski proces sledi gibljivemu cilju. Vsaka sprememba cenilne funkcije zahteva ponovno evaluacijo populacije, kar lahko pomeni za

posameznike nižjo novo stopnjo prileganja, saj so bili prej prilagojeni stari ciljni funkciji. Zato so pogosto koraki mutacije slabo adaptirani, kar pa proces samo-organizacije izboljšuje s tem, da povečuje velikost korakov mutacije v primeru slabših vrednosti cenilnih funkcij in da zmanjšuje njihove vrednosti v bližini novega optimuma.

Izkušnje s samo-organizacijo predstavljajo novo znanje, na osnovi katerega je mogoče identificirati potrebne pogoje za njeno uporabo:

- $\mu > 1$ zagotavlja možnost različnih strategij
- $\lambda > \mu$ pomeni, da število potomcev presega število prednikov
- $\lambda/\mu \approx 7$, oz. (15, 100), kar pomeni zmerni pritisk izbora preživetja
- $(\mu\ ,\ \lambda)$ izbor, ki zagotavlja iztrebitev slabih posameznikov
- Rekombinacija (posebno vmesna) strateških parametrov

4.4 Evolucijsko programiranje

Evolucijsko programiranje (EP) je zgodovinsko gledano naslednji član družine evolucijskega računanja (ER), ki pa se od ostalih (genetski algoritmi, ES, genetsko programiranje (GP)) precej razlikuje. Če za ostale obstajajo tipični predstavniki, pa je to težko reči za EP. Vseeno poglejmo njegove reprezentativne (in ne standardne) lastnosti:

- Predstavitev: vektorji realnih vrednosti
- Izbira prednikov: deterministična (vsak prednik ustvari enega potomca z mutacijo)
- Rekombinacija: ne obstaja
- Mutacija: Gauss-ova perturbacija
- Izbira preživetja: verjetnostna $(\mu + \mu)$
- Specialnost: samo-organizacija velikosti korakov mutacije

Prvotni namen evolucijskega programiranja je bil simulirati evolucijo kot učni proces s končnim ciljem generirati umetno inteligenco [30]. V tem kontekstu je inteligenca pomenila sposobnost sistema, da adaptira svoje obnašanje z namenom doseči določene cilje v zvezi z različnimi okolji. Pri tem je adaptivno obnašanje ključni izraz v definiciji inteligence, predpogoj za adaptivnost in s tem za inteligentno obnašanje pa je v sposobnosti napovedovanja okolja.

V klasičnem primeru EP je bila napoved predmet evolucije v obliki končnega avtomata KA (angl. finite state machine, FSM), ki sprejema na vhodu končno množico vhodnih znakov in odgovarja na izhodu s končno množico izhodnih znakov, ter ima končno število notranjih stanj, med katerimi je mogoče prehajanje na osnovi ustreznih pravil. KA v odvisnosti od stanja in vhodnega znaka generira izhodni znak in spremeni stanje.

Primer napovedi, ki naj se jo KA 'nauči' (s pomočjo evolucije), je napovedati ob vsakem vhodnem znaku x_n izhodni znak y_n, ter ga primerjati z naslednjim vhodnim znakom. Obnašanje KA se torej ocenjuje z deležem vhodov, kjer je $x_{n+1} = y_n$. Ta primer seveda zahteva, da sta vhodna in izhodna abeceda enaki. Populacija prednikov (KA) je po inicializaciji izpostavljena sekvenci vhodnih simbolov (iz okolja) do trenutnega časa. Po vsakem vhodnem znaku, vsak KA napove izhodni znak in ga primerja z naslednjim vhodnim znakom. Cenilna funkcija izmeri kvalitete napovedi za vse KA. Sledi generiranje potomcev tako, da je vsak KA (prednik) podvržen naključni mutaciji. V [30] je predlaganih pet možnih načinov naključne mutacije KA:

- Sprememba izhodnega simbola
- Sprememba prehoda stanj
- Dodajanje stanja
- Odstranitev obstoječega stanja (samo, če v KA več kot eno stanje)
- Sprememba začetnega stanja (samo, če v KA več kot eno stanje)

Operatorji mutacije so izbrani glede na določeno verjetnostno porazdelitev, ki je lahko ali uniformna, ali kakšna druga. Tudi število operatorjev mutacije na prednika je ali določeno s verjetnostno porazdelitvijo (npr. Poisson-ovo) ali pa je fiksno. Vsak dobljeni potomec je ocenjen na osnovi dejanskega okolja (niza vhodno-izhodnih parov), na isti način kot predniki.

Tipično je en potomec predviden za enega prednika. Potem, ko je generiranih μ potomcev iz μ prednikov, sledi izbira μ posameznikov za naslednjo generacijo izmed 2μ kandidatov (prednikov in potomcev).

Po letu 1990 so se pojavile variante EP z realno predstavitvijo genotipov, kar pomeni, da so se ustrezno modificirali tudi operatorji mutacije.

4.4.1 Predstavitev

Evolucijsko programiranje (EP) se danes uporablja za najrazličnejše aplikacije, zato je tudi izbira predstavitve genotipov zelo svobodna. Vendar pa se najpogosteje uporablja za optimizacijo funkcij oblike: $f : \mathbb{R}^n \rightarrow \mathbb{R}$, kjer v tem primeru EP uporablja naravno predstavitev s plavajočo vejico, kjer $(x_1, ..., x_n) \in R^n$ predstavlja posameznika oz. kandidata v optimizacijskem procesu. Pogosta lastnost današnjih EP je tudi samo-organizacija, kar pomeni, da je najbolj splošna oblika posameznika določena z:

$$< x_1, ..., x_n, \sigma_1, ..., \sigma_{n_\sigma} > ,$$

kjer so σ zopet strukturni parametri.

4.4.2 Mutacija

Zaradi različnih parametrov v genotipu imamo pri EP tudi več mutacijskih operatorjev. V primeru zgornje splošne oblike posameznika (genotipa ali kromosoma), predstavlja mutacija transformacijo od $< x_1, ..., x_n, \sigma_1, ..., \sigma_{n_\sigma} >$ k $< x'_1, ..., x'_n, \sigma'_1, ..., \sigma'_{n_\sigma} >$, kjer velja:

$$\sigma'_i = \sigma_i(1 + \alpha G(0, 1))$$
$$x'_i = x_i + \sigma'_i G_i(0, 1) ,$$

kjer je $\alpha \approx 0.2$, $G(0, 1)$ pa je naključna vrednost iz Gauss-ove porazdelitve pri srednji vrednosti 0 in standardni deviaciji 1.

4.4.3 Križanje

Opis križanja oz. rekombinacije v primeru EP je trivialen, saj se ne uporablja. Na začetku je bila sicer predvidena rekombinacija končnih avtomatov in sicer na osnovi volilnega mehanizma, vendar se ni nikoli zares uporabljala v kontekstu EP. Tudi danes so EP argumenti proti rekombinaciji prej konceptualni kot tehnični.

V devetdesetih letih je bila diskusija o prednosti uporabe mutacije pred kombinacijo mutacije in rekombinacije zelo intenzivna. Primerjani so bili eksperimentalni rezultati obeh variant na nizu linearnih funkcij in parametrskimi interakcijami med geni. Rezultati so govorili v prid sami mutaciji.

Danes splošno sprejeto mnenje zagovarja srednjo pot. Zadnji rezultati potrjujejo, da sta sposobnost križanja ali Gauss-ove mutacije, da generirata izboljšanega potomca, odvisna predvsem od stanja iskalnega procesa, pri čemer daje samo mutacija boljše začetne rezultate, križanje pa pridobiva v nadaljevanju evolucije.

Ob koncu poglavja o evolucijskem programiranju (EP) v okviru evolucijskega računanja (ER) velja omeniti, da obstajajo tudi druge adaptivne metode na osnovi končnih avtomatov. Eno od njih smo že spoznali v poglavju o učečih avtomatih (UA), kjer sta se v postopku učenja iterativno spreminjala dva strukturna parametra UA, verjetnost izbire izhodne akcije in verjetnost prehajanja stanj. Ustrezne korekcijske enačbe so spreminjale omenjene verjetnosti s ciljem izboljšati oceno iz okolja. Drugo metodo pa se pogosto uporablja v kontekstu sinteze celičnih avtomatov (CA) [31], kjer poteka razvoj oz. evolucija CA na osnovi algoritma celičnega programiranja (CPA), ki je predstavnik lokalnega evolucijskega algoritma. Pri njem gre za razvoj oz. evolucijo enega samega CA (in ne populacije) tako, da se uporabljajo lokalni operatorji mutacije in križanja nad strukturnimi parametri celic s ciljem izboljšati povprečno delovanje CA glede na cenilno funkcijo, ki jo določa okolje oz. aplikacija.

4.5 Genetsko programiranje

Genetsko programiranje (GP) je najmlajši član iz družine evolucijskega računanja (ER). Poleg razlike v predstavitvi genotipa, ki je v primeru GP drevesna

struktura, se bistveno razlikuje od ostalih evolucijskih algoritmih po področju aplikacij. Če je tipično področje uporabe ostalih evolucijskih algoritmov optimizacija, pa GP sodi bolj na področje strojnega učenja (modeliranja), ki ga je zato mogoče obravnavati kot poseben primer optimizacije. Tedaj lahko modele obravnavamo kot posameznike (genotipe), njihova ocena pa je kvaliteta modela, katere maksimum je cilj evolucije. Bistvene značilnosti GP so naslednje:

- Predstavitev: drevesna struktura
- Rekombinacija: zamenjava pod-dreves
- Mutacija: naključna sprememba v drevesu
- Izbira prednikov: proporcionalna cenilni funkciji
- Izbira preživetja: generacijska zamenjava (angl. generational replacement)

4.5.1 Teoretične osnove

Genetsko programiranje je oblika programske indukcije, ki se lahko uporablja za avtomatizirano iskanje programske rešitve za določen problem ali nalogo. Takšno vlogo je predvidel njen avtor J. Koza [32]. V principu so lahko posamezni programi izraženi v kateremkoli programskem jeziku. Vendar je sintaksa večine jezikov takšna, da bi GP operatorji kreirali ogromen procent sintaktično nepravilnih programov. Zaradi tega je Koza izbral sintakso v prefiksni obliki, analogno tisti pri programskem jeziku LISP in omejil jezik s primernim številom spremenljivk, konstant in operatorjev. Na ta način so lahko omejitve sintakse upoštevane, programski iskalni prostor pa je omejen. Omejen programski jezik je formiran z funkcijskim nizom oz. naborom operatorjev F in terminalnim nizom T, ki ju definira uporabnik. Izbrane funkcije so praviloma izbrane tako, da so z vidika aplikacije oz. naloge uporabne, terminali pa so spremenljivke ali konstante. Vsaka funkcija iz niza F mora biti sposobna sprejeti kot argumente možnih programskih rešitev je konstituiran z nizom vseh možnih kompozicij funkcij, ki jih je mogoče rekurzivno formirati iz elementov nizov F in T.

Kot preprost primer si oglejmo slučaj, ko tvorijo funkcijski niz osnovne aritmetične operacije, terminalni niz pa sestavljajo štiri spremenljivke:

$$\text{F} = \{+, -, *, /\}$$
$$\text{T} = \{\text{A, B, C, D}\}$$

Tedaj so naslednji primeri legalni programski moduli:

(+ (* A B) (/ C D))
(* (− (+ A C) B) A)

Potrebno je poudariti, da GP ne potrebuje nujno programov v LISP-u. Katerikoli drug programski jezik, ki lahko predstavi programe interno kot razčlenjeno drevo (angl. parse tree), je tudi primeren. Zato je danes večina GP aplikacij napisanih v jezikih C, C++ ali Java, namesto v LISP-u.

4.5.2 Predstavitev

Kot je bilo omenjeno na začetku, je osnovna ideja pri GP, da se za genotipe vzame razčlenjena drevesa. Takšnim drevesom ustreza kompaktna formalna sintaksa, ki je lahko aritmetični izraz, predikatna logika ali koda programskega jezika. Naslednji trije primeri sintaks so osnova za razčlenjena drevesa, ki jih prikazuje Slika 4.2.

A. Aritmetična formula:

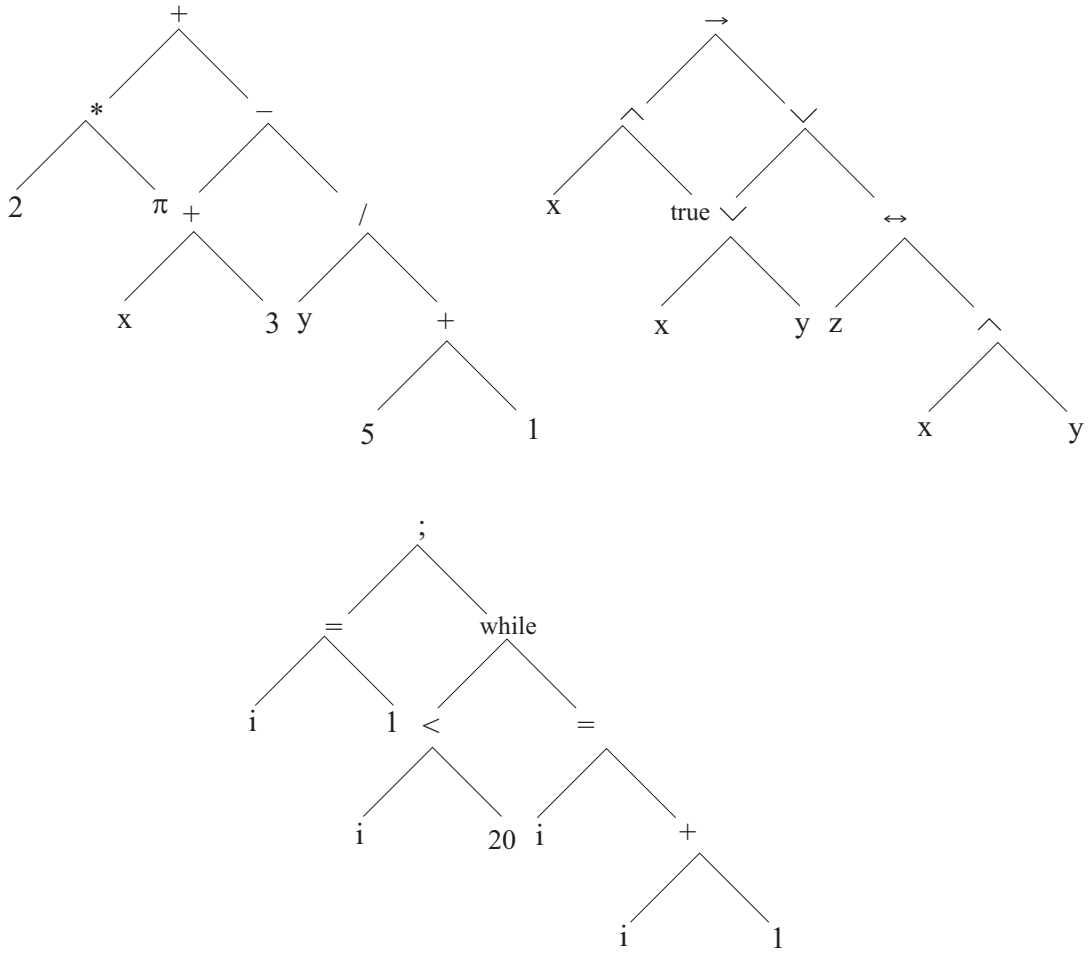
$$2\pi + ((x + 3) - \frac{y}{5 + 1})$$

B. Logična formula:

$$(x \wedge true) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$

C. Programska koda:

```
i = 1;
while (i < 20)
{
    i = i + 1;
}
```



Slika 4.2: Razčlenjena drevesa za primere izrazov A, B, C.

Zgornji primeri ilustrirajo, kako so lahko razčlenjena drevesa uporabljena in interpretirana. V primeru, da GP obravnavamo kot varianto genetskega algoritma, ki deluje z drugačno podatkovno strukturo (drevesa namesto bitov), se interpretacija dreves, ki je odvisna od aplikacije, izgubi. Zato je pomembno, da ima vsako drevo ustrezno sintakso, kar omogoča prefiksna ali poljska sintaksa, ki je tudi sintaksa funkcijskega programiranja (torej LISP-a). Formula A zgoraj ima npr. naslednjo prefiksno sintakso:

$$+(.(2,\pi),-(+(x,3),/(y,+(5,1)))) \ ,$$

medtem ko je izvršljiva LISP koda praktično enaka:

$$(+(* \ 2 \ \pi) \ (-(+ \ x \ 3) \ (/ \ y(+ \ 5 \ 1)))) \ .$$

Če privzamemo takšno interpretacijo za drevesa, potem je GP mogoče obravnavati kot programiranje računalnikov na osnovi naravne selekcije [32] ali pa kot avtomatično evolucijo računalniških programov [33].

Pred opisom operatorjev mutacije in rekombinacije povejmo, da se pri GP pogosto obe operaciji izvajata v enem koraku, za razliko od genetskih algoritmov in ES, kjer si obe operaciji sledita ena za drugo.

4.5.3 Mutacija

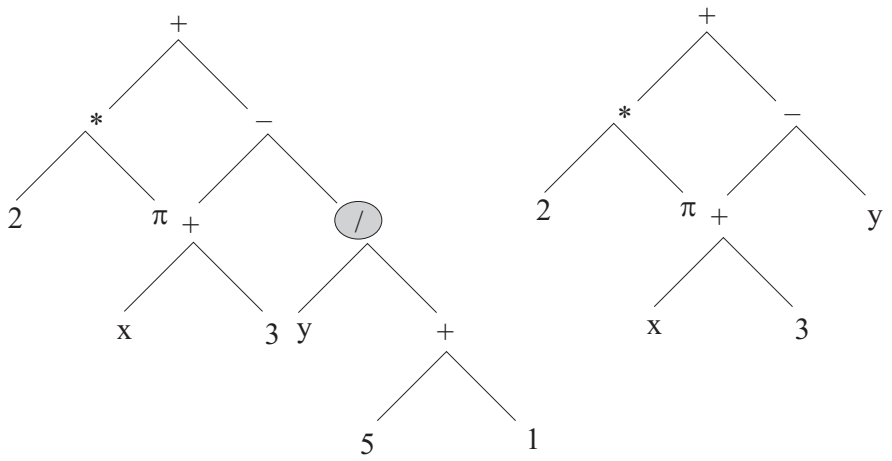
Operacija mutacije je pri GP podobna kot pri ostalih algoritmih ER, saj izvede majhno naključno spremembo nad posameznikom (genotipom), ki je v primeru GP drevo. Običajno to pomeni, da se naključno izbere točka v drevesu, v kateri se poddrevo zamenja z novim poddrevesom, ki je generirano na enak način, kot drevesa pri inicializaciji začetne populacije. Slika 4.3 ilustrira mutacijo drevesa, ki ustreza aritmetičnemu izrazu A zgoraj in ki se spremeni v drevo, ki pripada izrazu: $2 \cdot \pi + ((x + 3) - y)$. Na sliki je obkroženo vozlišče na levem drevesu izbrana točka, v kateri se poddrevo zamenja z novim, v tem primeru z listom.

Mutacija pri GP ima torej dva parametra:

- Verjetnost izbire mutacije p_m
- Verjetnost izbire notranje točke v drevesu kot korena za poddrevo, ki bo zamenjano

4.5.4 Križanje

Križanje ali rekombinacija pri GP ustvarja potomce tako, da zamenja poddrevesa obeh prednikov, ki ju določata dve naključno izbrani točki. Slika 4.4 ilustrira takšno križanje dveh dreves.



Slika 4.3: Ilustracija mutacije pri GP.

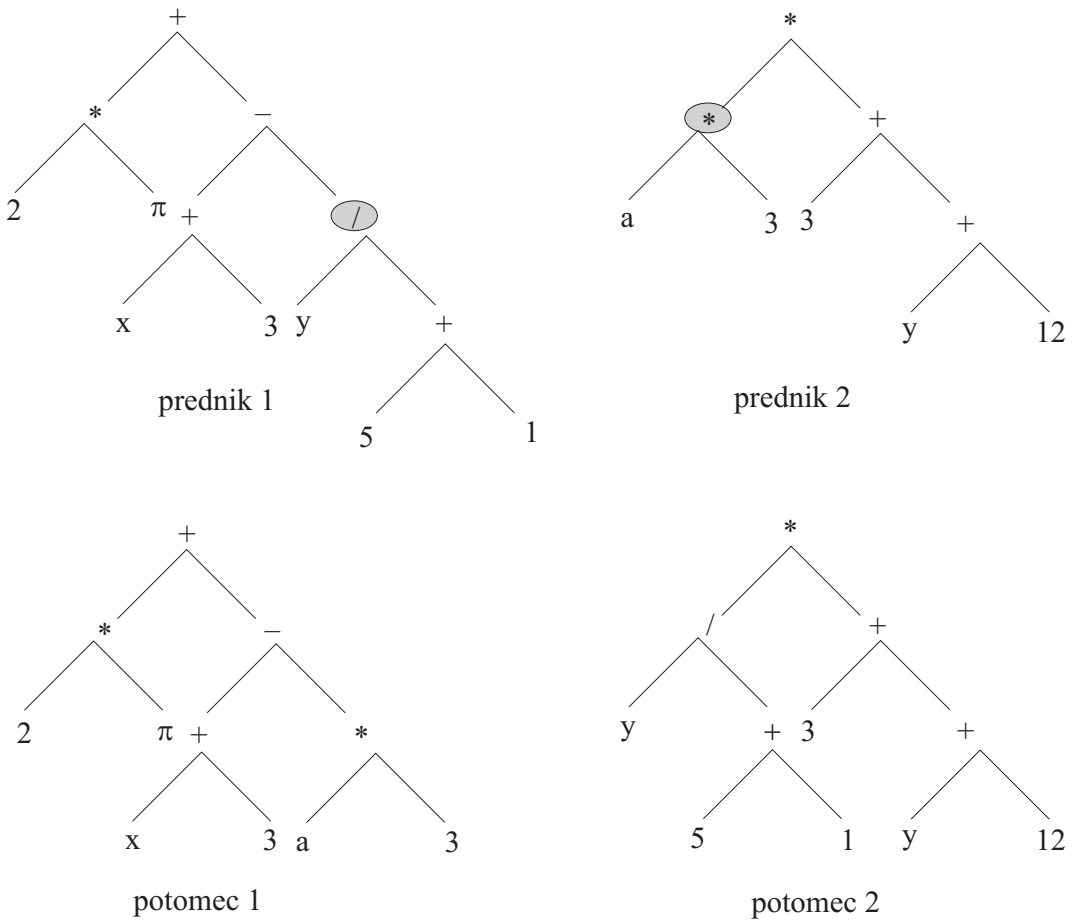
Rekombinacija pri GP ima dva parametra:

- Verjetnost izbire križanja p_c
- Verjetnost izbire notranjih točk obeh prednikov

4.5.5 Izbira staršev

Izbira staršev se pri GP izvaja na osnovi proporcionalnih vrednosti cenilnih funkcij. Kadar pa je populacija zelo velika (več kot tisoč posameznikov), pa se uporablja metoda 'nad-izbire' (angl. over-selection). Pri tej metodi se populacija po rangiranju razdeli v dve skupini; ena vsebuje zgornjih $x\%$ in druga ostalih $(100 - x)\%$. Ko so predniki tako izbrani, jih iz prve skupine pride 80%, iz druge pa 20%. Vrednost x je izbrana empirično (angl. rule of thumb) in zavisi od velikosti populacije. Spodnja tabela kaže tipične vrednosti za x v odvisnosti od velikosti populacije:

Velikost populacije	Delež v skupini x
1000	32%
2000	16%
4000	8%
8000	4%



Slika 4.4: Križanje dveh prednikov (dreves) pri GP v naključnih (obkroženih) točkah.

Kot je iz tabele razvidno, se število posameznikov v skupini x, iz katere se izbira večina prednikov za rekombinacijo, ohranja oz. je konstantno, kar pomeni, da se pomen izbire dramatično povečuje z velikostjo populacije.

4.5.6 Izbira preživetja

GP tipično uporablja generacijsko strategijo brez t.i. elitizma, kar pomeni, da je število izbranih potomcev enako velikosti populacije in da je življenjska doba posameznika ena generacija. To seveda tehnično ni potrebno, je le dogovor.

V zadnjem času se kaže trend po enakovredni uporabi stabilnega oz. stacionarnega (angl. steady-state) modela populacije, predvsem zaradi potrebe po elitizmu, ki se zahteva zaradi destruktivne vloge križanja.

4.5.7 Inicializacija

Inicializacija populacije pri GP poteka tako, da se najprej določi maksimalno začetno globino dreves d_{max}, nato pa se zgradi vsak posameznik začetne populacije iz množic F in T na osnovi dveh postopkov z enako verjetnostjo:

- Polna metoda (angl. full method): tukaj ima vsaka veja drevesa globino d_{max}. Vsebina vozlišča na globini d je izbrana iz F, če je $d < d_{max}$, in iz T, če je $d = d_{max}$
- Rastoča metoda (angl. grow method): veje drevesa imajo lahko različne globine, do d_{max}. Drevo se prične graditi pri korenu, vsebina vozlišča pa se izbere stohastično iz unije obeh množic, $F \cup T$, če je $d < d_{max}$.

Med procesom izbire posameznikov (dreves) v okviru GP, se kaže poseben trend rasti dreves. Ta efekt je znan kot 'preživetje najdebelejšega' (angl. survival of the fattest), katerega vzrok se zdi v neenakomernih drevesih. Postopki za preprečevanje tega efekta so precej zapleteni. Najenostavnejši predstavlja omejevanje velikosti drevesa z neko maksimalno globino, ki preprečuje povečevanje dreves preko te vrednosti, drugačen postopek pa zahteva izraz za kaznovanje velikih dreves, ki vpliva na cenilno funkcijo. V obeh primerih to zahteva nov strukturni parameter v genotipu drevesa.

Računanje prileganja dreves oz. programov se nekoliko razlikuje od numeričnih genotipov, ker v primeru GP iščemo program, ki zadošča danemu številu N vhodno/izhodnih relacij, oz. primerom prileganja. Za nek program p_i je prileganje j-temu primeru določeno z $f_j(p_i)$, ki predstavlja razliko med izhodom programa g_j in med pravilnim odgovorom G_j za isti j-ti primer. Celotno prileganje $f(p_i)$ je vsota preko vseh N primerov:

$$f(p_i) = \sum_{k=1}^{N} (g_k - G_k)^2 \,.$$

Seveda bo imel boljši program nižjo vrednost prileganja, saj to pomeni manjše odstopanje od želenih vrednosti.

Poglavje 5

Mehka logika

Osnovo mehke logike predstavlja teorija mehkih množic, ki jo je v šestdesetih letih zasnoval L. Zadeh [34]. V začetku je bila deležna obsežnih kritik, v katerih je prednjačil tedaj že uveljavljeni Kalman, ki enostavno ni videl smisla v 'poenostavljenem' obravnavanju problemov, saj natančnost res ne more in ne sme 'škodovati' pri njihovem reševanju. V sedemdesetih pa se je najbolj po zaslugi japonskih raziskovalcev teorija uveljavila, predvsem na področjih procesne kontrole, procesiranja signalov, telekomunikacij, medicine, pa tudi financ, trgovine in servisiranja. Danes so njeni rezultati prisotni v večini sodobnih produktov bele tehnike, audio-video naprav, optičnih in telekomunikacijskih izdelkov, itd. Mehki sistemi omogočajo opisovanje zapletenih problemov z simboličnimi (lingvističnimi) izrazi, podobno, kot jih v naravnem jeziku opisuje človek. Skupaj s postopki mehkih pravil, mehkega sklepanja, odločitev in krmiljenja, predstavljajo nov način reševanja problemov, ki sledi človeški obravnavi, sklepanju in odločanju.

5.1 Mehke množice

Tradicionalen način predstavitve elementov u v množici $A \subset U$, kjer je A podmnožica univerzalne množice (glede na problemsko domeno) U, uporablja

t.i. karakteristično funkcijo $x_A(u)$, ki je definirana z zapisom:

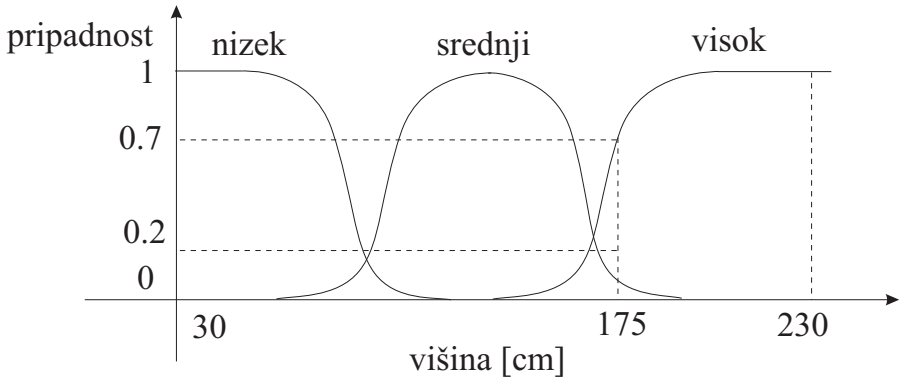
$$x_A(u) = \begin{cases} 1\,, & \text{če } u \in A \\ 0\,, & \text{sicer}\,. \end{cases} \tag{5.1}$$

Namesto takšne diskretne karakteristične funkcije, je Zadeh definiral zvezno pripadnostno funkcijo (angl. membership function) $\mu_A(u)$:

$$\mu_A(u) : U \rightarrow [0, 1]\,,$$

ki pove, kolikšna je stopnja pripadnosti spremenljivke u mehki množici A. Vrednosti pripadnostne funkcije so realna števila, običajno v intervalu [0,1], kjer 0 pomeni, da objekt u ni član množice A, 1 pa, da ji v celoti pripada. Vsaka njena vrednost se imenuje stopnja pripadnosti (angl. membership degree).

Slika 5.1 prikazuje tri pripadnostne funkcije, ki predstavljajo tri mehke množice, označene s pridevniki 'nizek', 'srednji' in 'visok', kar se lahko nanaša na velikost kakšne osebe, ali na vrednost določene merjene veličine. Če abscisna os pred-



Slika 5.1: Prikaz treh mehkih množic z njihovimi funkcijami pripadnosti.

stavlja veličino, npr. telesno višino, potem mehke množice s svojimi pripadnostnimi funkcijami pokrivajo določene predele vrednosti spremenljivke tako, da je mogoč njihov enostaven 'lingvističen' opis. Majhni telesni višini dobro ustreza pridevnik 'nizek', ki tem bolj drži, čim manjša je višina. Zato je tudi stopnja pripadnosti oz. vrednost pripadnostne funkcije večja pri majhni višini in se zmanjšuje proti 0, ko višina narašča. Mehki množici 'nizek' pripadajo vse vrednosti pripadnostne funkcije f_{nizek}. V splošnem opisujemo mehko množico s pari (vrednost pripadnostne funkcije/vrednost spremenljivke), preko vseh vrednosti spremenljivke:

$$A = \{(\mu_A(u)/u) \mid u \in U\}\,. \tag{5.2}$$

Običajno obstaja presek med mehkimi množicami, kar se ujema z izražanjem, s katerim človek opisuje numerične vrednosti. Zgornji zapis mehke množice 'visok' s Slike 5.1, je mogoče podati ali kot množico ali pa kot unijo parov:

$$\begin{aligned} \text{visok} &= \{0/50,\ 0.3/160,\ 0.68/170,\ 0.9/180,\ 1/250\} \\ &= 0/150 \cup\ 0.3/160 \cup\ 0.68/170 \cup\ 0.9/180 \cup\ 1/250\,, \end{aligned}$$

kjer $\cup$ pomeni operacijo unije. Presek med mehkima množicama 'nizek' in 'srednji' je posledica subjektivne ocene telesne višine in zato neizrazite meje med njima.

5.1.1 Lastnosti mehkih množic

1. *Podpora* (angl. support) $S(A)$ mehke množice A je podmnožica univerzalne množice U, katere vsak element ima stopnjo pripadnosti različno od 0. Npr. podpora množici 'srednja temperatura' je interval (10, 30) v stopinjah Celzija.

2. *Kardinalnost* (angl. cardinality) klasične množice je število elementov v množici, pri mehki množici pa je definirana kot:

$$M(A) = \sum_{u \in U} \mu_A(u)\,.$$

3. *Potenčna množica* (angl. power set) mehke množice A je množica vseh mehkih podmnožic množice A.

4. *Normalna* mehka množica A ima vsaj eno stopnjo pripadnosti z vrednostjo 1.

5. *Mehki posameznik* (angl. fuzzy singleton) je mehka množica, katere podpora vsebuje eno samo točko $u \in U$, pri kateri je $\mu_A(u) = 1$.

Teorijo mehkih množic lahko obravnavamo kot posplošitev klasične teorije množic. Zato veljajo ustrezne posplošitve tudi za osnovne operacije nad množicami, t.j. za unijo, presek in komplement.

5.1.2 Osnovne operacije nad mehkimi množicami

Klasične množice so poseben primer mehkih množic, pri katerih sta le dve stopnji pripadnosti ali vrednosti pripadnostne funkcije, 0 in 1. Zato morajo vse definicije, ki glasijo na mehke množice, veljati tudi za klasične ('trde', angl. crisp) množice. V nadaljevanju bodo podane osnovne mehke operacije nad dvema mehkima množicama A in B, definirane nad univerzalno množico U:

1. *Unija* $A \cup B$:

$$\mu_{A \cup B}(u) = \mu_A(u) \vee \mu_B(u)\,, \quad \text{za vse } u \in U$$

   $\vee$ pomeni operacijo MAX

2. *Presek* $A \cap B$:

$$\mu_{A \cap B}(u) = \mu_A(u) \wedge \mu_B(u)\,, \quad \text{za vse } u \in U$$

   $\wedge$ pomeni operacijo MIN

3. *Komplement* $\neg A$:

$$\mu_{\neg A}(u) = 1 - \mu_A(u)\,, \quad \text{za vse } u \in U$$

4. *Algebraični produkt* $A \cdot B$:

$$\mu_{AB}(u) = \mu_A(u) \cdot \mu_B(u)\,, \quad \text{za vse } u \in U$$

5. *Algebraična vsota* $A + B$:

$$\mu_{A+B}(u) = \mu_A(u) + \mu_B(u)\,, \quad \text{za vse } u \in U$$

6. *Enakost* $A = B$:

$$\mu_A(u) = \mu_B(u)\,, \quad \text{za vse } u \in U$$

Za operacije nad mehkimi množicami veljajo naslednje lastnosti: asociativnost, komutativnost in distributivnost:

- Asociativnost: $(a \circ b) \circ c = a \circ (b \circ c) = a \circ b \circ c$

- Komutativnost: $a \circ b = b \circ a$
- Distributivnost: $a \circ (b \bullet c) = (a \circ b) \bullet (a \circ c)$

kjer sta $\circ$ in $\bullet$ poljubni mehki operaciji.

Pomembna lastnost mehkih množic (napram klasičnim množicam) je v neveljavnosti pravila *izločene sredine* (angl. excluded middle) in pravila protislovja (angl. contradiction):

$$\begin{aligned} A \cup \neg A &\neq U\,, \\ A \cap \neg A &\neq \{\}\,. \end{aligned}$$

Unija mehke množice in njenega komplementa torej ne dajeta nujno celotne univerzalne množice U in njun presek ni nujno prazna množica.

Zanimiva je tudi mera za *nejasnost* (angl. ambiguity) mehke množice, ki jo podaja *entropija* [35]:

$$H(A) = \frac{M(A \cap \neg A)}{M(A \cup \neg A)}\,,$$

kjer je M oznaka za moč množice. Čim večja je entropija, tem večja je nejasnost.

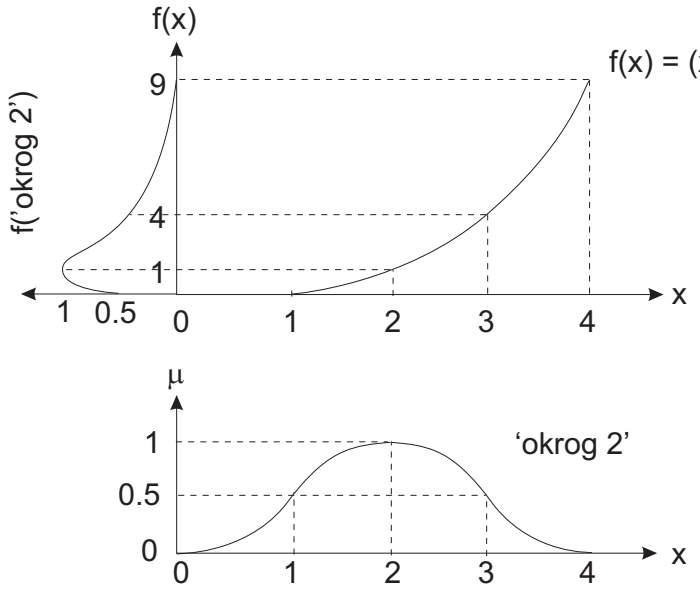
Zelo uporaben je *princip razširjave* (angl. extension principle), ki omogoča transformirati osnovno mehko množico v mehko množico funkcije nad osnovno mehko množico. Če je dana funkcija $f : X \to Y$, ki povezuje dve navadni množici X in Y, in če poznamo funkcijo pripadnosti μ_A podmnožice $A \subseteq X$, lahko dobimo mehko predstavitev $f(A)$ v Y s pomočjo izraza:

$$\mu_{f(A)}(f(x)) = \mu_A(x)\,.$$

Slika 5.2 ilustrira princip razširjave v primeru, ko je $f(x) = (x-1)^2$ in je A = 'okrog 2' = (0.5/1, 1/2, 0.5/3, 0/4). Iz slike je razvidno, kako je mogoče določiti vrednosti pripadnostne funkcije f('okrog 2').

5.2 Mehka logika

Mehka logika je odlično orodje za implementacijo znanja 'zdravega razuma' (angl. common-sense knowledge), nejasnega znanja v računalniških programih



Slika 5.2: Ilustracija principa razširjave.

in za posledično izvajanje določenih zaključkov. Mehka logika izhaja iz mehkih relacij in mehkih trditev (angl. fuzzy propositions), ki so definirane na osnovi mehkih množic [36].

5.2.1 Mehke relacije

Mehke relacije omogočajo opisovanje nejasnih razmerij s tem, da povezujejo mehke množice na nek dogovorjen način. Če je mehka množica A definirana v okviru univerzalne množice U in mehka množica B v okviru univerzalne množice V, potem je *mehka relacija* (angl. fuzzy relation) $R(A, B)$ katerakoli mehka množica, definirana nad kartezijskim produktom $U \times V = \{(u, v) \mid u \in U, v \in V\}$. Mehka relacija je določena s svojo pripadnostno funkcijo:

$$\mu_{R(u,v)} : \quad U \times V \rightarrow [0, 1] \, .$$

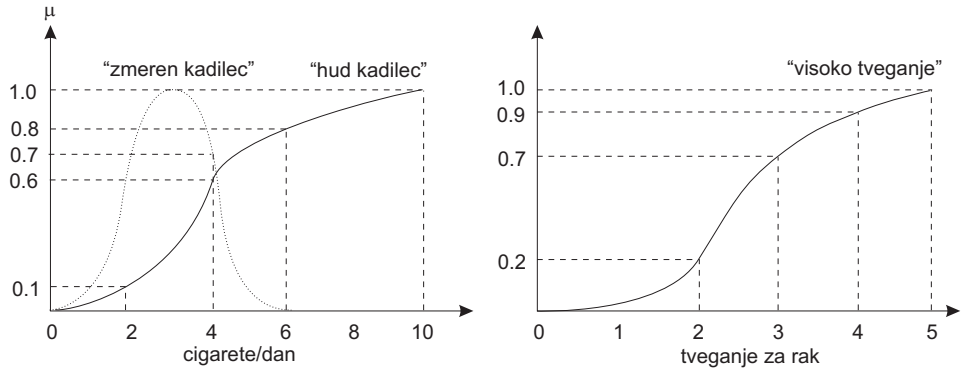
Zelo pomembna mehka relacija je *mehka implikacija* (angl. fuzzy implication), ki jo označujemo z $A \rightarrow B$. V mehki logiki obstaja veliko načinov za definicijo implikacije, kar je v nasprotju z običajno preklopno logiko, kjer velja ena sama definicija oz. pravilnostna tabela ($x \rightarrow y = \overline{x} \vee y$ ali disjunkcija negacije x in

y). Spodnja tabela podaja nekaj najzanimivejših mehkih implikacij. V tabeli je zaradi enostavnosti uporabljena substitucija: u namesto $\mu_A(u)$, v namesto $\mu_A(v)$, $\wedge$ pomeni operacijo 'minimum', $\vee$ pa 'maksimum', $+$ je algebraična vsota, $-$ pa algebraična razlika.

Tabela mehkih implikacijskih relacij (R):

R_a:	$1 \wedge (1 - u + v)$	definiral Zadeh
R_m:	$(u \wedge v) \vee (1 - u)$	Zadeh
R_c:	$u \wedge v$	Mamdani
R_b:	$(1 - u) \vee v$	Mizumoto & Zimmermann
R_s:	$(u \rightarrow v)_s = \begin{cases} 1\,, & \text{če } u \leq v \\ 0\,, & \text{če } u > v \end{cases}$	Mizumoto & Zimmermann
R_g:	$(u \rightarrow v)_g = \begin{cases} 1\,, & \text{če } u \leq v \\ v\,, & \text{če } u > v \end{cases}$	Mizumoto & Zimmermann

Mehke relacije lahko podajamo v matrični obliki ali z mehkim grafom. Slika 5.3 podaja primer R_c implikacije 'hud kadilec' $\rightarrow$ 'visoko tveganje' za rak. Pri-



Slika 5.3: Ilustracija mehke relacije v grafični in matrični obliki.

padnosti mehkima množicama 'hud kadilec' in 'visoko tveganje' označujeta vektorja u in v, ki ju lahko razberemo s slike. Mamdani-jeva relacija v matrični obliki se glasi:

$$u \wedge v^T = \min\left(\begin{bmatrix} 0.0 \\ 0.1 \\ 0.6 \\ 0.8 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.0 & 0.2 & 0.7 & 0.9 & 1 \end{bmatrix} \right) = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.0 & 0.2 & 0.6 & 0.6 & 0.6 \\ 0.0 & 0.2 & 0.7 & 0.8 & 0.8 \\ 0.0 & 0.2 & 0.7 & 0.9 & 1.0 \end{bmatrix}$$

Tolmačimo jo takole:

		tveganje				
		1	2	3	4	5
	0	0.0	0.0	0.0	0.0	0.0
	2	0.0	0.1	0.1	0.1	0.1
cigarete	4	0.0	0.2	0.6	0.6	0.6
	6	0.0	0.2	0.7	0.8	0.8
	10	0.0	0.2	0.7	0.9	1.0

Kompozicija

Relacija kompozicije ali kratko *kompozicija* dveh mehkih relacij $R_1(A, B)$ in $R_2(B, C)$ je relacija $R(A, C)$, ki jo dobimo po zaporedni izvedbi relacij R_1 in R_2. Tipična kompozicija je MAX-MIN, ki jo je predlagal Zadeh:

$$R(A, C) : \mu_{R(a,c)} = \vee\{\mu_{R_1}(a, b) \wedge \mu_{R_2}(b, c)\},$$

kjer $\vee$ pomeni MAX in $\wedge$ MIN operacijo, $a \in A$, $b \in B$, $c \in C$.

Kot primer kompozicije lahko opazujemo kompozicijo, ki je izvedena nad pripadnostno funkcijo 'zmeren kadilec' in implikacijo 'hud kadilec' $\rightarrow$ 'visoko tveganje'. Rezultat kompozicije je pripadnostna funkcija, ki opisuje tveganje za rak pri zmernem kadilcu:

tveganje = 'zmeren kadilec' $\circ$ ('hud kadilec' $\rightarrow$ 'visoko tveganje') .

V tem primeru gre za mehko sklepanje, ki je rezultat uporabe implikacije in kompozicije na naslednji način: pri znani relaciji $R : A \rightarrow B$ in kompoziciji $\circ$, sledi v primeru mehke množice A' logična posledica $B' = A' \circ R = A' \circ (A \rightarrow B)$.

Iz Slike 5.3 vidimo, da ima vektor 'zmeren kadilec' sledeče pripadnosti:

	0	0
	2	0.7
cigaret	4	0.7
	6	0
	10	0

Izračunajmo najprej min('zmeren kadilec', ('hud kadilec' → 'visoko tveganje')):

$$\min\left(\begin{bmatrix}0\\0.7\\0.7\\0\\0\end{bmatrix},\begin{bmatrix}0.0&0.0&0.0&0.0&0.0\\0.0&0.1&0.1&0.1&0.1\\0.0&0.2&0.6&0.6&0.6\\0.0&0.2&0.7&0.8&0.8\\0.0&0.2&0.7&0.9&1.0\end{bmatrix}\right)=\begin{bmatrix}0.0&0.0&0.0&0.0&0.0\\0.0&0.1&0.1&0.1&0.1\\0.0&0.2&0.6&0.6&0.6\\0.0&0.0&0.0&0.0&0.0\\0.0&0.0&0.0&0.0&0.0\end{bmatrix}.$$

Ko izvedemo še maksimum po stolpcih, dobimo tveganje:

$$[0.0, 0.2, 0.6, 0.6, 0.6]\,.$$

Lahko narišemo tudi takole:

		tveganje				
		1	2	3	4	5
cigarete	0	0.0	0.0	0.0	0.0	0.0
	2	0.0	0.1	0.1	0.1	0.1
	4	0.0	0.2	0.6	0.6	0.6
	6	0.0	0.0	0.0	0.0	0.0
	10	0.0	0.0	0.0	0.0	0.0
	max	0.0	0.2	0.6	0.6	0.6

Omenjeno sklepanje se imenuje tudi mehko sklepanje naprej, saj iščemo posledično mehko množico B', ali GMP (generalized modus ponens, operator $\therefore$). Možno je tudi mehko sklepanje nazaj, kjer iščemo vzročno mehko množico A', ali GMT (generalized modus tolens). V tem primeru poznamo $A \to B$ in B', iščemo pa $A' = (A \to B) \circ B'$.

Lastnosti mehkih relacij

Vzemimo, da je R mehka relacija nad univerzalnim prostorom $R \subseteq U \times U$. Tedaj veljajo za R naslednje lastnosti:

1. *Refleksivnost*: $\mu_R(x, x) = 1$
2. *Simetrija*: $\mu_R(x, y) = \mu_R(y, x)$
3. *Tranzitivnost*: $\mu_R(x, z) = \min(\mu_R(x, y), \mu_R(y, z))\,,$ za vse $x, y, z \in U$

4. *DeMorgan-ov teorem*:

$$\begin{aligned} \neg(A \wedge B) &= \neg A \vee \neg B \\ \neg(A \vee B) &= \neg A \wedge \neg B \end{aligned}$$

5.2.2 Mehka pravila in mehko sklepanje

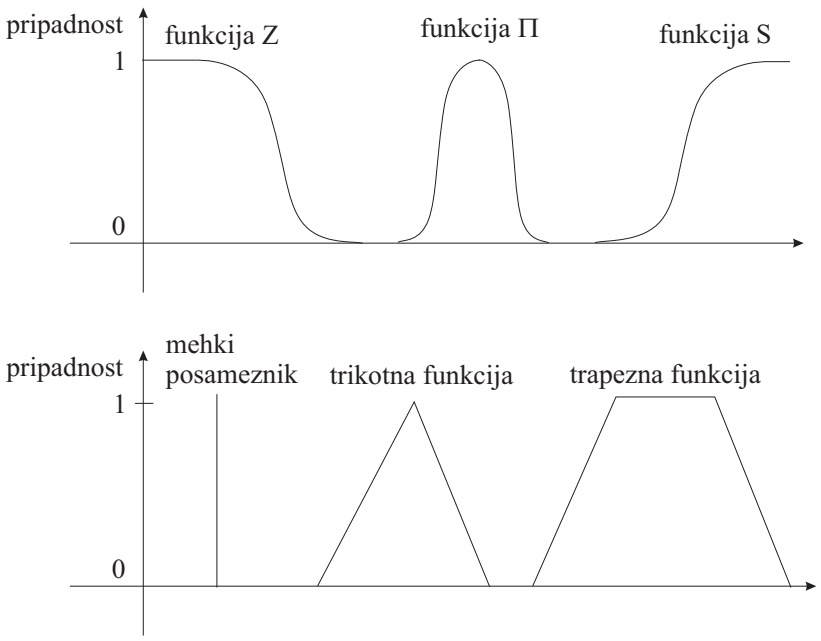
Mehki sistem sestoji iz treh delov:

- Iz mehkih vhodnih in izhodnih spremenljivk ter njihovih mehkih vrednosti
- Mehkih pravil
- Mehkih metod sklepanja ali inferenčnih metod, ki vsebujejo še postopke mehčanja in ostrenja (angl. fuzzification, defuzzification)

Mehke spremenljivke

Mehke spremenljivke so v bistvu mehke množice oz. množice vrednosti pripadnostnih funkcij posameznim 'lingvističnim' ali opisnim množicam. Postopek kreiranja mehkih vhodnih in izhodnih spremenljivk imenujemo mehčanje (angl. fuzzification). Primer mehke spremenljivke podaja lingvističen opis 'nizek' v kontekstu opisovanja telesne višine, ki smo ga spoznali pri uvodni predstavitvi mehkih množic. V sklop mehčanja sodi tudi določitev števila mehkih množic, ki pokrivajo vhodne in izhodne spremenljivke, ter izbira oblike ustreznih funkcij pripadnosti. Čeprav oblika teh funkcij ni bistvena in ne vpliva mnogo na uspešnost mehkega sistema, se v praksi uporabljajo najbolj trikotna, trapezna in normalna (Gauss-ova) oblika za funkcije pripadnosti.

Slika 5.4 prikazuje značilne možne oblike funkcij pripadnosti. Na sliki so tri funkcije, ki pokrivajo notranje področje vhodnih/izhodnih spremenljivk (trapezna, trikotna, zvončasta) in dve funkcije, ki pokrivata robna področja. Funkcija $S(u)$ je sigmoidna, ki smo jo spoznali v poglavju o nevronskih mrežah, $Z = 1 - S(u)$, zvončasta funkcija Π je enaka $S(u)$, če je $u \leq b$ in $Z(u)$, če je $u > b$.



Slika 5.4: Najpogostejše oblike funkcij pripadnosti.

Mehka pravila

Mehka pravila opisujejo vzroke in posledice v sistemu, ki ga modeliramo z mehko logiko. Pri tem so vzroki trenutne vhodne mehke spremenljivke oz. njihove funkcijske pripadnosti, posledice pa trenutne izhodne mehke spremenljivke or. ustrezne funkcijske pripadnosti. Obstaja več vrst mehkih pravil, vendar se največ uporabljata dva tipa: Zadeh-Mamdani-jevo pravilo in Takagi-Sugeno-vo pravilo. Oba sta oblike IF-THEN-(ELSE), kar pomeni, da v obravnavanem sistemu povezujejo vzroke s posledicami s pomočjo mehkih množic. Pogoj ELSE je zajet implicitno skozi druga pravila.

A. *Zadeh-Mamdani-jevo pravilo* (Z-M)

Mehko pravilo Z-M ima obliko:

IF x is A THEN y is B ,

kjer sta (x is A) in (y is B) dve mehki trditvi oz. propoziciji, ki pomenita,

da je vhodna spremenljivka x v območju mehke množice A z vrednostjo pripadnostne funkcije $\mu_A(x)$, izhodna spremenljivka y pa v območju mehke množice B z vrednostjo pripadnostne funkcije $\mu_B(y)$. Splošna oblika tega pravila, ki upošteva poljubno število vhodnih spremenljivk, ima obliko:

$$\text{IF } x_1 \text{ is } A_1 \text{ AND } x_2 \text{ is } A_2 \text{ AND ... AND } x_k \text{ is } A_k \text{ THEN } y \text{ is } B \ .$$

Takšnih pravil je v splošnem več, razlikujejo se po številu in vrstah uporabljenih vhodnih oz. izhodnih mehkih množicah. V primeru, da je pravil več, jih označujemo z R_i, $i = 1, 2, ...$

B. *Takagi-Sugeno-vo pravilo*

To pravilo se razlikuje v posledičnem delu, ki je tukaj podan s funkcijo vhodnih spremenljivk in ne z izhodno mehko množico. Pravilo R_i ima naslednjo obliko:

$$R_i\text{: IF } x \text{ is } A_i \text{ AND } y \text{ is } B_i \text{ THEN } z = f(x, y) \ .$$

V splošnem je vhodnih pogojev več, funkcija pa je pogosto linearna:

$$R_i\text{: IF } x_1 \text{ is } A_{1i} \text{ AND } x_2 \text{ is } A_{2i} \text{ AND ... AND } x_m \text{ is } A_{mi} \text{ THEN } z = C_{0i} + C_{1i}x_1 + ... + C_{mi}x_m \ .$$

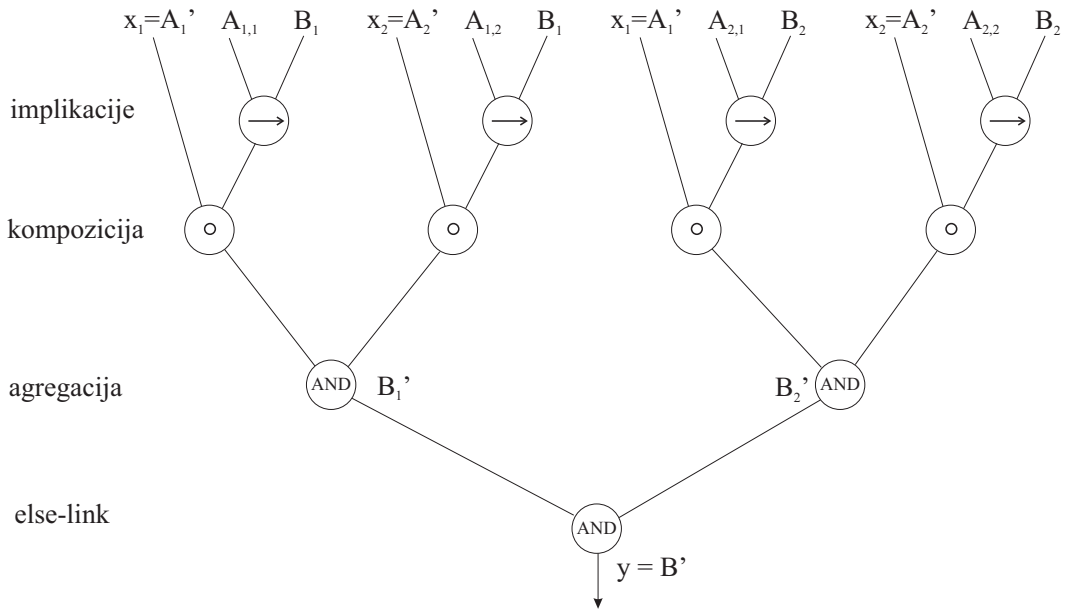
Mehko sklepanje

Mehko sklepanje je inferenčna metoda, ki uporablja mehke implikacijske relacije, mehke kompozicijske operatorje in operator ELSE, ki povezuje mehka pravila (angl. else-link operator), za katerega običajno izberemo OR-link ali AND-link. V prvem primeru (OR) povezujemo rezultate posameznih pravil z MAX operacijo, v drugem (AND) pa z MIN operacijo. Mehko sklepanje ali inferenco določa torej trojček:

$$F = < I, C, L > ,$$

kjer je I mehka implikacija (npr. $R_c = u \wedge v$), C mehka kompozicija (npr. MAX-MIN) in L povezovalni operator (npr. OR).

Ker smo si implikacijske relacije in kompozicijo že ogledali, skupaj z načinoma sklepanja GMP in GMT, poglejmo sedaj še, kako vse te elemente povežemo v t.i. dekompozicijsko referenčno strategijo. Pri tem predpostavljamo, da pravilo s k pogoji razstavimo (dekomponiramo) na k implikacij $A_{ji} \rightarrow B_i$, j = 1, 2, ..., k, v primeru R_i. Vsaka implikacija posebej daje rezultat sklepanja $B_i' = A_{ji}' \circ (A_{ji} \rightarrow B_i)$, j = 1, 2, ..., k. Delne rezultate B_i' združimo (angl. aggregate) z enim od agregacijskih operatorjev (običajno AND, OR). Končno s povezovalnim (else-link) operatorjem rezultate posameznih pravil pretvorimo v rezultat. Omenjeni postopek dekompozicijske strategije prikazuje inferenčno drevo. Na Sliki 5.5 je prikazano drevo sklepanja za primer dveh pravil:



Slika 5.5: Primer inferenčnega drevesa dekompozicijske strategije.

R_1: IF x_1 is $A_{1,1}$ AND x_2 is $A_{1,2}$ THEN y is B_1

R_2: IF x_1 is $A_{2,1}$ AND x_2 is $A_{2,2}$ THEN y is B_2

Mehko sklepanje na osnovi 'ostrih' podatkov

V primeru, da so vhodni podatki podani v t.i. 'ostri' (angl. crisp) obliki, oz. v numerični obliki, ter da se pričakujejo tudi rezultati v isti obliki, potem mehko sklepanje zahteva po vrsti: mehčanje vhodnih spremenljivk, ocenjevanje pravil in ostrenje izhodnih mehkih spremenljivk. V tem podpoglavju bomo spoznali ocenjevanje, ostrenje pa v naslednjem, ker presega tukajšnje okvirje.

Ocenjevanje mehkega pravila pomeni računanje izhodne pripadnostne funkcije B' na osnovi vrednosti vhodnih pripadnostnih funkcij $\mu_{A_1}(x'_1)$ in $\mu_{A_2}(x'_2)$. Pri tem se uporabljata dve metodi:

- Sklepanje z operacijo 'min':
$$B' = B \cdot \min(\mu_{A_1}(x'_1), \mu_{A_2}(x'_2))$$
- Sklepanje s produktom:
$$B' = B \cdot \mu_{A_1}(x'_1) \cdot \mu_{A_2}(x'_2)\,,$$

kjer je $\cdot$ algebraični produkt, min pa splošni agregacijski operator, ki združuje delne rezultate posameznih pogojev v okviru enega pravila. Pri tem sta običajna operatorja AND, OR posplošena s t.i. T-normo in S-normo (ali T-ko-normo), katerih tipični predstavniki so:

T-norma: $T(a,b) = a\ t\ b =$
- $\min(a,b)$
- $a \cdot b$
- $\max(0,\ a+b-1)$

S-norma: $S(a,b) = a\ s\ b =$
- $\max(a,b)$
- $a+b-a \cdot b$
- $\min(1,\ a+b)$

Pri tem sta t in s operatorja T-norme oz. S-norme.

Ostrenje

Rezultat, izhodno mehko spremenljivko oz. njeno funkcijo pripadnosti, dobljen z metodo mehkega sklepanja na osnovi mehkih pravil, je potrebno včasih pretvoriti še v končno realno numerično obliko. Postopek se imenuje ostrenje (angl. defuzzification). V rabi sta dve metodi ostrenja:

- Težišče oz. Center gravitacije (CG):

$$y' = \frac{\int y\mu_B(y)dy}{\int \mu_B(y)dy}$$

  Običajno vzamemo kar vsoto kot približek:

$$y' = \frac{\sum_y y \cdot \mu_B(y)}{\sum_y \mu_B(y)}$$

- Sredina maksimumov (SM): tukaj iščemo y', ki ustreza maksimumu pripadajoče pripadnostne funkcije B'. Če je maksimumov več, poiščemo njihovo povprečje:

$$y' = \frac{m+M}{2},$$

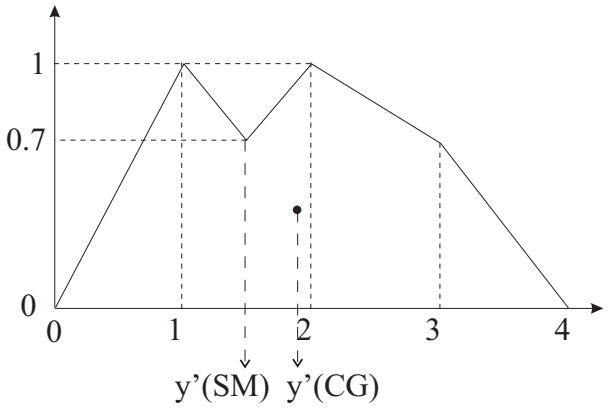
  kjer sta m najmanjši in M največji y, pri katerih ima pripadnostna funkcija lokalni maksimum.

Prva metoda je bolj standardna, prednost druge pa je v lažji realizaciji. Slika 5.6 ilustrira obe omenjeni metodi ostrenja, z računanjem centra gravitacije (CG) in računanja sredine maksimumov (SM).

Primer 1: Mehko sklepanje

Vožnja z avtomobilom. Tabelarično so podane tri mehke množice z lingvistično interpretacijo:

$$\begin{aligned} A_{11} &\equiv \text{počasi} = \{1/0,\ 0.67/20,\ 0.33/40,\ 0/60,\ 0/80,\ 0/100\},\\ A_{12} &\equiv \text{srednje} = \{0.33/0,\ 0.67/20,\ 1/40,\ 0.67/60,\ 0.33/80,\ 0/100\},\\ A_{13} &\equiv \text{hitro} = \{0/0,\ 0/20,\ 0.33/40,\ 0.67/60,\ 1/80,\ 1/100\}, \end{aligned}$$



$$y'(CG) = \frac{(0 \text{ x } 0) + (1 \text{ x } 1) + (1 \text{ x } 2) + (0.7 \text{ x } 3)}{0 + 1 + 1 + 0.7} \doteq 1.9$$

$$y'(SM) = \frac{1 + 2}{2} = 1.5$$

Slika 5.6: Primer dveh metod ostrenja, CG in SM.

ki pokrivajo vhodno mehko spremenljivko $u \in U$, ki predstavlja hitrost vožnje, ter tri mehke množice

$$\begin{aligned} A_{21} &\equiv \text{nizka} = \{0/3,\ 0.5/4,\ 1/5,\ 0.5/6,\ 0/7,\ 0/8,\ 0/9,\ 0/10,\ 0/11\}, \\ A_{22} &\equiv \text{srednja} = \{0/3,\ 0/4,\ 0/5,\ 0.5/6,\ 1/7,\ 0.5/8,\ 0/9,\ 0/10,\ 0/11\}, \\ A_{23} &\equiv \text{visoka} = \{0/3,\ 0/4,\ 0/5,\ 0/6,\ 0/7,\ 0.5/8,\ 1/9,\ 0.5/10,\ 0/11\}, \end{aligned}$$

nad izhodno spremenljivko $v \in V$, ki predstavlja porabo goriva. Vhodna mehka spremenljivka je X, izhodna pa Y:

$$\begin{aligned} X &\in \{\text{počasi, srednje, hitro}\} \quad \text{... hitrost (vzrok)} \\ Y &\in \{\text{nizka, srednja, visoka}\} \quad \text{... poraba (posledica)} \end{aligned}$$

Obstajajo pravila, ki povezujejo vhode in izhode s pomočjo relacij med mehkimi množicami:

R_1 : IF X is počasi THEN Y is visoka OR

R_2 : IF X is srednje THEN Y is nizka OR

R_3 : IF X is hitro THEN Y is srednja

Skupaj vsa tri pravila zapišemo krajše:

$$R : (\text{počasi} \wedge \text{visoka}) \vee (\text{srednje} \wedge \text{nizka}) \vee (\text{hitro} \wedge \text{srednja}) \,,$$

oz. s pomočjo pripadnostnih funkcij:

$$\mu_R(u,v) = \vee_i(\mu_{A_i'} \wedge \mu_{B_i'}) = \max_i(\min(\mu_{A_i'}, \mu_{B_i'})) = \max_i(\mu_{R_i}) \,.$$

Za podane mehke množice imajo posamezna pravila naslednje matrične vrednosti:

R_1 : počasi $\wedge$ visoka $= (1, 0.67, 0.33, 0, 0, 0)^T \wedge (0, 0, 0, 0, 0, 0.5, 1, 0.5, 0) =$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0.5 & 1 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0.67 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.33 & 0.33 & 0.33 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

R_2 : srednje $\wedge$ nizka $= (0.33, 0.67, 1, 0.67, 0.33, 0)^T \wedge (0, 0.5, 1, 0.5, 0, 0, 0, 0, 0) =$

$$\begin{bmatrix} 0 & 0.33 & 0.33 & 0.33 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0.67 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 1 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0.67 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.33 & 0.33 & 0.33 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

R_3 : hitro $\wedge$ srednja $= (0, 0, 0.33, 0.67, 1, 1)^T \wedge (0, 0, 0, 0.5, 1, 0.5, 0, 0, 0) =$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.33 & 0.33 & 0.33 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.67 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 1 & 0.5 & 0 & 0 & 0 \end{bmatrix}$$

Posamezna pravila združimo z operatorjem max ($\vee$) tako, da za vsako pozicijo treh matrik poiščemo maksimalno vrednost. Tako dobimo:

$$R: \begin{bmatrix} 0 & 0.33 & 0.33 & 0.33 & 0 & 0.5 & 1 & 0.5 & 0 \\ 0 & 0.5 & 0.67 & 0.5 & 0 & 0.5 & 0.67 & 0.5 & 0 \\ 0 & 0.5 & 1 & 0.5 & 0.33 & 0.33 & 0.33 & 0.33 & 0 \\ 0 & 0.5 & 0.67 & 0.5 & 0.67 & 0.5 & 0 & 0 & 0 \\ 0 & 0.33 & 0.33 & 0.5 & 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 1 & 0.5 & 0 & 0 & 0 \end{bmatrix}$$

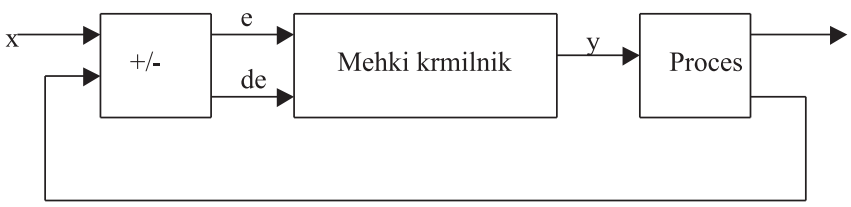
Sedaj uporabimo skupno pravilo R pri sklepanju na osnovi GMP, kar pomeni, da pri znani mehki množici na vhodu poiščemo mehko množico na izhodu. Vhodna mehka množica A' naj bo 'hitro'. Izvajamo postopek GMP: $R \therefore B'$.

$$\begin{aligned} B' &= \max(\min(\mu_{A'}, \mu_R)) \\ &= \max(\min(\begin{bmatrix} 0 \\ 0 \\ 0.33 \\ 0.67 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 & 0.33 & 0.33 & 0.33 & 0 & 0.5 & 1 & 0.5 & 0 \\ 0 & 0.5 & 0.67 & 0.5 & 0 & 0.5 & 0.67 & 0.5 & 0 \\ 0 & 0.5 & 1 & 0.5 & 0.33 & 0.33 & 0.33 & 0.33 & 0 \\ 0 & 0.5 & 0.67 & 0.5 & 0.67 & 0.5 & 0 & 0 & 0 \\ 0 & 0.33 & 0.33 & 0.5 & 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 1 & 0.5 & 0 & 0 & 0 \end{bmatrix})) \\ &= \max(\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.33 & 0.33 & 0.33 & 0.33 & 0.33 & 0.33 & 0.33 & 0 \\ 0 & 0.5 & 0.67 & 0.5 & 0.67 & 0.5 & 0 & 0 & 0 \\ 0 & 0.33 & 0.33 & 0.5 & 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 1 & 0.5 & 0 & 0 & 0 \end{bmatrix}) \\ &= (0, 0.5, 0.67, 0.5, 1, 0.5, 0.33, 0.33, 0) \end{aligned}$$

Priklic nam je dal mehko množico, ki ni povsem enaka tisti, ki jo predvideva pravilo R_3, ji je pa precej podobna. Slednje je posledica vpliva ostalih pravil, ki smo jih tudi upoštevali pri sklepanju.

Primer 2: Mehki krmilnik (diskretno-računsko)

Ogledali si bomo primer mehkega krmilnika, ki deluje v okviru regulatorja na Sliki 5.7. V mehki krmilnik vstopata dve mehki spremenljivki, e ('error' oz. napaka) in de ('difference of error' oz. sprememba napake), izhodna spremenljivka y pa krmili proces s ciljem slediti vhodni referenčni spremenljivki oz. njeni vrednosti.

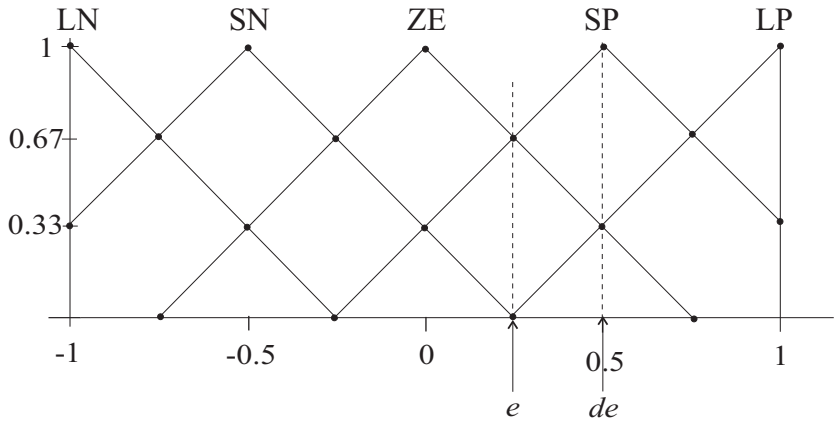


Slika 5.7: Mehki regulator.

Za vse tri spremenljivke bomo zaradi enostavnosti uporabili enake mehke množice:

LP: velika pozitivna ('large positive')
SP: majhna pozitivna ('small positive')
ZE: nič ('zero')
SN: majhna negativna ('small negative')
LN: velika negativna ('large negative')

Pokritja z mehkimi množicami v grafični obliki so na Sliki 5.8. Predpostavljamo



Slika 5.8: Mehke množice za regulator na Sliki 5.7.

obseg vrednosti med -1 in +1 z ločljivostjo (resolucijo) 0.25. Pokritost mehkih množic za vse tri spremenljivke podaja tabela:

	-1	-0.75	-0.5	-0.25	0	0.25	0.5	0.75	1
LN	1	0.67	0.33	0	0	0	0	0	0
SN	0.33	0.67	1	0.67	0.33	0	0	0	0
ZE	0	0	0.33	0.67	1	0.67	0.33	0	0
SP	0	0	0	0	0.33	0.67	1	0.67	0.33
LP	0	0	0	0	0	0	0.33	0.67	1

Predpostavimo, da mehki krmilnik določajo pravila:

R_1: IF e is ZE AND de is ZE THEN y is ZE

R_2: IF e is ZE AND de is SP THEN y is SN

R_3: IF e is SN AND de is SN THEN y is LP

R_4: IF e is LP OR de is LP THEN y is LN

Sedaj vzemimo, da imamo na vhodu krmilnika vrednosti: $e = 0.25$ in $de = 0.5$.

Pri takšnih vrednostih imajo pripadnostne funkcije posameznim mehkim množicam naslednje vrednosti (izhaja iz zgornje tabele in iz Slike 5.8):

$e = 0.25$		$de = 0.5$	
$\mu_{LP}(e)$	0	$\mu_{LP}(de)$	0.33
$\mu_{SP}(e)$	0.67	$\mu_{SP}(de)$	1
$\mu_{ZE}(e)$	0.67	$\mu_{ZE}(de)$	0.33
$\mu_{SN}(e)$	0	$\mu_{SN}(de)$	0
$\mu_{LN}(e)$	0	$\mu_{LN}(de)$	0

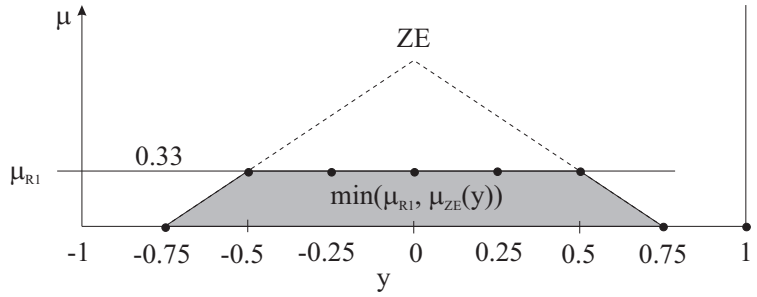
Poglejmo sedaj, v kolikšni meri se dani vrednosti vhodnih spremenljivk nahajata v območju pravil:

$$
\begin{aligned}
R_1 &: \mu_{R_1} = \min(\mu_{ZE}(e = 0.25), \mu_{ZE}(de = 0.5)) = \min(0.67, 0.33) = 0.33 \\
R_2 &: \mu_{R_2} = \min(\mu_{ZE}(e = 0.25), \mu_{SP}(de = 0.5)) = \min(0.67, 1) = 0.67 \\
R_3 &: \mu_{R_3} = \min(\mu_{SN}(e = 0.25), \mu_{SN}(de = 0.5)) = \min(0.0, 0.0) = 0.0 \\
R_4 &: \mu_{R_4} = \max(\mu_{LP}(e = 0.25), \mu_{LP}(de = 0.5)) = \max(0.0, 0.33) = 0.33
\end{aligned}
$$

Vrednost μ_{R_i} nam pove, v kolikšni meri je treba upoštevati i-to pravilo, oz. kakšna je njegova 'teža'. To je odvisno od tega, v kolikšni meri se vhodni spremenljivki nahajata v območju tega pravila.

Te vrednosti vplivajo na sklepne mehke množice na izhodu:

$$\begin{aligned}
\mu_{Y_1'}(y) &= \min(\mu_{R_1}, \mu_{ZE}(y)) = (0, 0, 0.33, 0.33, 0.33, 0.33, 0.33, 0, 0) \\
\mu_{Y_2'}(y) &= \min(\mu_{R_2}, \mu_{SN}(y)) = (0.33, 0.67, 0.67, 0.67, 0.33, 0, 0, 0, 0) \\
\mu_{Y_3'}(y) &= \min(\mu_{R_3}, \mu_{LP}(y)) = (0, 0, 0, 0, 0, 0, 0, 0, 0) \\
\mu_{Y_4'}(y) &= \min(\mu_{R_4}, \mu_{LN}(y)) = (0.33, 0.33, 0.33, 0, 0, 0, 0, 0, 0)
\end{aligned}$$



Slika 5.9: Izhodna mehka množica Y_1' - mehak sklep za pravilo R_1.

Določiti moramo še mehko izhodno množico, ki upošteva vsa pravila oz. vse delne rezultate:

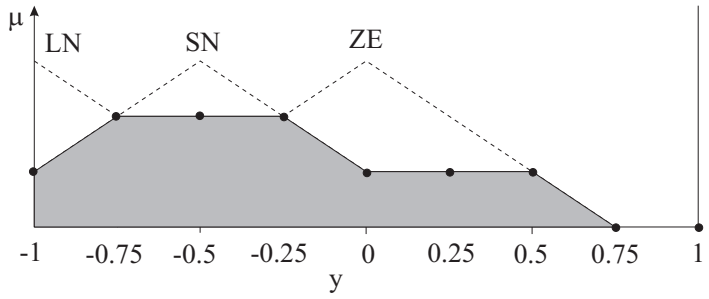
$$\begin{aligned}
Y' &= \max(\min(0.33, ZE), \min(0.67, SN), \min(0.0, LP), \min(0.33, LN)) \\
&= (0.33, 0.67, 0.67, 0.67, 0.33, 0.33, 0.33, 0, 0)\,.
\end{aligned}$$

Prikazana je tudi na Sliki 5.10. Izhodno (ostro) vrednost dobimo s pomočjo računanja težišča (metoda **CG**):

$$y = \frac{\sum_i y_i \mu_{Y'}(y_i)}{\sum_i \mu_{Y'}(y_i)}\,.$$

Za naš primer dobimo končno realno vrednost za izhodno spremenljivko:

$$\begin{aligned}
y' &= \frac{-1 \cdot 0.33 - 0.75 \cdot 0.67 - 0.5 \cdot 0.67 - 0.25 \cdot 0.67 + 0.25 \cdot 0.33 + 0.5 \cdot 0.33}{0.33 + 0.67 + 0.67 + 0.67 + 0.33 + 0.33 + 0.33} \\
&= -0.327 \\
&\approx -0.3
\end{aligned}$$



Slika 5.10: Izhodna mehka množica Y'.

Primer 3: Mehki krmilnik (zvezno-grafično)

Imamo vhodni spremenljivki x in y ter izhodno spremenljivko z; vse so realne. Nad vsako spremenljivko imamo tri mehke množice. Nad x in nad y so definirane množice 'nizek', 'srednji' in 'visok', nad z pa 'negativen', 'nič' in 'pozitiven' (Tabela 5.1). Pripadnostne funkcije mehkih množic, ki so odsekoma

Tabela 5.1: Mehke množice.

x		y		z	
N	... nizek	N	... nizek	NE	... negativen
S	... srednji	S	... srednji	NI	... nič
V	... visok	V	... visok	PO	... pozitiven

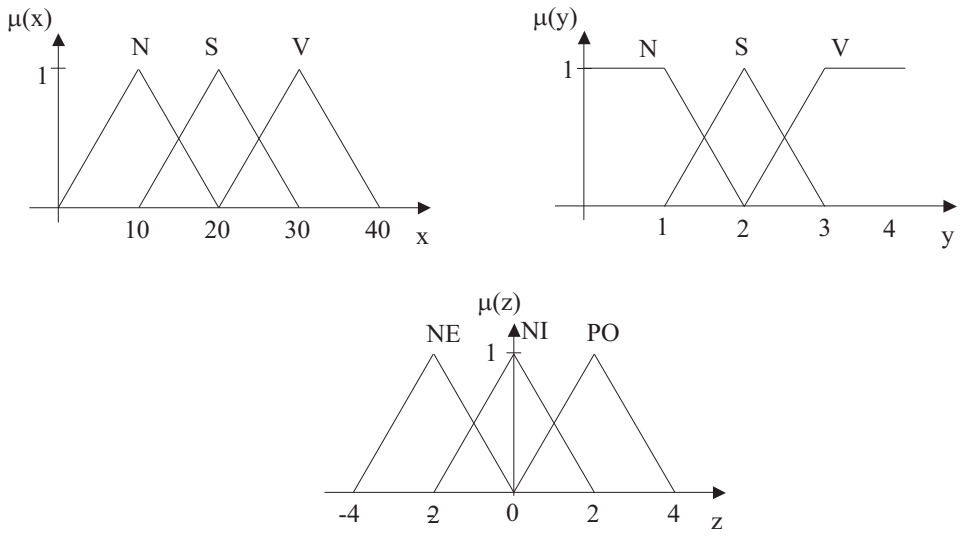
linearne, so prikazane na Sliki 5.11. Odsekoma linearne mehke množice lahko zapišemo tudi s štirimi parametri (razlaga je na Sliki 5.12):

$\mu_N(x) = (0, 10, 10, 20)$ $\mu_S(x) = (10, 20, 20, 30)$ $\mu_V(x) = (20, 30, 30, 40)$
$\mu_N(y) = (0, 0, 1, 2)$ $\mu_S(y) = (1, 2, 2, 3)$ $\mu_V(y) = (2, 3, 4, 4)$
$\mu_{NE}(z) = (-4, -2, -2, 0)$ $\mu_{NI}(z) = (-2, 0, 0, 2)$ $\mu_{PO}(z) = (0, 2, 2, 4)$

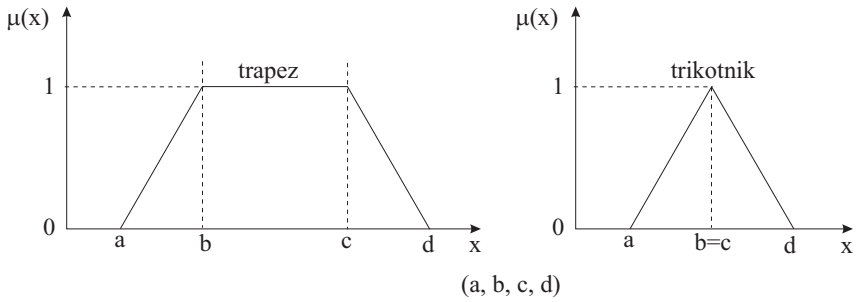
Poleg tega je podan nabor pravil tipa IF-THEN:

IF X=N AND Y=N THEN Z=PO
IF X=N AND Y=S THEN Z=PO
IF X=N AND Y=V THEN Z=PO

IF X=S AND Y=N THEN Z=PO
IF X=S AND Y=S THEN Z=NI
IF X=S AND Y=V THEN Z=NE



Slika 5.11: Mehke množice nad spremenljivkami x, y in z.



Slika 5.12: Način zapisa za trapezne ter trikotniške oblike pripadnosti s parametri a, b, c in d.

IF X=V AND Y=N THEN Z=NE
IF X=V AND Y=S THEN Z=NE
IF X=V AND Y=V THEN Z=NE

Bolj pregledno so pravila predstavljena na Sliki 5.13.

Na osnovi podanih podatkov izvedimo mehko sklepanje za primer vhodnih vrednosti x=12 in y=1.9.

y \ x	N	S	V
N	PO	PO	NE
S	PO	NI	NE
V	PO	NE	NE

Slika 5.13: Pravila.

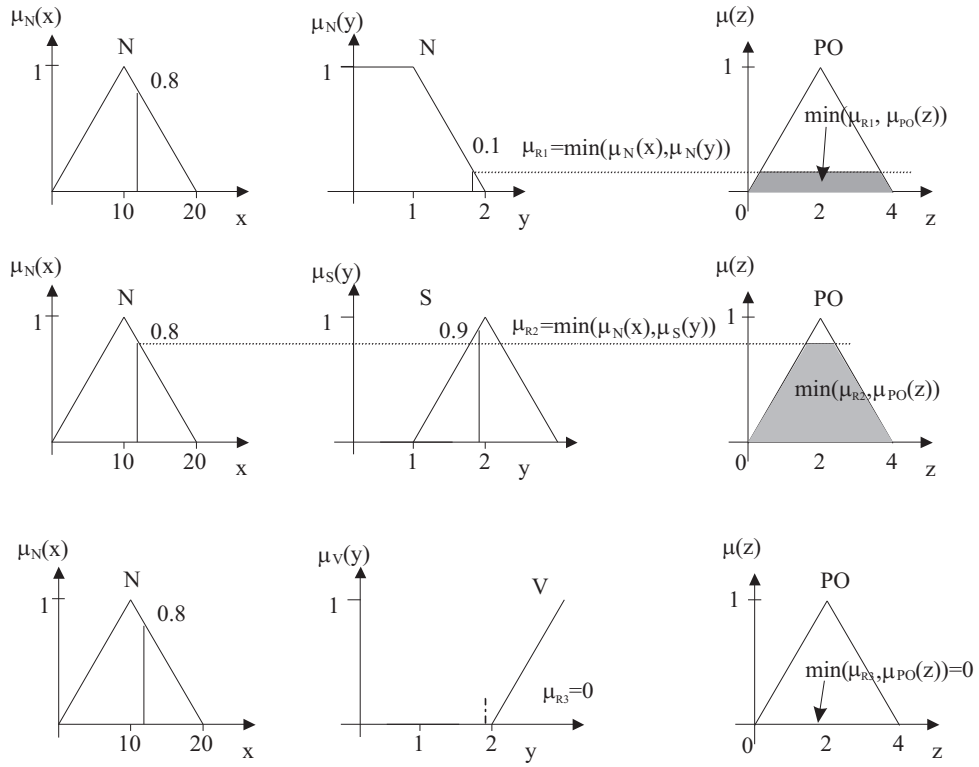
Za vsako pravilo ugotovimo pripadnost konkretnih vhodnih vrednosti mehkim množicam, ki nastopajo v pravilu. Nato vzamemo konjunkcijo obeh predpostavk (v mehki logiki se običajno uporablja operator minimum). Za prvo pravilo imamo torej $\mu_{R_1} = \min(\mu_N(x), \mu_N(y))$. Ta vrednost nam pove, v kolikšni meri je dotično pravilo izpolnjeno glede vhodnih spremenljivk, tj. v kolikšni meri sta podana x in y nizka. V tej meri bo upoštevan tudi izhod preko operacije $\min(\mu_{R_1}, m_{PO}(z))$. To je mehka množica, katere pripadnost dobimo tako, da $\mu_{PO}(z)$ omejimo z vrednostjo μ_{R_1}, kot kaže Slika 5.14.

Isto storimo za vsa ostala pravila, kot je prikazano na Slikah 5.14, 5.15 in 5.16.

Ugotovimo, da imamo le štiri množice z $\mu_{R_i} \neq 0$. To pomeni, da na izhod vplivajo le štiri pravila, kar je lepo vidno na Sliki 5.17.

Sedaj vse štiri izhodne množice združimo v mehko množico, ki predstavlja disjunkcijo pravil in s tem unijo izhodnih množic. Za unijo običajno vzamemo operator maksimum. Dobimo sestavljen lik, ki predstavlja izhodno mehko množico, kot kaže Slika 5.18. Seveda si v realnem svetu z mehko množico direktno ne moremo pomagati, zato jo je potrebno izostriti (ang. defuzzification). Ostrenje se običajno prevede na izračun težišča lika, ki predstavlja pripadnostno funkcijo, oz. (uteženega) povprečja

$$z' = \frac{\int z\mu(z)dz}{\int \mu(z)dz}$$



Slika 5.14: Pravila za x = N; y = N, S, V.

mehke množice. S tem dobimo konkretno vrednost z. Uteženo povprečje lahko približno izračunamo kar z vsoto namesto integrala:

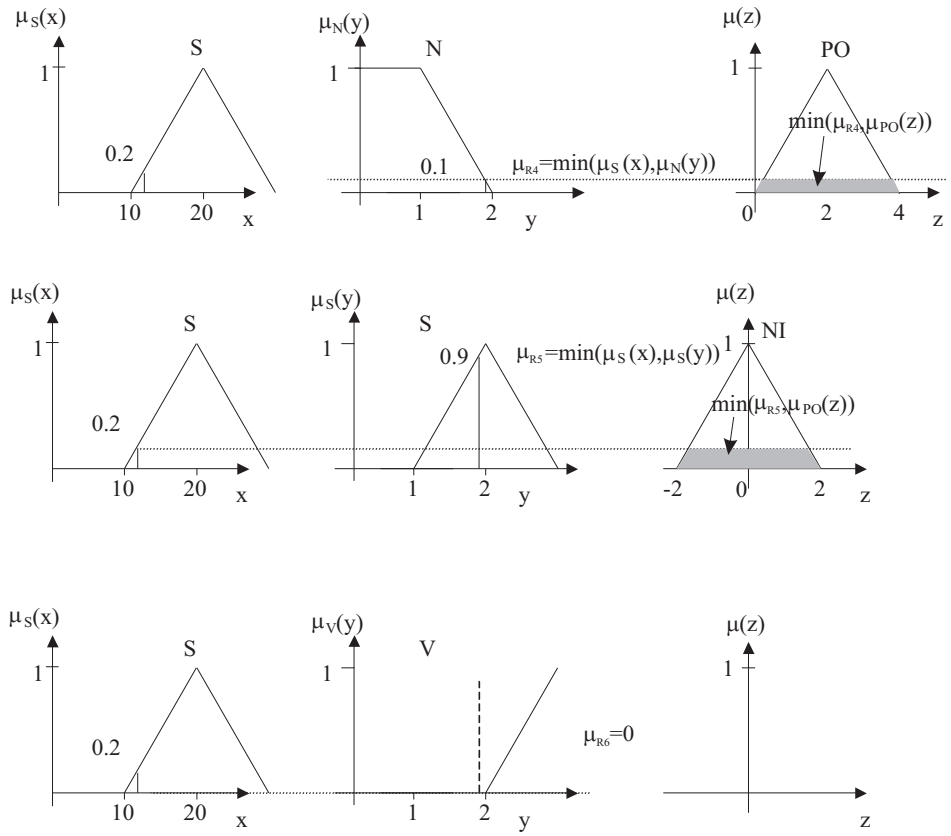
$$z' \approx \frac{\sum_{k=0}^{K} z_k \mu(z_k)}{\sum_{k=0}^{K} \mu(z_k)},$$

kjer je $z_k = z_{min} + k\Delta z$ in $K = (z_{max} - z_{min})/\Delta z$. Če vzamemo zaradi enostavnosti za korak Δz kar 0.2, dobimo

$$z = \frac{\sum_{k=0}^{40} (-4 + k * 0.2) m(-4 + 0.2k)}{\sum_{k=0}^{40} m(-4 + 0.2k)},$$

kot končni rezultat mehkega sklepanja $z' = 1.54$.

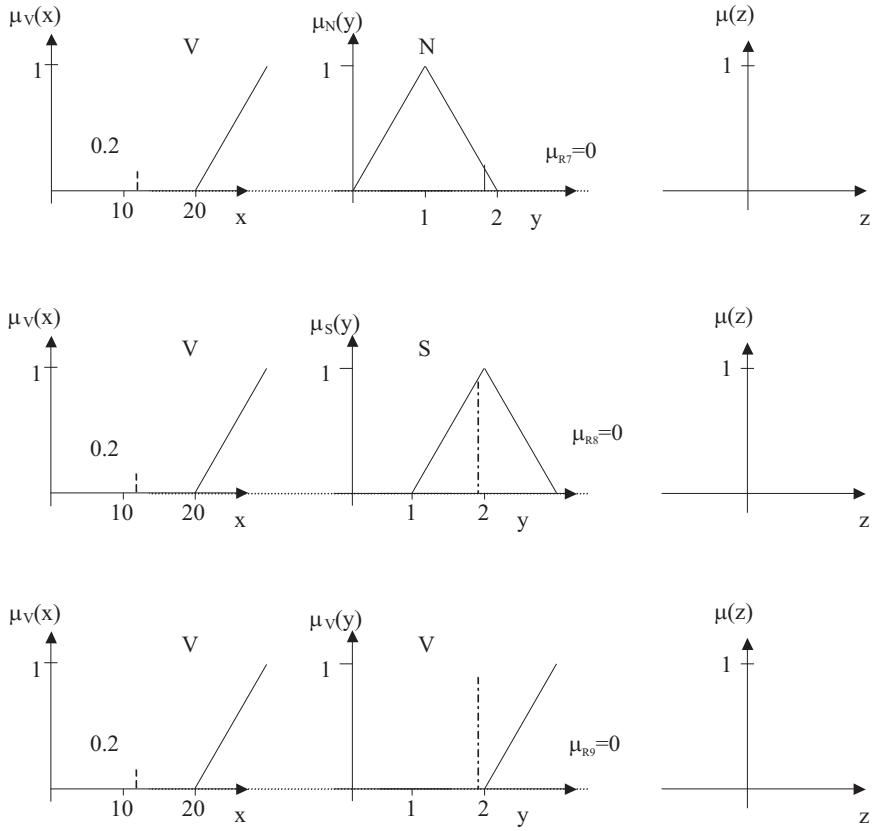
Primer 4: Vzvratno učenje in mehko sklepanje



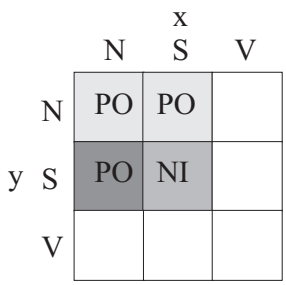
Slika 5.15: Pravila za x = S; y = N, S, V.

V enačbah za učenje mreže nastopa učilni parameter ('learning rate'), ki določa korak sprememb uteži. Le-ta je smiselno odvisen od prvega in drugega odvoda, kar je mogoče prikazati z mehkimi pravili ($\Delta\alpha$ = f(CE, CCE)), ki so prikazana v Tabeli 5.2. Pri tem je CE ('Change of Error') sprememba napake, CCE ('Change of CE') pa sprememba spremembe napake, kar smiselno ustreza drugemu odvodu. Primer mehkih množic za CE, CCE in $\Delta\alpha$ prikazuje Slika 5.19. Primer izračuna $\Delta\alpha$ na osnovi mehkih pravil (za vrednosti CE = 0.03, CCE = -0.15) pa je na sliki 5.20.

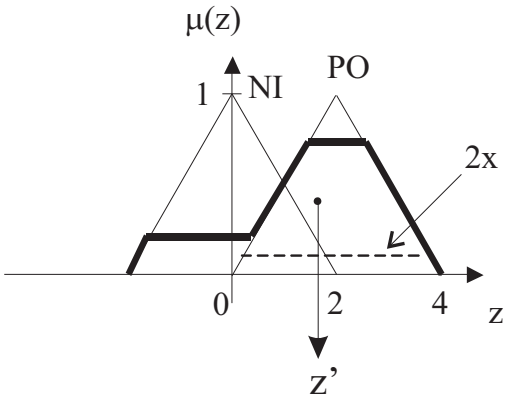
Primere mehkega sklepanja najdemo tudi v [37] in v [38].



Slika 5.16: Pravila za x = V; y = N, S, V.



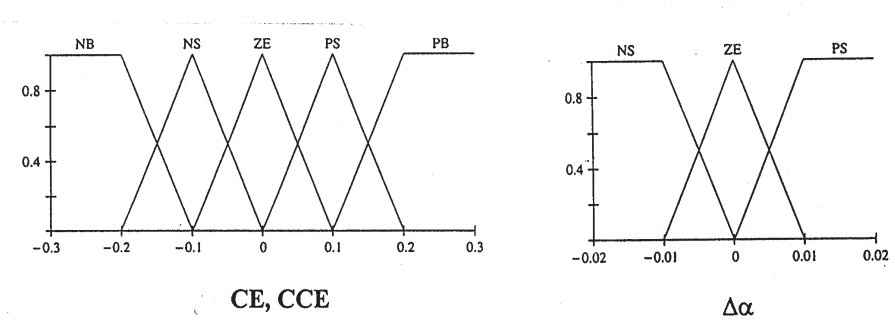
Slika 5.17: "Teža" pravil.



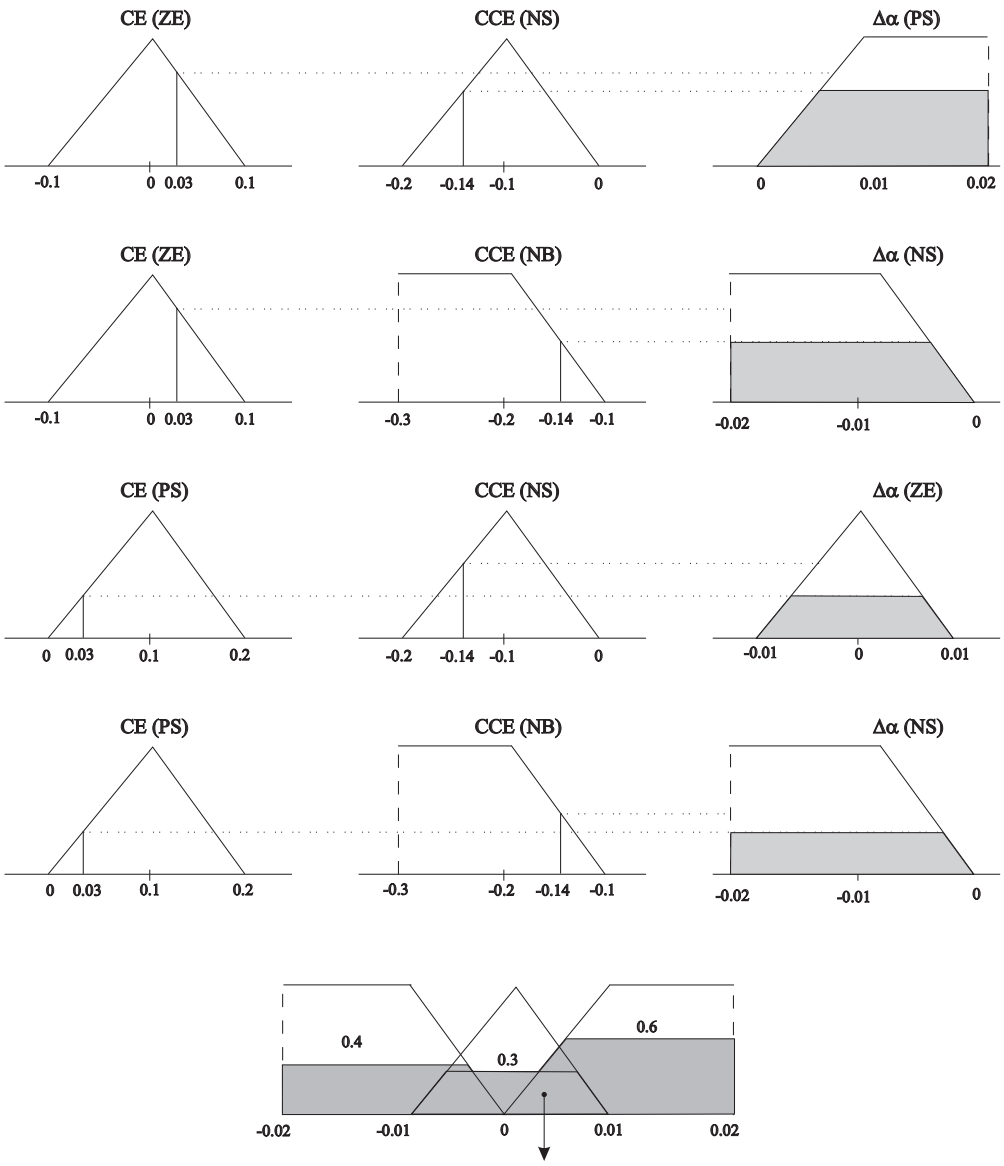
Slika 5.18: Ostrenje za Primer 3.

Tabela 5.2: Mehka pravila za Primer 4.

		CE				
		NB	NS	ZE	PS	PB
	NB	NS	NS	NS	NS	NS
	NS	NS	ZE	PS	ZE	NS
CCE	ZE	ZE	PS	PS	PS	ZE
	PS	NS	ZE	PS	ZE	NS
	PB	NS	NS	NS	NS	NS



Slika 5.19: Mehke množice za vhodni spremenljivki CE in CCE ter za izhodno spremenljivko $\Delta\alpha$.



Slika 5.20: Primer izračuna $\Delta\alpha$ na osnovi mehkih pravil.

Poglavje 6

Hibridne metode

Področja, ki smo si jih ogledali do sedaj, predvsem nevronske mreže, evolucijski algoritmi in mehka logika, imajo precej različne izvore svojega nastanka. Umetne nevronske mreže v bioloških živčnih sistemih, evolucijski algoritmi v naravni selekciji in mehka logika v simboličnem razmišljanju. Pa vendarle imajo vse omenjene metode, ki določajo t.i. mehko računanje, pomembno lastnost, to je reševanje problemov. Znotraj tega skupnega cilja pa je mogoče najti številne povezave med njimi. Za vse omenjene metode je značilno, da na različne načine izkoriščajo toleranco v natančnosti sledenja cilju, robustnost delovanja in nizka cena procesiranja.

V tem poglavju nas bo zanimalo predvsem sodelovanje glavnih metod mehkega računanja med seboj in ne njihovo zaporedno povezovanje. Po vrsti bomo spoznali evolucijsko snovanje nevronskih mrež, evolucijsko snovanje mehkih sistemov, nato kombinacijo nevronskih mrež in mehkih sistemov ter na koncu še mehke evolucijske algoritme.

6.1 Evolucijsko snovanje nevronskih mrež

V poglavju o nevronskih mrežah smo spoznali snovanje mrež na osnovi nadzorovanega učenja, samo-organizacije in spodbujevanega (angl. reinforcement) učenja. Vsi ti postopki so imeli za cilj, v različnih pogojih določiti notranje parametre nevronske mreže tako, da je mreža sposobna izvajati določeno nal-

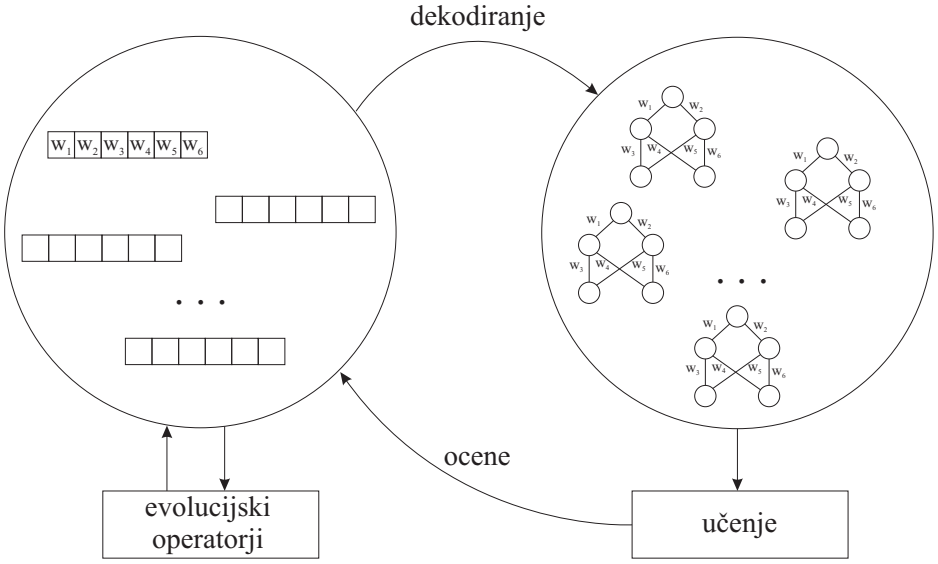
ogo. Pri tem je rezultat močno odvisen od izbrane topologije mreže, od izbranih začetnih uteži in od parametrov postopka učenja. Običajno se omenjene parametre izbira izkustveno na osnovi splošnih navodil ali pa s pomočjo obsežnega eksperimentiranja.

Evolucijski algoritmi se lahko uporabljajo pri nevronskih mrežah na več načinov. Najpomembnejši so naslednji: določitev uteži pri fiksni topologiji mreže, iskanje topologije, razvoj (evolucija) učilnega pravila, izbira vhodnih lastnosti (angl. features).

6.1.1 Evolucija uteži v fiksni topologiji nevronske mreže

Pri najbolj uporabljenemu algoritmu učenja nevronske mreže, t.i. vzvratnemu pravilu BP (angl. backpropagation), obstaja kar nekaj težav (npr. lokalni minimum), zaradi katerih se zdi stohastično učenje z evolucijskim algoritmom dobra alternativa gradientnemu postopku učenja. Čeprav je možnosti več, se bomo omejili le na uporabo GA, katerega prednost pred BP je tudi v tem, da deluje pri nezvezni (npr. pragovni) izhodni funkciji nevrona, v primeru rekurentnih ali splošno povezljivih mrež ter v primeru okolja, ki omogoča le oceno delovanja mreže (spodbujevano snovanje).

Namesto adaptiranja uteži na lokalne izboljšave kriterijske funkcije (vsota kvadratov napak na izhodu mreže čez učno množico), GA evolucijsko spreminja uteži glede na prileganje mreže [39, 40]. Genotip nevronske mreže je seznam uteži, vsaka utež je predstavljena kot binarna serija oz. koda realnega števila v območju $[-1,+1]$. Takšno binarno kodiranje uteži pa ima precej slabosti, saj s številom uteži genotip naraste tako, da postane postopek evolucije nesprejemljivo dolg. Poleg tega večja natančnost zahteva daljše binarne zapise, kar tudi zmanjšuje skalabilnost postopka. Rešitev je v predstavitvi uteži direktno z realnimi števili, vendar to pomeni ustrezno uporabo genetskih operatorjev mutacije in križanja (glej poglavje III). Fenotip mreže pomeni njeno obnašanje (ob uporabi uteži iz genotipa) nad učnimi podatki, funkcija prileganja pa je enaka akumulirani napaki. Slika 6.1 ilustrira evolucijsko snovanje nevronske mreže.



Slika 6.1: Shema evolucijskega snovanja nevronske mreže.

Genetski algoritmi in spodbujevano snovanje nevronskih mrež

Velikokrat v procesu snovanja nimamo želenih vrednosti, na osnovi katerih bi učili ali evolucijsko snovali nevronsko mrežo. Kadar lahko le ocenjujemo izhodne vrednosti mreže kot njen odgovor na določene vhodne vrednosti, tedaj govorimo o spodbujevanem snovanju mreže. Ocena okolja (aplikacije) je lahko le pozitivna (nagrada, glej poglavje o učečih avtomatih) ali negativna (kazen). Genetski algoritem reagira na pozitivno oceno z povečanjem vrednosti aktivnih povezav (uteži, ki vodijo v vzbujene nevrone), na negativno oceno pa z zmanjšanjem aktivnih povezav [41]. Genetski algoritmi so bili uspešno uporabljeni v primeru krmiljenja obrnjenega nihala in nekaterih nalog avtonomnega robota [42].

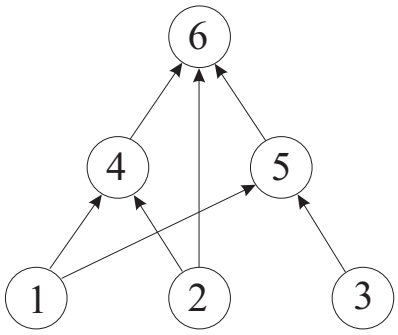
6.1.2 Evolucija nevronskih topologij ali arhitektur

Izbira pravilne topologije ali arhitekture nevronske mreže je zelo pomembna za njen uspeh pri reševanju določenega problema. To pomeni določitev števila vozlišč (nevronov) v nevronski mreži in izbira vzorca njihovih povezav. Obstajata dva načina, ki omogočata iskanje primerne topologije s pomočjo evolucijskega

algoritma: z direktnim ali indirektnim kodiranjem.

Direktno kodiranje

Pri tej kodirni shemi je topologija nevronske mreže predstavljena s sosednostno matriko A reda $N \times N$, kjer je N število nevronov. Element matrike $a_{ij} = 1$ pomeni, da je povezava med nevronoma i in j, njegova vrednost 0 pa, da povezave ni. Genotip je tedaj binarni niz, ki povezuje zaporedne vrstice v matriki med seboj. Primer topologije in ustrezne matrike prikazuje Slika 6.2. Elementi matrike so lahko tudi realna števila, v tem primeru je mogoče evolu-



	1	2	3	4	5	6
1	0	0	0	1	1	0
2	0	0	0	1	0	1
3	0	0	0	0	1	0
4	0	0	0	0	0	1
5	0	0	0	0	0	1
6	0	0	0	0	0	0

Slika 6.2: Primer direktnega kodiranja topologije nevronske mreže.

cijsko spreminjati tako arhitekturo kot tudi uteži [43].

Indirektno kodiranje

Zaradi problema skalabilnosti pri direktnem kodiranju, je postalo zanimivo kodiranje, ki bi omogočalo opis t.i. rastoče mreže (angl. growing network). Takšno kodiranje določa gramatika s pravili, s katerimi generira graf oz. nevronsko mrežo. Ker postopek posega v povsem novo področje, podajamo le ustrezno referenco [44].

6.1.3 Evolucija učilnih pravil

Pri učenju nevronskih mrež se uporablja več vrst algoritmov, kar smo spoznali v poglavju o nevronskih mrežah. Videli smo, da učenje običajno sledi izboru topologije ali arhitekture nevronske mreže. Zato lahko nastopi problem, kadar ne vemo, katera topologija je najboljša za obravnavani problem. Tedaj postane avtomatizirana adaptacija učilnega pravila koristna in evolucijski algoritmi so pri tem lahko dobra izbira. Chalmersa [45] je zanimalo, ali je mogoče z GA priti do znanega delta pravila učenja (spoznali smo ga pri učenju perceptrona v poglavju o nevronskih mrežah) ali njegove izboljšave. To mu je tudi uspelo, vendar s številnimi omejitvami, npr. z linearnimi funkcijami parametrov (vhodi, izhodi, želene vrednosti) in spreminjanja uteži, itd. Uporabil je le vnaprej povezano mrežo brez skrite plasti nevronov, kar omejuje njeno uporabo le na linearno ločljive preslikave. Kljub temu odpira omenjen eksperiment prostor za nove raziskave. Dosedanji povzetek dosežkov v tej smeri podaja [39].

6.2 Evolucijsko snovanje mehkih sistemov

Eden od razlogov za uspeh mehke logike na številnih področjih, je v uporabi lingvističnih spremenljivk, vrednosti in pravil, kar omogoča preslikavo človeškega znanja v delujoče sisteme. Vendar v primeru, ko ekspertno znanje ne obstaja, ne obstaja način, kako zgraditi mehki sistem. Tedaj lahko uporabimo evolucijske algoritme za iskanje primerne rešitve.

Prvi poizkusi v tej smeri so iz devetdesetih let [46]. Pri tem so bile uporabljene preproste oblike za funkcije pripadnosti (trikotne, trapezne), ki jih je mogoče opisati le z dvema ali tremi parametri, poleg tega pa je bilo število mehkih množic (domen) na spremenljivko fiksno. Mehka pravila so bila v obliki kartezijskih produktov nad vhodnimi spremenljivkami in binarne vrednosti so opisovale njihovo prisotnost ali odsotnost v množici pravil. Omenjene poenostavitve so precej zmanjšale prostostno stopnjo iskanja optimalnih parametrov.

Pri mehkih sistemih obstaja precej lastnosti, ki jih je mogoče obravnavati kot parametre za evolucijsko iskanje. Poleg oblik funkcij pripadnosti in njihovega števila, so tukaj še strukturne značilnosti, to pa pomeni oblika in vsebina mehkih pravil, ki so jedro mehkega sistema. V nadaljevanju nas bo zani-

mala predvsem evolucija mehkih pravil, ker smo se z običajno optimizacijo parametrov spoznali že v prejšnjih poglavjih.

6.2.1 Evolucija mehkih pravil

Obstaja več oblik kodiranja mehkih pravil, ki so primerne za evolucijsko iskanje oz. snovanje. Ogledali si bomo le dve, tabelarično in identifikacijsko kodiranje. Kodirano pravilo predstavlja genotip, ki ga postopek evolucije spreminja na običajen način (glej poglavje o evolucijskem računanju), s ciljem poiskati najboljše pravilo za dani problem, oz. takšno, ki daje nad testno množico najboljše rezultate.

Tabelarično kodiranje pravil

Tabelarično kodiranje pravil je predlagal Thrift [47] s tem, da je predpostavljal mehko pravilo v tabelarični obliki. Če so x_1, x_2, ..., x_N vhodne spremenljivke v sistem in ima vsaka m_i domen (mehkih množic), $i = 1, ..., N$, potem je niz pravil podan s pomočjo matrike R reda $m_1 \times m_2 \times ... \times m_N$, kjer v matriki nastopajo lingvistične spremenljivke izhodne spremenljivke y. Takšna tabelarična predstavitev je predvsem primerna za manjše sisteme, npr. z dvema vhodnima in eno izhodno spremenljivko, kjer je pravilo oblike:

IF x_1 is A_j AND x_2 is B_k THEN y is C

Tedaj je C element matrike R na mestu (j, k). Elementi v matriki so lahko tudi prazni, kar pomeni, da ustreznega pravila ni. Genotip pravila izhaja direktno iz matrike tako, da si vrstice matrike v njem sledijo po vrsti. Nad takšnim genotipom je mogoče uporabiti standardno mutacijo in križanje, vendar je Thrift uporabil posebno mutacijo, ki je spremenila labelo izhodne mehke množice v sosednjo ali pa v prazno labelo. V primeru prazne labele je mutacija izbrala naključno labelo, vendar ne prazno.

Identifikacijsko kodiranje pravil

Identifikacijsko kodiranje je alternativa tabelaričnemu. Vsako pravilo je označeno s celoštevilčno oznako ali identiteto (ID) tako, da množico pravil predstavlja seznam njihovih ID oznak [48]. Ob predpostavki, da imajo vse vhodne in izhodna spremenljivka po m domen, je število možnih vhodnih kombinacij m^N. Zato vsako pravilo vsebuje eno od m^N vhodnih kombinacij in eno od m možnih izhodnih posledic. Zato je v celoti možnih m^{N+1} različnih pravil, ki jih označujejo števila med 0 in $m^{N+1} - 1$. Takšne oznake (ID) je mogoče kodirati z $\lceil (N+1) \log_2 m \rceil$ biti.

Pri znani ID pravila d je mogoče z naslednjo proceduro pravilo poiskati v seznamu pravil:

1. $i \leftarrow 1$;
2. pripadnostna funkcija spremenljivke x_i je A_{ij}, $j = (d \mod m) + 1$;
3. $d \leftarrow \lfloor d/m \rfloor$;
4. $i \leftarrow i + 1$; če $i \leq N$, nadaljuj s korakom 2, sicer nadaljuj s korakom 5
5. izhodna funkcija pripadnosti je B_d

6.3 Nevronski mehki sistemi

Nevronski mehki sistemi združujejo lastnosti umetnih nevronskih mrež in mehke logike. Seveda je cilj združevanja privzeti od vsake metodologije le dobre lastnosti. Npr. mehki sistemi lahko uspešno delujejo z nenatančno informacijo in tudi omogočajo dobro razlago za svoje početje. Po drugi strani pa morajo biti mehka pravila integrirana v sistem, da je z njim mogoče izvajati sklepe. Mehki sistemi se torej ne morejo učiti, kar pa je bistvena lastnost nevronskih mrež. Vendar pa je relativno težko analizirati pridobljeno znanje iz nevronskih mrež, kar smo spoznali v poglavju o nevronskih mrežah. V principu zato lahko od kombinacije nevronskih mreži in mehke logike pričakujemo sistem, ki je sposoben dojemanja (učenja) na nizkem nivoju signalne percepcije in ki ima visoko nivojsko sposobnost razmišljanja in lingvističnega izražanja.

Obstajata dva načina, kako izvesti omenjeno sinergijo. Prvi predpostavlja modifikacijo nevronske mreže v t.i. mehko nevronsko mrežo (angl. fuzzy neural network), drugi pa nevronska obogatitev mehkih sistemov z učilnimi sposobnostmi (angl. neuro-fuzzy system). Pri prvem načinu je več možnih nivojev, na katerih izvajamo modifikacijo, na nivoju uteži, nivoju nevronov oz. njihovih funkcij ali na nivoju algoritmov učenja. Pri drugem načinu pa je nevronska mreža predvidena za učenje pravil in/ali pripadnostnih funkcij mehkega sistema.

6.3.1 Mehke nevronske mreže

Vpliv mehke logike je mogoče integrirati v nevronske mreže na različnih nivojih. Najbolj primeren se zdi nivo nevrona, kar pomeni vpeljavo modela mehkega nevrona.

Mehki nevron

Osnovni model nevrona predpostavlja združevalno ali agregacijsko računanje utežene vsote oz. skalarnega produkta med vhodnim in utežnim vektorjem, temu pa sledi izhodna ali aktivacijska funkcija. Zaradi potrebe po odvajanju, se je osnovna pragovna aktivacijska funkcija zamenjala s sigmoidno funkcijo. Korak v smeri mehkega nevrona je lahko v izbiri drugačnih oblik združevanja vhodnega in utežnega vektorja, npr.:

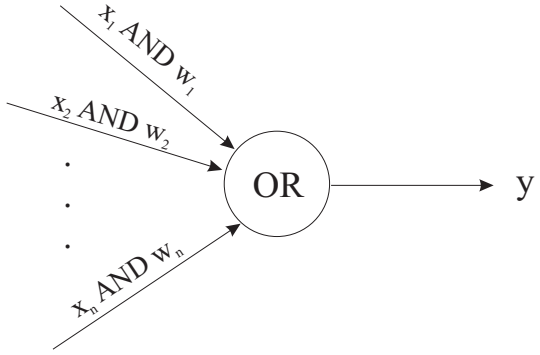
$$y = g(A(\boldsymbol{\omega}, \mathbf{x}))\,,$$

kjer je g prenosna (aktivacijska) funkcija in y skalarna izhodna vrednost nevrona. Namesto utežene vsote je mogoče izbrati mehko unijo, presek, ali v splošnem S-normo oz. T-normo (glej poglavje o mehki logiki). Izmed številnih možnosti si oglejmo dva posebna primera, mehka nevrona OR in AND.

A. Mehki nevron OR

   Slika 6.3 podaja model mehkega nevrona OR. Mehki nevron OR realizira preslikavo od enotine hiperkocke do vrednosti, ki predstavlja mehko pripadnostno funkcijo znotraj enotinega intervala:

$$OR: \quad [0,1]^n \times [0,1]^n \rightarrow [0,1]\,,$$



Slika 6.3: Model mehkega nevrona OR.

kjer so uteži tudi definirane znotraj enotinega intervala. Mehki nevron OR uporablja agregacijsko funkcijo, ki ustreza maksimalni vrednosti uteženih vhodov. To pomeni, da na izhodu daje rezultat mehko disjunkcijo mehkih konjunkcij med posameznimi vhodi in utežmi:

$$\text{y} = \text{OR } (x_1 \text{ AND } \omega_1, x_2 \text{ AND } \omega_2, ..., x_n \text{ AND } \omega_n) \ .$$

Ker se mehki logični operatorji S-norme in T-norme (glej poglavje o mehki logiki) označujejo z ∇ in Δ, je enačba nevrona OR podana lahko tudi z:

$$y = \nabla_{j=1}^{n}(x_j \Delta \omega_j) \, .$$

B. Mehki nevron AND

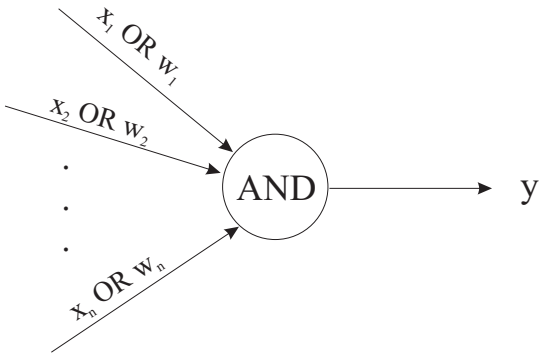
Mehki model AND, ki ga prikazuje Slika 6.4, je dualen modelu OR. Vhodi so disjunktivno povezani z utežmi, vse skupaj pa združi v izhod konjunkcija. Prenosna funkcija nevrona AND je:

$$\text{y} = \text{AND } (x_1 \text{ OR } \omega_1, x_2 \text{ OR } \omega_2, ..., x_n \text{ OR } \omega_n) \ ,$$

oziroma bolj konvencionalno:

$$y = \Delta_{j=1}^{n}(x_j \nabla \omega_j) \, .$$

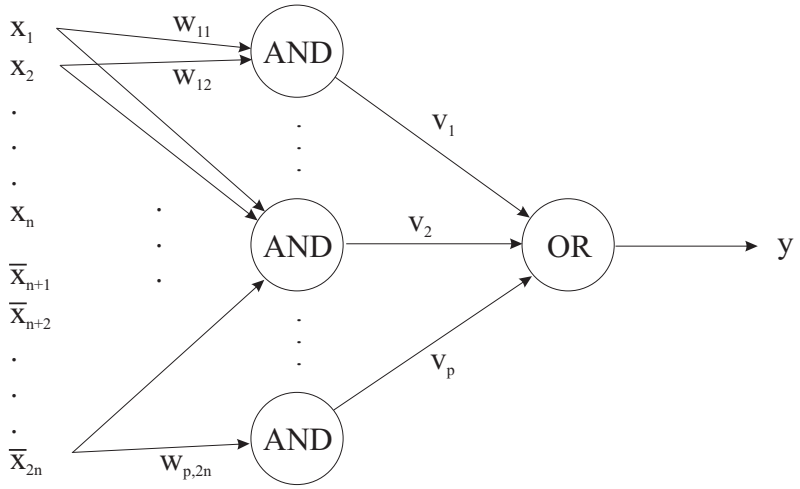
Če v obeh modelih upoštevamo še utež pri konstantni vrednosti oz. zamik (angl. bias), potem v obeh enačbah teče indeks od 0 do n.



Slika 6.4: Model mehkega nevrona AND.

Večnivojska mehka nevronska mreža

Če uporabimo mehke nevrone AND in OR v zaporednih plasteh, dobimo mehko večnivojsko mrežo. Pri njej na vhodu nastopa $2n$ spremenljivk, ker poleg pravih vrednosti vhodnih spremenljivk nastopajo še njihove negacije. Če prvi nivo tvorijo mehki nevroni OR in če je nevron drugega nivoja tipa AND, potem je to primer konjunktivne mehke nevronske mreže, v zamenjanem slučaju pa primer disjunktivne mehke nevronske mreže. Slika 6.5 prikazuje mehko dvonivojsko disjunktivno nevronsko mrežo.



Slika 6.5: Mehka disjunktivna dvonivojska nevronska mreža.

Učenje v mehkih nevronskih mrežah

Nadzorovano učenje mehkih nevronskih mrež pomeni spreminjanje povezovalnih uteži v mreži tako, da se ciljna funkcija (napaka) nad testnimi vhodno-izhodnimi pari zmanjšuje. Dodatna zahteva pri učenju je, da je dobljena mreža sposobna posploševanja, kar pomeni, da se njeno obnašanje ohranja, ko se odziva na nove vhodne podatke.

Vzemimo, da imamo testno množico sestavljeno iz parov $(\mathbf{x}_k, d_k)$, $k = 1, 2, ..., n$, kjer je $\mathbf{x}_k$ k-ti vhodni vektor in d_k želena k-ta izhodna skalarna vrednost. Učenje mehke nevronske mreže ima cilj minimizirati ciljno funkcijo napake:

$$E(\boldsymbol{\omega}) = \frac{1}{2}\sum_{k=1}^{n}(d_k - y_k)^2$$

s spreminjanjem uteži z gradientnim zmanjševanjem uteži:

$$\Delta\omega_{ij} = -\eta\frac{\partial E}{\partial\omega_{ij}}\,,$$

kjer je η učilni parameter oz. skalirni faktor, ki nadzira velikost spreminjanja uteži. Računanje sprememb uteži zahteva uporabo verižnega pravila odvajanja, kot smo to spoznali v poglavju o nevronskih mrežah, zato podrobnosti na tem mestu izpuščamo. Detajlno izpeljavo učenja mehkim nevronskih mrež je mogoče najti v [49].

6.3.2 Nevronski mehki sistemi

Tudi v okviru nevronskega učenja mehkih sistemov obstaja več načinov. V nadaljevanju si bomo ogledali učenje mehkih sistemov, ki uporabljajo mehka pravila tipa Takagi-Sugeno [50].

Predpostavljajmo, da so lingvistične vrednosti, ki nastopajo v pravilu kot vhodni pogoji, definirane s parametriziranimi trikotnimi mehkimi množicami na naslednji način:

$$\mu(x) = \begin{cases} 1 - 2\frac{|x-C|}{b}\,, & \text{če je } C - \frac{b}{2} \le x \le C + \frac{b}{2} \\ 0\,, & \text{sicer}\,, \end{cases}$$

kjer je C center in b širina trikotnika. Posledica v pravilu je trda vrednost zamika (angl. bias) ω_0, kar je degeneriran primer pravila T-S, vhodne pogoje pa povezuje operator T-norme. Učilni algoritem sloni na gradientnem sestopu z naslednjo mero za napako:

$$\text{hse} = \frac{1}{2}\sum_{k=1}^{m}(y_k - d_k)^2\,,$$

kjer je hse 'polovičen kvadrat napake' (angl. half squared error), y_k je izhodna vrednost, ki jo izračuna mehki sistem, d_k pa želena vrednost iz učnih podatkov. Računanje sprememb za parametre C in b za pogojne funkcije pripadnosti (mehke množice) in posledico ω_0 se za vsa pravila računa ekvivalentno vzvratnemu pravilu (angl. backpropagation) pri nevronski mreži MLP. Ker uporabljene trikotne pripadnostne funkcije v treh točkah niso odvedljive, je potrebno z uporabo hevristike rešiti omenjeni problem.

Slaba stran opisanega pristopa je v tem, da je semantika lingvističnih vrednosti odvisna od pravil, v katerih se pojavljajo. Če so vse funkcije pripadnosti na začetku enakih oblik, se le-te ob koncu učenja razlikujejo, zaradi spreminjanja parametrov, ki jih opisujejo. To pa ni najbolj zaželeno, saj spremeni interpretacijo končnih pravil. Da se temu izognemo, je potrebno uporabiti identične lingvistične izraze za enake funkcije pripadnosti [51].

6.4 Mehki evolucijski algoritmi

Sinergija med evolucijskimi algoritmi in mehko logiko se pojavlja v treh komplementarnih oblikah.

Najbolj očitna oblika izkorišča optimalno iskalno lastnost evolucijskih algoritmov za sintezo in optimizacijo mehkih sistemov.

Druga možnost se kaže v uporabi mehkega znanja za nadzor evolucijskega procesa, kar preprečuje neželena obnašanja, npr. prezgodnjo konvergenco.

Tretja možnost pa vgrajuje mehke lastnosti v evolucijske algoritme. Npr. preciznost računanja prileganja je mogoče žrtvovati za ceno ohranjanja procesorskih virov tako, da se uporabi mehka funkcija prileganja ali mehki genetski operatorji.

V nadaljevanju bosta podrobneje opisani zadnji dve obliki.

6.4.1 Mehko krmiljenje evolucije

Definicija parametrov evolucijskega algoritma je ključni faktor pri določanju razmerja med izkoriščanjem in raziskovanjem in bistveno vpliva na njegovo uspešnost. Zato evolucijski algoritmi zahtevajo med svojim delovanjem določen (človeški) nadzor in po potrebi korekcijo parametrov, ki ohranja njegovo zadovoljivo obnašanje. Tudi odločitev o končanju se velikokrat prepusti človeški presoji, ki se odloča med kvaliteto rezultatov in časovnimi omejitvami.

Alternativa človeškemu nadzoru je *adaptivni evolucijski algoritem*, ki je sposoben spreminjati kontrolne parametre med evolucijo tako, da je zagotovljeno njegovo najboljše obnašanje. Ker je evolucijski proces kompleksen sistem z nelinearnim obnašanjem, se zdi izbira mehkega krmiljenja spreminjanja parametrov za izvedbo evolucijskega algoritma primerna.

Mehka vlada

Mehka vlada (angl. fuzzy government) pomeni zbirko mehkih pravil in rutin, ki kontrolirajo evolucijo populacije, detektirajo primernost rešitev, spreminjajo parametre in preprečujejo neželeno obnašanje.

Mehki krmilnik pri izbiri svojih akcij upošteva obnašanje eksperta. Mehka vlada pa dinamično spreminja izbrane krmilne parametre ali genetske operatorje med delovanjem evolucijskega algoritma tako, da omogoča najprimernejšo raziskovalno (prostorsko) in izkoriščevalno (časovno) obnašanje. Privlačnost uporabe mehkega krmiljenja je v možnosti enostavne vključitve ekspertnega (nepreciznega) znanja v sistem.

Arhitektura adaptivnega evolucijskega algoritma

Bistvo adaptivnega evolucijskega algoritma je v uporabi mehkega krmilnika, mehkega vladanja, kjer so vhodi statistika evolucijskega algoritma in izhodi krmilni parametri.

Statistika evolucijskega algoritma, ki se izvaja z določeno hitrostjo, npr. po r generacijah, se pošilja procesu mehkega vladanja, ki izvede določeno sklepanje, s katerim generira nove vrednosti za vse krmilne parametre. Ti so nato ažurirani z novimi vrednostmi in kontrola se vrne evolucijskemu algoritmu.

A. Statistika evolucijskega algoritma se deli na dva razreda: na statistiko genotipov in fenotipov. Merjenja različnosti so osnova za statistiko genotipov. Temeljijo na definiciji mehke podobnostne mere:

$$\mu_{podoben}(\gamma, \kappa)\,,$$

kjer sta γ in κ dva genotipa. Za statistiko fenotipov pa je značilna fenotipska mera različnosti, npr. območje prileganja, razmerje najboljši / povprečen, varianca prileganja. Možne so še druge statistike, npr. hitrost uspeha kot razmerje med mutacijami, ki vodijo k izboljšanju prileganja proti vsem mutacijam, itd.

B. Krmilni parametri so lahko običajni parametri evolucijskega algoritma (npr. velikost populacije, hitrost mutacije in križanja, itd.) ali parametri iz aplikacije.

C. Mehka pravila v mehki vladi so določena z vrednostmi lingvističnih spremenljivk (ali vrednostmi pripadajočih funkcij prileganja). V njih se združuje ekspertno znanje (o aplikaciji) in splošno znanje o evolucijskem algoritmu. Alternativa takšnim mehkim pravilom predstavlja avtomatsko učenje mehkih pravil in/ali funkcij pripadnosti.

D. Ekspertno znanje lahko uporabi proces mehkega vodenja pri reševanju številnih potencialnih težav evolucijskega algoritma, npr. prepočasne konvergence ali prezgodnje konvergence. Vzroki so lahko naslednji: parametri na začetku niso dobro izbrani, parametri se ne spreminjajo med delovanjem, pa bi se morali, interakcija med parametri je kompleksna in težko razumljiva. Mehko vladanje lahko pomaga v vseh treh slučajih. Tabela 6.1 v ilustracijo prikazuje dva niza pravil za dinamično spreminjanje hitrosti mutacije in križanja pri GA.

Avtomatsko učenje

Avtomatsko učenje je proces iskanja mehkih pravil skupaj z definicijami ustreznih funkcij prileganja. Avtorja Lee in Takagi [48] sta uporabila mehki sistem s tremi vhodnimi spremenljivkami:

p_c	Velikost populacije		
Št. generacij	majhna	srednja	velika
majhno	srednja	majhna	majhna
srednje	velika	velika	srednja
veliko	zelo velika	zelo velika	velika

p_m	Velikost populacije		
Št. generacij	majhna	srednja	velika
majhno	velika	srednja	majhna
srednje	srednja	majhna	zelo majhna
veliko	majhna	zelo majhna	zelo majhna

Tabela 6.1: Ilustracija mehkega vodenja operatorjev križanja in mutacije GA.

- razmerje med povprečnim in najboljšim prileganjem: $\overline{f}/f^*$
- razmerje med najslabšim in povprečnim prileganjem: $f_{min}/\overline{f}$
- sprememba prileganja od zadnje krmilne akcije

in tremi izhodnimi spremenljivkami:

- velikost populacije
- hitrost mutacije (verjetnost njene izvedbe)
- hitrost križanja

Rezultati eksperimenta so pokazali, da evolucijski algoritem (z uporabo mehkega vladanja) za razvoj mehkega krmilnika obrnjenega nihala daje boljše rezultate kot 'ročno' izdelani (statični) algoritem.

6.4.2 Evolucijski algoritmi z mehkimi komponentami

Pri določenih aplikacijah je težko natančno izmeriti kvaliteto posameznih rešitev. To je posledica velikega (ali neskončnega) števila primerov, nad katerimi bi bilo potrebno testirati rešitve. To je pogost primer pri genetskem programiranju

(GP), kjer je vzorec za testiranje izbran vnaprej, v upanju da je reprezentativen. Dostikrat je računanje prileganja podvrženo tudi šumu, kar pomeni, da rezultat pri istem vzorcu v različnih časih ni enak.

V takšnih primerih je mogoče priti do boljše ocene prileganja za ceno ponavljanja računanja tolikokrat, kot je potrebno. Vendar je mogoče priti do istega rezultata in hkrati prihraniti procesorski čas, če se uporabijo instrumenti z vgrajeno mehko logiko.

Mehki genetski operatorji

Mehka rekombinacija in mehka mutacija sta posplošitev diskretnih operatorjev [52].

A. Mehka rekombinacija:

   Pri danih genotipih prednikov $(x_1, ..., x_N)$ in $(y_1, ..., y_N)$ je verjetnost, da ima potomec vrednost z_i, $i = 1, ..., N$, podana z bimodalno porazdelitvijo:

$$p(z_i) \in \{\Phi(x_i), \Phi(y_i)\}\,,$$

   kjer je $\Phi(r)$ trikotna verjetnostna porazdelitev in r način (angl. mode), definiran v področju $[r - d\,|y_i - x_i|,\ r + d\,|y_i - x_i|]$:

$$\Phi(z) = \begin{cases} \frac{z-r+d\,|y_i-x_i|}{d^2(y_i-x_i)^2}\,, & z \le r \\ \frac{r-z+d\,|y_i-x_i|}{d^2(y_i-x_i)^2}\,, & z > r \end{cases}\,,$$

   kjer je $d \ge 1/2$.

B. Mehka mutacija:

   Pri realnem genu z vrednostmi iz intervala $[a, b]$ mehka mutacija generira novo vrednost gena:

$$x' = x + \Delta\,,$$

   kjer ima Δ porazdelitev $p(\Delta)$, naključno izbrano iz družine:

$$\{\Phi(\pm A\beta^{\pi}), \Phi(\pm A\beta^{\pi+1}), ..., \Phi(\pm A\beta^{0})\}\,,$$

   kjer je A amplituda mutacije, $A \ll b - a$, $\pi = \lfloor \log_\beta R_{min} \rfloor < 0$ z $\beta > 1$ je osnova (angl. base) mutacije, R_{min} spodnja limita relativne spremembe mutacije, $\Phi(x)$ pa je definirana podobno kot v primeru mehke rekombinacije.

Literatura

[1] M. L. Tsetlin: On the Behaviour of Finite Automata in Random Media, Automation and Remote Control 22, 1210–1219, 1962. Originalno v Avtomatika i Telemekhanika 22, 1345–1354, 1961.

[2] K. Narendra, M. Thathachar: Learning Automata: An Introduction, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[3] V. I. Varshavskii, I. P. Vorontsova: On the Behaviour of Stochastic Automata with a Variable Structure, Automation and Remote Control 24, 327–333, 1963.

[4] R. R. Bush, F. Mosteller: Stochastic Models for Learning, John Wiley & Sons, New York, 1958.

[5] I. J. Shapiro, K. S. Narendra: Use of Stochastic Automata for Parameter Self-Optimization with Multi-Modal Performance Criteria, IEEE Transactions on Systems Sci. and Cybernetics SSC-5, 352–360, 1969.

[6] S. Lakshmivarahan, M. A. L. Thathachar: Absolutely Expedient Learning Algorithms for Learning Automata, IEEE Transactions on Systems, Man and Cybernetics SMC-3, 281–286, 1973.

[7] W.S. McCulloch, W. Pitts: A Logical Calculus of the Ideas Immanent in Nervous Activity, Bull. Math. Biophys. 5, 115–133, 1943.

[8] F. Rosenblatt: Principles of Neurodynamics, Spartan, New York, 1962.

[9] M. Minsky, S. Papert: Perceptrons: An Introduction to Computational Geometry, MIT Press, 1969.

[10] D. E. Rumelhart, G. E. Hinton, R. J. Williams: Learning internal representations by error propagation, v D. E. Rumelhart, J. L. McClelland (ur.): Parallel Distributed Processing 1: 318–362, The MIT Press, Cambridge, Massachusetts, 1986.

[11] S. Haykin: Neural Networks: A Comprehensive Foundation, 2. izdaja, Prentice Hall Int., 1999.

[12] D. Lowe: Adaptive radial basis function nonlinearities, and the problem of generalisation, 1st IEE International Conference on Artificial Neural Networks, 171–175, London, UK, 1989.

[13] G.H. Golub, C.G. Van Loan: Matrix Computations, 3.izdaja, John Hopkins Univ. Press, 1996.

[14] J. Moody, C.J. Darken: Fast learning in networks of locally-tuned processing units, Neural Computation 1(2), 281–294, 1989.

[15] T. Poggio, F. Girosi: Networks for Approximation and Learning, Proceedings of the IEEE 78, 1481–1497, 1990.

[16] R.O. Duda, P.E. Hart: Pattern Classificationand Scene Analysis, Wiley, 1973.

[17] S. Chen: Nonlinear time series modelling and prediction using Gaussian RBF networks with enhanced clustering and RLS learning, Electronics Letters, 31(2), 117–118, 1995.

[18] T. Kohonen: Self-Organization and Associative Memory, Springer, 1984.

[19] R. J. Williams, D. Zipser: A Learning Algorithm for Continually Running Fully Recurrent Neural Networks, Neural Computation 1(2), 270–280, 1989.

[20] I. Gabrijel, A. Dobnikar: On-line identification and reconstruction of finite automata with generalized recurrent neural networks, Neural Networks 16: 101–120, 2003.

[21] H. Tsukimoto: Extracting Propositions from Trained Neural Networks, Int. Joint Conf. on Artif. Int., IJCAI'97.

[22] S. Bornholdt, H.G. Schuster (ur.): Handbook of Graphs and Networks, Wiley-VCH, 2003

[23] A. Dobnikar, B. Šter: Structural Properties of Recurrent Neural Networks, Neural Proc. Lett., poslano v objavo.

[24] V. Batagelj, A. Mrvar: Pajek - Program for Large Network Analysis, Connections 21(2): 47–57, 1998.

[25] A.E. Eiben, J.E. Smith: Introduction to Evolutionary Computing, Springer, 2003.

[26] J. Holland, Adaptation in natural and artificial systems, University of Michigan Press, 1975.

[27] I. Rechenberg: Evolutionstrategie: Optimierung Technicher Systeme nach Principien des Biologishen Evolution, Fromman-Hezlbog Verlag, 1973.

[28] H.P. Schwefel: Numerical Optimization of Computer Models, Wiley, 1981.

[29] H.G. Beyer: The Theory of Evolution Strategies, Springer, 2001.

[30] L.J. Fogel, A.J. Owens, M.J. Walsh: Artificial Intelligence through Simulated Evolution, Wiley, 1966.

[31] A.Dobnikar, U. Lotrič: Porazdeljeni sistemi, FRI, 2008.

[32] J.R. Koza: Genetic programming, MIT Press, 1992.

[33] W. Banzhaf, P. Nordin, R.E. Keller, F.D. Francone: Genetic Programming: An Introduction, Morgan Kaufman, 1998.

[34] L. Zadeh: Fuzzy Sets, Information and Control, 1965.

[35] B. Kosko: Fuzzy entropy and conditioning, Information Sciences, 1986.

[36] N.K. Kasabov: Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering, A Bradford Book, MIT Press, 1998.

[37] A. Dobnikar, U. Lotrič, B. Šter, M. Trebar: Inteligentni sistem vodenja proizvodne linije gumijevih profilov. Organizacija (Kranj), 37(4), 234–241, 2004.

[38] I. Gabrijel, A. Dobnikar, N.C. Steele: RBF neural networks and fuzzy logic based control - a case study. V: P. Borne, M. Ksouri, A. El Kamel (ur.): Proceedings of the 2nd IMACS International Multiconference on Computational Engineering in Systems Applications - CESA'98, Nabeul-Hammamet, Tunisia, April 1-4 1998, vol. 2, str. 284–289, 1998.

[39] X.Yao: Evolving artificial neural network, Proc. of the IEEE, 1999.

[40] A. Dobnikar: Evolutionary design of application-specific neural networks: a genetic approach, Neural Netw. World, 5(1), 41–50, 1995.

[41] D. Whitley: Genetic algorithms and neural networks, v M. Galan in sod.(ur.): Genetic algorithms in Engineering and Computer Science, J.Wiley, 1995.

[42] D. Floreano, J.Urzelai: Evolution and learning in autonomous robotic agents, v D. Mange, M. Tomassini (ur.): Bio-Inspired Computing Machines: Towards Novel Computational Architectures, Presses Polytechniques et Universitaires Romandes, Lausanne, 1998.

[43] D. Whitley, T. Starkweather, C. Bogart: Genetic algorithms and neural networks: optimizing connections and connectivity, Parallel Computing 14(3), 1990.

[44] H. Kitano: Designing neural networks by genetic algorithms using graph generation systems, Complex Systems, 1990.

[45] D.J. Chalmers: The evolution of learning: an experiment in genetic connectionism, v D.S. Touretzky in sod.(ur.): Connectionist Models: Proc. Of the 1990 Summer School, Morgan Kaufmann, 1990.

[46] C.L. Karr: Genetic algorithms for fuzzy controllers, AI Expert, 1991.

[47] P. Thrift: Fuzzy logic synthesis with genetic algorithms, v R.K. Below, L.B. Booker (ur.): Proc. of 4-th Int. Conf. On Genetic Algorithms, 1991.

[48] M. Lee, H. Takagi: Dynamic control of genetic algorithms using fuzzy logic techniques, v S. Forrest (ur.): Proceedings of the 5'th Int. Conf. on Genetic Alg., 76-83, Morgan Kaufmann, 1993

[49] W. Pedrycz: Fuzzy Control and Fuzzy Systems, J. Wiley & Sons, 1993.

[50] H. Pedrycz, I. Hayashi, N. Wakami: A learning method of fuzzy inference rules by descent method, v Proc. of the IEEE Int. Conf. on Fuzzy Systems, 1992.

[51] H. Bersini, J.P. Nordvik, A. Bonarini: A simple direct adaptive fuzzy controller derived from its neural equivalent, v Proc. of the IEEE Int. Conf. on Fuzzy Systems, 1993.

[52] H.M. Voigt: Soft genetic operators in evolutionary algorithms, v W. Banzhaf, F.H. Eeckman (ur.): Evolution and Biocomputation, Springer-Verlag, 1995.

Stvarno kazalo