Poglavlje 13

Oblikovanje i učenje neuronske mreže algoritmima evolucijskog računanja

Kroz prethodna poglavlja upoznali smo se s pojmom umjetne neuronske mreže te smo naveli dvije tipične arhitekture – unaprijedne slojevite neuronske mreže te samoorganizirajuće neuronske mreže. Od mnoštva različitih arhitektura mreža izabrali smo ta dva kako bismo ilustrirali mogućnosti neuronskih mreža. Prisjetimo se, unaprijedne slojevite neuronske mreže mogli smo primijeniti na probleme funkcijske regresije i aproksimacije kao i klasifikacije. S druge pak strane, samoorganizirajuća neuronska mreža, poput Kohonenovog SOM-a, omogućavala je rješavanje zadataka poput grupiranja podataka.

Ako se fokusiramo na samo jedan od tih tipova, primjerice, unaprijedne slojevite mreže, ili još općenitije, unaprijedne mreže, postavlja se sljedeće pitanje: za zadatak koji mreža treba riješiti, koja je optimalna arhitektura neuronske mreže te koji su optimalni parametri takve mreže a uz koje neuronska mreža dobro rješava zadani problem, i pri tome zadržava dobro svojstvo generalizacije? U današnjem trenutku egzaktan odgovor na to pitanje ne postoji i odabir i učenje neuronske mreže svodi se na niz pokušaja i pogrešaka. Stoga je upravo prirodno razmotriti mogućnost kombiniranja evolucijskog procesa i razvoja neuronskih mreža.

Istraživanja na temu uporabe evolucijskih procesa za razvoj neuronskih mreža traju već preko dvadesetak godina, pri čemu su intenzivnija istraživanja započela ranih devedesetih. Primjena evolucijskih procesa na razvoj neuronskih mreža može se kategorizirati u tri područja.

1. *Evolucija težinskih faktora.* Klasični algoritmi učenja izvedeni su za pojedine arhitekture neuronskih mreža; primjerice, samoorganizirajuće neuronske mreže ne možemo učiti algoritmom *Backpropagation*. Evolucijski algoritmi kao univerzalni optimizacijski algoritmi primjenjivi

su na evoluciju parametara praktički neograničenog broja različitih arhitektura neuronskih mreža čime se izbjegava potreba za izvođenjem algoritama prilagođenih pojedinim arhitekturama.

2. *Evolucija arhitektura.* Poznato je da će prejednostavna neuronska mreža koju se primijeni na rješavanje nekog zadatka pronaći neprikladno rješenje – rješenje koje ima veliku pogrešku. S druge pak strane, preveliku neuronsku mrežu teško ćemo naučiti da kvalitetno rješava zadatak a nakon postupka učenja mreža će ispoljavati svojstvo pretreniranosti i vrlo loše generalizirati. Evolucijske algoritme moguće je stoga primijeniti na razvoj arhitekture neuronske mreže koja će biti jednostavna, a opet dovoljno složena kako bi rješavala zadani problem i zadržala svojstvo generalizacije.

3. *Evolucija pravila učenja.* Proučavanjem unaprijedne slojevite neuronske mreže u poglavlju 8 izveli smo algoritam *Backpropagation*, odnosno pokazali na koji se način računa korekcija svakog težinskog faktora. Međutim, izraz koji smo dobili nije jedini koji se može koristiti za postupak učenja. Evolucijske algoritme stoga je moguće primijeniti upravo na razvoj pravila učenja uz koje će neuronska mreža učiti brzo i efikasno. Uprabom evolucijskih algoritama pokušava se naučiti kako treba učiti.

13.1 Evolucija težinskih faktora

Unaprijedne neuronske mreže najčešće se koriste tako da se fiksira arhitektura mreže (broj ulaznih, izlaznih te skrivenih neurona; između koji neurona postoje veze te koje se prijenosne funkcije koriste) te se temeljem zadanog skupa uzoraka za učenje algoritmom učenja pokušava pronaći optimalan skup težinskih faktora uz koje mreža radi minimalnu pogrešku (sve drugo je fiksirano). Ovdje govorimo o učenju s učiteljem, a najpoznatiji algoritam koji se koristi za postupak učenja je *Backpropagation* [Rumelhart et al.(1986)Rumelhart, Hinton, and Williams] koji se zasniva na uporabi gradijenta pogreške. Osim tog algoritma i niza njegovih modifikacija, postoji još niz drugih koji se koriste, poput algoritma QuickProp [Fahlman(1989), Craig Veitch and Holmes(1991)], algoritma konjugiranog gradijenta [Moller(1990)] te Levenberg-Marquardt algoritma [Hagan and Menhaj(1994), Suratgar et al.(2005)Suratgar, Tavakoli, and Hoseinabadi].

Zajednička karakteristika svih ovih algoritama je njihova utemeljenost na gradijentu – takvi algoritmi često zapinju u lokalnim optimumima; algoritmi koji se temelje samo na informaciji o gradijentu također su vrlo neefikasni odnosno zahtijevaju velik broj iteracija. Dodatno, neprimjenjivi su na probleme minimizacije funkcije pogreške koja nije derivabilna.

Evolucijski algoritmi, primjerice genetski algoritam stohastički su optimizacijski algoritmi koji su poprilično robusni na lokalne optimume te ne zahtijevaju da funkcija koja se optimira bude derivabilna. Stoga je jedan od načina zaobilaženja problema koje sa sobom donosi algoritam *Backpropagation* uporaba evolucijskog algoritma primijenjenom na pronalazak optimalnog skupa težinskih faktora u fiksiranoj mreži. U literaturi se može pronaći dosta radova na temu usporedbe algoritma *Backpropagation* i *genetskog algoritma* kojima je trenirana ista mreža i rezultati nisu jasni – dio literature pokazuje da je algoritam *Backpropagation* efikasniji od genetskog algoritma, drugi dio literature pokazuje upravo suprotno. Međutim, treba napomenuti da su usporedbe uvijek rađene nad različitim problemima te između različitih vrsta algoritma *Backpropagation* i genetskih algoritama, što u konačnici samo potvrđuje *No-free-lunch* teorem: različiti algoritmi nad različitim problemima ponašaju se različito.

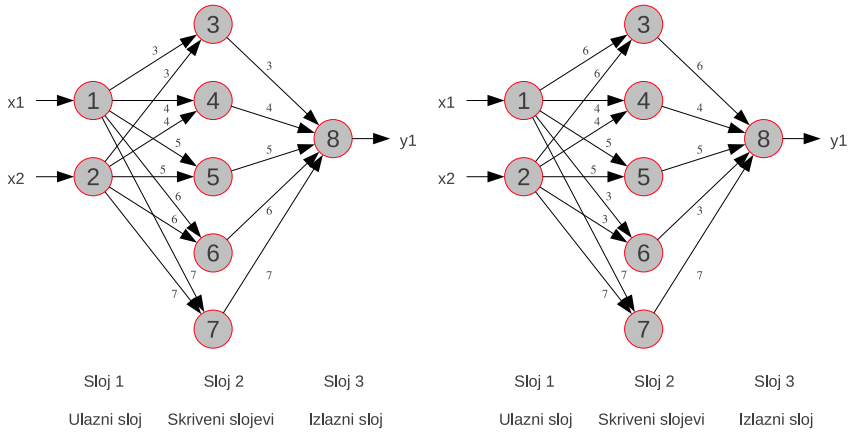
13.1.1 Primjena genetskog algoritma na evoluciju težinskih faktora

U fiksnoj arhitekturi neuronske mreže broj težinskih faktora koje treba naučiti je fiksan i ne mijenja se tijekom postupka učenja. Stoga se u praksi koriste dvije reprezentacije rješenja s kojima radi genetski algoritam: binarni prikaz te prikaz vektorom decimalnih brojeva.

Kod binarnog prikaza sve se težine numeriraju (primjerice od 0 do $n-1$), za svaku se težinu uzme k bitova i potom se izgradi kromosom od $n \cdot k$ bitova. Da bi se omogućilo dekodiranje težina, potrebno je još utvrditi minimalni i maksimalni iznos težine te kod koji se koristi (primjerice, binarni kod, Grayev kod i slično). Prednost ovakvog prikaza je izostanak potrebe za definiranjem posebnih operatora križanja i mutacije. Jedan od nedostataka ovog prikaza je ograničen prostor pretraživanja – naime, za težine je unaprije potrebno zadati raspon unutar kojeg se obavlja pretraga. Prisjetimo se, kod algoritma *Backpropagation* to nije bio slučaj – težine smo tretirali kao realne brojeve i nismo (svjesno) postavljali nikakva ograničenja jer često ne znamo unutar kojeg raspona će se težine kretati.

Prikaz s vektorom decimalnih brojeva u ovom slučaju je nešto prirodniji iako sa sobom donosi niz problema: više ne možemo koristiti uobičajene binarne operatore već je potrebno razviti skup operatora koji rade nad decimalnim brojevima.

Relativno opsežno istraživanje na temu uporabe genetskog algoritma za treniranje neuronske mreže prikazano je u [Montana and Davis(1989)], gdje autori definiraju niz različitih operatora mutacije i križanja od kojih su neki prikazani u nastavku. Međutim, prije detaljnijeg pregleda tih operatora potrebno je spomenuti i problem koji se javlja prilikom učenja neuronskih mreža populacijskim algoritmima – radi se o problemu permutacija kod neuronskih mreža (engl. *Permutation problem*, također poznat i pod nazivom



(a) Originalna mreža (b) Drugačija mreža za isto preslikavanje

Slika 13.1: Ilustracija simetričnosti unutar neuronske mreže – problem permutacija

engl. *Competing conventions problem*) [Hancock(1992)]. Kako smo do sada govorili samo o algoritmu *Backpropagation* koji nije populacijski, nismo ga do sada spominjali. A radi se o sljedećem: neuronske mreže pravilnih struktura (primjerice troslojna unaprijedna potpuno-povezana neuronska mreža) izuzetno su simetrične, odnosno postoji mnoštvo načina raspodjele težina uz koje mreža obavlja identično preslikavanje. Da je tome tako, možemo se uvjeriti jednostavnim primjerom koji prikazuje slika 13.1.

Obje prikazane neuronske mreže su troslojne neuronske mreže s dva ulaza, pet skrivenih neurona i jednim izlazom. Svi neuroni su numerirani; neuroni 1 i 2 su ulazni i ništa ne računaju, neuroni 3 do 7 su skriveni i neuron 8 je izlazni. Mreža prikazana na slici 13.1b dobivena je direktno iz mreže prikazane na slici 13.1a tako da su sve težine na ulazima i izlazima neurona 3 i 6 zamijenjene. Uočimo, time neuron 6 sada računa ono što je u mreži prikazanoj na slici 13.1a računao neuron 3, a neuron 3 sada računa ono što je prije računao neuron 6. Kako su prilikom zamjene zamijenjene i težine kojima izračunati izlazi utječu na neuron 8, slijedi da obje mreže rade identično funkcijsko preslikavanje iz $\mathbb{R}^2 \to \mathbb{R}$.

Posljedica ove simetričnosti jest postojanje više globalnih optimuma funkcije pogreške koja se minimizira. U prikazanom primjeru na slici 13.1 u skrivenom sloju se nalazi 5 neurona; to znači težine u mreži možemo raspodijeliti na 5! različitih načina a da pri tome imamo identično ponašanje neuronske mreže. Već uz jedan skriveni sloj, broj globalnih optimuma jednak je dakle faktorijeli broja neurona u tom skrivenom sloju. Kako za iole veći broj neurona u tom skrivenom sloju broj rješenja raste vrlo brzo (a da ne spominjeno neuronske mreže s dva ili više skrivenih slojeva), jasno je da će

populacijski optimizacijski postupci koji međusobno kombiniraju dva ili više rješenja imati težak posao. Iz tog razloga dosta istraživača u ovom području pokušava izbjegavati takve operatore – kod genetskog algoritma to je upravo operator križanja, i koristiti samo operatore koji rade direktno nad jednim rješenjem (kod genetsko algoritma to bi bio operator mutacije).

Pogledajmo i na dva primjera zašto je križanje problematično u ovom slučaju. Pretpostavimo da su rješenja vektori decimalnih brojeva. Križajmo dvije mreže prikazane na slikama 13.1a i 13.1b najprije na način da za svaku težinu slučajno biramo hoćemo li je uzeti iz jednog djeteta ili iz drugog. Kako su težine kod svih neurona osim neurona 3 i 6 identične, dijete će ih naslijediti. Pretpostavimo sada da u djetetu neuron 3 dobije jednu težinu od jednog roditelja a drugu od drugog (dakle dobije težine 3 i 6) a to isto nastupi i kod težina neurona 6. Evo u čemu je problem: dotadašnji postupak evolucije neurone 3 i 6 u oba roditelja specijalizirao je za određeno područje ulaznog prostora i pronašao im prikladne težine; križanjem smo međutim dobili dva neurona od kojih niti jedan više nije niti sličan onome za što su polazni neuroni bili specijalizirani, i takva će mreža tipično raditi bitno lošije od roditelja; u tom smislu, operator križanja je naprosto previše destruktivan.

Pretpostavimo sada da smo izveli drugačije križanje: svjesni smo da u mreži postoji struktura pa operatorom mutacije mijenjamo težine, a operatorom križanja u dijete prenosimo kompletne neurone s njihovim vezama. Razmotrimo opet križanje dviju mreža prikazanih na slikama 13.1a i 13.1b. Neka dijete od prve mreže pokupi neurone 3, 4 i 5 a od druge mreže neurone 6 i 7. Tada će kod djeteta neuron 3 imati težine (3,3), neuron 4 težine (4,4), neuron 5 težine (5,5), neuron 6 težine (3,3) i neuron 7 težine (7,7). Uočimo sada: neuron koji je bio specijaliziran za ulazni dio prostora pokriven težinama (6,6) je kod djeteta izgubljen, a umjesto njega, dijete je dobilo dvije kopije neurona koji ima težine (3,3). Takvo dijete opet će vrlo vjerojatno biti lošije od roditelja pa križanje niti na koji način neće pomoći u evoluciji rješenja.

Opisani problem kod manjih mreža ne dolazi toliko do izražaja kao kod onih većih. Dapače, dio istraživača pokazao je da genetski algoritam u određenim slučajevima može biti dovoljno robustan da se nosi s ovim problemima i da opet pronalazi zadovoljavajuća rješenja. Međutim, ukazani problem jasno ukazuje i na potrebu da se, ako se već koriste operatori koji kombiniraju više rješenja, izvedbi takvih operatora posveti posebna pažnja.

U nastavku ćemo dati još i prikaz različitih izvedbi operatora mutacije i križanja za genetski algoritam koji traži optimalne težinske faktore neuronske mreže, kako su definirani u [Montana and Davis(1989)]. Operatori su izvedeni uz pretpostavku da se za prikaz rješenja koristi vektor decimalnih brojeva. Krenimo najprije s operatorima mutacije.

- *Mutacija bez pristranosti.* Za svaki element vektora operator s vjerojat-

nošću p_m mijenja tu vrijednost nekom drugom nasumično generiranom vrijednosti.

- *Mutacija s pristranošću.* Za svaki element vektora operator s vjerojatnošću p_m toj vrijednosti nadodaje neku drugu nasumično generiranu vrijednost.

- *Mutacija neurona.* Operator odabire n neurona koji nisu ulazni. Potom svim dolaznim težinama odabranih neurona nadodaje neku nasumično generiranu vrijednost. Ovim se promjene lokaliziraju na razini pojedinih neurona.

- *Mutacija najslabijeg neurona.* Jakost neurona definira se kao razlika između izlaza neuronske mreže u kojoj promatrani neuron normalno funkcionira i izlaza lobotomizirane neuronske mreže u kojoj je izlaz promatranog neurona pritegnut na vrijednost 0. Neuron koji je najslabiji očito nije specijaliziran niti za jedno korisno područje i ne obavlja važnu ulogu u mreži – stoga se takav neuron mutacijom pokušava aktivirati.

Od operatora križanja, spomenut ćemo 3 opisana u nastavku.

- *Križanje težina.* Vektor težina djeteta tvori se tako da se svaka komponenta vektora nasumično preuzme od jednog ili drugog roditelja.

- *Križanje neurona.* Za svaki se neuron koji nije ulazni nasumično odabere jedan od roditelja i od njega se preuzme kompletan skup ulaznih težina tog neurona; naime, kako je pretpostavka da je svaki neuron u određenom smislu specijaliziran za određeno područje, u dijete se prenosi kompletan skup težina za neuron od izabranog roditelja čime se ta specijalizacija prenosi u dijete.

- *Križanje značajki.* Za svaki neuron u prvom roditelju pokušava se pronaći neuron koji obavlja istu ili najsličniju ulogu tog neurona u drugom roditelju. To se radi tako da se na ulaze obje mreže dovodi niz različitih ulaza i potom se promatra odziv fiksiranog neurona iz prve mreže i kandidata neurona iz druge mreže. Jednom kada se za svaki neuron iz prvog roditelja pronađe njegov par, poredak neurona u drugom roditelju se presloži tako da mreža i dalje obavlja istu funkciju ali da poredak neurona u obje mreže bude jednak u funkcijskom smislu. Tada se roditelji križaju na prethodno opisan način ("križaj neurone").

Za detalje se zainteresirane čitatelje upućuje na rad [Montana and Davis(1989)]. Spomenimo i da se u određenim slučajevima rad genetskog algoritma može dosta ubrzati ugradnjom lokalne pretrage. Lokalna pretraga je postupak koji se pokreće nakon generiranja djece (kada, nad kojoj točno djecom i koliko dugo dodatni su parametri koje tada treba podesiti) a čiji je cilj

pokušati popraviti stvoreno rješenje. Kao postupak lokalne pretrage moguće je implementirati upravo algoritam *Backpropagation* (pa odraditi nekoliko iteracija) ili neki drugi postupak. Algoritam *Backpropagation* izuzetno je osjetljiv na inicijalni odabir težina pa je česta situacija u praksi da se algoritam mora pokrenuti više puta kako bi pronašao neko dovoljno dobro rješenje, a ne zapeo u neprihvatljivo lošem lokalnom optimumu. Genetski algoritam je pak algoritam koji dosta grubo pretražuje prostor i koji je upravo sposoban pronaći dobar skup težina od kojih algoritam lokalne pretrage može dalje generirati dobra rješenja. Stoga ovakva kombinacija u praksi može davati dobre rezultate.

13.2 Evolucija arhitektura

Kako bi neuronska mreža zadani problem rješavala optimalno dobro, potrebno je prije postupka učenja težina odabrati mrežu prikladne složenosti. Naime, prejednostavna mreža neće biti sposobna zadatak rješavati uz malu pogrešku. S druge strane, kod presložene mreža postupak učenja će trajati vrlo dugo a dobivena mreža, iako će raditi malu pogrešku na skupu za učenje, generalno govoreći iskazivat će svojstvo vrlo slabe generalizacije. Pitanje je, stoga, kako odabrati optimalnu arhitekturu koja je prikladna za rješavanje postavljenog zadatka. Ovaj "meta-zadatak" – pronalazak optimalne arhitekture mreže – može se definirati kao zaseban optimizacijski problem koji je potrebno riješiti. Kriterijska funkcija koja se pri tome definira može uključivati faktore poput minimalne složenosti mreže, brzine učenja mreže i slično. Dva su uobičajena načina rješavanja tog problema: *konstruktivnim algoritmima* te *destruktivnim algoritmima*. Konstruktivni algoritmi počinju s neuronskom mrežom minimalne složenosti i po potrebi dodaju nove neurone, nove veze i nove slojeve kako bi mrežu učinili prikladnijom za rješavanje zadanog problema. Destruktivni algoritmi kreću od vrlo općenite i složene arhitekture mreže i potom pokušavaju uklanjati veze, neurone i slojeve pazeći da pri tome mreža i dalje dobro rješava zadani problem. Međutim, i kod jedih i kod drugih vrsta algoritama nema jasnog pravila (i opravdanja) na koji način treba točno raditi već su takvi algoritmi vođeni heurističkim pravilima.

Umjesto uporabe konstruktivnih i destruktivnih algoritama, problem pronalaska optimalne arhitekture može se rješavati evolucijskim algoritmima koji su prema [Miller et al.(1989)Miller, Todd, and Hedge] posebno pogodni za ovaj zadatak iz nekoliko razloga.

- Površina koju razapinje kriterijska funkcija a koju pretražuje optimizacijski algoritam beskonačno je velika jer broj mogućih arhitektura nije ograničen.

- Površina koju razapinje kriterijska funkcija nije derivabilna jer postoji

niz promjena arhitektura koje su diskretne (primjerice, dodavanje ili uklanjanje jedne veze, jednog neurona ili jednog sloja neurona ili pak promjena prijenosne funkcije u neuronu) .

- Površina koju razapinje kriterijska funkcija je kompleksna i sadrži veliku količinu šuma jer nemamo način egzaktne procjene kvalitete pojedine arhitekture već se ta ocjena računa indirektno – primjerice, mjereći vrijeme potrebno da mreža nauči, što je izuzetno podložno utjecaju slučajnosti.

- Površina koju razapinje kriterijska funkcija je decepcijska, budući da slične arhitekture mogu imati bitno različite iznose kriterijske funkcije.

- Površina koju razapinje kriterijska funkcija je višemodalna, budući da različite arhitekture mogu imati slične iznose kriterijske funkcije.

Prilikom dizajniranja evolucijskog algoritma prikladnog za rješavanje problema pronalaska optimalne arhitekture potrebno je odlučiti se *koliko će informacija biti kodirano u kromosom* s kojim evolucijski algoritam radi. Postoje dvije mogućnosti: svaki kromosom može do u posljednji detalj specificirati sve detalje neuronske mreže – sve neurone, sve veze među njima, iznose težina, odabrane prijenosne funkcije i slično. U tom slučaju govorimo o *direktnom kodiranju neuronske mreže*. Alternativa je u kromosom pohraniti samo najvažnije informacije, primjerice, koliko slojeva mreža treba imati i koliko neurona u svakom sloju (ako razvijamo slojevite mreže). U tom slučaju govorimo o *indirektnom kodiranju*.

Jednom kada je odabran način reprezentacije neuronske mreže, razvoj arhitektura može se obavljati prema pseudokodu prikazanom u nastavku.

1. Svaki kromosom dekodiraj u pripadnu neuronsku mrežu. Ako se koristi indirektno kodiranje, parametre koji nedostaju postavi na neki način (primjerice, dodatnim pravilom koje se koristi ili naprosto postupkom učenja težina).

2. Svaku neuronsku mrežu treniraj puno puta, svaki puta počevši od različitih početnih težina, različitih parametara u pravilu učenja (npr. ako se koristi *Backpropagation*, pokušavati s više različitih stopa učenja i faktora inercije).

3. Svakoj mreži (kromosomu) dodijeli pripadni iznos funkcije dobrote koji je izračunat temeljem obavljenih pokušaja učenja iz koraka 2 i u skladu s definiranom kriterijskom funkcijom.

4. Obavi križanje roditelja i mutaciju djece u skladu s odabranom vrstom evolucijskog algoritma a sukladno dodijeljenim iznosima funkcije dobrote te primijeni i eventualno neke druge operatore čime se u konačnici generira sljedeća generacija roditelja.

U nastavku ćemo se još osvrnuti na načine kodiranja neuronskih mreža.

Tablica 13.1: Prikaz arhitekture neuronske mreže matricom veza

	1	**2**	**3**	**4**
1	$c_{1,1}$	$c_{1,2}$	$c_{1,3}$	$c_{1,4}$
2	$c_{2,1}$	$c_{2,2}$	$c_{2,3}$	$c_{2,4}$
3	$c_{3,1}$	$c_{3,2}$	$c_{3,3}$	$c_{3,4}$
4	$c_{4,1}$	$c_{4,2}$	$c_{4,3}$	$c_{4,4}$

13.2.1 Direktno kodiranje neuronske mreže

Najjednostavniji primjer direktnog kodiranja neuronske mreže koja sadrži N neurona jest matricom dimenzija $N \times N$. Primjer takvog zapisa za neuronsku mrežu koja se sastoji od ukupno 4 neurona dan je u tablici 13.1. Koeficijenti $c_{i,j} \in \{0, 1\}$ određuju postoji li veza između izlaza neurona i i ulaza neurona j (tj. da li neuron i pobuđuje neuron j); ako je $c_{i,j} = 1$ tada neuron i pobuđuje neuron j; u suprotnom neuron i ne pobuđuje neuron j. Ako ovakva mreža obavlja preslikavanje iz $\mathbb{R}^m \to \mathbb{R}^n$ (tj. ima m ulaza i n izlaza, možemo se pridržavati konvencije da su prvih m neurona ulazni (neuroni 1, 2, ..., m) a posljednjih n neurona izlazni (neuroni $4 - n + 1$, $4 - n + 2$, ..., 4).

Umjesto restrikcije elemenata matrice na binarne zastavice, matrični prikaz možemo koristiti i direktno za pohranu težina (vidi tablicu 13.2). U tom slučaju elemente matrice ćemo označavati s $w_{i,j}$ i držat ćemo se sljedeće konvencije: ako je $w_{i,j} = 0$ tada ne postoji veza kojom neuron i pobuđuje neuron j; u suprotnom, neuron i pobuđuje neuron j i pri tome je iznos težinskog faktora te veze upravo $w_{i,j}$. Primjetimo još da svaki neuron u neuronskoj mreži osim težinskih faktora koji se nalaze na njegovim ulazima i skaliraju pobude koje neuron dobiva od drugih neurona neuron sadrži i slobodnu težinu koju smo obično označavali s w_0. Kako bismo mogli pamtiti i ove težine, matrični prikaz koji je dan u tablici 13.2 uvodi virtualni neuron indeksiran s 0 koji nema nikakvih ulaza i na svojem izlazu generira fiksnu jedinicu. Tada za sve ostale neurone vrijednost slobodnog težinskog faktora možemo zapisati kao težinu između tog virtualnog neurona i svakog od preostalih neurona (prvi redak tablice). Primijetimo također da, ako se radi o unaprijednim mrežama, pripadna matrica ne može biti skroz popunjena. Primjerice, ako je neuron 1 ulazni neuron, tada njega ne pobuđuje niti jedan drugi neuron što znači da $\forall i \; w_{i,1} = 0$. Naš virtualni neuron također nitko ne pobuđuje, pa mora vrijediti $\forall i \; w_{i,0} = 0$. Ako je neuron 4 izlazni, tada on nikoga ne pobuđuje, a to znači da mora vrijediti $\forall i \; w_{4,i} = 0$. I konačno, da bi mreža bila unaprijedna, dozvoljeno je postojanje veza od neurona i prema neuronu j ako je $i < j$ ali ne i obratno. Naime, ako se dozvoli da neuron i pobuđuje neuron j i da neuron j pobuđuje neuron i, tada izlaz svakog od neurona ovisi o izlazu upravo onog drugog, pa više nemamo unaprijednu mrežu već smo dobili mrežu koja ima vremenski promjenjiv odziv (sjetite se primjerice mreže *Winner-Takes-All* koja je upravo takva). Iz istog razloga,

Tablica 13.2: Prikaz arhitekture neuronske mreže matricom težina

	0	**1**	**2**	**3**	**4**
0	$w_{0,0}$	$w_{0,1}$	$w_{0,2}$	$w_{0,3}$	$w_{0,4}$
1	$w_{1,0}$	$w_{1,1}$	$w_{1,2}$	$w_{1,3}$	$w_{1,4}$
2	$w_{2,0}$	$w_{2,1}$	$w_{2,2}$	$w_{2,3}$	$w_{2,4}$
3	$w_{3,0}$	$w_{3,1}$	$w_{3,2}$	$w_{3,3}$	$w_{3,4}$
4	$w_{4,0}$	$w_{4,1}$	$w_{4,2}$	$w_{4,3}$	$w_{4,4}$

mora vrijediti $\forall i \; w_{i,i} = 0$. Međutim, uočimo da ova ograničenja postoje ako želimo dobiti unaprijednu mrežu; ako to nije slučaj, matrični je prikaz dovoljno ekspresivan za dobivanje bilo kakve arhitekture.

Genetski algoritam za rješavanje problema XOR uz direktno kodiranje

Problem XOR definirat ćemo kao problem pronalaska neuronske mreže koja korektno radi na sljedećem skupu uzoraka za učenje: {(-1,-1)→-1, (-1,1)→1, (1,-1)→1, (1,1)→-1}.

Kako bismo ilustrirali genetski algoritam koji koristi direktno kodiranje, napisana je implementacija troturnirskog genetskog algoritma koji trenira neuronsku mrežu koja se sastoji od neurona kod kojih se kao prijenosna funkcija koristi funkcija skoka oblika:

$$f(net) = \begin{cases} 1, & net > 0 \\ -1, & \text{inače.} \end{cases}$$

Pokušaj 1 Kako je poznato da neuronska mreža koja se sastoji od jednog neurona (ne računajući ulazne neurone) ne može obaviti taj posao, genetskim algoritmom pokušali smo pronaći slojevitu potpuno-povezanu troslojnu neuronsku mrežu koja u skrivenom sloju ima dva neurona i koja rješava taj problem, tj. mrežu oblika $2 \times 2 \times 1$. Pripadna matrica težina prikazana je u tablici 13.3. S obzirom na postavljeni zahtjev slojevitosti, neuronska mreža ima relativno malo težina koje se mogu postavljati. Neuron 0 je virtualni neuron koji na izlazu generira fiksnu vrijednost 1 i služi za podešavanje slobodnih težinskih faktora, neuroni 1 i 2 su ulazni neuroni koji na svoj izlaz preslikavaju dovedeni uzorak, odnosno komponente x_1 i x_2, neuroni 3 i 4 su neuroni skrivenog sloja i prvi koji doista nešto računaju i neuron 5 je izlazni neuron.

Operatori križanja izveden je tako da dijete svaki element matrice s 50% šanse preuzme od jednog roditelja odnosno s 50% šanse od drugog roditelja. Mutacija je izvedena na način da se svakom elementu matrice s vjerojatnošću p_m nadoda slučajno generirani broj iz normalne distribucije s parametrima

Tablica 13.3: Matrični prikaz arhitekture neuronske mreže za problem XOR

	0	**1**	**2**	**3**	**4**	**5**
0	0.0	0.0	0.0	$w_{0,3}$	$w_{0,4}$	$w_{0,5}$
1	0.0	0.0	0.0	$w_{1,3}$	$w_{1,4}$	0.0
2	0.0	0.0	0.0	$w_{2,3}$	$w_{2,4}$	0.0
3	0.0	0.0	0.0	0.0	0.0	$w_{3,5}$
4	0.0	0.0	0.0	0.0	0.0	$w_{4,5}$
5	0.0	0.0	0.0	0.0	0.0	0.0

$(0, \sigma)$. Nakon križanja i mutacije, nad matricom je pokrenut dodatni postupak koji je sve težine za koje je $c_{i,j} = 0$ resetirao ponovno na 0 kako bi se osiguralo da je mreža i dalje unaprijedna slojevita.

Konkretno, genetski algoritam je pokrenut uz sljedeće parametre:

- veličina populacije: 50,
- $p_m = 0.33$ te
- $\sigma = 0.25$.

Inicijalno, težine su birane uniformno iz intervala $[-5,5]$. Algoritam je pronašao rješenje koje problem rješava ispravno već u 4. generaciji nakon čega je ostatak vremena proveo popravljajući njegovu kvalitetu s obzirom na zadanu kriterijsku funkciju. Kriterijska funkcija pri tome je definirana kao negativan iznos funkcije kazne. Funkcija kazne računana je na sljedeći način:

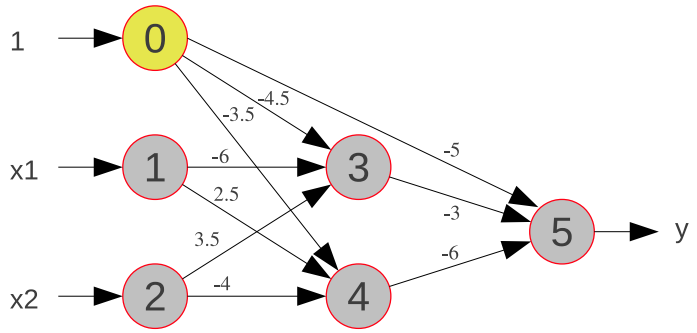
- Označimo s K_1 iznos sume kvadratnog odstupanja izračunatog na čitavom skupu za učenje. U našem slučaju minimalni iznos ove komponente je 0 a maksimalni $4 \cdot 2^2 = 16$, koji se postiže ako mreža sva 4 uzorka pogrešno klasificira.
- Označimo s K_2 iznos sume kazni zbog neprikladnog oblika težina. Naime, htjeli bismo da sve težine budu ili cijeli brojevi, ili decimalni brojevi oblika $x.5$. Komponenta K_2 je suma apsolutnog odstupanja svakog elementa matrice od željenog oblika težine. Primjerice, ako je neka težina jednaka 3.4, najbliži "poželjni" broj je 3.5 pa će za tu težinu u K_2 biti nadodan iznos $|3.4 - 3.5| = 0.1$.
- Označimo s K_3 iznos sume kazni zbog veličine težina. Naime, htjeli bismo postići da je što više težina jednako 0 ili da su vrlo male. Stoga komponentu K_3 računamo kao sumu kvadata svih težina u matrici.
- Ukupna kazna tada se računa kao:

$$K_1 + \frac{1}{2}\phi(K_2) + \frac{1}{2}\phi(K_3)$$

gdje je funkcija $\phi(x) = \frac{2}{\pi} \cdot arctan(x)$.

Tablica 13.4: Matrični prikaz arhitekture neuronske mreže za problem XOR

	0	**1**	**2**	**3**	**4**	**5**
0	0.0	0.0	0.0	-4.5	-3.5	-5.0
1	0.0	0.0	0.0	-6.0	2.5	0.0
2	0.0	0.0	0.0	3.5	-4.0	0.0
3	0.0	0.0	0.0	0.0	0.0	-3.0
4	0.0	0.0	0.0	0.0	0.0	-6.0
5	0.0	0.0	0.0	0.0	0.0	0.0



Slika 13.2: MLP s dva neurona u skrivenom sloju za problem XOR-a naučen genetskim algoritmom

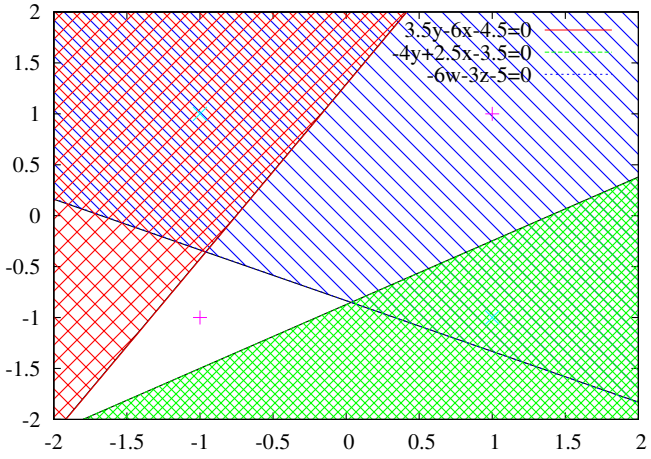
Razvijeni genetski algoritam prilikom rada pokušava maksimizirati dobrotu generiranih neuronskih mreža odnosno pronaći mrežu koja ima minimalni iznos funkcije kazne. Dobiveni rezultat prikazan je u tablici 13.4 a dekodirana neuronska mreža prikazana je na slci 13.2.

Interesantno je pogledati na koji je način genetski algoritam podesio težine, odnosno što točno pronađena neuronska mreža radi. Za potrebe pojašnjenja, uvedimo oznaku z za izlaz neurona 3 te oznaku w za izlaz neurona 4. Uz pronađeni skup težina vrijedi:

$$z = step(-6x_1 + 3.5x_2 - 4.5)$$

$$w = step(2.5x_1 - 4x_2 - 3.5).$$

Decizijske funkcije koje se dobiju za $z = 0$ (crvena linija) i $w = 0$ (zelena linija) prikazane su na slici 13.3. Dodatno, čitav podprostor u kojem je *net* koji računa neuron 3 pozitivan (pa stoga izlaz $z = 1$) obojan je crvenim uzorkom dok je čitav podprostor u kojem je *net* koji računa neuron 4 pozitivan (pa stoga izlaz $w = 1$) obojan je zelenim uzorkom. Sada vidimo da je neuron 3 specijaliziran za detekciju uzorka $(x_1, x_2) = (-1, 1)$ i samo za njega daje na izlazu vrijednost 1 a neuron 4 je specijaliziran za detekciju uzorka $(x_1, x_2) = (1, -1)$ i samo za njega daje na izlazu vrijednost 1. Čitava neuronska mreža sada treba na izlazu dati -1 samo ako nije prepoznat



Slika 13.3: Decizijske funkcije naučene za svaki od tri neurona u MLP-u za problem XOR-a

uzorak (-1,1) niti uzorak (1,-1); drugim riječima, samo kada su oba izlaza w i z jednaka -1; ako je prepoznat bilo koji od navedena dva uzorka, mreža treba dati izlaz 1. No taj je posao rješiv jednim neuronom – onim izlaznim. Naime, taj neuron treba obaviti sljedeće preslikavanje:{(w,z)$\rightarrow \phi(w,z)$}: {(-1,-1)$\rightarrow$-1, (-1,1)$\rightarrow$1, (1,-1)$\rightarrow$1, (1,1)$\rightarrow$1} što je najobičnija funkcija ILI i ona je linearno razdvojiva. Taj posao obavlja upravo neuron 5 čija je decizijska funkcija:

$$y = step(-3z - 6w - 5)$$

koja je na slici 13.3 prikazana plavom linijom a područje za koje pripadni *net* poprima pozitivnu vrijednost (pa je tada $y = 1$) obojano je plavim uzorkom.

Na koji je dakle način genetski algoritam podesio težine neuronske mreže? Neurone skrivenog sloja specijalizirao je tako da iz prostora uzoraka detektiraju određene zanimljive uzorke (ili općenito kombinacije uzoraka); potom je iskoristio sljedeći sloj neurona (kod nas i posljednji, izlazni) kako bi temeljem aktivacija tih detektora (a ne više direktnim promatranjem ulaznih uzoraka) generirao konačnu klasifikaciju. Sada je jasno zašto neuronske mreže s više slojeva i malim brojem neurona mogu naučiti obavljati komplicirane klasifikacije: svaki sljedeći sloj neurona postaje detektor koji svoj izlaz gradi temeljem prethodnog sloja detektora i time gradi sve kompliciranije decizijske funkcije.

Pokušaj 2 Umjesto troslojne potpuno povezane neuronske mreže tipa $2 \times 2 \times 1$, genetskim algoritmom smo pokušali naučiti jednostavniju troslojnu potpuno-povezanu neuronsku mrežu: $2 \times 1 \times 1$. Međutim, postupak učenja nije mogao pronaći niti jedan skup težina uz koje bi mreža korektno klasificirala sve uzorke – razmislite je li to krivica algoritma učenja? Potom

Tablica 13.5: Matrični prikaz arhitekture neuronske mreže s dva operativna neurona za problem XOR

	0	**1**	**2**	**3**	**4**
0	0.0	0.0	0.0	$w_{0,3}$	$w_{0,4}$
1	0.0	0.0	0.0	$w_{1,3}$	$w_{1,4}$
2	0.0	0.0	0.0	$w_{2,3}$	$w_{2,4}$
3	0.0	0.0	0.0	0.0	$w_{3,4}$
4	0.0	0.0	0.0	0.0	0.0

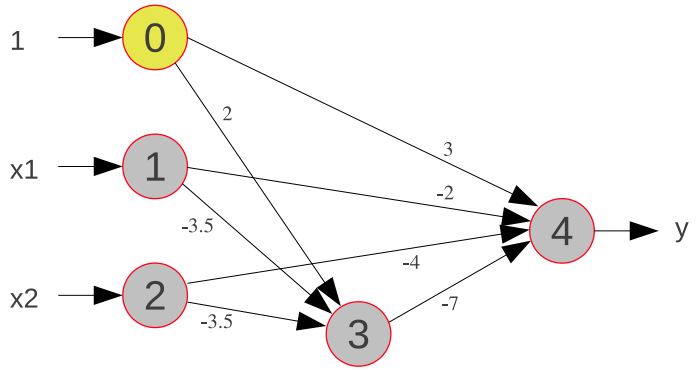
Tablica 13.6: Matrični prikaz arhitekture neuronske mreže za problem XOR

	0	**1**	**2**	**3**	**4**
0	0.0	0.0	0.0	2.0	3.0
1	0.0	0.0	0.0	-3.5	-2.0
2	0.0	0.0	0.0	-3.5	-4.0
3	0.0	0.0	0.0	0.0	-7.0
4	0.0	0.0	0.0	0.0	0.0

smo odustali od zahtjeva da mreža bude slojevita, i pokušali naučiti neuronsku mrežu koja je unaprijedna i koja ima dva ulaza te još dva neurona na raspolaganju (od koji je jedan ujedno i izlazni). Opći oblik matrice koja predstavlja ovakvu mrežu prikazan je u tablici 13.5.

Genetski algoritam pokrenuli smo s istim skupom parametara; jedina je razlika u dozvoljenom obliku matrice koja se evoluira. I opet, genetski algoritam je uspješno pronašao rješenje. Evoluirana matrica te dekodirana neuronska mreža prikazane su u tablici 13.6 i na slici 13.4.

Prokomentirajmo i ovo rješenje. Kako sada imamo nedovoljno neurona kako bismo direktno razvili detektore za pojedine dijelove ulaznog prostora



Slika 13.4: Unaprijedna (ali neslojevita) neuronska mreža s ukupno 2 operativna neurona za problem XOR-a naučen genetskim algoritmom

pa klasifikaciju radili temeljem njih, genetski algoritam je evoluirao drugačiju strategiju rješavanja problema: umjesto da se problem pokuša razriješiti direktno u dvodimenzijskom ulaznom prostoru gdje problem nije linearno razdvojiv, evolucijski proces je najprije napravio preslikavanje iz dvodimenzijskog ulaznog prostora u trodimenzijski prostor i potom klasifikaciju pokušao napraviti u tom trodimenzijskom prostoru. Kako to vidimo? Izlazni neuron dobiva kao ulaze x_1, x_2 i z; stoga on pokušava napraviti klasifikaciju u trodimenzijskom prostoru. Spomenuto preslikavanje radi se tako da se dvije komponente iz ulaznog prostora direktno proslijede kao dvije komponente za trodimenzijski prostor dok se treća komponenta računa uporabom za to predviđenog neurona 3. Ako izlaz tog neurona 3 označimo sa z, tada uzorke $(x_1,x_2) \in \mathbb{R}^2$ preslikavamo u uzorke $(x_1,x_2,z) \in \mathbb{R}^3$. Uz težine prikazane na slici 13.4, izlazni neuron pokušava naučiti sljedeće preslikavanje: {(-1,-1,1)→-1, (-1,1,1)→1, (1,-1,1)→1, (1,1,-1)→-1}. Lako se je uvjeriti da je taj zadatak rješiv jer je ovako definirani ulazni skup linearno razdvojiv, i taj posao upravo uspješno odrađuje izlazni neuron.

Zaključak Direktno kodiranje neuronske mreže najjednostavniji je oblik zapisa neuronske mreže; činjenica da se u zapisu nalaze sve potrebne informacije koje u potpunosti određuju neuronsku mrežu bitno olakšavaju postupak evaluacije kvalitete razvijene arhitekture neuronske mreže. Međutim, ovakav zapis, nažalost, ne funkcionira dobro ako se pokušaju razvijati velike neuronske mreže jer broj elemenata matrice raste kvadratno s brojem korištenih neurona. Također, iako neka istraživanja pokazuju da operatori koji kombiniraju više neuronskih mreža prilikom stvaranja potomaka mogu pozitivno utjecati na evolucijski proces, činjenica je da je direktno kodiranje izravno podložno prethodno opisanome problemu permutacija. Indirektno kodiranje pokušava jednim dijelom riješiti upravo te probleme.

13.2.2 Indirektno kodiranje neuronske mreže

Za razliku od direktnog kodiranja, indirektno kodiranje u kromosom ne pohranjuje sve detalje o neuronskoj mreži već samo određen dio. Nakon što se izgradi mreža koja je u skladu s tim informacijama, sve što nedostaje, definira se nekim unaprijed zadanim pravilom ili se pak uči. Postoji više načina indirektnog kodiranja od kojih ćemo spomenuti samo neke.

Parametarski prikaz

Pretpostavimo da razvijamo unaprijednu slojevitu potpuno-povezanu neuronsku mrežu. Takva se mreža tada može opisati definiranjem broja neurona u svakom od slojeva; primjerice, radi li se o mreži tipa $2 \times 2 \times 3$ ili $2 \times 5 \times 3$ ili $2 \times 3 \times 4 \times 3$ i slično. Kod ovakvog prikaza u kromosom se upisuju samo informacije o broju slojeva te o broju neurona unutar svakog

sloja. Uočimo da se sada više ne pamte težine koje odgovaraju svakoj od veza između međusobno povezanih neurona. Operatori križanja i mutacije u ovom slučaju rade operacije nad ovako zapisanim parametrima.

Problem koji se sada javlja jest kako utvrditi dobrotu arhitekture koju predstavlja pojedini kromosom. Da bi to bilo moguće, potrebno je više puta uz različite početne vrijednosti težina obaviti postupak učenja takve neuronske mreže i temeljem performansi postupka učenja donijeti ocjenu kvalitete ove arhitekture. Ovakav pristup ocjenjivanju u sebi inherentno sadrži veću količinu šuma nego što je to slučaj kod direktnog kodiranja – moguće je da naprosto nismo imali sreće pa smo puno puta odabrali nepovoljne početne težine uz koje je postupak učenja bio dugotrajan i uz koje je zapeo u nepovoljnom lokalnom optimumu, no osim toga ta je arhitektura inače vrlo dobra za problem koji rješavamo.

Kod ovakvog prikaza osim parametara koji određuju mrežu moguće je istovremeno evoluirati i parametre koji su prisutni u pravilu učenja; primjerice, ako mrežu učimo algoritmom *Backpropagation* uz moment inercije, u kromosom se mogu upakirati i parametri poput stope učenja te faktora inercije čime će evolucijski algoritam evoluirati istovremeno i prikladnu arhitekturu mreže i njoj i problemu za koji se mreža koristi prilagođen algoritam učenja.

Prikaz uporabom pravila razvoja

Razvojna pravila su pravila koja govore na koji će se način izgraditi neuronska mreža. Zamislimo to ovako: umjesto da evoluiramo samu mrežu, mi evoluiramo program koji kada pokrenemo generira određenu neuronsku mrežu. Kromosomi su tada različita razvojna pravila. U literaturi je zabilježen niz prednosti ovakvog pristupa pred direktnim kodiranjem: bolja skalabilnost te imunost na problem permutacija budući da kromosomi ne predstavljaju direktno neuronske mreže.

Razvojna pravila najčešće dolaze u dva oblika: ili kao rekurzivne jednadžbe [Mjolsness et al.(1989)Mjolsness, Sharp, and Alpert] ili kao produkcijska pravila [Kitano(1990)] koja generiraju konačnu neuronsku mrežu. U oba slučaja, ono što se gradi jest matrica težina koja u konačnici određuje oblik neuronske mreže.

Primjer uporabe produkcijskih pravila prikazan je na slikama 13.5 i 13.6 gdje se pravilima gradi matrica veza.

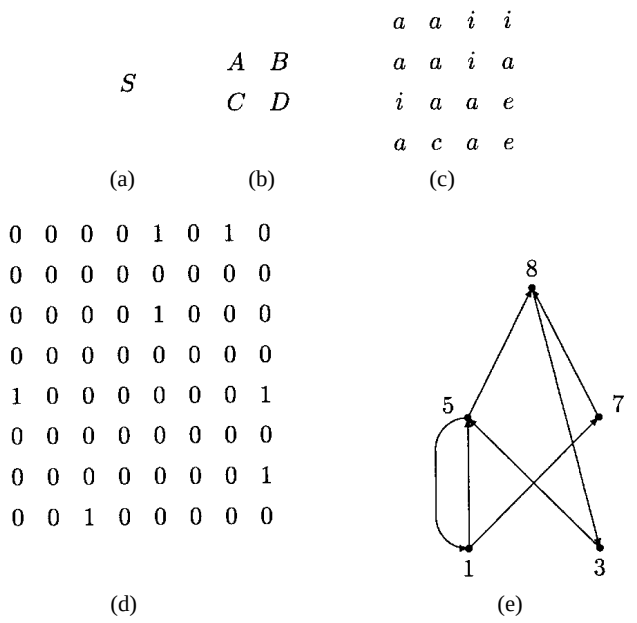
Produkcijska pravila prikazana na slici 13.5 sastoje se od nezavršnih simbola (velika slova i mala slova) te završnih simbola (znamenke). Pravila su podijeljena u dvije grupe: pravila prve razine kod kojih se s lijeve strane nalazi veliko slovo te pravila druge razine kod kojih se s lijeve strane nalazi malo slovo. Početni simbol od kojeg se kreće je `S`. Pravila prve razine s desne strane imaju matrice dimenzija 2×2 čiji su elementi mala slova. Pravila druge razine s desne strane imaju matrice dimenzija 2×2 čiji su

$$S \longrightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

$$A \longrightarrow \begin{pmatrix} a & a \\ a & a \end{pmatrix} \quad B \longrightarrow \begin{pmatrix} i & i \\ i & a \end{pmatrix} \quad C \longrightarrow \begin{pmatrix} i & a \\ a & c \end{pmatrix} \quad D \longrightarrow \begin{pmatrix} a & e \\ a & e \end{pmatrix} \quad \cdots$$

$$a \longrightarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad c \longrightarrow \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad e \longrightarrow \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad i \longrightarrow \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \cdots$$

Slika 13.5: Primjer produkcijskih pravila za izgradnju matrice veza neuronske mreže



Slika 13.6: Primjer uporabe produkcijskih pravila za izgradnju matrice veza neuronske mreže

elementi znamenke 0 ili 1.

Primjer izgradnje konkretne neuronske mreže prikazan je na slici 13.6. Kreće se od simbola `S` (a) koji se potom ekspandira u matricu 2×2 (b). Potom se svaki od elemenata te matrice mijenja podmatricama 2×2 (c) čime se dobiva matrica 4×4. Konačno, svaki se nezavršni element u toj matrici supstituira novom matricom 2×2 (d) čime se dobiva konačna matrica veza dimenzija 8×8. Ova matrica ne definira vrijednosti težinskih faktora već samo određuje između kojih neurona postoje veze. Točne iznose težina trebat će utvrditi posebno, primjerice algoritmom učenja nad konkretnim problemom koji se rješava. Konačna neuronska mreža prikazana je pod (e).

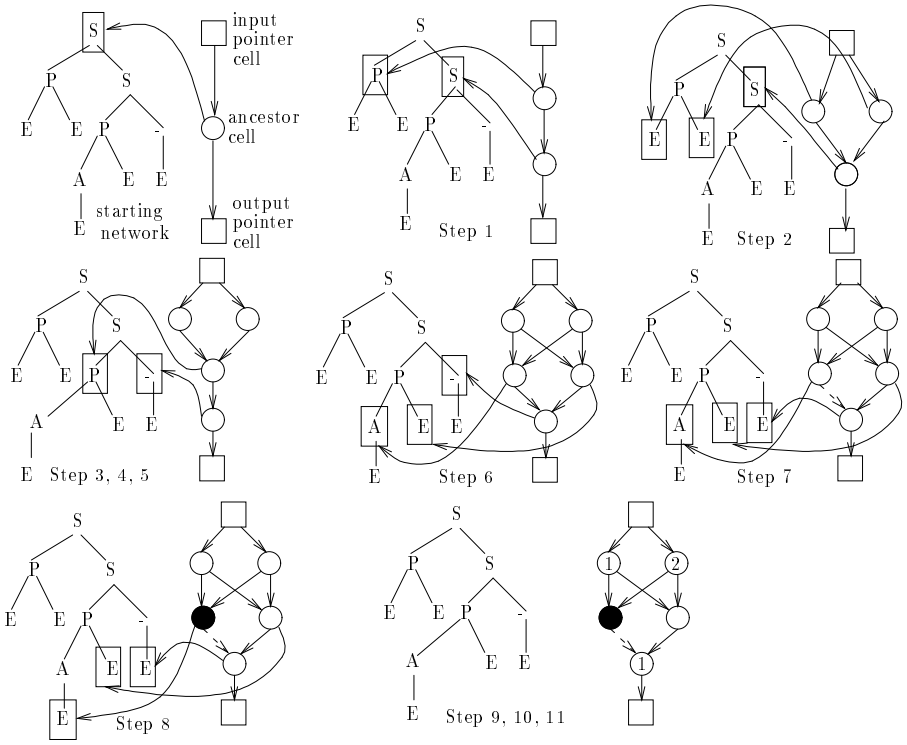
Pretpostavimo sada da su pravila druge grupe zadana unaprijed (pravila koja s lijeve strane imaju mala slova) i pretpostavimo da ih ima 16 (od `a` do `p`). Evolucijskim algoritmom uče se samo pravila prve grupe. Ako koristimo uobičajen skup slova, onda takvih pravila može biti ukupno 26. Kako svako pravilo s desne strane sadrži matricu od 4 elementa, slijedi da jedan kromosom koji specificira kompletan skup pravila treba sadržavati ukupno $26 \cdot 4 = 104$ simbola. Primjerice, početak kromosoma koji predstavlja upravo skup pravila prikazan na slici 13.5 bio bi `ABCDaaaaiiiaiaacaeae` (prva 4 simbola predstavljaju matricu pravila `S`, sljedeća 4 matricu pravila `A` itd.).

Zadaća evolucijskog algoritma primijenjenog na ovakav zapis neuronske mreže jest pronaći produkcijski sustav koji će izgraditi optimalnu neuronsku mrežu.

Pristup koji umjesto produkcijskih pravila koristi rekurzivne jednadžbe prikazan je u [Mjolsness et al.(1989)Mjolsness, Sharp, and Alpert]. Rekurzivne jednadžbe definirane su nad cjelobrojnim matricama i one određuju kako se iz manje matrice u sljedećem koraku gradi veća matrica. Na taj način iteraciju po iteraciju dolazi se do konačne matrice koja predstavlja neuronsku mrežu. Zadaća algoritma učenja je tada pronaći optimalne vrijednosti koeficijenata u rekurzivnim jednadžbama. Ovaj pristup dalje nećemo pojašnjavati.

Konačno, zanimljiv pristup uporabom staničnog razvoja opisan je u [Gruau(1992)]. Svaki neuron ima pridruženu kopiju genetskog koda koji upravlja njegovim razvojem. Genetski kod je stablo koje određuje kako se neuron transformira. Stablo se sastoji od niza simbola koje neuron izvršava. Svaki simbol predstavlja određenu operaciju koja modificira neuron koji je izvede. Razvoj počinje od najjednostavnije moguće neuronske mreže koja se sastoji samo od ulaznih neurona, izlaznih neurona te jednog *početnog* neurona koji je spojen na sve ulaze i sve izlaze i koji jedini dobiva genetski kod koji počinje izvršavati; ulazni i izlazni neuroni već su oformljeni neuroni koji se dalje ne razvijaju.

Simbol `S` označava da se neuron treba podijeliti u dva neurona. Prvi neuron pri tome preuzima sve ulazne veze početnog neurona a drugi neuron preuzima sve izlazne veze početnog neurona. Dodatno izlaz prvog neurona spaja se kao jedini ulaz drugog neurona. U stablu, simbol `S` ujedno je i točka



Slika 13.7: Primjer staničnog razvoja neuronske mreže

grananja: prvi stvoreni neuron koji je nastao ovom podjelom nastavit će izvoditi lijevu granu pridruženu simbolu `S` a drugi desnu granu (vidi sliku 13.7).

Simbol `P` predstavlja paralelno dijeljenje. Kad neuron obavi instrukciju `P`, podijelit će se u dva neurona pri čemu oba novonastala neurona preuzimaju sve ulazne i izlazne veze od neurona koji se je podijelio. U stablu, i simbol `P` predstavlja točku grananja: prvi stvoreni neuron koji je nastao ovom podjelom nastavit će izvoditi lijevu granu pridruženu simbolu `P` a drugi desnu granu.

Simbol `E` označava kraj programa; kad neuron pročita i izvrši taj simbol, neuron postaje nepromjenjiv i gubi pridruženi genetski kod. Simbol `A` povećava vrijednost praga u neuronu (težina w_0), a simbol `-` obrće predznak ulazne težine. Osim ovih, definiran je još niz simbola koji obavljaju različite akcije, uključivo i rekurzivne pozive i koje dalje nećemo opisivati. Primjer razvoja jedne mreže detaljnije je prikazan na slici 13.7.

Primjenom evolucijskog algoritma na ovakvu vrstu pravila razvoja pokušava se razviti program prikazan u obliku stabla koji će generirati neuronsku mrežu optimalnih karakteristika.

U praksi se osim navedenih primjera dosta radi i na evoluciji prijenosnih funkcija kao i na istovremenoj evoluciji i arhitektura i pripadnih težinskih faktora. Međutim, u okviru ovog poglavlja nećemo razmatrati ta područja.

13.3 Evolucija pravila učenja

Algoritam *Backpropagation* jedan je od primjera algoritama za učenje težina neuronske mreže; naravno – postoje i drugi. Stoga se postavlja pitanje možemo li evolucijski proces iskoristiti za pronalaženje novih algoritama učenja? Kada govorimo o pravilima učenja, evolucijski postupci se primjenjuju na dva mjesta.

13.3.1 Evolucija parametara algoritma učenja

Koji god algoritam učenja koristili, uobičajeno je da postoji jedan ili više parametara koji upravljaju radom tog algoritma. Kod algoritma *Backpropagation* to su, primjerice, stopa učenja i faktor inercije. Uz fiksirani algoritam učenja, evolucijski se proces može upotrijebiti kako bi pronašao optimalni skup tih parametara uz koje će postupak učenja biti maksimalno djelotvoran i učinkovit.

13.3.2 Evolucija novih algoritama učenja

Prilikom razvoja algoritama učenja, uobičajeno se kreće od sljedećih pretpostavki:

1. ažuriranje težine ovisi samo o lokalno dostupnim informacijama poput aktivacije dolaznog neurona, aktivacije promatranog neurona i slično te

2. algoritam učenja (odnosno pravilo) jednako je za sve težine u mreži.

Pravilo učenja tada se formulira kao linearna kombinacija dostupnih lokalnih varijabli i njihovih produkata:

$$\Delta w(t) = \sum_{k=1}^{n} \sum_{i_1,i_2,\ldots,i_k=1}^{n} \left(\theta_{i_1,i_2,\ldots,i_k} \prod_{j=1}^{k} x_{i,j}(t-1) \right)$$

gdje je t vrijeme odnosno iteracija, $\Delta w(t)$ promjena težine koju treba napraviti $x_1, \ldots, x_n$ su varijable koje čuvaju lokalno dostupne informacije, a $\theta_{i_1,i_2,\ldots,i_k}$ koeficijenti. Uz različite vrijednosti koeficijenata $\theta_{i_1,i_2,\ldots,i_k}$ dobivaju se različita pravila učenja, i zadaća evolucijskog procesa je utvrditi optimalne iznose ovih koeficijenata kako bi se dobio maksimalno učinkovit i djelotvoran algoritam učenja.

Poglavlje 14

Kombiniranje neuro-fuzzy sustava i evolucijskog računanja

Primjena algoritama evolucijskog računanja na neuro-fuzzy sustave postoji čitav niz. U okviru ovog poglavlja opisat ćemo nekoliko takvih primjena (radovi [Lin et al.(2011)Lin, Peng, and Lee, Ghomsheh et al.(2007)Ghomsheh, Shoorehdeli, and Teshnehlab]).

14.1 Primjena algoritma PSO na otkrivanje optimalnih parametara ANFIS-a

U radu [Ghomsheh et al.(2007)Ghomsheh, Shoorehdeli, and Teshnehlab] opisana je primjena algoritma PSO na treniranje neuro-fuzzy sustava ANFIS. U okviru ove knjige već smo opisali sustav ANFIS u poglavlju 12.2. Algoritam za treniranje tog sustava razvili smo polazeći od gradijentnog spusta koji zahtjeva da su funkcije koje se njime optimiraju derivabilne te da možemo relativno jednostavno i efikasno računati njihov gradijent.

Kako bi se zaobišla ta ograničenja, a ujedno i povećala otpornost algoritma učenja na lokalne optimume, u radu [Ghomsheh et al.(2007)Ghomsheh, Shoorehdeli, and Teshnehlab] opisana je primjena algoritma roja čestica. Primjena je ilustrirana na primjeru sustava ANFIS koji koristi zaključivanje tipa 3 (TSK-zaključivanje); takav sustav još je jednom prikazan na slici 14.1.

Sustav ANFIS za koji je razvijen algoritam treniranja opisan je sljedećim parametrima.

- I - broj ulaznih varijabli odnosno dimenzionalnost ulaza (na slici 14.1 $I = 2$).
- N^* - broj jezičnih izraza po jezičnoj varijabli, odnosno broj funkcija