Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Web-orijentirani simulator modela neuronskih mreža s primjerom raspoznavanja novčanica i generatorom uzoraka

Autori

Marko Čupić
apsolvent studija računarstva
matični broj: 0036363469

Jan Šnajder
apsolvent studija računarstva
matični broj: 0036350084

Mentor

Doc.dr.sc. Bojana Dalbelo Bašić

Zavod za elektroniku,
mikroelektroniku,
računalne i inteligentne
sustave

Zagreb, 24. travnja 2001.

PREDGOVOR

Rad sačinjavaju tri povezane cjeline organizirane u tri poglavlja. U prvom je poglavlju dan teorijski pregled neuro-računarstva s naglaskom na one dijelove na koje se oslanjaju kasnije implementacije. Drugo poglavlje daje kratki prikaz simulatora modela mekog računarstva koji je u nastavku korišten za generiranje neuronske mreže. U trećem je poglavlju prikazan konkretan primjer uporabe neuronske mreže za raspoznavanje uzoraka. Svojstva neuronske mreže (poput robusnosti) provjerena su uporabom umjetno generiranih uzoraka dobivenih za tu svrhu posebno napisanim programom.

Simulator, prateći programi i veći dio ovog materijala nastali su tijekom dvogodišnjeg rada na predmetima *Neizrazito, evolucijsko i neuro-računarstvo* i *Strojno učenje* pod stručnim vodstvom doc.dr.sc. Bojane Dalbelo Bašić. Rad nije nastao u sklopu diplomskih radova autora, niti je vezan uz ta područja.

Posebno se zahvaljujemo doc.dr.sc. Bojani Dalbelo Bašić za svesrdnu pomoć i podršku prilikom izrade svih segmenata ovog rada. Prilika koju nam je pružila omogućila nam je da mnogo naučimo iz ovog vrlo interesantnog područja, te da se okušamo u izradi vlastitih programskih rješenja.

SADRŽAJ

1. UMJETNE NEURONSKE MREŽE

1.1. UVOD U NEURO-RAČUNARSTVO

Iako danas većinu podataka koje ne obrađujemo snagom vlastita uma obrađujemo digitalnim računalom, pogrešna je slika svijeta u kojem upravo računala obrađuju većinu podataka. Digitalna računala svakodnevno obrađuju doista ogromnu količinu podataka - zapravo gotovo sva *automatizirana* obrada radi se pomoću računala. No, to je tek malen dio podataka naspram onih koji se neprestano obrađuju u mozgovima živih bića što nastoje preživjeti u svojoj okolini. Ako obradu podatka promatramo na spomenuti način, shvaćamo želju za ostvarenjem drugačijeg koncepta kojim bi bilo moguće imitirati obradu podataka kakva već milijunima godina postoji u prirodi.

Područje umjetne inteligencije stremi ka ostvarenju imitacije mozga, odnosno njegovu pretakanju u umjetan oblik. Mozak, napose ljudski, sadrži maksimum inteligencije koju danas poznajemo, možda čak i svu inteligenciju koju tim istim mozgom uopće *možemo spoznati*. Čovjek pokazuje kreativnost koja se očituje u sposobnosti izbora ispravnih hipoteza i pokretanja iskustava i vođenja tih iskustava na temelju logičkih pravila. Čovjek je osim toga sposoban učiti koristeći se različitim strategijama, a učenje je još jedan bitan aspekt umjetne inteligencije koji sustavu omogućava da obavlja promjene nad samim sobom. Stoga je opravdano smatrati da bi sustav koji uspješno oponaša rad ljudskog mozga bio upravo – inteligentan.

Danas znamo da se ljudski mozak sastoji od velikog broja živčanih stanica (neurona), koji pri obradi različitih vrsta informacija rade paralelno. Neurofiziološka istraživanja, koja su nam omogućila bolje razumijevanje strukture mozga, pa čak i kognitivna psihologija – koja promatra obradu podataka čovjeka na makro-razini - daju naslutiti da je modelu mozga najsličniji model u kojem brojni procesni elementi podatke obrađuju paralelno. Područje računarstva koje se bavi tim aspektom obrade informacija zovemo *neuro-računarstvo,* a paradigmu obrade podatka *umjetnom neuronskom mrežom* (engl. Artifical Neural Network, ANN).

Ideja potiče još iz 1940. g, kada McCulloch i Pitts (Massachusetts Institute of Technology), istražujući neurofiziološke karakteristike živih bića, objavljuju matematički model neuronske mreže u okviru teorije automata. Međutim, procesna moć ondašnjih računala nije još bila dorasla implementaciji umjetne neuronske mreže. Tek u kasnim pedesetima, pojavom LSI računala, pojavila su se i prva praktička ostvarenja. Neuronske mreže zatim opet padaju u zaborav, te u trajanju od dvadeset godina istraživanje tog područja gotovo da je bilo zaustavljeno. Umjetne neuronske mreže vraćaju se na scenu umjetne inteligencije 1990., da bi danas postale vodeći i gotovo nezaobilazan koncept pri razvoju inteligentnih sustava.

Posebna istraživanja rade se na području arhitekture računala koja bi na pogodniji način od konvencionalne von Neumannove arhitekture omogućila učinkovitu implementaciju umjetne neuronske mreže. Konvencionalna danas raširena arhitektura računala temelji se na sekvencijalnoj obradi podataka, koja nema mnogo zajedničkog sa strukturom i načinom funkcioniranja mozga. Inherentne razlike između konvencionalnih digitalnih računala i ljudskog mozga naznačene su u tablici 1.1.

Tablica 1.1.

Inherentne razlike između digitalnog računala i ljudskog mozga

Atribut	Mozak	Računalo
tip elementa za procesiranje	neuron (100 različitih vrsta)	bistabil
brzina prijenosa	2 ms ciklus	ns ciklus
broj procesora	oko 10^{11}	10 ili manje
broj veza među procesorima	10^3-10^4	10 ili manje
način rada	serijski, paralelno	serijski
signali	analogni	digitalni
informacije	ispravne i neispravne	ispravne
pogreške	nefatalne	fatalne
redundancija	stotine novih stanice	eventualno rezervni sustav

Ipak, možemo se koristiti i konvencionalnim računalima pri implementaciji umjetne neuronske mreže, odbacujući pri tome formalizam rješavanja problema putem *algoritama*, odnosno manipulacije simbolima po definiranim pravilima. Na taj se način u stvari nalazimo u hibridnom području u kojem sekvencijalni stroj tek imitira neuronsku mrežu kao visoko-paralelnu arhitekturu. Takav je sustav fizički von Neumannovo računalo, ali se na razini obrade podataka odriče simboličke paradigme u korist paradigme neuronskih mreža. Bitne karakteristike koje razlikuju te dvije paradigme dane su u tablici 1.2.

Uspjeh simboličkog pristupa u umjetnoj inteligenciji ovisi o ishodu prve navedene stavke u tablici 1.2 koja pretpostavlja da postoji rješenje u vidu algoritma i da nam je to rješenje znano. Ispada, međutim, da su mnogi svakodnevni zadaci preteški da bi ih se na taj način formaliziralo. Npr. raspoznavanje uzorka kojeg smo već vidjeli ali ne baš točno u obliku kakav nam je predstavljen, poput rukopisa, zatim prepoznavanje lica bez obzira na njegov izraz i sl. – primjera ima bezbroj. Konvencionalne tehnike očigledno su previše siromašne da bi obuhvatile svu različitost stvarnih podataka u cilju postizanja generalizacije.

Tablica 1.2.

Usporedba karakteristika paradigmi

Von Neumann	Neuronska mreža
Računalu se unaprijed detaljno mora opisati algoritam u točnom slijedu koraka (program)	Neuronska mreža uči samostalno ili s učiteljem
Podaci moraju biti precizni – nejasni ili neizraziti podaci ne obrađuju se adekvatno	Podaci ne moraju biti precizni (gotovo uvijek su neprecizni)
Arhitektura je osjetljiva – kod uništenja nekoliko memorijskih ćelija računalo ne funkcionira	Obrada i rezultat ne mora puno ovisiti o pojedinačnom elementu mreže
Postoji eksplicitna veza između semantičkih objekata (varijabli, brojeva, zapisa u bazi...) i sklopovlja računala preko pokazivača na memoriju	Pohranjeno znanje je implicitno, ali ga je teško interpretirati

1.2. UMJETNA NEURONSKA MREŽA

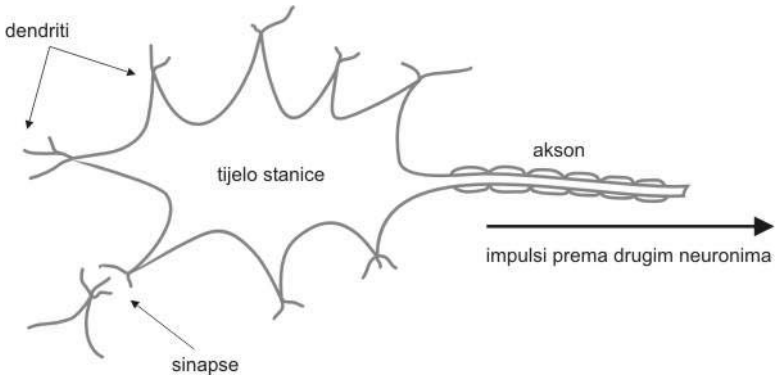
1.2.1. Neuron: biološki i umjetni

Za razumijevanje sposobnosti mozga nužno je upoznati građu njegova sastavna dijela: neurona (živčane stanica). Ljudski mozak sastavljen je od oko 10^{11} neurona kojih ima više od 100 vrsta i koji su shodno svojoj funkciji raspoređeni prema točno definiranom rasporedu. Svaki je neuron u prosjeku povezan s 10^4 drugih neurona. Četiri su osnovna dijela neurona: tijelo stanice (soma), skup dendrita (ogranaka), aksona (dugačke cjevčice koje prenose električke poruke) i niza završnih članaka. Slika 1.1 prikazuje građu neurona.

Tijelo stanice sadrži informaciju predstavljenu električkim potencijalom između unutrašnjeg i vanjskog dijela stanice (oko –70 mV u neutralnom stanju). Na sinapsama, spojnom sredstvu dvaju neurona kojim su pokriveni dendriti, primaju se informacije od drugih neurona u vidu post-sinaptičkog potencijala koji utječe na potencijal stanice povećavajući (hiperpolarizacija) ili smanjujući ga (depolarizacija). U tijelu stanice sumiraju se post-sinaptički potencijali tisuća susjednih neurona, u ovisnosti o vremenu dolaska ulaznih informacija. Ako ukupni napon pređe određeni prag, neuron "pali" i generira tzv. akcijski potencijal u trajanju od 1 ms. Kada se informacija akcijskim potencijalom prenese do završnih članaka, onda oni, ovisno o veličini potencijala, proizvode i otpuštaju kemikalije, tzv. neurotransmitere. To zatim ponovno inicira niz opisanih događaja u daljnjim neuronima. Iz navedenog je očigledno da je propagacija impulsa jednosmjerna.

Slika 1.1.

Građa neurona

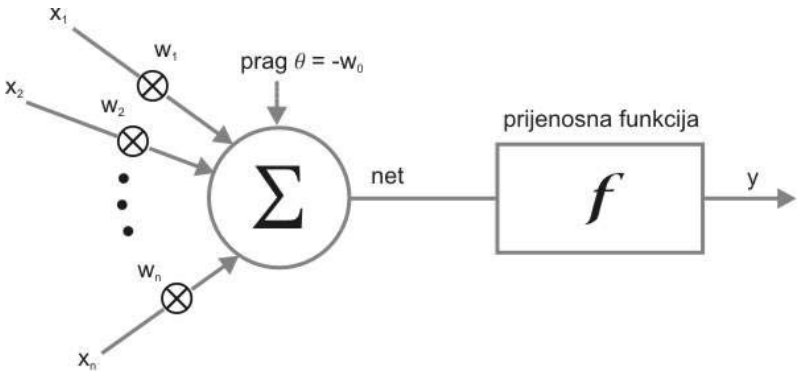


Funkcionalnost biološkog neurona imitira McCulloch-Pitts model umjetnog neurona, tzv. *Threshold Logic Unit* (TLU). Model koristi slijedeću analogiju: signali su opisani numeričkim iznosom i na ulazu u neuron množe se težinskim faktorom koji opisuje jakost sinapse; signali pomnoženi težinskim faktorima zatim se sumiraju analogno sumiranju potencijala u tijelu stanice; ako je dobiveni iznos iznad definirana praga, neuron daje izlazni signal.

U općenitom slučaju, umjetni neuron umjesto funkcije praga može imati i neku drugu funkciju, tzv. prijenosnu funkciju (transfer funkcija, aktivacijska funkcija). Općeniti model umjetnog neurona dan je na slici 1.2. U nastavku ćemo za pojam *umjetni neuron* ravnopravno koristiti i istovjetne pojmove: *procesni element* (PE), *čvor* ili *jedinica*.

Slika 1.2.

Umjetni neuron



Ulazne signale, njih ukupno n, označavamo sa $x_1, x_2, \ldots x_n$. Težine označavamo sa $w_1, w_2, \ldots w_n$. Ulazni signali općenito su realni brojevi u intervalu [-1,1], [0,1] ili samo elementi iz {0,1}, kada govorimo o Booleovom ulazu. Težinska suma *net* dana je s

$$net = \omega_1 x_1 + \omega_2 x_2 + \ldots + \omega_n x_n - \theta \tag{1.1}$$

ali se zbog kompaktnosti često dogovorno uzima da je vrijednost praga $q = -w_0$ te se dodaje ulazni signal x_0 s fiksiranom vrijednošću 1, pa pišemo jednostavnije

$$net = \omega_0 x_0 + \omega_1 x_1 + \omega_2 x_2 + \ldots + \omega_n x_n = \sum_{i=0}^{n} \omega_i x_i \tag{1.2}$$

dok je izlaz *y* rezultat prijenosne funkcije primijenjen na izraz 1.2:

$$y = f(\sum_{i=1}^{n} \omega_i x_i) = f(net) \tag{1.3}$$

1.2.2. Definicija i osobitosti umjetne neuronske mreže

Umjetna neuronska mreža u širem je smislu riječi umjetna replika ljudskog mozga kojom se nastoji simulirati postupak učenja. To je paradigma kojom su implementirani pojednostavljeni modeli što sačinjavaju biološku neuronsku mrežu. Analogija s pravim biološkim uzorom zapravo je dosta slaba jer uz mnoga učinjena pojednostavljena postoje još mnogi fenomeni živčanog sustava koji nisu modelirani umjetnim neuronskim mrežama, kao što postoje i karakteristike umjetnih neuronskih mreža koje se ne slažu s onima bioloških sustava.

Neuronska mreža jest skup međusobno povezanih jednostavnih procesnih elemenata, *jedinica* ili *čvorova*, čija se funkcionalnost temelji na biološkom neuronu. Pri tome je obradbena moć mreže pohranjena u snazi veza između pojedinih neurona tj. *težinama* do kojih se dolazi postupkom prilagodbe odnosno *učenjem* iz skupa podataka za učenje. Neuronska mreža obrađuje podatke distribuiranim paralelnim radom svojih čvorova.

Neke osobitosti neuronskih mreža naspram konvencionalnih (simboličkih) načina obrade podataka su slijedeće:

- Vrlo su dobre u procjeni nelinearnih odnosa uzoraka.
- Mogu raditi s nejasnim ili manjkavim podacima tipičnim za podatke iz različitih senzora, poput kamera i mikrofona, i u njima raspoznavati uzorke.
- Robusne su na pogreške u podacima, za razliku od mnogih konvencionalnih metoda koje pretpostavljaju neku raspodjelu obilježja u ulaznim podacima.
- Stvaraju vlastite odnose između podataka koji nisu zadani na eksplicitan simbolički način.
- Mogu raditi s velikim brojem varijabli ili parametara.
- Prilagodljive su okolini.
- Moguća je njihova jednostavna VLSI implementacija.
- Sposobne su formirati znanje učeći iz iskustva (tj. primjera).

Neuronske mreže odlično rješavaju probleme *klasifikacije* i *predviđanja*, odnosno općenito sve probleme kod kojih postoji odnos između prediktorskih (ulaznih) i zavisnih (izlaznih) varijabli, bez obzira na visoku složenost te veze (nelinearnost). Danas se neuronske mreže primjenjuju u mnogim segmentima života poput medicine, bankarstva, strojarstva, geologije, fizike itd., najčešće za slijedeće zadatke:

- raspoznavanje uzoraka,
- predviđanje (npr. kretanje dionica),

- obrada slike,
- obrada govora,
- problemi optimizacije,
- nelinearno upravljanje,
- obrada nepreciznih i nekompletnih podataka,
- simulacije i sl.

1.2.3. Vrste umjetnih neurona

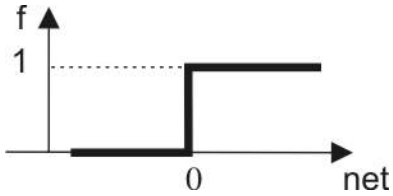
Općeniti model umjetnog neurona kakav je prikazan na slici 2.2 možemo dalje razmatrati prema ugrađenoj prijenosnoj funkciji. Slijede neki najčešći oblici te funkcije.

Najjednostavnija moguća aktivacijska funkcija je

$$f(net) = net \tag{1.4}$$

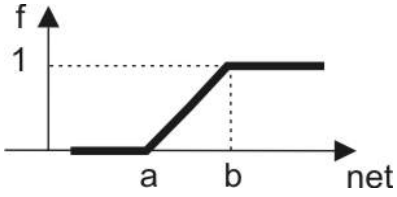
Takva je funkcija svojstvena modelu umjetnog neurona ADALINE (od engl. Adaptive Linear Element). Izlaz iz takve jedinice upravo je, dakle, težinska suma njegovih ulaza. Druga je mogućnost korištenje funkcije skoka ili praga (engl. threshold function, hard-limiter) čime dobivamo procesnu jedinicu koja daje Booleov izlaz (engl. Threshold Logic Unit, TLU):

$$f(net) = \begin{cases} 0 & za\ net < 0 \\ 1 & inace \end{cases} \tag{1.5}$$



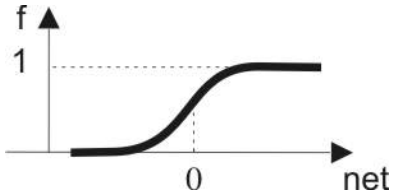
pri čemu znak nejednakosti može prema potrebi uključiti i jednakost. Prijenosna funkcija može biti definirana po dijelovima linearno:

$$f(net) = \begin{cases} 0 & za\ net \le a \\ net & za\ a < net < b \\ 1 & za\ net \ge b \end{cases} \tag{1.6}$$



Najčešći oblik prijenosne funkcije jest sigmoidalna funkcija. Za razliku od prethodnih funkcija, ova je funkcija derivabilna što je, kako će se pokazati, bitna prednost pri postupku učenja umjetne neuronske mreže. Sigmoidalna funkcija definirana je kao:

$$f(net) = \frac{1}{1 + e^{-a \cdot net}} \tag{1.7}$$



uz parametar *a* koji određuje nagib funkcije. Sigmoidalna funkcija ponekad se naziva i *logističkom funkcijom*.

Umjetne neurone nadalje možemo razvrstati i prema vrsti signala koje prosljeđuju (realni brojevi ili Booleove vrijednosti) te prema obliku integrirajuće funkcije (ovdje smo se ograničili samo na razmatranje težinske sume). U slučaju da se od neuronske mreže zahtijeva rad s podacima čije vrijednosti bilo na njezinu ulazu ili izlazu nisu u uobičajenom intervalu [-1,1], najjednostavnije je rješenje provesti pred-procesiranje odnosno post-procesiranje podataka (njihovo linearno preslikavanje u spomenuti interval).

1.2.4. Postupak učenja mreže

Prije postupka obrade podatka umjetnu je neuronsku mrežu potrebno *naučiti* ili *trenirati*. Za razliku od konvencionalnih tehnika obrade podataka gdje je postupak obrade potrebno analitički razložiti na određeni broj algoritamskih koraka, kod neuronske mreže takav algoritam ne postoji. Znanje o obradi podataka, tj. znanje o izlazu kao funkciji ulaza, pohranjeno je implicitno u težinama veza između neurona. Te se težine postupno prilagođavaju kroz postupak učenja neuronske mreže sve do trenutka kada je izlaz iz mreže, provjeren na skupu podataka za testiranje, zadovoljavajući. Pod postupkom učenja kod neuronskih mreža podrazumijevamo iterativan postupak predočavanja ulaznih primjera (koje često nazivamo i *uzorcima mreže* ili *iskustvom*) i eventualno očekivana izlaza.

Ovisno o tome da li nam je u postupku učenja á priori znan izlaz iz mreže, pa ga pri učenju mreže koristimo uz svaki ulazni primjer, ili nam je točan izlaz nepoznat, razlikujemo dva načina učenja:

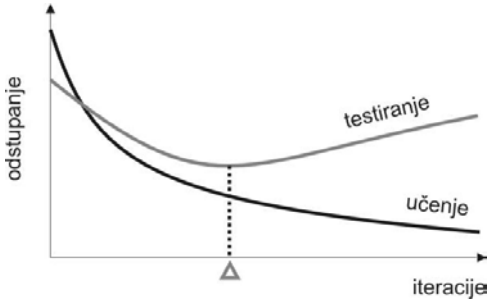
- učenje s učiteljem (engl. supervised learning) – učenje mreže provodi se primjerima u obliku para *(ulaz, izlaz),*
- učenje bez učitelja (engl. unsupervised learning) – mreža uči bez poznavanja izlaza.

Skup primjera za učenje često se dijeli na tri odvojena skupa: *skup za učenje*, *skup za testiranje* i *skup za provjeru* (validaciju). Primjeri iz prvog skupa služe za učenje u užem smislu (podešavanje težinskih faktora). Pomoću primjera iz drugog skupa vrši se tijekom učenja provjera rada mreže s trenutnim težinskim faktorima kako bi se postupak učenja zaustavio u trenutku degradacije performanse mreže. Umjetnu neuronsku mrežu moguće je, naime, *pretrenirati* - nakon određenog

broja iteracija mreža gubi svojstvo generalizacije i postaje stručnjak za obradu podatka iz skupa primjera za učenje dok preostale podatke obrađuje loše. Stalnim praćenjem izlaza iz mreže dobivenog pomoću primjera iz skupa za testiranje moguće je otkriti iteraciju u kojoj dobiveni izlaz najmanje odstupa od željenog (slika 1.3). Točnost i preciznost obrade podataka moguće je naposljetku provjeriti nad trećim skupom primjera – skupom za provjeru.

Slika 1.3.

Odstupanje od stvarnog izlaza kroz iteracije



Uz pojam učenja umjetne neuronske mreže vezani su pojmovi *iteracije* i *epohe*. Pod iteracijom podrazumijevamo korak u algoritmu postupka za učenje u kojem se odvija podešavanje težinskih faktora, dok je epoha jedno predstavljanje cjelokupnog skupa za učenje. Ovisno o broju primjera predočenih mreži za trajanje jedne iteracije, razlikujemo:

- pojedinačno učenje (engl. on-line training)– u jednoj iteraciji predočavamo samo jedan primjer za učenje (tj. kod svakog primjera za učenje vrši se prilagodba težinskih faktora),
- grupno učenje (engl. batch training) –u jednoj iteraciji predočavamo sve primjere za učenje (tj. iteracije se podudaraju s epohama).

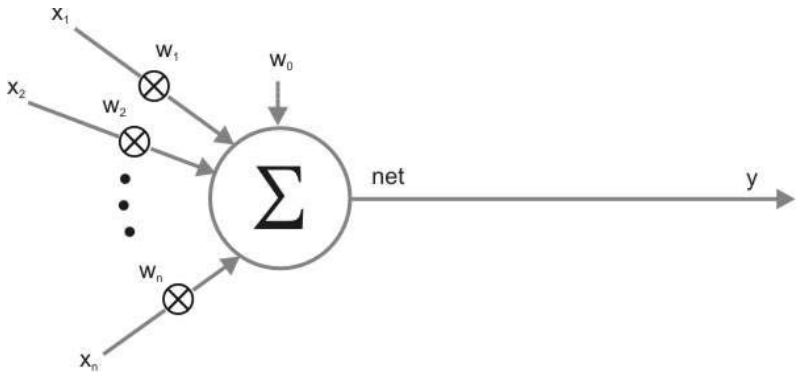
U nastavku ćemo prvo razmotriti metode učenja za neurone kao osnovne elemente umjetne neuronske mreže. Zatim ćemo razmotriti primjenu tih metoda u postupku učenja acikličkih neuronskih mreža. Zanimljivo područje neuro-računarstva - mreže koje uče bez učitelja - ostat će ovim radom neobuhvaćeno, jer ono nadilazi njegove okvire te nije neposredno vezano uz primjer uporabe koji će biti naveden kasnije.

1.3. ADALINE

ADALINE je najjednostavniji procesni element kod kojega je nelinearna funkcija izbačena, tako da neuron zapravo računa težinsku sumu ulaza, prema 1.4. ADALINE procesnu jedinicu prikazuje slika 1.4.

Slika 1.4.

ADALINE procesna jedinica

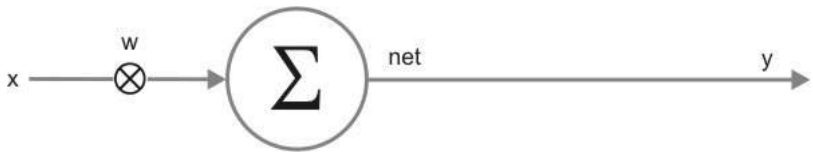


1.3.1. Učenje ADALINE jedinice

Na ovoj najjednostavnijoj procesnoj jedinici pokazat ćemo osnovne metode učenja umjetnih neurona. Naime, podešavanjem težinskih faktora w_i možemo sami odrediti kakav izlaz želimo uz određeni ulaz. Postupak kojim se ovo provodi naziva se učenje umjetnog neurona. Učenje možemo ilustrirati slijedećim primjerom: Neka je zadano N uređenih parova (x_i, d_i), pri čemu pretpostavljamo da je između njih odnos linearan, tj. da te točke leže na pravcu, a radi jednostavnosti pretpostavimo još i da taj pravac prolazi kroz ishodište. Tada možemo pisati: $d_i \approx w \cdot x_i$. Koristimo znak ≈ jer nismo sigurni da su sve točke baš na pravcu. Uočite da se ovo može realizirati kao ADALINE jedinica koja ima točno jedan ulaz i njemu pridružen težinski faktor w, prema slici 1.5. Takvu ADALINE jedinicu nazvat ćemo *reduciranom ADALINE jedinicom*.

Slika 1.5.

Reducirana ADALINE procesna jedinica



Uvođenjem oznake za grešku ε_i, izraz možemo napisati kao $d_i = w \cdot x_i + \varepsilon_i$, pri čemu je greška definirana kao: $\varepsilon_i = d_i - w \cdot x_i$. Želimo pronaći takav koeficijent w da pravac koji dobijemo na taj način radi najmanju pogrešku u odnosu na sve točke. Ukupnu pogrešku možemo definirati kao polovicu prosječne sume kvadrata pojedinačnih pogrešaka, polovica od MSE (engl. Mean Square Error):

$$J = \frac{1}{2N}\sum_{i=1}^{N}\varepsilon_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(d_i - w\cdot x_i\right)^2 \tag{1.8}$$

Želimo naći takav pravac za koji je ova pogreška minimalna. Problem možemo riješiti analitički:

$$\begin{aligned}
&\frac{\partial J}{\partial w} = 0 \\
&\Rightarrow \frac{\partial}{\partial w}\left(\frac{1}{2N}\sum_{i=1}^{N}\left(d_i - w\cdot x_i\right)^2\right) = 0 \\
&\Rightarrow -\frac{1}{N}\sum_{i=1}^{N}\left(d_i - w\cdot x_i\right)\cdot x_i = 0 \\
&\Rightarrow w = \frac{\sum_{i=1}^{N} d_i\cdot x_i}{\sum_{i=1}^{N} x_i^2}
\end{aligned} \tag{1.9}$$

Međutim, vidjeti ćemo da će u nastavku stvari postati dovoljno složene da ovakav način rješavanje učine nemogućim. Zbog toga ćemo postupak minimizacije provesti iterativnim postupkom: *gradijentnim spustom*. Naime, pogrešku *J* ćemo minimizirati tako da parametre o kojima pogreška *J* ovisi mijenjamo u smjeru suprotnom od smjera gradijenta funkcije: $w(k+1) = w(k) - \eta\cdot\nabla J(k)$, pri čemu *k* označava trenutni korak. Inicijalna vrijednost *w(0)* može se odabrati proizvoljno. Koeficijent η naziva se *stopa učenja*, i obično je dosta malen broj (između 0 i 1). Kako *J* ovisi samo o parametru *w*, gradijent je:

$$\nabla J = \frac{\partial J}{\partial w} = \frac{\partial}{\partial w}\left(\frac{1}{2N}\sum_{i=1}^{N}\varepsilon_i^2\right) == \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i\frac{\partial}{\partial w}\varepsilon_i \tag{1.10}$$

Vidimo da gradijent u izrazu 1.10 ovisi o SVIM pojedinačnim pogreškama, te njegovo izračunavanje na ovaj način zahtjeva dosta računskih operacija. Umjesto ovakvog načina izračuna gradijenta, Widrow je ponudio vrlo elegantnu aproksimaciju gradijenta: kada već kroz iterativni postupak radimo niz iteracija, tada gradijent u jednoj iteraciji možemo aproksimirati pomoću samo jednog ulaznog uzorka koji ćemo koristiti u toj aproksimaciji:

$$\nabla J \approx \varepsilon(k)\frac{\partial}{\partial w}\varepsilon(k) = -\varepsilon(k)\cdot x(k) \tag{1.11}$$

Uz ovu aproksimaciju iterativna formula za izračun koeficijenta *w* prelazi u:

$$w(k+1) = w(k) + \eta\cdot\varepsilon(k)\cdot x(k) \tag{1.12}$$

tzv. LMS pravilo.

Važno je imati na umu da je osnova izraza 1.12 aproksimacija. Kako nam je izračun gradijenta činio poteškoće, odlučili smo se na njegovu jednostavnu aproksimaciju pomoću trenutnog uzorka, a

ne pomoću svih uzoraka. Ova odluka ima za posljedicu unošenje šuma u gradijent. Upravo zbog tog šuma mi se ne spuštamo prema minimumu stvarnim gradijentom već nečim što ga aproksimira.

Drugi način minimizacije pogreške kreće od toga da se u svakom koraku iterativnog postupka računaju gradijenti sviju predočenih uzoraka. U tom slučaju krećemo od općenitog izraza za gradijent 1.10 i izraza 1.11, pa dobivamo:

$$\nabla J = \frac{\partial J}{\partial w} = \frac{\partial}{\partial w}\left(\frac{1}{2N}\sum_{i=1}^{N}\varepsilon_i^2\right) = \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i\frac{\partial}{\partial w}\varepsilon_i = -\frac{1}{N}\sum_{i=1}^{N}\varepsilon_i\cdot x_i \tag{1.13}$$

Ubacimo li ovu definiciju gradijenta u formulu za minimizaciju 1.12, dobiti ćemo iterativnu formulu:

$$\begin{aligned} &w(k+1) = w(k) - \eta\cdot\nabla J(k) = \\ &w(k) + \eta\cdot\sum_{i=1}^{N}\varepsilon_i\cdot x_i = \\ &w(k) + \eta\cdot\sum_{i=1}^{N}\left[\left(d_i - w(k)\cdot x_i\right)\cdot x_i\right] \end{aligned} \tag{1.14}$$

Faktor *1/N* ispred sume apsorbiran je, radi kompaktnosti zapisa, u konstantu η koja je ionako proizvoljna. Budući da se ovdje gradijent računa na temelju svih točaka, nema nikakvog šuma zbog aproksimacije Ovaj način učenja naziva se grupno (engl. batch) učenje, dok se direktna primjena LMS-a naziva pojedinačnim učenjem (engl. on-line). Karakteristika pojedinačnog učenja jest sposobnost učenja uzorak-po-uzorak. To znači da za pojedinačno učenje ne moramo predočiti cijeli skup ulaznih podataka. Grupno učenje pak zahtjeva za svaki korak predočavanje cijelog skupa ulaznih podataka. Međutim, iako je pojedinačno učenje jednostavnije i poželjnije, grupno učenje ima jednu veliku prednost – puno bolju stabilnost, što, ako zanemarimo glavni problem – potrebu za predočavanjem svih uzoraka u svakom koraku – ponekad može biti presudno.

Grupno se učenje može dobiti i uporabom klasične LMS formule, samo što tijekom jedne epohe (ciklusa u kojem predočavamo sve uzorke) izračunati koeficijent *w* držimo konstantnim, a akumuliramo promjene koje diktiraju gradijenti za svaki uzorak. Onda na kraju epohe izvršimo izmjenu za *w*. Naime, šumoviti gradijent koji nastaje kod LMS-a tijekom jedne epohe uprosječi se te šum nestane, što opet rezultira pravim gradijentom.

1.3.2. Konvergencija LMS algoritma

Pitanje koje se nameće uz LMS pravilo dano izrazom 1.12 jest: konvergira li ovaj izraz uopće i, ako konvergira, pod kojim uvjetima? Razmotrimo još jednom kako smo izveli 1.12. Do izraza smo došli vođeni težnjom za minimizacijom pogreške:

$$J = \frac{1}{2N}\sum_{i=1}^{N}\varepsilon_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(d_i - w\cdot x_i\right)^2 = \frac{1}{2N}\sum_{i=1}^{N}d_i^2 - \frac{1}{2N}2\cdot w\cdot\sum_{i=1}^{N}x_i\cdot d_i + w^2\frac{1}{2N}\sum_{i=1}^{N}x_i^2 \tag{1.15}$$

Analitičkim rješavanjem minimuma ove funkcije došli smo do:

$$w^* = \frac{\sum_{i=1}^{N}d_i\cdot x_i}{\sum_{i=1}^{N}x_i^2} \Rightarrow \sum_{i=1}^{N}d_i\cdot x_i = w^*\cdot\sum_{i=1}^{N}x_i^2 \tag{1.16}$$

Uvrštavanjem se dalje dobije:

$$J = \frac{1}{2N}\sum_{i=1}^{N}d_i^2 - \frac{1}{2N}2\cdot w\cdot w^*\cdot\sum_{i=1}^{N}x_i^2 + w^2\frac{1}{2N}\sum_{i=1}^{N}x_i^2$$
$$J = \frac{1}{2N}\sum_{i=1}^{N}\left(d_i^2 - x_i^2\cdot w^{*2}\right) + \frac{1}{2N}\left(w - w^*\right)^2\sum_{i=1}^{N}x_i^2 = J_{\min} + \frac{\lambda}{2}\left(w - w^*\right)^2 \tag{1.17}$$

uz

$$\lambda = \frac{1}{N}\sum_{i=1}^{N}x_i^2 \tag{1.18}$$

Iz izraza se vidi da je pogreška kvadratna funkcija koja ovisi o varijabli w. Riječ je o paraboli okrenutoj prema gore i koja je uvijek pozitivna. Minimum se postiže u tjemenu za $w = w^*$. Problem kako izračunati w^* riješili smo i analitički, i iterativno. Od iterativne formule očekujemo, dakako, da kada k teži u beskonačnost, da w teži ka w^*.

Gradijent funkcije pogreške možemo napisati i ovako:

$$\nabla J = \lambda\left(w - w^*\right) \tag{1.19}$$

pa uvrštavanjem dobivamo iterativnu formulu za koeficijente analognu LMS-u:

$$w(k+1) = w(k) - \eta\cdot\nabla J = w(k) - \eta\cdot\lambda\left(w(k) - w^*\right) = \left(1 - \eta\cdot\lambda\right)\cdot w(k) + \eta\cdot\lambda\cdot w^* \tag{1.20}$$

Od lijeve i desne strane oduzmimo w^*:

$$w(k+1) - w^* = \left(1 - \eta\cdot\lambda\right)\cdot\left(w(k) - w^*\right) \tag{1.21}$$

Ovo je jednadžba diferencija prvog reda. Ispišimo prvih nekoliko koraka rješavanja:

$$\begin{aligned} w(1)-w^* &= (1-\eta\cdot\lambda)\cdot(w(0)-w^*) \\ w(2)-w^* &= (1-\eta\cdot\lambda)\cdot(w(1)-w^*)= \\ &= (1-\eta\cdot\lambda)\cdot((1-\eta\cdot\lambda)\cdot(w(0)-w^*)+w^*-w^*)= \\ &= (1-\eta\cdot\lambda)^2\cdot(w(0)-w^*) \\ &\dots \\ w(k)-w^* &= (1-\eta\cdot\lambda)^k\cdot(w(0)-w^*) \end{aligned} \tag{1.22}$$

Odavde je rješenje očito:

$$w(k) = (1-\eta\cdot\lambda)^k\cdot(w(0)-w^*)+w^* \tag{1.23}$$

Kada k teži u beskonačnost, $w(k)$ mora težiti prema w^*.

$$\lim_{k\to\infty} w(k) = \lim_{k\to\infty}(1-\eta\cdot\lambda)^k\cdot(w(0)-w^*)+w^* = w^* \Rightarrow \lim_{k\to\infty}(1-\eta\cdot\lambda)^k \to 0 \tag{1.24}$$

Da bi traženi eksponencijalni član težio k nuli, modul mora biti manji od jedan, pa imamo:

$$|1-\eta\lambda|<1 \Rightarrow \eta<\frac{2}{\lambda} \Rightarrow \eta_{\max}=\frac{2}{\lambda} \tag{1.25}$$

Može se pokazati da se najbrža konvergencija postiže za $\eta=\frac{1}{\lambda}$.

Odavde je očigledno zašto je ispravan odabir stope učenja od presudne važnosti za brzu i ispravnu konvergenciju algoritma. Naime, razlikujemo tri karakteristična slučaja stope učenja:

- *Stopa učenja je vrlo mala.* Rezultat je monotona, izuzetno spora konvergencija prema w^*.
- *Stopa učenja je velika, ali manja od maksimalne.* Rezultat je alternirajuće približavanje vrijednosti w^*. Alternirajuće znači da niz poprima vrijednosti koje su veće od w^*, zatim manje od w^*, zatim opet veće, pa opet manje i tako redom ali svaki puta sve bliže vrijednosti w^*.
- *Stopa učenja je veća od maksimalne.* Rezultat je divergencija algoritma. Pronalaze se vrijednosti za w koje su svakim korakom sve dalje od w^*. Divergencija je najgori slučaj koji se može dogoditi, čak i ako nadziremo rad algoritma pa pri pojavi divergencije počnemo smanjivati stopu učenja. Naime, pri pojavi divergencije iskačemo iz točke u kojoj smo bili u neku vrlo daleku točku čime smo iz točke blizu minimuma skočili u "nepoznato" i traženju minimuma napravili veliku štetu, jer je algoritam jednostavno zaboravio gdje je bio.

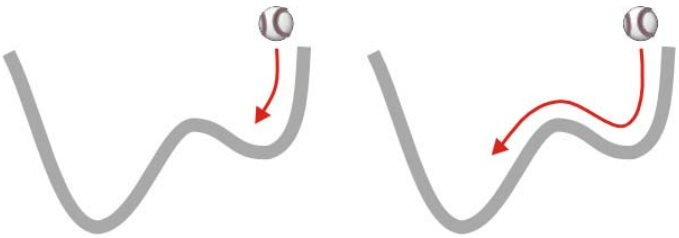
Na sličan način može se pokazati da i greška geometrijskim nizom pada ka vrijednosti J_{min}.

Kriterij opisan izrazom 1.25 vrijedi za pojedinačno učenje. Kakva je situacija kod grupnog učenja? Kod grupnog učenja u jednoj epohi akumuliramo promjenu faktora w ukupno N puta (za svaki od N uzoraka) čuvajući pri tome fiksan w i na kraju epohe radimo izmjenu. Nažalost, može se pokazati da je ova strategija, što se tiče stabilnosti, ekvivalentna obavljanju N izmjena faktora w. Efekt ovoga je prividni porast stope učenja N puta, pa da bi postupak ostao stabilan, stopu učenja kod grupnog učenja treba još podijeliti s N.

1.3.3. Popravljanje svojstava konvergencije

U prethodnom poglavlju pokazali smo kako brzina kojom se stiže do minimuma ovisi o stopi učenja η. Međutim, problem lokalnih minimuma, naslijeđen iz samog gradijentnog postupka, i dalje je ostao neriješen. Ukoliko se tijekom optimizacije dođe u blizinu lokalnog minimuma, postupak će vrlo vjerojatno tamo i zaglaviti (naime, gradijent je tada približno nula i neće dolaziti do korekcije težinskih faktora). Kako bi se ovo izbjeglo, postupak se modificira imajući u vidu jednu analogiju iz stvarnoga svijeta (slika 1.6).

Slika 1.6.
Moment i lokalni minimum
(a) Bez momenta
(b) S momentom



Trenutni položaj na plohi pogreške možemo zamisliti kao položaj loptice. Ukoliko ta loptica nema momenta, tada će se ona spustiti u prvi lokalni minimum i u njemu će ostati. No, ukoliko loptica ima moment, ona će se po inerciji nastaviti gibati pa ako je lokalni minimum dovoljno malen, loptica će ga napustiti i nastaviti gibanje prema globalnom minimumu. Ovakvo razmišljanje dovodi nas do modificirane LMS formule:

$$w(k+1) = w(k) + \eta \cdot \varepsilon(k) \cdot x(k) + \gamma \cdot \big(w(k) - w(k-1)\big) \tag{1.26}$$

Pri tome se γ naziva momentom, i obično je njegova vrijednost između 0 i 1.

1.3.4. Poopćenje reduciranog ADALINE elementa

U uvodnom dijelu poglavlja uveli smo pojam ADALINE jedinice, ali smo zbog jednostavnosti daljnju analizu proveli za ADALINE sa samo jednim ulazom, kojeg smo nazvali reduciranim (slika 1.5). Sada ćemo razmotriti kako se ovaj isti element ponaša kada dopustimo višestruke ulaze, kao što to pokazuje slika 1.4. Označimo broj ulaza jedinice s D. Sada ADALINE na temelju D ulaznih varijabli i D težinskih faktora računa težinsku sumu i nju prosljeđuje na izlaz. Za $D=1$ već smo pokazali da ADALINE možemo naučiti kako da obavi linearnu regresiju ulaznih točaka na pravac kroz ishodište. Za $D=2$ umjesto pravca dobiti ćemo ravninu, a za $D>2$ govorimo o hiper-ravninama. Zajedničko svojstvo svima je da prolaze kroz ishodište, što je glavni razlog loše regresije za slučaj kada točke ne tvore hiper-ravninu koja prolazi kroz ishodište. Da bi se ovo izbjeglo, uvodi se dodatni težinski faktor w_0 koji predstavlja pomak (engl. bias) čime se dodaje još jedan stupanj slobode. U tom slučaju ADALINE računa funkciju:

$$y = w_D \cdot x_D + w_{D-1} \cdot x_{D-1} + \cdots + w_1 \cdot x_1 + w_0 \tag{1.27}$$

Za slučaj $D=1$ sada imamo najopćenitiji oblik pravca: $y=ax+b$. Prethodnu formulu kompaktno ćemo zapisivati na slijedeći način:

$$y = \sum_{i=0}^{D} w_i \cdot x_i \tag{1.28}$$

pri čemu je $x_0=1$ (uvijek; x_0 nije stvarni ulaz ADALINE-a). Formula je identična formuli 1.2. Pretpostavimo sada da imamo na raspolaganju N ulaznih točaka (D komponentne stupčaste vektore $\vec{x}_i$) i odgovarajuće izlaze (d_i), i pretpostavimo da te točke tvore hiper-ravninu. ADALINE ćemo naučiti željenoj funkciji tako da obavimo linearnu regresiju. U tom slučaju ADALINE će davati izlaz:

$$y_i = w_D \cdot x_{i,D} + w_{D-1} \cdot x_{i,D-1} + \cdots + w_1 \cdot x_{i,1} + w_0 \quad \text{(i=1,2,…,N)} \tag{1.29}$$

pri čemu je $x_{i,j}$ j-ta komponenta vektora $\vec{x}_i$, tj. i-tog uzorka. Kako i-tom uzorku odgovara izlaz d_i, ADALINE radi pogrešku $\varepsilon_i=d_i-y_i$, tj.

$$\varepsilon_i = d_i - \left(w_D \cdot x_{i,D} + w_{D-1} \cdot x_{i,D-1} + \cdots + w_1 \cdot x_{i,1} + w_0\right) = d_i - \sum_{k=0}^{D} w_k \cdot x_{i,k} \tag{1.30}$$

(uz $x_{i,0}=1$).

Pogreška za sve uzorke tada je definirana izrazom:

$$J = \frac{1}{2N} \sum_{i=1}^{N} \left(d_i - \sum_{k=0}^{D} w_k \cdot x_{i,k} \right)^2 \tag{1.31}$$

Vidimo da pogreška ovisi o $D+1$ težinskom faktoru koje treba odrediti tako da postignemo minimum pogreške. Ovo ćemo obaviti na isti način kao što smo to radili i kada je ADALINE imao samo jedan ulaz: naći ćemo gradijent i zatim doći do iterativne formule za gradijentni spust. Kako J ovisi o $D+1$ težinskom faktoru, gradijent je $D+1$ komponenti stupčasti vektor:

$$\nabla J = \begin{bmatrix} \dfrac{\partial J}{\partial w_D} \\ \vdots \\ \dfrac{\partial J}{\partial w_0} \end{bmatrix} \tag{1.32}$$

Parcijalna derivacije po w_j $(j=0,1,2,...,D)$ iznosi:

$$\frac{\partial J}{\partial w_j} = \frac{\partial}{\partial w_j} \left(\frac{1}{2N} \sum_{i=1}^{N} \left(d_i - \sum_{k=0}^{D} w_k \cdot x_{i,k} \right)^2 \right) = \frac{1}{N} \sum_{i=1}^{N} \left(\left(d_i - \sum_{k=0}^{D} w_k \cdot x_{i,k} \right) \cdot \left(- x_{i,j} \right) \right) \tag{1.33}$$

Sada se vidi da za izračun gradijenta trebamo sve ulazne uzorke. Međutim, ako opet primijenimo Widrowovu ideju i gradijent aproksimiramo u i-toj iteraciji samo pomoću i-tog ulaznog uzorka, izraz 1.33 prelazi u:

$$\frac{\partial J}{\partial w_j} \approx -\left(d_i - \sum_{k=0}^{D} w_k \cdot x_{i,k} \right) \cdot x_{i,j} = -\varepsilon_i \cdot x_{i,j} \tag{1.34}$$

Tada gradijent u *i*-tom koraku (1.32) možemo zapisati kao:

$$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial w_D} \\ \vdots \\ \frac{\partial J}{\partial w_0} \end{bmatrix} \approx \begin{bmatrix} -\varepsilon_i \cdot x_{i,D} \\ \vdots \\ -\varepsilon_i \cdot x_{i,0} \end{bmatrix} = -\varepsilon_i \cdot \begin{bmatrix} x_{i,D} \\ \vdots \\ x_{i,0} \end{bmatrix} \tag{1.35}$$

pa slijedi iterativna formula gradijentnog spusta:

$$\vec{w}(k+1) = \vec{w}(k) - \eta \cdot \nabla J(k) = \vec{w}(k) + \eta \cdot \varepsilon(k) \cdot \vec{X}(k) \tag{1.36}$$

tj. LMS formula, pri čemu je $\vec{X}(k)$ vektor ulaznog uzorka proširen jedinicom:

$$\vec{X}(k) = \begin{bmatrix} x_{k,D} \\ \vdots \\ x_{k,1} \\ 1 \end{bmatrix} \tag{1.37}$$

a $\varepsilon(k)$ je greška u *k*-tom koraku. Maknemo li se iz ovog vektorskog oblika, vidjeti ćemo zapravo da sada imamo identičnu formulu gradijentnog spusta za svaki faktor w_k.

1.4. TLU PERCEPTRON

TLU (Threshold Logic Unit) jest procesni element koji kao nelinearnost koristi step funkciju. Izlaz ovog perceptrona dan je izrazom 1.5.

Alternativno se za vrijednosti izlaza umjesto 0 i 1 mogu koristiti vrijednosti –1 i 1. Ovakav tip perceptrona najčešće se koristi kao element za klasifikaciju: pripada li zadani uzorak klasi (tada je izlaz 1), ili ne (tada je izlaz 0). Isto tako, ovaj se perceptron često koristi za ostvarivanje logičkih funkcija I te ILI.

1.4.1. Pravilo perceptrona

Osnovni postupak učenja pravilom perceptrona (engl. perceptron rule) je slijedeći: Zadaje se niz uzoraka s pripadnom klasifikacijom d_i (pretpostavimo perceptron s izlazima 0 i 1):

$$\begin{matrix}(x_{1,1}, x_{1,2}, \cdots, x_{1,D}, d_1)\\ (x_{2,1}, x_{2,2}, \cdots, x_{2,D}, d_2)\\ \vdots\\ (x_{k,1}, x_{k,2}, \cdots, x_{k,D}, d_k)\end{matrix} \tag{1.38}$$

Za učenje se koristi LMS pravilo, ali na slijedeći način.

- Ukoliko se uzorak klasificira ispravno, ne radi korekciju.
- Ukoliko se uzorak klasificira neispravno, primjeni LMS pravilo.
- Ciklički uzimaj sve uzorke redom, a postupak zaustavi kada sve uzorke klasificiraš ispravno za redom.

Ovo matematički možemo zapisati na slijedeći način:

$$\vec{w}(k+1) = \vec{w}(k) + \begin{cases}\eta \cdot \varepsilon(k) \cdot \vec{X}(k), \text{ako je } (net < 0 \;\&\; \&d = 1) \,||\, (net \geq 0 \;\&\; \&d = 0)\\ 0, \text{ako je } (net < 0 \;\&\; \&d = 0) \,||\, (net \geq 0 \;\&\; \&d = 1)\end{cases} \tag{1.39}$$

Navodimo i pseudo-kod algoritma ekvivalentnog izrazu 1.38:

Pseudo-kod za pravilo perceptrona

```
Ponavljaj za svaki uzorak x(i) ciklički
        δ = d(i) - y(i)
                Ako je δ ≠ 0 tada
                ε = d(i) - y(i)
                w(j) = w(j) + η · ε · x(i,j)
        Kraj ako
Sve dok sve uzorke ne klasificiraš ispravno za redom
```

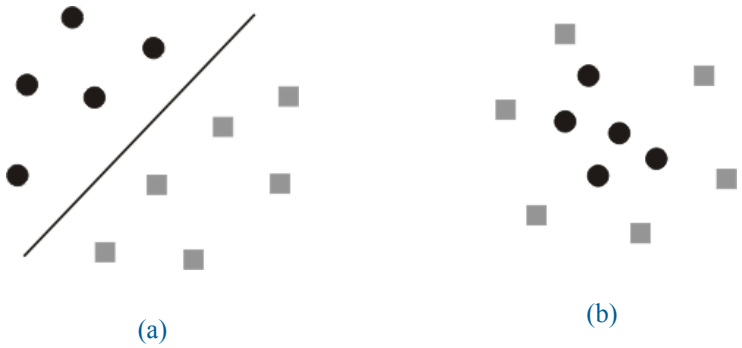
Već smo pokazali da je ADALINE linearni element. Kako je TLU perceptron izveden iz ADALINE perceptrona, ima granicu za *net=0*, može se pokazati da je TLU perceptron linearni klasifikator. To znači da će moći ispravno klasificirati razrede koji su linearno odvojivi. Primjer linearno odvojivih razreda prikazuje 1.7a, dok razrede koji nisu linearno odvojivi prikazuje 1.7b. Postupak učenja pravilom perceptrona konvergirati će ispravnom rješenju ako i samo ako su razredi međusobno linearno odvojivi. U suprotnom postupak nikada neće završiti.

Slika 1.7.

Linearna odvojivost

(a) Odvojivi razredi

(b) Neodvojivi razredi



(a) (b)

Ovdje se javlja značajan problem: Pretpostavimo da smo započeli učenje perceptrona, i da je ono trajalo izvjesno duže vrijeme. U jednom se trenutku ipak moramo zapitati da li nastaviti postupak učenja (koje je možda uzaludno, jer uzorci nisu linearno razdvojivi) ili ga prekinuti (iako možda postupak traje toliko dugo samo zato jer imamo vrlo sporu konvergenciju). Ako koristimo ovu metodu učenja, problem ostaje nerješiv.

1.4.2. Delta pravilo

Ideja za uvođenje *delta pravila* (engl. delta rule) dolazi kao rješenje prethodno opisanog problema. Pravilo promatra TLU perceptron kao ADALINE element i pokušava postići najbolju linearnu regresiju. Naime, osim što uzorke možemo zadati na način opisan izrazom 1.38, problem možemo promatrati i ovako: zadana je funkcija od D varijabli na slijedeći način:

$$\begin{aligned} f\left(x_{1,1}, x_{1,2}, \cdots, x_{1,D}\right) &= d_1 \\ f\left(x_{2,1}, x_{2,2}, \cdots, x_{2,D}\right) &= d_2 \\ &\vdots \\ f\left(x_{k,1}, x_{k,2}, \cdots, x_{k,D}\right) &= d_k \end{aligned} \tag{1.40}$$

Funkciju f možemo nazvati klasifikacijskom funkcijom; to je funkcija koja uzorku pridružuje njegov razred. Upravo to je ono što očekujemo i od TLU perceptrona. Nadalje, algoritam pretpostavlja da je funkcija f linearna, pa se učenje perceptrona može svesti na postupak linearne regresije. Taj nam je postupak poznat kao učenje ADALINE jedinice (klasični LMS postupak). Funkcija f će biti linearna, ukoliko su razredi međusobno odvojivi. Ukoliko razredi nisu linearno odvojivi, funkcija nije linearna, i postupak učenja nikada neće uspjeti sve uzorke ispravno razvrstati. Međutim, postupak će ipak svakom iteracijom pogrešku smanjivati prema nekoj minimalnoj vrijednosti koju linearni klasifikator u tom slučaju može postići. Ovo je bitna razlika u odnosu na prethodno opisani postupak pravila perceptrona.

Postupak učenja se provodi na slijedeći način:

- kao izlaz TLU perceptrona uzimaj *net* (tj. izlaz ADALINE elementa),
- provodi LMS postupak učenja.

Postupak ostvaruje slijedeći pseudo-kod:

Pseudo-kod za *delta pravilo* - pojedinačno učenje

```
Ponavljaj za svaki uzorak x(i) ciklički
        ε = d(i) - Σ w(j)x(i,j)
        w(j) = w(j) + η · ε · x(i,j)
Sve dok sve uzorke ne klasificiraš ispravno za redom
```

$Y(i)=\Sigma w(j)x(i,j)$ je izlaz TLU perceptrona, $d(i)$ je željeni izlaz za uzorak $x(i)$, $x(i,j)$ je j-ta komponenta uzorka $x(i)$.

Prethodni pseudo-kod zapravo je implementacija pojedinačnog učenja iz poglavlja 1.3. Delta pravilo može se implementirati i kao postupak grupnog učenja na način kako slijedi:

Pseudo-kod za *delta pravilo* - grupno učenje

```
Ponavljaj
        ∀j Dw(j) = 0
        Ponavljaj za svaki uzorak x(i)
                ε = d(i) - Σ w(j)x(i,j)
                Dw(j) = Dw(j) + η · ε · x(i,j)
        Kraj ponavljaj
        Ako si sve uzorke u gornjoj petlji klasificirao ispravno
                Tada zavrsi postupak.
        ∀j w(j) = w(j) + Dw(j)
Kraj ponavljaj
```

1.5. VIŠESLOJNE NEURONSKE MREŽE

1.5.1. Struktura mreže

Način na koji su neuroni međusobno organizirani i povezani u mreži određuju njezinu *arhitekturu*. Razlikujemo četiri osnovne arhitekture:

- unaprijedna mreža (engl. feedforward network) odnosno aciklička mreža,
- mreža s povratnom vezom (engl. recurrent network),
- lateralno povezana mreža (rešetkasta),
- hibridne mreže.

Aciklička mreža nema povratnih veza između neurona pa signali koji krenu od ulaznih neurona nakon određenog broja prijelaza dolaze do izlaza mreže, tj. propagacija signala je jednosmjerna. Odatle i engleski naziv *feedforward* mreže. Kod ovakve vrste mreža razlikujemo ulazni sloj neurona, izlazni sloj i skriveni sloj. Neuroni ulaznog sloja nemaju ulaznih signala - nemaju funkcionalnost neurona - i obično pod ulaznim slojem podrazumijevamo podatke organizirane u vektor konkretnih vrijednosti. Struktura mreže obično se zadaje kao n-torka u zapisu $n_1 \mu n_2 \mu .. \mu n_n$ kojom se označava mreža od *n slojeva* kod koje n_1 neurona čini ulazni sloj, n_2 neurona prvi skriveni sloj itd.

Posebna podvrsta acikličkih mreža su mreže kod kojih je moguće oblikovati slojeve njezinih neurona. U tom slučaju ne postoji skup od tri neurona *a*, *b*, *c* takav da je ulaz na *c* izlaz iz *a* i *b*, te da je istovremeno izlaz iz *a* spojen na ulaz neurona *b*. Mrežu kod kojih je ovaj zahtjev ispunjen nazivamo *slojevitom acikličkom*.

Neuronske mreže s povratnom vezom sadrže u svojoj strukturi barem jednu povratnu vezu, tj. postoji barem jedan čvor takav da ako pratimo njegov izlaz kroz sve moguće putove, nakon konačnog broja koraka čvor ćemo ponovno obići. Kod mreža s povratnom vezom ne možemo govoriti o

ulaznom i izlaznom sloju, već govorimo o *vidljivim čvorovima,* koji interagiraju s okolinom, i *skrivenim čvorovima*, kao kod acikličkih mreža. Primjer mreže s povratnom vezom je Hopfieldova mreža (nema skrivenog sloja) i Boltzmannov stroj (dozvoljava skriveni sloj).

Spomenute strukture mreže prikazane su na slici 1.8.

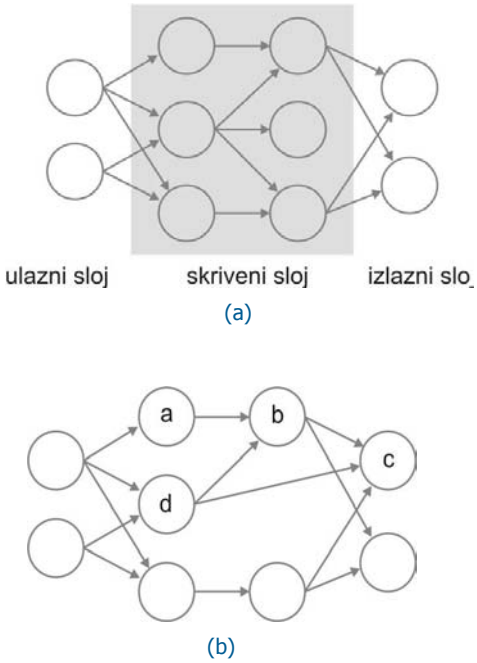
Slika 1.8.

Neke strukture mreža

(a) Aciklička slojevita 2µ3µ3µ2 mreža

(b) Aciklička koja ne ispunjava uvjet slojevitosti radi povezanosti neurona b-c-d

(c) Mreža s povratnom vezom b-a

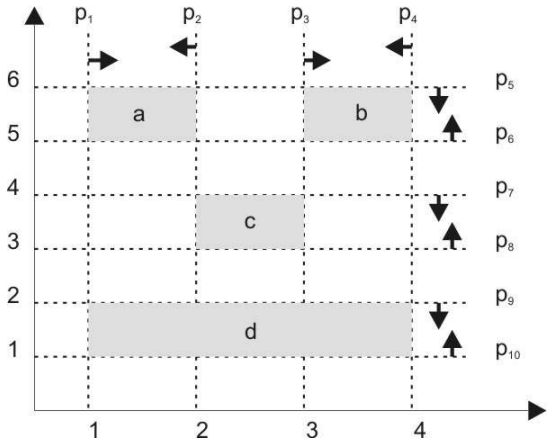


1.5.2. Višeslojna mreža perceptrona

Uzevši u obzir ograničenu primjenjivost perceptrona na predočavanje samo linearne funkcije, potrebno je za predočavanje složenijih odnosa koristiti neuronsku mrežu koja se sastoji od više međusobno povezanih perceptrona. Pogledajmo jednostavan primjer kako se može načiniti neuronska mreža za klasifikaciju uzoraka pomoću TLU perceptrona. Ovakva mreža može se vrlo jednostavno projektirati ako unaprijed znamo raspodjelu uzoraka (slika 1.9).

Slika 1.9.

Raspodjela uzoraka

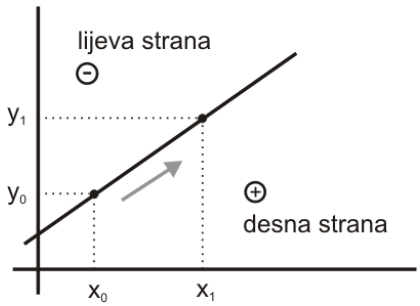


Pretpostavimo da uzorcima koji se nalaze u poljima *a*, *b*, *c* i *d* treba dodijeliti razred *1*, a svim ostalim uzorcima razred *-1*. Treba pronaći odgovarajući način spajanja TLU perceptrona u mrežu kako bismo ostvarili ovu klasifikaciju.

TLU perceptron jest linearni klasifikator, tj. granična linija kojom perceptron odvaja uzorke je hiper-ravnina (odnosno pravac u 2D prostoru). Prvo moramo pogledati kako možemo sami odrediti težinske faktore jednog TLU perceptrona kada znamo kako izgleda granica. Prema slici 1.10, ukoliko zadajemo pravac kroz dvije točke (x_0,y_0) i (x_1,y_1), možemo zamisliti da je pravac prividno dobio usmjerenje od točke (x_0,y_0) prema točki (x_1,y_1).

Slika 1.10.

Granica područja TLU perceptrona



Raspišemo li jednadžbu pravca kroz dvije točke

$$y - y_0 = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) \tag{1.41}$$

dobit ćemo

$$\begin{gathered}(y - y_0)(x_1 - x_0) - (y_1 - y_0)(x - x_0) = 0 \\ \cdots \\ -x(y_1 - y_0) + y(x_1 - x_0) + x_0(y_1 - y_0) - y_0(x_1 - x_0) = 0\end{gathered} \tag{1.42}$$

Izjednačavanjem s težinskom sumom koju računa perceptron:

$$w_1 x + w_2 y + w_0 = 0 \tag{1.43}$$

Možemo pročitati težinske faktore:

$$\begin{aligned} w_1 &= -(y_1 - y_0) \\ w_2 &= (x_1 - x_0) \\ w_0 &= x_0(y_1 - y_0) - y_0(x_1 - x_0) \end{aligned} \tag{1.44}$$

Uočavamo odmah da TLU za sve uzorke koji su na pozitivnoj strani pravca daje izlaz *1*, a za sve koji su na negativnoj strani daje izlaz *-1* (ili *0*). Sada možemo definirati mrežu.

Područje *a* na slici 1.9 ograđeno je pozitivnim dijelovima pravaca p_1, p_2, p_5 i p_6. Izvedimo jednadžbe tih pravaca. Npr. p_1 prolazi točkama (1,6) i (1,5). Uočite kako smo redoslijed točaka odabrali tako da se krećemo prema dolje, tako da nam strana na kojoj se nalazi područje *a* bude pozitivna. Uz ove dvije točke koeficijenti pravca glase:

$$\begin{aligned} w_1 &= -(y_1 - y_0) = -(5-6) = 1 \\ w_2 &= (x_1 - x_0) = 1 - 1 = 0 \\ w_0 &= x_0(y_1 - y_0) - y_0(x_1 - x_0) = -1 \end{aligned} \tag{1.45}$$

te jednadžba pravca glasi:

$$p_1 \ldots \; w_1 x + w_2 y + w_0 = 0 \Rightarrow 1 \cdot x + 0 \cdot y - 1 = 0 \tag{1.46}$$

Zapamtimo koeficijente w_i za ovaj pravac jer će nam trebati za definiranje ulaza TLU perceptrona. Na sličan način dolazi se i do preostalih jednadžbi pravaca.

$$\begin{aligned} p_1 \ldots &\quad 1 \cdot x + 0 \cdot y - 1 = 0 \\ p_2 \ldots &\quad -1 \cdot x + 0 \cdot y + 2 = 0 \\ p_3 \ldots &\quad 1 \cdot x + 0 \cdot y - 3 = 0 \\ p_4 \ldots &\quad -1 \cdot x + 0 \cdot y + 4 = 0 \\ p_5 \ldots &\quad 0 \cdot x - 1 \cdot y + 6 = 0 \\ p_6 \ldots &\quad 0 \cdot x + 1 \cdot y - 5 = 0 \\ p_7 \ldots &\quad 0 \cdot x - 1 \cdot y + 4 = 0 \\ p_8 \ldots &\quad 0 \cdot x + 1 \cdot y - 3 = 0 \\ p_9 \ldots &\quad 0 \cdot x - 1 \cdot y + 2 = 0 \\ p_{10} \ldots &\quad 0 \cdot x + 1 \cdot y - 1 = 0 \end{aligned} \tag{1.47}$$

Uočite samo da se *c* nalazi sa negativnih strana pravaca p_2 i p_3, budući da smo njih izveli tako da im je pozitivna strana okrenuta prema područjima *a* odnosno *b*. Krenimo sada u definiranje strukture mreže. U prvom sloju (ulazni sloj) nalaze se dva perceptrona koja direktno prosljeđuju vrijednosti x, odnosno y. Prvi skriveni sloj definiran je upravo preko 10 pravaca koje smo izveli: prvi skriveni sloj

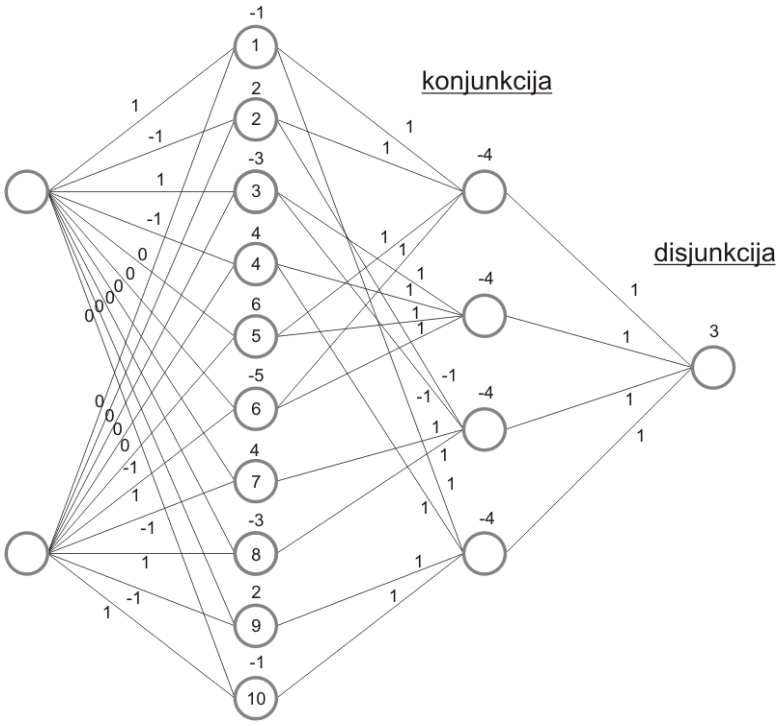
dakle ima 10 TLU perceptrona. Svaki taj perceptron odgovara jednom pravcu, i s ulazom *x* je povezan težinom w_1 tog pravca, s ulazom *y* je povezan težinom w_2 tog pravca, a prag mu je w_0.

Područje *a* definirano je kao ono područje koje se nalazi sa pozitivnih strana pravaca p_1, p_2, p_6 i p_7 (dakle, odgovarajući perceptroni svi moraju dati *1* na izlazu). Ovaj problem riješit ćemo uporabom novog TLU perceptrona koji će računati logičku I operaciju: uzorak pripada području *a* ako perceptron p_1 kaže da pripada I ako perceptron p_2 kaže da pripada I ako perceptron p_6 kaže da pripada I ako perceptron p_7 kaže da pripada. Logička funkcija I pripada funkcijama vrste *m-od-n*, gdje je funkcija I zapravo *n-od-n*, tj. daje izlaz *1* tek kada su svi ulazi *1*. Funkcija vrste *n-od-n* može se ostvariti kao TLU perceptron kome je prag jednak *–n*, a svi težinski faktori *+1* (za –1,1 TLU jedinice). Tako će prvi perceptron u drugom skrivenom sloju imati prag $w_0 = -4$, i biti će povezan sa perceptronima p_1, p_2, p_6 i p_7 s težinama *+1*. Na sličan se način definira i područje *b* kao područje za koje p_3, p_4, p_5 i p_6 daju na izlazu *1* što se rješava dodavanjem drugog perceptrona u drugi skriveni sloj s pragom *–4* i težinama *1* prema perceptronima p_3, p_4, p_5 i p_6. Područje *c* definira se kao ono za koje p_7 i p_8 kažu 1, ali p_2 i p_3 kažu *0*. Ovo ćemo riješiti opet I sklopom koji će na ulazima imati izlaze perceptrona p_7 i p_8 i komplemente izlaza perceptrona p_2 i p_3. Zbog toga ovaj treći perceptron u drugom skrivenom sloju opet ima prag *–4*, a preko težina 1 spojen je s izlazima perceptrona p_7 i p_8, dok je s težinama *–1* spojen s izlazima perceptrona p_2 i p_3. Ostalo je još područje 'd' koje je definirano kao ono područje za koje p_1, p_4, p_9 i p_{10} daju *1*, što se rješava četvrtim perceptronom u drugom skrivenom sloju koji ima prag *–4*, i spojen je težinama *1* s perceptronima p_1, p_4, p_9 i p_{10}.

Sada smo definirali četiri perceptrona od kojih će svaki imati na izlazu *1*, ako uzorak pripada području koje on nadgleda, a inače nula. Mrežu završavamo jednim perceptronom u završnom sloju koji će na izlazu dati *1* ako barem jedan od perceptrona iz drugog skrivenog sloja da *1*. Naime, uzorak pripada razredu *1* ako pripada bilo kojem području u kojem se nalaze uzorci koji pripadaju ovom razredu. Izlaz treba, dakle, biti *1* ako uzorak pripada području *a* ILI ako uzorak pripada području *b* ILI ako uzorak pripada području *c* ILI ako uzorak pripada području *d*. Funkciju ILI (vrste *1-od-n*) možemo realizirati tako da prag postavimo na *+(n-1)*, a sve težine na *+1*. Tako će naš izlazni perceptron imati prag *+3*, i s težinama *+1* biti će povezan sa svakim od 4 perceptrona drugog skrivenog sloja.

Slika 1.11.

Mreža za klasifikaciju sastavljena od TLU perceptrona



Konstruirana mreža prikazana je na slici 1.11. Provjerimo je li konstrukcija mreže valjana. Dovedimo na ulaz mreže uzorak (2.5, 3.5). To je uzorak koji pripada području *c* pa mu stoga treba dodijeliti razred *1*.

- *1. Korak.* Perceptroni ulaznog sloja imaju vrijednosti [2.5, 3.5].
- *2. Korak.* Prvi skriveni sloj tada će imati izlaze [1, -1, -1, 1, 1, -1, 1, 1, -1, 1]
- *3. Korak.* Drugi skriveni sloj tada će imati izlaze [-1, -1, 1, -1]
- *4. Korak.* Izlaz mreže je [1]

Vidimo da mreža ispravno klasificira dani uzorak. Slično se može ponoviti za neki drugi primjer i promatrati odziv mreže.

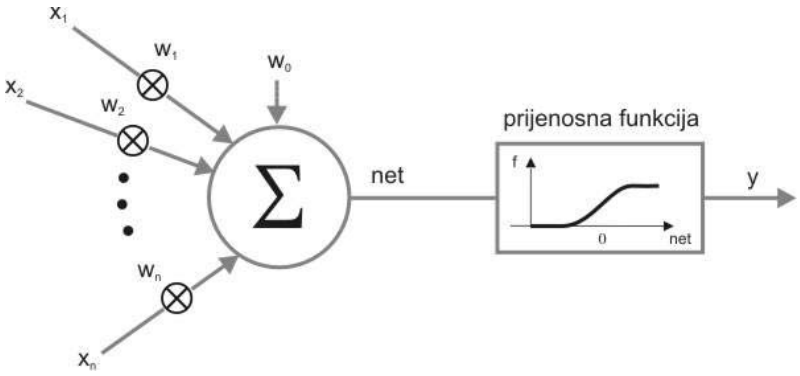
1.5.3. BACKPROPAGATION algoritam

Neuronska mreža s perceptronom kao procesnom jedinicom ipak može predstaviti jedino linearne odnose. Kako bi neuronska mreža mogla predstaviti visoko nelinearne funkcije, potrebno je da prijenosna funkcija njezinih procesnih elemenata i sama bude nelinearna funkcija svojih ulaza. Nadalje, radi primjene gradijentne metode pri postupku učenja mreže, potrebno je da prijenosna funkcija bude derivabilna funkcija težinskih faktora. Funkcija koja ispunjava oba navedena uvjeta je ranije spomenuta sigmoidalna funkcija (1.7). Stoga će višeslojna neuronska mreža sa sigmoidalnom funkcijom kao prijenosnom funkcijom procesnih elemenata biti u stanju predstaviti nelinearne odnose

ulaznih i izlaznih podataka. Procesni element prikazan je na slici 1.12. No kako naučiti takvu višeslojnu mrežu? Učinkovita i popularna metoda učenja višeslojnih mreža jest *algoritam sa širenjem pogreške unazad* - BACKPROPAGATION algoritam.

Slika 1.12.

Sigmoidalna jedinica



Algoritam koristi metodu gradijentnog spusta kako bi minimizirao nastalu pogrešku. Kod višeslojne mreže izlazni sloj može sačinjavati veći broj neurona, te je potrebno proširiti definiciju pogreške (1.31) za višestruke izlaze:

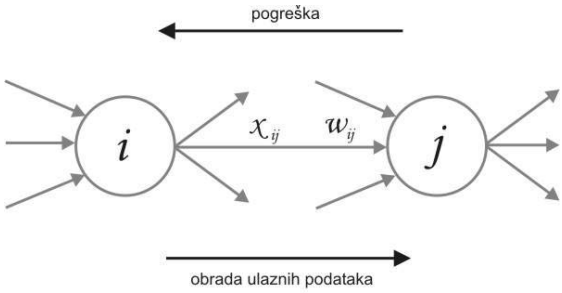
$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2 \qquad (1.48)$$

pri čemu su t_{kd} i o_{kd} ciljna i stvarna izlazna vrijednost za k-ti neuron izlaznog sloja dobivene s primjerom za učenje d.

Učenje višeslojne mreže pomoću BACKPROPAGATION algoritma svodi se na pretraživanje u n-dimenzionalnom prostoru hipoteza, gdje je n ukupan broj težinskih faktora u mreži. Pogrešku u takvom prostoru možemo vizualizirati kao hiper-površinu koja, za razliku od parabolične površine jednog procesnog elementa, može sadržavati više lokalnih minimuma. Zbog toga postupak gradijentnog spusta lako može zaglaviti u nekom lokalnom minimumu. U praksi se ipak pokazalo da algoritam unatoč tome daje vrlo dobre rezultate. Postoje razne tehnike izbjegavanja lokalnih minimuma. Jedna takva tehnika – uključenje momenta – opisana je u poglavlju 1.3.3.

Slika 1.13.

Povezani neuron



Algoritam BACKPROPAGATION pikazan je u nastavku. Prikazana je stohastička verzija algoritma za pojedinačno (engl. on-line) učenje. Korištena je slijedeća notacija: x_{ij} je ulaz s jedinice i u jedinicu j (izlaz jedinice i), w_{ij} je odgovarajuća težina, δ_n je pogreška izlaza jedinice n. Veličine su skicirane na

slici 1.13. Algoritam kao parametre uzima skup za učenje D, stopu učenja h, broj čvorova ulaznog sloja n_i, broj čvorova izlaznog sloja n_o i broj čvorova skrivenog sloja n_h. Mreži se predočavaju primjeri za učenje u obliku para $(\underline{x}, \underline{t})$ gdje je $\underline{x}$ vektor ulaznih vrijednosti a $\underline{t}$ vektor ciljnih izlaznih vrijednosti.

Algoritam nakon inicijalnog postavljanja težina u glavnoj petlji ponavlja predstavljanje sviju primjera mreži sve dok nije ispunjen uvjet zaustavljanja. Kao uvjet može poslužiti maksimalni dozvoljeni iznos pogreške dobivene obradom primjera iz skupa za učenje ili skupa za testiranje, zatim postupak se može zaustaviti nakon fiksnog broja iteracija ili epoha i sl. Uvjet zaustavljanja ključan je parametar jer premalo iteracija rezultira lošom obradbenom sposobnosti mreže dok preveliki broj iteracija dovodi do njezina pretreniranja.

Za svaki predstavljeni primjer računa se izlaz iz mreže na način da se signali prosljeđuju od ulaznih čvorova ka izlaznima te računa izlaz svakog pojedinog čvora. U ovoj fazi algoritma signali propagiraju unaprijed, od ulaznog sloja ka izlaznom. Na osnovi odstupanja stvarnog izlaza od ciljnog, računa se pogreška i ugađaju svi težinski faktori u cilju njezine minimizacije.

Stohastčki BACKPROPAGATION algoritam

Inicijaliziraj težinske faktore slučajne vrijednosti
Dok nije ispunjen uvjet zaustavljanja **čini**
  Za svaki $(\underline{x}, \underline{t})$ iz D **čini**
  Izračunaj izlaz o_u *za svaku jedinicu u*
  Za svaki izlazni čvor k izračunaj pogrešku d_k

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k) \tag{1.49}$$

  Za svaku skrivenu jedinicu izračunaj pogrešku

$$\delta_h \leftarrow o_h(1-o_h)\sum_{s\in Downstream(h)}\omega_{hs}\delta_s \tag{1.50}$$

  Ugodi svaki težinski faktor w_{ij}

$$\omega_{ij} \leftarrow \omega_{ij} + \Delta\omega_{ij}$$

  gdje je

$$\Delta\omega_{ij} = \eta\delta_j x_{ij} \tag{1.51}$$

  Kraj
Kraj

Zato što primjeri za učenje određuju ciljne vrijednosti samo izlaznog sloja neurona, poznata nam je jedino pogreška izlaznog sloja (1.49). Kako ugađati težinske faktore neurona u skrivenom sloju? BACKPROPAGATION algoritam računa pogrešku bilo kojeg skrivenog neurona h tako da zbraja pogreške d_s svih onih neurona s na koje utječe izlaz neurona h, uz dodatno množenje težinskim faktorom w_{hs}. Faktor ukazuje na to u kojoj je mjeri skriveni neuron h pridonio nastanku pogreške na izlazu jedinice s. Skup *Downstream(h)* jest skup svih neurona *nizvodno* od neurona h, tj. svi oni

neuroni čiji ulazi uključuju izlaz iz neurona *h*. U slučaju da je neuronska mreža slojevita, tada izraz (1.50) možemo napisati jednostavnije kao

$$\delta_h \leftarrow o_h(1-o_h)\sum_{s\in sloj\ m+1}\omega_{hs}\delta_s \tag{1.52}$$

gdje *m* sloj u kojem se nalazi jedinica *h,* a *m+1* je idući sloj u smjeru izlaznog sloja.

Računajući pogrešku svakog neurona, algoritam propagira pogrešku od izlaznog sloja ka ulaznome, dakle unazad kroz mrežu. Odatle i naziv BACKPROPAGATION algoritam. Zbog toga, strogo razmatrajući, ne možemo reći da se signali kod višeslojnih acikličkih mreža treniranih pomoću ovog algoritma rasprostiru samo unaprijed kroz mrežu. Oni se pri računanju pogreške prosljeđuju unazad. Mreža, međutim, opravdano može zadržati naziv *aciklička* (engl. feedforward) jer pod tim nazivom podrazumijevamo smjer kretanja samo ulaznih podataka.

Prikazana je stohastička verzija algoritma, što znači da se težine ugađaju postepeno nakon svakog predočenog primjera. Alternativni pristup, koji odgovara izvornoj gradijentnoj metodi, bio bi da se pogreške za svaki primjer zbrajaju pa tek onda ugađaju težinski faktori. Kod prvog načina govorimo o pojedinačnom, a kod drugog o grupnom učenju.

1.5.4. Interpretacija skrivenog sloja

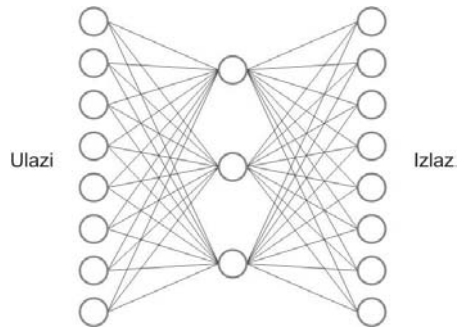
Znanje o obradi podataka pohranjeno je kod umjetne neuronske mreže u obliku različitih iznosa sviju težinskih faktora. Takvo implicitno znanje teško je interpretirati i predočiti ga čovjeku u obliku pravila. Konkretni primjeri pokazuju zanimljivo svojstvo BACKPROPAGATION algoritma koji je u stanju pronaći karakteristična obilježja ulaznih primjera koja nisu eksplicitno zadana, ali koja su bitna za pravilno učenje ciljne funkcije. Budući da algoritam ni na koji način nije ograničen pri postavljanju iznosa težina za skriveni sloj, težine će postaviti tako da minimiziraju izlaznu pogrešku.

Slika 1.14.

Mreža 8µ3µ8

(a)
Struktura mreže

(b)
Vrijednosti ulaza, izlaza i izlaza neurona skrivenog sloja nakon učenja



(a)

Ulaz	Skriveni sloj			Izlaz
10000000	0.996	0.025	0.128	10000000
01000000	0.011	0.981	0.994	01000000
00100000	0.996	0.996	0.995	00100000
00010000	0.883	0.992	0.003	00010000
00001000	0.172	0.005	0.002	00001000
00000100	0.003	0.009	0.751	00000100
00000010	0.836	0.003	0.995	00000010
00000001	0.003	0.921	0.069	00000001

(b)

Kao primjer neka posluži umjetna neuronska mreža sa slike 1.14a. Osam ulaznih neurona spojeno je na tri neurona skrivenog sloja, te zatim na ponovno osam neurona izlaznog sloja. Na ulaz mreže

dovodimo vektor $\underline{x}$ koji sadrži sedam nula i jednu jedinicu. Isti vektor dovodimo i na izlaz mreže. Mrežu, dakle, treniramo da nauči jednostavnu ciljnu funkciju $f(\underline{x})=\underline{x}$. Radi svoje strukture sa samo tri neurona u skrivenom sloju, mreža je prisiljena na neki način kodirati osam ulaznih vrijednosti koristeći tri neurona. Kako pokazuje slika 1.14b, izlazi neurona skrivenog sloja nakon jedne epohe učenja mreže odgovaraju nakon zaokruživanja na cjelobrojnu vrijednost upravo binarnom kôdu za osam različitih ulaznih vrijednosti.

2. FSIT - SIMULATOR I GENERATOR KÔDA

2.1. UVOD

Zajednička karakteristika vezana uz modele mekog računarstva jest potreba za intenzivnim eksperimentiranjem i simuliranjem, budući da razvijeni modeli pružaju vrlo mnogo stupnjeva slobode, pa je prije razvoja nekog specifičnog modela koji rješava određeni problem potrebno eksperimentalno odrediti niz parametara koji će osigurati da se model ponaša na željeni način. Teorijska osnova jednog od područja mekog računarstva (neuronske mreže) izložena je u poglavlju 1. Međutim, kako ovo područje iziskuje simulacijske metode, u ovom je radu predložen simulator modela mekog računarstva koji će biti opisan u nastavku.

Zbog svoje atraktivnosti neki od modela mekog računarstva već su određeno vrijeme prisutni i u popularnim alatima koji su namijenjeni rješavanju širokog spektra problema poput sustava *Mathematica* i *MatLab*. Međutim, veliki nedostatak ovakvih alata je njihova izuzetna kompleksnost i prilično dugo vrijeme potrebno za njihovo savladavanje prije no što se može krenuti na uporabu željenih modela. Isto tako, neki od alata implementiraju određene modele uz dosta ograničenja, pa iste takvim alatima nije moguće proširivati niti vršiti željene eksperimente.

Iz ovih razloga, tijekom kolegija "Neizrazito, evolucijsko i neuro-računarstvo" te kolegija "Strojno učenje" razvijen je simulator modela mekog računarstva (FSIT), koji na vrlo intuitivan način dozvoljava opisivanje modela, određivanje parametara modela (odnosno učenje potrebnih vrijednosti), uporabu modela lokalno na računalu, ili pak distribuirano preko mreže, kao i edukaciju korisnika, pri čemu simulator prilikom učenja određenih parametara može objašnjavati korisniku način na koji se ti parametri uče.

Isto tako, budući da ljudi uče puno lakše ukoliko se problem može prikazati grafički, simulator - gdje to ima smisla - nudi grafičke prikaze postupaka učenja kao i mogućnost animacije djelovanja različitih parametara na određene modele.

Prilikom izrade simulatora posebna pažnja posvećena je pitanjima poput:

- prenosivost kao uvjet za *učenje na daljinu* (engl. distance-learning),
- jednostavnost uporabe i
- dostupnost.

Kako bi se zadovoljili ovi zahtjevi, simulator je izveden kao web-orijentirana aplikacija (tzv. *Java applet*). Naime, u današnje doba jedini programski jezik koji je dovoljno rasprostranjen i podržan na

velikom broju različitih platformi (kako programskih, tako i strojnih) jest Java, zbog činjenice da se Java programi ne prevode u strojne instrukcije za određen tip procesora, već u instrukcije "virtualnom" Java procesoru koji je posebno napisan za svaku platformu. Na taj je način omogućeno izvođenja Java programa na širokom spektru platformi.

Izvedba simulatora kao Java applet također doprinosi jednostavnosti uporabe, budući da se simulator može pokretati direktno s mreže uporabom web-preglednika. Na ovaj je način korisnik simulatora oslobođen poslova poput instalacije programa, pribavljanja potrebnih datoteka i sl., jer taj cjelokupni posao za njega obavlja web-preglednik.

Treći bitan razlog izvedbe simulatora kao Java applet jest on-line dostupnost. Simulator postoji na Internetu na određenoj adresi i tamo je dostupan 24h dnevno, zbog čega korisnik simulatora ne mora cijeli simulator nositi sa sobom na svako računalo.

Da bi se omogućilo izvođenje što šireg spektra eksperimenata, simulator u potpunosti koristi prednosti web-orijentiranih aplikacija i dozvoljava uporabu u distribuiranom načinu, čime korisnički objekti koji koriste usluge simulatora mogu putovati Internetom neovisno o simulatoru.

Trenutno simulator podržava rad s neizrazitom logikom u širem smislu, kao i rad s perceptronima i neuronskim mrežama. Podržani su *adaline*, *adaline_r* (reducirani ADALINE o kojem je bilo riječ u poglavlju 1.3.1.), *tlu* te *sigmoidalni perceptroni*, *unaprijedne neuronske mreže* sa sigmoidalnim perceptronima, rad sa neizrazitom logikom u širem smislu, te sustavi neizrazitog upravljanja. Za potrebe ovog rada nama je bitan isključivo model neuronske mreže sa sigmoidalnim perceptronima.

Prilikom uporabe simulatora, korisnik simulator može koristiti u tri načina rada:

- Osnovni način
- Napredni način
- Distribuirani način

Kada se simulator koristi u osnovnom načinu rada, korisnik je ograničen isključivo na uporabu korisničkog sučelja koje nudi simulator. Pri tome se neke od funkcionalnosti korištenih modela gube, ali budući da je ovo najjednostavniji način rada sa simulatorom, korisnik ga može koristiti za edukativne svrhe. Naime, korisniku je omogućen opis svih modela kao i prikaz učenja određenih parametara. Korisnik također može eksperimentirati s animiranim prikazom djelovanja određenih parametara na modele.

Međutim, pred korisnika su postavljena i određena ograničenja. Budući da korisnik mora koristiti isključivo korisničko sučelje simulatora, očito je da nije u mogućnosti opisane modele koristiti na načine koji su inače mogući, ali ih korisničko sučelje ne dozvoljava. Npr. vrlo očita primjena perceptrona jest uporaba u svojstvu klasifikatora. Perceptron se najprije na temelju uzoraka za učenje istrenira, a potom koristi za klasifikaciju. U osnovnom načinu rada, simulator korisniku omogućava da opiše perceptron, i zada uzorke za učenje. Prilikom stvaranja modela, simulator koristeći zadane uzorke za učenje obavlja postupak učenja perceptrona i time ga pripremi za uporabu. Međutim, tu postupak staje, i u osnovnom načinu korisnik ne može koristiti perceptron za klasifikaciju.

Napredni i distribuirani način proširuju funkcionalnost simulatora dozvoljavajući korisniku da sam stvara svoje objekte u programskom jeziku Java, i zatim preko tih objekata maksimalno iskorištava simulator, odnosno sve modele koje simulator podržava. Detaljniji prikaz ovih modela biti će naveden u poglavlju 2.3.

2.2. EDUKACIJA

Bitan aspekt sustava FSIT jest demonstracija načela rada modela mekog računarstva u edukativne svrhe. Korisniku je omogućeno da, nakon što definira željeni model, na interaktivan i inutuitivan način prati kako on funkcionira. Napose za modele neizrazitog računarstva, kojima se u ovom radu bavimo, FSIT nudi korisniku učenje pojedinih paremetara modela uz istovremeno objašnjavanje samog postupka. Teorija time odmah biva poduprijeta odgovarajućim primjerom, kojeg je uvijek moguće pokrenuti nanovo te u različitim varijacijama. Web-orijentiranost ovog sustava i ovdje dolazi do izražaja: korisnik može pristupiti web stranici na kojoj se FSIT izvršava kao *Java applet*, izvršiti simulaciju modela i zatim pogledati dobivene rezultate.

Kako bismo demonstrirali edukacijske mogućnosti sustava FSIT, u nastavku je prikazan primjer modeliranja perceptrona. U primjeru se koristi dvoulazni TLU perceptron kojeg učimo metodom *Basic* (što je istovjetno metodi učenja opisanoj u poglavlju 1.4.1.).

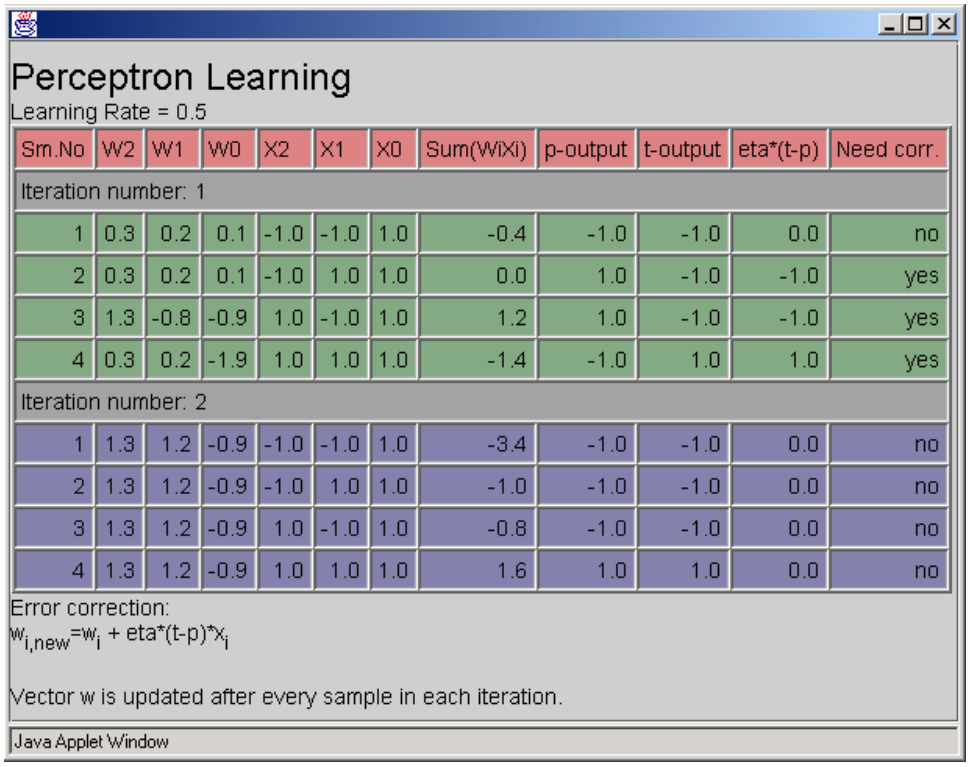
Opis modela perceptrona s uzorcima i parametrima za učenje

```
Perceptron p (
       SampleDimension(2),
       Samples((-1,-1,-1),(-1, 1,-1),( 1,-1,-1),( 1, 1, 1)),
       Type("TLU"),
       LearnMethod("Basic"),
       InputWeights(0.3,0.2,0.1),
       Options(
              IterationLimit=5, LearningRate=0.5, ProduceRaport=true,
              LearnNow=true
       )
);
```

Rezultati izvođenja ovog programa su tablični (slika 2.1) i grafički (slika 2.2.) prikaz učenja perceptrona iz koraka u korak, zajedno sa svim potrebnim parametrima, poput pogrešaka i sl.

Slika 2.1.

Tablični prikaz rezultata izvođenja

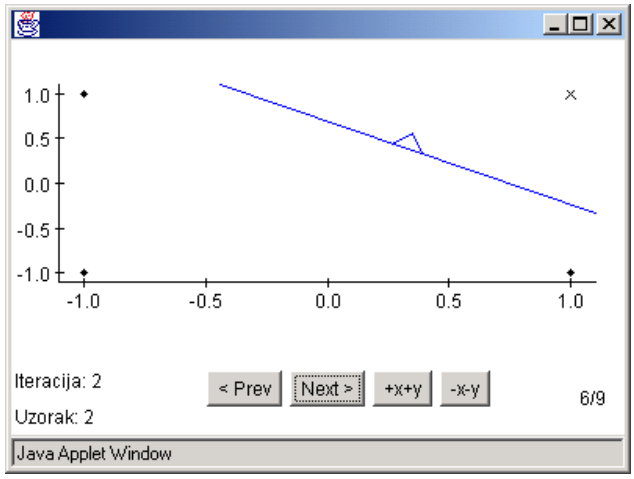


Sm.No	W2	W1	W0	X2	X1	X0	Sum(WiXi)	p-output	t-output	eta*(t-p)	Need corr.
Iteration number: 1											
1	0.3	0.2	0.1	-1.0	-1.0	1.0	-0.4	-1.0	-1.0	0.0	no
2	0.3	0.2	0.1	-1.0	1.0	1.0	0.0	1.0	-1.0	-1.0	yes
3	1.3	-0.8	-0.9	1.0	-1.0	1.0	1.2	1.0	-1.0	-1.0	yes
4	0.3	0.2	-1.9	1.0	1.0	1.0	-1.4	-1.0	1.0	1.0	yes
Iteration number: 2											
1	1.3	1.2	-0.9	-1.0	-1.0	1.0	-3.4	-1.0	-1.0	0.0	no
2	1.3	1.2	-0.9	-1.0	1.0	1.0	-1.0	-1.0	-1.0	0.0	no
3	1.3	1.2	-0.9	1.0	-1.0	1.0	-0.8	-1.0	-1.0	0.0	no
4	1.3	1.2	-0.9	1.0	1.0	1.0	1.6	1.0	1.0	0.0	no

Iz grafičkog prikaza sa slike 2.2. vidljivo je da su zadani uzorci međusobno linearno odvojivi (vidi sliku 1.7.). Kada to ne bi bio slučaj, postupak učenja perceptrona teoretski nikada ne bi završio, što je objašnjeno u poglavlju 1.4.1. Sustav FSIT tada bi zaustavio učenje perceptrona nakon što broj iteracija prekorači onaj zadan parametrom *IterationLimit* (u primjeru je to vrijednost 5).

Slika 2.2.

Grafički prikaz rezultata izvođenja



2.3. DISTRIBUIRANI NAČIN RADA

2.3.1. Proširivost sustava

Kako bi se prevladala ograničenja osnovnog načina rada, i korisniku omogućilo da opisane modele koristi u što širem spektru aplikacija, očito je da simulator mora ponuditi nekakav model proširivosti. Naime, čak i kada je riječ o programskim alatima, ne postoji jezik koji je najbolji za sve. Ako se, međutim, programeru omogući da dijelove svojih programa piše u različitim programskim jezicima, i osigura mehanizam povezivanja napisanih dijelova u jednu cjelinu, programer može iskoristiti najbolja svojstva svakog od jezika i time znatno olakšati razvoj željene aplikacije, i implementirati svojstva koja možda neki od jezika ne podržava. Imajući ovu ideju u vidu, simulator korisniku osim osnovnog načina rada nudi još dvije mogućnosti: napredni način i distribuirani način. Pri tome simulator u oba načina omogućava korisniku da razvije svoje vlastite objekte u programskom jeziku Java, i da te objekte koristi pri radu sa opisanim modelima. Naime, prilikom obrade programa zadanog od strane korisnika, simulator stvara internu reprezentaciju svakog od modela koji korisnik opiše.

Slika 2.3.

Model kako ga pamti simulator

```
model m(in x1, in x2, in x3, ..., out y1, out y2, out y3, ...)
{
 // complex program here which transforms
 // inputs x1, x2, x3, ..., xm to outputs y1, y2, y3, ..., yn
 // Example: neural network, fuzzy control system, ...
}
```

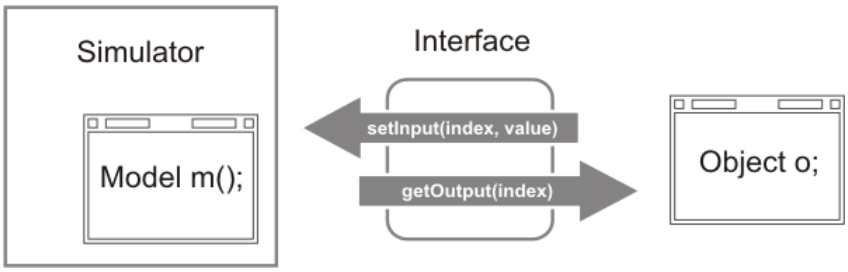
Nakon što ova faza završi, svi su modeli spremni za korištenje. Simulator interno za svaki od modela stvara i sučelje prema tom modelu. Mehanizmom sučelja simulator od korisnika skriva svu kompleksnost koja se krije u pozadini (poput načina na koji model transformira ulaze u izlaze, načina na koji se model uči i sl.), i po principu apstrakcije korisniku nudi korištenje modela kao crne kutije (slika 2.4.). Sve što korisnik mora znati o modelu jest da postoji *m* ulaza i *n* izlaza, te funkcije koje osigurava sučelje. Te funkcije nude mogućnost postavljanja određenog uzorka na ulaze crne kutije i dohvaćanje uzorka s izlaza crne kutije, pri čemu sav posao transformacije ulaza i izlaze obavlja simulator.

Slika 2.4.

Korisnički objekt model vidi kao crnu kutiju

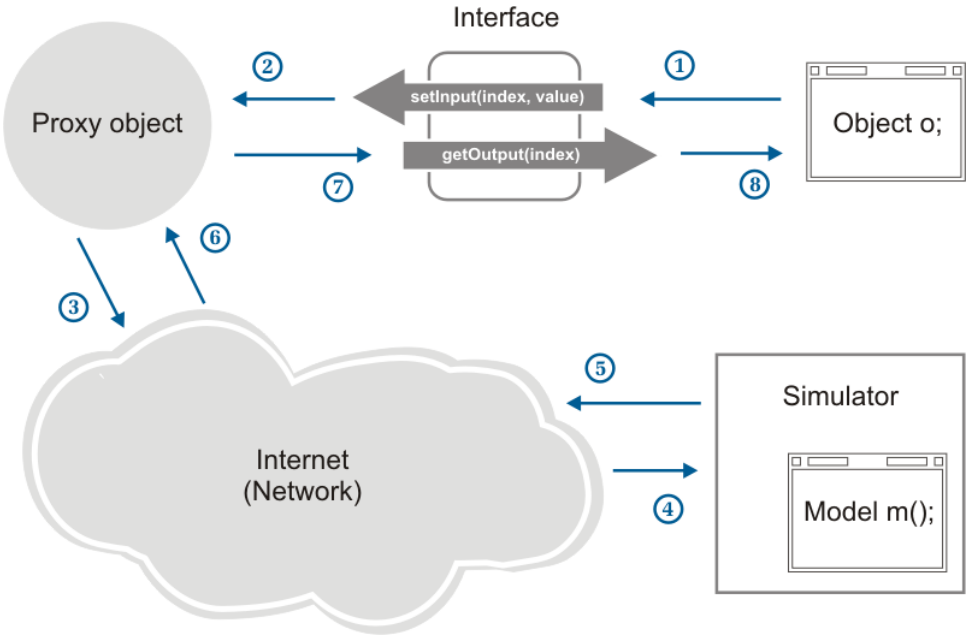
Slika 2.5.

Korisnički objekt 'o' model 'm' vidi kroz sučelje



Čitav simulator korisničkom se objektu 'o' predstavlja također preko sučelja. Naime, u trenutku kada simulator učita korisnički objekt 'o', objektu se predaje samo referenca na sučelje *FSITI*, što predstavlja sučelje prema simulatoru. Na ovaj način korisnički objekt 'o' i čitav simulator vidi kao crnu kutiju koja nudi neke njemu zanimljive funkcije. Sučelje *FSITI* korisničkom objektu 'o' nudi funkcije pomoću kojih korisnički objekt 'o' može dobiti sučelja prema modelima koji su trenutno opisani u simulatoru. Korištenjem tih funkcija korisnički objekt 'o' može dohvatiti sučelje 's' prema modelu 'm' koji ga interesira, i preko tako dobivenog sučelja 's' objekt 'o' može započeti sa korištenjem modela 'm' (slika 2.5.), prepuštajući simulatoru da odradi posao transformacije ulaza modela 'm' u odgovarajuće izlaze.

Korisnik se može odlučiti za korištenje simulatora u distribuiranom načinu (slika 2.6.). U tom slučaju simulator se pokreće kao server na računalu. Korisnički objekt 'o' tada se ne mora čak niti nalaziti na istom računalu, već može biti bilo gdje na mreži. U trenutku kada korisnički objekt 'o' zatreba usluge simulatora, objekt 'o' stvara *proxy objekt 'p'* predajući mu pri tome adresu gdje se nalazi simulator. Proxy objekt 'p' spaja se na zadanu adresu, i simulatoru prosljeđuje program koji simulator treba izvesti. Nakon toga, za objekt 'o' nema velike razlike u načinu korištenja simulatora u odnosu na napredni način (koji zahtjeva da se simulator i objekt nalaze na istom računalu, te koriste druge mehanizme komunikacije). Naime, objekt 'o' od proxy objekta 'p' zahtjeva usluge dobivanja sučelja prema određenim modelima, i zatim preko tih sučelja opet koristi modele. U ovom se slučaju sam proxy objekt 'p' brine za to da objektu 'o' vrati sučelje pomoću kojega će objekt 'o' moći koristiti model kao da je on lokalno dostupan, a same funkcije sučelja će se pobrinuti za to da odgovarajući pozivi budu preusmjereni mrežom do simulatora, i da se odgovarajući povratni rezultati mrežom prenesu do objekta 'o'. Ovo je moguće ostvariti zahvaljujući mogućnosti apstrakcije objekata koji je inherentno svojstvo Java programskog jezika u kojem je napisan i sam simulator. Detaljna specifikacija sučelja biti će navedena u nastavku, uz svaki od podržanih modela.



Slika 2.6.

Korisnički objekti simulator vide transparentno kroz proxy objekt (distribuirani način)

Npr. pretpostavimo da korisnički objekt 'o' želi pomoću perceptrona klasificirati određeni uzorak. Događa se slijedeći niz koraka:

- Sučelje prema modelu perceptrona korisnički objekt 'o' zahtjeva i dobiva od proxy objekta 'p' (sučelje s).
- Korisnički objekt 'o' poziva metodu sučelja 's' setInput kojom na ulaze perceptrona želi postaviti uzorak. (Slika 2.6., ①.)
- Umjesto do simulatora, poziv stiže do proxy objekta 'p' koji je na istom računalu kao i korisnički objekt 'o'. (Slika 2.6., ②.)
- Proxy objekt 'p' poziv preusmjerava u mrežu (npr. Internet) (Slika 2.6., ③.)
- Poziv stiže do simulatora na udaljenom računalu. Simulator postavlja zadani uzorak na ulaze perceptrona. (Slika 2.6., ④.)
- Simulator informaciju o uspješnosti obavljanja tražene akcije vraća u mrežu. (Slika 2.6., ⑤.)
- Proxy objekt 'p' s mreže preuzima informaciju. (Slika 2.6., ⑥.)
- Proxy objekt 'p' informaciju prosljeđuje prema korisničkom objektu 'o'. (Slika 2.6., ⑦.)
- Korisnički objekt 'o' preko sučelja 's' dobiva informaciju o uspješnosti obavljanja tražene akcije. (Slika 2.6., ⑧.)
- Korisnički objekt 'o' poziva metodu sučelja 's' kojom želi pročitati klasifikaciju, što rezultira ponovnim izvođenjem prethodnih 9 koraka (samo što je u pitanju druga metoda).

2.3.2. FSITI

Kako bi se osigurala što lakša metoda proširivanja funkcionalnosti simulatora, prethodno opisani napredni način kao i distribuirani način izvedeni su potpuno transparentno u odnosu na korisnički objekt 'o'. Naime, bilo da će objekt 'o' direktno komunicirati sa simulatorom na istom računalu, bilo da će se komunikacija odvijati preko mreže, objekt 'o' ponuđača usluga simulatora vidi preko zajedničkog sučelja *FSITI*, koje je opisano u nastavku. Kada korisnički objekt 'o' koristi napredni način rada, sučelje *FSITI* povezuje korisnički objekt 'o' i simulator direktno. Prilikom uporabe distribuiranog načina rada sučelje *FSITI* direktno povezuje korisnički objekt 'o' i proxy objekt 'p' koji se nalazi lokalno na istom računalu kao i korisnički objekt 'o', dok se za komunikaciju sa simulatorom brine sam proxy objekt 'p', o čemu korisnički objekt 'o' nema nikakvih informacija.

```
package hr.fer.zemris.dfsit;

public interface FSITI {
 public FSITUnaryFunctionI getUnaryFunctionByName(
   String name
 ) throws FSITCommError;
 public FSITPerceptronI getPerceptronByName(
   String name
 ) throws FSITCommError;
 public FSITMultiLayerPerceptronI getMultiLayerPerceptronByName(
   String name
 ) throws FSITCommError;
 public FSITFuzzyControlSystemI getFuzzyControlSystemByName(
   String name
 ) throws FSITCommError;
}
```

Ovaj mehanizam sučelja omogućava izradu korisničkih objekata koji se vrlo jednostavno mogu prilagođavati potrebnom načinu rada, bez da se mora modificirati veći dio koda objekta. Dodatna objašnjenja i specifikacije sučelja biti će navedena u nastavku.

2.4. IZRADA OBJEKTA ZA DISTRIBUIRANI NAČIN RADA

Prilikom rada u distribuiranom načinu rada, simulator se pokreće kao server na računalu X. Korisnički objekt 'o' se u tom slučaju može nalaziti na bilo kojem računalu Y koje je mrežom spojeno do računala X (alternativno, moguće je da svi budu na istom računalu).

Kada korisnički objekt 'o' želi koristiti usluge simulatora, objekt 'o' stvara primjerak proxy objekta 'p'. Proxy objekt implementiran je u klasi *fsitServerProxy*. Ova klasa sastavni je dio paketa *hr.fer.zemris.dfsit*. Proxy objekt 'p' korisnički objekt 'o' također vidi preko sučelja *FSITI*, pa baš kao i

kod rada u naprednom načinu korisničkom objektu 'o' stoje na raspolaganju svi modeli. Jedna od razlika je način na koji korisnik zadaje opis modela simulatoru. U naprednom načinu to se događa kroz korisničko sučelje simulatora. U distribuiranom načinu opis se zadaje prilikom stvaranja novog proxy objekta 'p', pa dobiveno sučelje *FSITI* omogućava rad s modelima koji su na taj način opisani simulatoru.

Primjer objekta koji se može koristiti u distribuiranom načinu rada prikazan je u nastavku.

```
import hr.fer.zemris.dfsit.*;

public class ObjectForDist {
 // definiraj ulaznu funkciju koju poziva Java
 public static void main( String argv[]) {
  try {
   // u varijablu "program" treba upisati program koji se inace
   // upisuje u simulator…
   String program = "…";
   // Kreirajmo proxy objekt i dohvatimo sucelje FSITI
   FSITI fsit =
     fsitServerProxy.newFsitServerProxy("www.somehost.com", 8020,
     program
   );
   // ovdje koristi sucelje FSITI za svoj rad
   // …
  } catch(FSITCommError e) {
   // … obradi pogresku
  }
 }
}
```

2.4.1. Primjena perceptrona u distribuiranom načinu

Kako bi omogućio korištenje modela perceptrona u naprednom i distribuiranom radu, simulator modele perceptrona nudi preko sučelja *FSITPerceptronI*. Koristeći sučelje *FSITI* koje korisnički objekt 'o' dobije od simulatora tijekom faza učitavanja i inicijalizacije, sučelje prema pojedinom modelu perceptrona 'o' može dobiti korištenjem metode:

```
public FSITPerceptronI getPerceptronByName(String name) throws
FSITCommError;
```

Argument metode je naziv modela perceptrona koji se želi dohvatiti, jer simulator zahtjeva da se svakom modelu koji se opisuje dodijeli naziv, upravo zbog ovakvih načina korištenja.

Sučelje *FSITPerceptronI* opisano je u nastavku. Sučelje pripada paketu *hr.fer.zemris.dfsit*, i definirano je na slijedeći način.

```
package hr.fer.zemris.dfsit;

public interface FSITPerceptronI {
 public double getResponseFor(double [] inputs) throws FSITCommError;
}
```

Sučelje definira metodu *getResponseFor*, koja kao argument prima ulazni uzorka koji se stavlja na ulaz perceptrona, pri čemu je uzorak zapisan u polju kao vektor. Metoda vraća vrijednost koja nastaje na izlazu perceptrona kao rezultat djelovanja ulaznog uzorka. U slučaju pogreške (npr. u distribuiranom načinu rada prilikom prekida veze) generirati će se iznimka *FSITCommError*.

Primjer izvedbe objekta za uporabu u distribuiranom načinu rada

```
import hr.fer.zemris.dfsit.*;

public class TestPercDist {
 // definiraj ulaznu funkciju koju poziva Java
 public static void main( String argv[]) {
  try {
   // u varijablu "program" treba upisati program koji se inace
   // upisuje u simulator…
   String program = "…";
   // Kreirajmo proxy objekt i dohvatimo sucelje FSITI
   FSITI fsit =
     fsitServerProxy.newFsitServerProxy("www.somehost.com", 8020,
     program
   );
   // Dalje ide uporaba kao da nista nije distribuirano…
   // dohvati mrezu pod nazivom "mreza1"
   FSITPerceptronI p =
     fsit. .getPerceptronByName("percX");
   // Neka je perceptron četveroulazni: 4D vektor (0.7, -0.3, 0.1, 0.5)
   double inputs[] = new double[4];
   inputs[0] = 0.7; inputs[1] = -0.3;
   inputs[2] = 0.1; inputs[3] = 0.5;
   // Posalji ulaze i dohvati izlaz perceptrona
   double output = perc.getResponseFor(inputs);
   // … obavi nesto sa rezultatom
  } catch(FSITCommError e) {
   // … obradi pogresku
  }
 }
}
```

2.4.2. Primjena neuronske mreže u distribuiranom načinu

Za uporabu u naprednom načinu rada simulator modele neuronskih mreža nudi preko sučelja *FSITMultiLayerPerceptronI*. Korisnički objekt 'o' koji od simulatora tijekom faza učitavanja i inicijalizacije dobije referencu na sučelje FSITI (preko kojega vidi simulator), uporabom tog sučelja koristeći metodu:

```
public FSITMultiLayerPerceptronI getMultiLayerPerceptronByName(String
name) throws FSITCommError;
```

može dohvatiti sučelje prema opisanom modelu, koristeći se pri tome nazivom modela koji je zadan u simulatoru. Jednom kada objekt 'o' dobije referencu na sučelje prema modelu, model neuronske mreže može se koristiti metodama koje definira sučelje *FSITMultiLayerPerceptronI*. Ovo sučelje opisano je u nastavku.

Sučelje *FSITMultiLayerPerceptronI* pripada *hr.fer.zemris.dfsit* paketu, i definirano je na slijedeći način.

```
package hr.fer.zemris.dfsit;

public interface FSITMultiLayerPerceptronI {
 public double[] getResponseFor(double [] inputs) throws FSITCommError;
}
```

Sučelje definira metodu *getResponseFor*, kojom se zadaje ulazni uzorak mreže, i dohvaća izlazni uzorak koji mreža generira kao odziv na zadani ulazni uzorak. U slučaju pogreške pri radu (npr. u distribuiranom načinu rada prilikom prekida veze) generirati će se iznimka *FSITCommError*.

Primjer izvedbe objekta za uporabu u distribuiranom načinu rada

```
import hr.fer.zemris.dfsit.*;

public class TestMLPDist {
 // definiraj ulaznu funkciju koju poziva Java
 public static void main( String argv[]) {
  try {
   // u varijablu "program" treba upisati program koji se inace
   // upisuje u simulator…
   String program = "…";
   // Kreirajmo proxy objekt i dohvatimo sucelje FSITI
   FSITI fsit =
     fsitServerProxy.newFsitServerProxy("www.somehost.com", 8020,
     program
   );
   // Dalje ide uporaba kao da nista nije distribuirano…
   // dohvati mrezu pod nazivom "mreza1"
   FSITMultiLayerPerceptronI mlp =
     fsit.getMultiLayerPerceptronByName("mreza1");
   // Neka je mreza dvoulazna: 2D vektor (0.7, -0.3)
   double inputs[] = new double[2];
   inputs[0] = 0.7; inputs[1] = -0.3;
   // Posalji ulaze i dohvati izlaze mreze
   double [] outputs = mlp.getResponseFor(inputs);
   // … obavi nesto sa rezultatima
  } catch(FSITCommError e) {
   // … obradi pogresku
  }
 }
}
```

2.5. GENERIRANJE C KODA

Problematika neuronskih mreža sa sobom nosi nekoliko zanimljivih detalja na koje treba obratiti pažnju. Neuronska mreža sama je po sebi vrlo složena struktura u kojoj su perceptroni međusobno povezani na različite načine. Ukoliko se radi o unaprijednim mrežama, mreže imaju svojstvo acikličnosti, i još neka ograničenja koja osiguravaju da se za učenje takvih mreža može primijeniti algoritam BACKPROPAGATION. Međutim, osim što je ovaj način učenja mreže numerički vrlo intenzivan i zbog toga spor, dodatno usporenje donosi svojstvo algoritma koje zahtjeva da se pojedini neuroni u mreži obrađuju točno zadanim redoslijedom koji je dinamički potrebno otkrivati (ovisi o samoj mreži). Zbog toga se postupak učenja mreže svodi na postupak pretraživanja pravog puta učenja, kao i postupak samog računanja.

Slijedeće loše svojstvo BACKPROPAGATION algoritma jest potreba za vrlo velikim brojem ponavljanja ulaznih uzoraka (nerijetko se broj iteracija mjeri tisućama, desetcima tisuća pa i više). Ovi svi razlozi jasno ukazuju da je učenje mreže BACKPROPAGATION algoritmom vrlo, vrlo sporo.

Konačno, BACKPROPAGATION algoritam kod mreža sa većim brojem slojeva inherentno je spor, zbog toga što se korekcijsko djelovanje širi od kraja mreže prema početku, a svaki prolaz kroz pojedini sloj reducira iznos korekcije, što dovodi do znatnog pada brzine učenja ukoliko mreža ima veći broj slojeva.

Imajući u vidu prethodno razmatranje, može se utvrditi da je programski jezik Java relativno loš izbor, jer je on dosta spoa (iako se u posljednje vrijeme radi i na tom problemu, pa su razvijeni JIT-ovi, tj. *Just In-Time Compilers*, koji prije izvođenja kod prevode na način da bude prilagođeniji ciljnoj platformi).

Kako bi se omogućila što šira podrška eksperimentiranju s neuronskim mrežama, simulator korisnicima nudi mogućnost da se na temelju opisa mreže koju korisnik zadaje simulatoru generira C kod koji implementira takvu neuronsku mrežu, i odabrani postupak učenja.

Budući da je najznačajniji nedostatak učenja neuronske mreže upravo brzina, prilikom generiranja C koda pažnja je posvećena upravo optimiranju koda kako bi se dobilo maksimalno ubrzanje.

Tako su poduzeti slijedeći koraci:

- *Izbjegnuta je uporaba polja.* Naime, poznato je da dohvaćanje elemenata polja sa sobom nosi i dodatnu aritmetiku (potrebno je proračunati na temelju indeksa elementa gdje se određeni element doista nalazi u memoriji), što bi usporilo proces učenja. Umjesto toga, svaki se parametar mreže definira kao zasebna varijabla što daje maksimalno ubrzanje prilikom dohvata, ali postavlja velike zahtjeve na prevodioce koda koji moraju biti u stanju raditi s tako velikim brojem varijabli.
- *Izbjegnuta je pretjerana uporaba petlji.* Naime, petlje se također ostvaruju određenim nizom instrukcija. Imajući u vidu da se određeni parametri mreže izračunavaju jednostavnim aritmetičkim operacijama, često izvođenje instrukcija petlje moglo bi bitno narušiti performanse postupka učenja. Ako se može staviti više instrukcija "učenja" prije no što se dođe do instrukcija petlje, efikasnost "učenja" u smislu povećanja brzine se povećava. Negativna posljedica izbjegavanja petlji je porast količine instrukcija programa, ali to je problem koji je poznat i od prije. Naime, prilikom izrade programa, programer je uvijek stavljen pred izbor memorijsko zauzeće nasuprot brzine izvođenja. Ukoliko želimo što manji kod kako bismo zauzeli što manje memorije, koristiti ćemo više petlji, i time postići smanjenje broja instrukcija. Negativna posljedica jest povećanje učestalosti instrukcija petlji, koje smanjuju brzinu rada programa. Ukoliko želimo postići što veću brzinu, moramo se pomiriti sa time da treba potrošiti i više memorije.

  Kako je osnovni zahtjev na generiranu mrežu bio što veća brzina, odlučeno je da se ne štedi na prostoru.

U nastavku će biti prikazane specifikacije jednostavne neuronske mreže i kod koji se generira pomoću simulatora.

Specifikacija neuronske mreže

```
// MultiLayerPerceptron struktura

MultiLayerPerceptron Mreza1 (
 SampleDimension(2),          // Dimenzija ulaznog vektora == broj ulaza
 Perceptrons(                 // Definicije svih perceptrona
  Perceptron("p1","Sigm"),    // Perceptron imena "p1", sigmoidalni
  Perceptron("p2","Sigm")     // Perceptron imena "p2", sigmoidalni
 ),
 Connections(                 // Definicije tko je spojen na koga
  // prvi ulaz spojen sa ... i tezinski faktor je ...
  (Input(1), "p1", 0.50 ), (Input(1), "p2", 0.40 ),
  // drugi ulaz spojen sa ... i tezinski faktor je ...
  (Input(2), "p1", 0.20 ), (Input(2), "p2", 0.60 ),

  // Ovo ne kaze da je perceptron spojen na samog sebe, vec
  // se ovime postavlja vrijednost praga (w0) za zadani perceptron
  ("p1", -0.3 ), ("p2", 0.1 )
 ),
 Outputs("p1","p2"),          // Perceptroni p7 i p8 su izlazni
 Samples(Dummy),              // Uzorci
 LearnMethod("Backpropagation"),   // Ucimo sa backpropagationom
 Options(                     // I malo opcija
  LearnNow=true,              // Odmah pokreni postupak ucenja
  LearningRate=0.3,           // Stopa ucenja je 0.3
  InertionRate=0.0,           // Stopa tromosi je 0.0
  ProduceRaport=false,        // Zelim izvjestaj
  RaportStyle=Code,           // Obican tekst a ne HTML stranicu
  Task=GenerateCCode,         // Zadatak: generiraj C kod
  GenerateCCode=true,         // Generiraj C kod
  AllowedError=0.01,          // Dopustiva pogreska je 0.01
  IterationLimit=1            // Ucenje smijes obaviti kroz 1 iteraciju
 )
);
```

Generirani opis neuronske mreže zajedno s BACK-PROPAGATION algoritmom

```
#include <math.h>
#include <stdio.h>

double inputs_0[2] = {
0.0, 0.0
};

double outputs_0[2] = {
0.0, 0.0
```

```
typedef struct {
  double p_2_1_output;
  double p_2_1_error;
  double w_2_1;
  double w_2_1_old;
  double w_1_1_2_1;
  double w_1_1_2_1_old;
  double w_1_2_2_1;
  double w_1_2_2_1_old;
  double p_2_2_output;
  double p_2_2_error;
  double w_2_2;
  double w_2_2_old;
  double w_1_1_2_2;
  double w_1_1_2_2_old;
  double w_1_2_2_2;
  double w_1_2_2_2_old;
  double p_1_1_output;
  double p_1_2_output;
} mlpnetwork;
mlpnetwork net1;

double *pinputs[1] = {
inputs_0
};

double *poutputs[1] = {
outputs_0
};

int numberOfSamples = 1;

long EpohLimit = 5;
long IterationLimit = 1;
long i1, e1;
double ukupna_pogreska;
double allowedError = 0.01;
double eta = 0.30000000000000004;
double eta_trom = 0.0;
double *inputs;
double *outputs;

double q;
int uzorak_br;
```

```
double getTotalError(void);
void IspisiFaktore(void);
void IspisiUlazeIzlaze(void);

void calcSample( int sampno, double *err, double *aoutputs );

void InitWeights(void) {

 net1.w_2_1=-0.30000000000000004;
 net1.w_2_1_old=net1.w_2_1;
 net1.w_1_1_2_1=0.5;
 net1.w_1_1_2_1_old=net1.w_1_1_2_1;
 net1.w_1_2_2_1=0.2;
 net1.w_1_2_2_1_old=net1.w_1_2_2_1;
 net1.w_2_2=0.1;
 net1.w_2_2_old=net1.w_2_2;
 net1.w_1_1_2_2=0.4;
 net1.w_1_1_2_2_old=net1.w_1_1_2_2;
 net1.w_1_2_2_2=0.6000000000000001;
 net1.w_1_2_2_2_old=net1.w_1_2_2_2;
}

void main(void) {

 InitWeights();
 learn();

}

void learn(void) {
 e1 = 1;
 while(1==1) {
  for( uzorak_br=0; uzorak_br < numberOfSamples; uzorak_br++) {
   // Postavi kao trenutni uzorak uzorak broj: uzorak_br
   inputs = pinputs[uzorak_br];
   outputs = poutputs[uzorak_br];
   i1 = 1;
   while(1==1) {
    net1.p_1_1_output = inputs[0];
    net1.p_1_2_output = inputs[1];
    net1.p_2_1_output = 1./(1.+exp(-(+net1.w_2_1+net1.p_1_1_output *
       net1.w_1_1_2_1+net1.p_1_2_output * net1.w_1_2_2_1)));
    net1.p_2_2_output = 1./(1.+exp(-(+net1.w_2_2+net1.p_1_1_output *
       net1.w_1_1_2_2+net1.p_1_2_output * net1.w_1_2_2_2)));
    net1.p_2_1_error = net1.p_2_1_output * ( 1 - net1.p_2_1_output ) *
       ( outputs[0]-net1.p_2_1_output );
    net1.p_2_2_error = net1.p_2_2_output * ( 1 - net1.p_2_2_output ) *
```

```
    ( outputs[1]-net1.p_2_2_output );

ukupna_pogreska = (outputs[0]-net1.p_2_1_output)*
          (outputs[0]-net1.p_2_1_output) +
          (outputs[1]-net1.p_2_2_output)*
          (outputs[1]-net1.p_2_2_output);

// Total sample error equals ukupna_pogreska.
if( ukupna_pogreska < allowedError ) break;
q = net1.w_1_1_2_1;
net1.w_1_1_2_1 += eta * net1.p_2_1_error * net1.p_1_1_output +
    eta_trom*(net1.w_1_1_2_1-net1.w_1_1_2_1_old);
net1.w_1_1_2_1_old = q;

q = net1.w_1_1_2_2;
net1.w_1_1_2_2 += eta * net1.p_2_2_error * net1.p_1_1_output +
    eta_trom*(net1.w_1_1_2_2-net1.w_1_1_2_2_old);
net1.w_1_1_2_2_old = q;

q = net1.w_1_2_2_1;
net1.w_1_2_2_1 += eta * net1.p_2_1_error * net1.p_1_2_output +
    eta_trom*(net1.w_1_2_2_1-net1.w_1_2_2_1_old);
net1.w_1_2_2_1_old = q;

q = net1.w_1_2_2_2;
net1.w_1_2_2_2 += eta * net1.p_2_2_error * net1.p_1_2_output +
    eta_trom*(net1.w_1_2_2_2-net1.w_1_2_2_2_old);
net1.w_1_2_2_2_old = q;

q = net1.w_2_1;
net1.w_2_1 += eta * net1.p_2_1_error + eta_trom * (net1.w_2_1-
   net1.w_2_1_old);
net1.w_2_1_old = q;

q = net1.w_2_2;
net1.w_2_2 += eta * net1.p_2_2_error + eta_trom * (net1.w_2_2-
   net1.w_2_2_old);
net1.w_2_2_old = q;

if((IterationLimit>0) && (i1>=IterationLimit)) {
 printf("Iteration limit reached: %ld epoha, %ld
          iteration.\r\n",e1,i1);
 break;
}
i1++;
}
```

```
  if((EpohLimit>0) && (e1>=EpohLimit)) break;
  e1++;
 }
 printf("Done after %ld epohs.\r\n",e1);
 ukupna_pogreska = getTotalError();
 printf("Ukupna pogreska za sve uzorke je %g.\r\n",ukupna_pogreska);
 IspisiFaktore();
 IspisiUlazeIzlaze();
}

void IspisiFaktore(void) {
   printf("net1.w_2_1=%g\r\n",net1.w_2_1);
   printf("net1.w_1_1_2_1 = %g\r\n", net1.w_1_1_2_1);
   printf("net1.w_1_2_2_1 = %g\r\n", net1.w_1_2_2_1);
   printf("net1.w_2_2=%g\r\n",net1.w_2_2);
   printf("net1.w_1_1_2_2 = %g\r\n", net1.w_1_1_2_2);
   printf("net1.w_1_2_2_2 = %g\r\n", net1.w_1_2_2_2);
}

void IspisiUlazeIzlaze(void) {
 for( uzorak_br=0; uzorak_br < numberOfSamples; uzorak_br++) {
    inputs = pinputs[uzorak_br];
    outputs = poutputs[uzorak_br];
    net1.p_1_1_output = inputs[0];
    net1.p_1_2_output = inputs[1];
    net1.p_2_1_output = 1./(1.+exp(-(+net1.w_2_1+net1.p_1_1_output *
       net1.w_1_1_2_1+net1.p_1_2_output * net1.w_1_2_2_1)));
    net1.p_2_2_output = 1./(1.+exp(-(+net1.w_2_2+net1.p_1_1_output *
       net1.w_1_1_2_2+net1.p_1_2_output * net1.w_1_2_2_2)));
    printf("Uzorak %d\r\n",uzorak_br);
    printf("Ulazi:        %.6lf,%.6lf\r\n",inputs[0],inputs[1]);
    printf("Tocni izlazi: %.6lf,%.6lf\r\n",outputs[0],outputs[1]);
    printf("Izlazi mreze:
              %.6lf,%.6lf\r\n",net1.p_2_1_output,net1.p_2_2_output);
 }
}

double getTotalError(void) {
 static double ukp;
 static int ubr;
 static double *inps;
 static double *outs;
 ukp = 0.; inps = inputs; outs = outputs;
 for( ubr=0; ubr < numberOfSamples; ubr++) {
    inputs = pinputs[ubr];
    outputs = poutputs[ubr];
```

```
    net1.p_1_2_output = inputs[1];
    net1.p_2_1_output = 1./(1.+exp(-(+net1.w_2_1+net1.p_1_1_output *
net1.w_1_1_2_1+net1.p_1_2_output * net1.w_1_2_2_1)));
    net1.p_2_2_output = 1./(1.+exp(-(+net1.w_2_2+net1.p_1_1_output *
net1.w_1_1_2_2+net1.p_1_2_output * net1.w_1_2_2_2)));
    net1.p_2_1_error = net1.p_2_1_output * ( 1 - net1.p_2_1_output ) * (
outputs[0]-net1.p_2_1_output );
    net1.p_2_2_error = net1.p_2_2_output * ( 1 - net1.p_2_2_output ) * (
outputs[1]-net1.p_2_2_output );
    ukp += ( outputs[0]-net1.p_2_1_output )*( outputs[0]-
net1.p_2_1_output ) + ( outputs[1]-net1.p_2_2_output )*( outputs[1]-
net1.p_2_2_output );
 }
 inputs = inps; outputs = outs;
 return ukp;
}

void calcSample( int sampno, double *err, double *aoutputs ) {
 static double *inps;
 static double *outs;
 inps = inputs; outs = outputs;
 inputs = pinputs[sampno];
 outputs = poutputs[sampno];
    net1.p_1_1_output = inputs[0];
    net1.p_1_2_output = inputs[1];
    net1.p_2_1_output = 1./(1.+exp(-(+net1.w_2_1+net1.p_1_1_output *
       net1.w_1_1_2_1+net1.p_1_2_output * net1.w_1_2_2_1)));
    net1.p_2_2_output = 1./(1.+exp(-(+net1.w_2_2+net1.p_1_1_output *
       net1.w_1_1_2_2+net1.p_1_2_output * net1.w_1_2_2_2)));
    net1.p_2_1_error = net1.p_2_1_output * ( 1 - net1.p_2_1_output ) *
       ( outputs[0]-net1.p_2_1_output );
    net1.p_2_2_error = net1.p_2_2_output * ( 1 - net1.p_2_2_output ) *
       ( outputs[1]-net1.p_2_2_output );
  if( err != NULL ) {
   *err = (outputs[0]-net1.p_2_1_output)*
              (outputs[0]-net1.p_2_1_output)+
              (outputs[1]-net1.p_2_2_output)*
              (outputs[1]-net1.p_2_2_output );
  }
  if(aoutputs!=NULL) {
   aoutputs[0]=net1.p_2_1_output;   aoutputs[1]=net1.p_2_2_output;
  }
 inputs = inps; outputs = outs;
}
```

Uporaba polja nije u potpunosti izbačena – polja se i dalje koriste za pohranu ulaznih uzoraka, i izlaznih klasifikacija, zbog toga što se na taj način osigurava lakša manipulacija uzorcima. S druge

strane, ulazi i izlazi se koriste samo u dijelovima izračuna koji su vezani uz vanjske slojeve, što kod većih višeslojnih mreža čini razmjerno mali dio ukupnog izračuna.

Sve težine i potrebni parametri mreže pohranjeni su u strukturu *mlpnetwork*. Definirano je i šest automatski generiranih funkcija, od kojih su tri vrlo bitne: `learn`, `getTotalError` i `calcSample`, a tri pomoćne: `IspisiFaktore`, `IspisiUlazeIzlaze` i `InitWeights`.

Funkcija `learn` obavlja postupak učenja koji je zadan prilikom generiranja mreže. Valja uočiti da funkcija ima samo tri petlje: prva beskonačna petlja ponavlja postupak učenja cijele mreže iz epohe u epohu, druga petlja u svakoj epohi dohvaća redom uzorke, a treća petlja ponavlja učenje dok pogreška uzorka ne padne ispod dozvoljene ili se dosegne zadano ograničenje (ova petlja posljedica je odabranog postupka učenja). Međutim, nikakvih petlji nema u računski najintenzivnijem dijelu – učenju jednog uzorka, tj. propagaciji ulazne vrijednosti prema izlazu, povrata pogreške unatrag i korigiranja težinskih faktora.

Funkcija `InitWeights` poziva se iz funkcije `learn` na početku učenja kako bi postavila početne vrijednosti za težinske faktore.

Funkcija `getTotalError` računa i vraća ukupnu pogrešku koju mreža radi nad cijelim skupom uzoraka za učenje.

Funkcija `calcSample` računa odgovor koji daje mreža za određeni uzorak.

Funkcija `IspisiFaktore` i `IspisiUlazeIzlaze` omogućavaju ispis na zaslon težinskih faktora, kao i uzoraka za učenje.

3. PRIMJER KLASIFICIRANJA NOVČANICA

3.1. O PRIMJERU

U ovom je poglavlju opisana konkretna implementacija unaprijedne neuronske mreže čiji je zadatak klasificiranje uzoraka. Implementacija mreže u potpunosti se oslanja kako na teorijski uvod iz prvog poglavlja, posebice BACKPROPAGATION algoritam kojim je mreža učena, tako i na opis simulatora modela mekog računarstva, kojim je mreža modelirana i programski ostvarena, a koji je dan u drugom poglavlju. U tom smislu, primjer koji slijedi svojevrsna je sinteza iznesene teorije i predloženog programskog rješenja u obliku alata FSIT. Već je rečeno da je to programsko rješenje vrlo općenito, pa tako uz neuronske mreže podržava i brojne druge modele mekog računarstva. U okviru ovog rada, međutim, zanimaju nas prvenstveno neuronske mreže. Nadalje, u okviru ovog primjera, zanima nas isključivo funkcionalnost FSIT-a kao generatora koda neuronskih mreža (poglavlje 2.5). Simulacija modela neuro-računarstva nije njime obuhvaćena i ona je spomenuta kao edukativni aspekt programa u poglavlju 2.2.

Primjer je programski ostvaren kroz dva međusobno odvojiva modula: prvi sačinjava kod neuronske mreže, generiran alatom FSIT, a drugi je korisničko sučelje mreže u koje je ugrađen i generator umjetnih uzoraka. Tako je korisniku na jednostavan i intuitivan način omogućeno dohvaćanje uzoraka i provjera rada prethodno trenirane mreže nad tim uzorcima.

3.2. POSTUPAK KLASIFIKACIJE

3.2.1. Zadatak mreže

Zadatak mreže jest klasifikacija četiri različite vrsta papirnatih novčanica. Skup primjera za učenje sačinjavaju novčanice od 5, 10, 20 i 50 kuna, svaka u četiri moguće varijante ovisno o orijentaciji (slika 3.1). Pojedina se novčanica može pojaviti na ulazu neuronske mreže bilo licem ili naličjem, kao što može biti i rotirana za 180 stupnjeva. Mreža, nakon što joj se na ulazu predoči odgovarajućim postupkom prethodno pripremljeni uzorak, na izlazu ima odrediti o kojoj je novčanici riječ. Klasifikacija mora biti neovisna o orijentaciji.

Slika 3.1.

Skup primjera za učenje

Zato jer su uzorci u stvari digitalizirane slike, oni će se međusobno razlikovati po intenzitetu korespondentnih slikovnih elemenata, što je posljedica razlike u stupnju istrošenosti novčanica, izgužvanosti papira, pomaku uzorka u *x* ili *y* smjeru u odnosu na otvor optičkog instrumenta i sl. Od neuronske se mreže očekuje da učini generalizaciju, pa da sve primjerke jedne te iste novčanice klasificira ispravno, neovisno o prisutnim varijacijama u intenzitetu komponenti pojedinih slikovnih elemenata. Od mreže se također zahtjeva određeni stupanj tolerancije na očekivana oštećenja novčanica, poput nečistoća, poderanosti i sl.

3.2.2. Priprema uzorka za klasifikaciju

Prije nego što se uzorak novčanice proslijedi ulazima neuronske mreže, potrebno ga je obraditi odgovarajućim postupcima. Digitalna slika u memoriji računala pohranjena je kao matrica slikovnih elemenata, a svaki je slikovni element opisan s više komponenti. Ovisno o primijenjenom modelu, te komponente mogu biti intenzitet crvene, zelene i plave boje (tzv. RGB model), zatim intenzitet krome i svjetline, ili neki drugi uobičajeni model. Najčešće se u elektroničkim optičkim uređajima, kao što su zaslon i skener, koristi RGB model. Kada bi uzorak bez prethodne obrade bio proslijeđen neuronskoj mreži onako kako je skeniran, onda bi odgovarajuća neuronska mreža u ulaznom sloju morala imati vrlo veliki broj neurona. Učenje i korištenje takve mreže bilo bi suviše složeno i sporo. Iz tog se razloga digitalnim postupkom obrade smanjuje količina informacije uzorka.

Postupak prikazuje slika 3.2. Novčanicu je najprije potrebno pretvoriti u digitalni oblik, što se ostvaruje skeniranjem na određenoj rezoluciji. Pretpostavimo da je skeniranje učinjeno skenerom u boji s rezolucijom od 300 točaka po inču i da je prosječna veličina novčanice 14x7 cm. Nakon tog postupka dobivena digitalna slika uzorka dimenzija je otprilike 1500x800 slikovnih elemenata. Slika se zatim nanovo uzorkuje na veličinu 45x22 slikovnih elemenata. Veličina je odabrana proizvoljno i, kako će se pokazati, dovoljno je mala da omogućava brzu klasifikaciju, a opet dovoljno velika da bi klasifikacija bila ispravna. Uzorkovanje se općenito može provoditi raznim algoritmima, npr. bikubičnim filtrom, ili jednostavno uprosječivanjem vrijednosti intenziteta skupine susjednih

slikovnih elemenata. Nakon što je digitalna slika smanjena na prihvatljivu veličinu, provodi se još i transformacija u crno-bijelu sliku. Općenito, ovaj korak može izostati i mrežu se može učiti tako da klasificira i na temelju boja. U tom slučaju sliku bi trebalo dodatno smanjiti ako želimo zadržati isti broj ulaznih neurona. U našem primjeru, međutim, u potpunosti smo odbacili informaciju o bojama pa će neuronska mreža uzorke vidjeti crno-bijelima. Transformacija boja također se može provoditi na različite načine, najjednostavniji od kojih je računanje srednje vrijednosti RGB komponenti slikovnog elementa. U primjeru je korišten nešto složeniji (i točniji) način transformacije putem izračuna komponente svjetline. Komponenta svjetline označava se s *Y* i uz dvije kromatske komponente *U* i *V* čini tzv. *YUV* model koji predstavlja čestu alternativu RGB modelu u mnogim primjenama obrade digitalne slike. Komponenta *Y* računa se prema formuli:

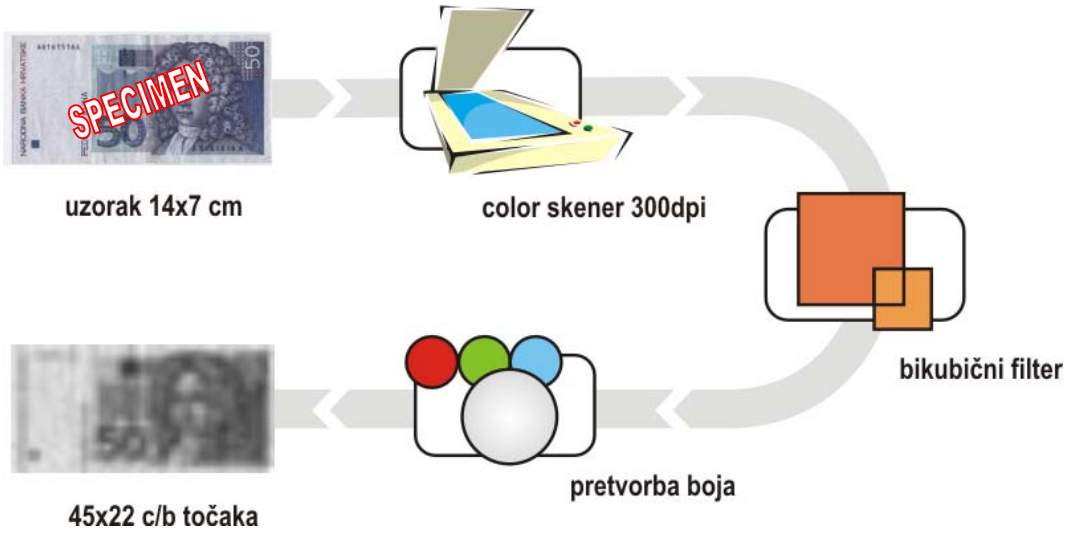
$$Y = 0.299\,R + 0.587\,G + 0.114\,B \tag{3.1.}$$

i daje vrijednosti u intervalu [0, 255], ako su vrijednosti R, G i B u istom intervalu.

Nakon ovog postupka dobivamo matricu slikovnih elemenata dimenzije 22 retka i 45 stupaca. Naposljetku linearnom transformacijom svodimo vrijednosti elemenata na vrijednosti iz intervala [0, 1], te ih takve možemo proslijediti neposredno na ulazne neurone mreže. Kako matrica ima ukupno 45x22=990 elemenata, mora i neuronska mreža u ulaznom sloju imati upravo 990 neurona.

Slika 3.2.

Postupak pripreme uzorka

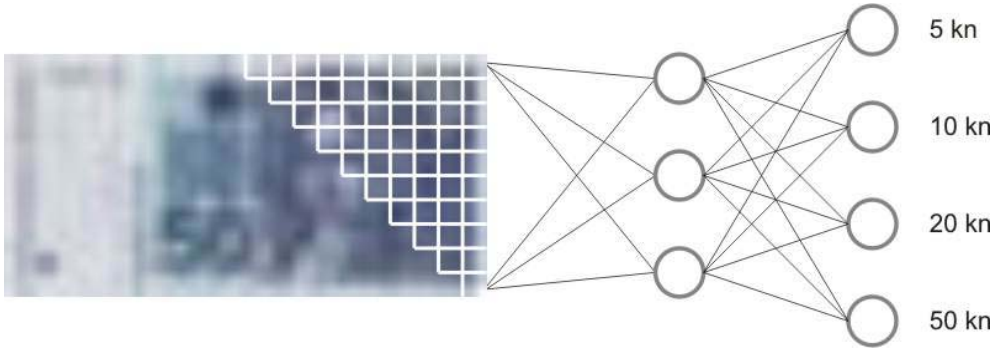


3.2.3. Arhitektura, parametri i učenje mreže

Mreža koja će klasificirati uzorke jest aciklička slojevita potpuno povezana mreža strukture 990×3×4. Četiri neurona izlaznog sloja određuju rezultat klasifikacije novčanice na način da se onaj neuron čiji je izlaz najveći proglašava *pobjednikom*. Ako je to prvi neuron, onda je na ulazu prvi primjer za učenje (novčanica od 5 kuna), ako je pobjednik drugi neuron onda se radi o novčanici od 10 kuna itd. Slika 3.3 prikazuje opisanu neuronsku mrežu.

Slika 3.3.

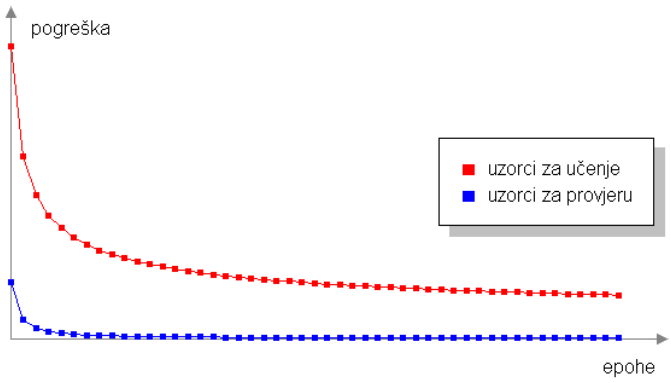
Umjetna neuronska mreža za klasifikaciju novčanica



Neuronska mreža učena je sa stopom učenja h=0.02 i momentom g=0.02. Svakih 2500 epoha rad mreže provjeren je nad skupom primjera za testiranje. Smanjivanje pogreške na izlazu iz mreže prikazuje slika 3.4. Učenje se provodilo tako da je za uzorak novčanice od 5 kn na izlaznom sloju samo izlaz prvog neurona postavljen na jedinicu, a svi ostali na nule. Za uzorak od 10 kn svi su neuroni postavljeni na nulu, osim drugog koji je postavljen na jedinicu, i tako redom za uzorak od 20 kn i 50 kn. Važno je uočiti da je mreža učena samo s primjerima iz skupa za učenje prikazanim na slici 3.1. Mreža nije učena s antiprimjerima, što znači da se za svaki uzorak doveden na ulaz pretpostavlja da je iz skupa 5, 10, 20 ili 50 kn. Kada uzorak ne bi bio iz tog skupa (primjerice neka strana valuta), on će unatoč tome biti klasificiran kao jedna od novčanica iz poznatog skupa. U tom slučaju ne možemo govoriti o pogrešnoj klasifikaciji, jer namjena konstruirane mreže nije dati odgovor na pitanje pripada li novčanica skupu za učenje ili ne.

Slika 3.4.

Pogreška mreže pri postupku učenja



Mrežu se, međutim, moglo trenirati i s antiprimjerima. Jedna je mogućnost da se za svaki predočeni antiprimjer kao izlaz zahtijeva vrijednost nula na svim neuronima izlaznog sloja, sugerirajući tako da predočeni uzorak ne odgovara niti jednoj poznatoj novčanici. Druga je mogućnost dodavanje petog neurona u izlazni sloj, koji bi, kada mu je izlazna vrijednost jednaka jedinici, upućivao na to je novčanica nepoznata. No u oba slučaja potrebno je skup primjera sa slike 3.1. proširiti antimprimjerima - a njih teoretski ima beskonačno mnogo. Kompromisno rješenje je proširivanje skupa za učenje samo onim antiprimjerima koji se opravdano mogu pojaviti na ulazu mreže (npr. postojeće strane valute ili novčanice koje su povučene iz opticaja).

Kada mreža nije učena s antiprimjerima, kao što je slučaj u ovoj implementaciji, dobar pokazatelj pouzdanosti klasifikacije može biti međusoban odnos vrijednosti neurona izlaznog sloja. Ako je, primjerice, vrijednost neurona pridruženog novčanici od 50 kn blizu jedinici, a vrijednost ostalih triju neurona blizu nuli, onda je sasvim sigurno riječ o uzorku od 50 kn. U situaciji kada ne postoji velika razlika u vrijednosti pobjedničkog neurona i preostalih neurona, klasifikacija uzorka manje je pouzdana i uzorak, ovisno o nekom usvojenom pragu tolerancije, može biti proglašen nepoznatim.

3.3. GENERIRANJE KODA MREŽE

Programski kod mreže generiran je alatom FSIT. U poglavlju 2.5. navedene su prednosti ovakvog načina programske implementacije mreže. Iako je navedeni primjer računski manje zahtjevan u fazi korištenja mreže, u fazi treniranja to nije slučaj i spomenute optimizacije postaju značajnima. Model mreže opisan u sustavu FSIT prikazan je u nastavku. Izvršavanje ovog modela rezultira generiranjem C koda. Dobiveni kôd vrlo je opsežan (veličinom nadmašuje 2 MB) i iz tog je razloga izostavljen.

Izvorni kod mreže za sustav FSIT

```
MultiLayerPerceptron Mreza1 (
 SampleDimension(990),
 Perceptrons(
  FullFFNetwork("Sigm",990,3,4)
 ),
 Connections(
  Random(-2.4,2.4)
 ),
 Outputs(default),
 Samples( Dummy ),
 LearnMethod("BackpropagationStohastic"),
 Options(
  LearnNow=true,
  LearningRate=0.3,
  InertionRate=0.0,
  RaportStyle=Code,
  Task=GenerateCCode,
  AllowedError=0.000001,
  IterationLimit=1000,
  EpohLimit=500
 )
);
```

3.4. GENERIRANJE UZORAKA

3.4.1. Prednosti i nedostaci generiranja uzoraka

Generator umjetnih uzoraka omogućava korisniku da na jednostavan način generira uzorke novčanica različitog stupnja oštećenja.. Nad tim uzorcima - koji predstavljaju imitaciju primjera kakvi bi se pojavljivali u stvarnoj primjeni ovakvog sustava - moguće je provjeriti ispravnost rada neuronske mreže. To može biti osobito korisno kada je skup primjera za učenje relativno malen, pa iz njega nije moguće izdvojiti skup primjera za provjeru kako je to opisano u poglavlju 1.2.4. Generator umjetnih uzoraka omogućava i praćenje ponašanja mreže ovisno o različitim obilježjima uzorka, npr. promatranje pouzdanosti klasifikacije u ovisnosti o istrošenost papira novčanice i sl. Na temelju takvih razmatranja moguće je definirati željeni prag tolerancije oštećenja uzoraka, kako je opisano u poglavlju 3.2.3. Prednosti korištenja generatora umjetnih uzoraka očigledno su slijedeće:

- dostupnost velikog broja uzoraka za provjeru,
- mogućnost promatranja odziva mreže u ovisnosti o promjeni samo jednog parametra uzorka,
- mogućnost definiranja praga tolerancije oštećenih uzoraka.

U primjeni, međutim, provjera rada mreže isključivo generiranjem uzoraka ne bi bila zadovoljavajuća. Umjetno generirane uzorke valjalo bi kombinirati s pravim uzorcima, i to iz slijedećih razloga:

- generirani uzorci nisu u potpunosti stohastičke prirode (generirana obilježja su pseudo-slučajna),
- generatorom se rijetko mogu obuhvatiti sva obilježja pravih uzoraka.

Nadalje, kvaliteta umjetno generiranih uzoraka u smislu njihova podudaranja s pravim primjerima ovisi o primijenjenom postupku generiranja. Postupak, primjerice, može biti previše pojednostavljen da bi rezultirao kvalitetnih uzorcima.

3.4.2. Generator uzoraka novčanica

Generator uzoraka ugrađen u ovaj primjer generira novčanice s različitim stupnjem oštećenja. Korisniku je omogućeno da preko grafičkog sučelja, opisanog kasnije, odredi stupnjeve pojedinih oštećenja. Generatorom je moguće odrediti:

- istrošenost papira novčanice,
- poderanosti papira,
- izgužvanost papira,
- stupanj nečistoće papira i
- išaranost papira.

Dodatno je moguće prikazati i naličje novčanice, rotirati je za 180 stupnjeva te dodati samoljepljivu traku. Ovako ostvareni generator pokriva u velikoj mjeri moguće stvarne primjere, no spomenuti nedostaci postupka i dalje vrijede. Čak štoviše, pri generiranju se koristi generator slučajnih brojeva s unaprijed zadanom početnom vrijednošću (engl. seed value). To je učinjeno namjerno, kako bi program uvijek generirao isti skup uzoraka.

Oštećenja novčanice stvaraju se postupcima obrade digitalne slike. Ona je pohranjena kao matrica triju komponenti dimenzija 367x182 slikovnih elemenata. Korišten je model YUV. Svako pojedino oštećenje vezano je implementacijski za jedan grafički filtar ili više njih. Pojedini filtar djeluje bilo na sve tri komponente slike ili samo na neke, kombinirajući slikovne elemente matrice uzorka sa slikovnim elementima pripremljenih matrica. Pripremljene matrice oponašaju strukturu raznih oštećenja (npr. mrlja, izgužvanosti), dobivene su obradom dijelova slike pravih novčanica i specifične su za svaki filtar. U nastavku ćemo razmotriti pojedine filtre.

Istrošenost novčanice generira se modifikacijom svih triju komponenti matrice uzoraka. Filtar funkcionira na način da smanjuje kontrast novčanice što je stupanj istrošenosti veći. Uz to se, prema formuli eksponencijalnog porasta s ograničenjem, povećava zatamnjenje novčanice (smanjivanje komponente *Y*). Izgužvanost papira generira se kombinacijom *Y* komponente matrice uzorka s istom komponentom pripremljene matrice. Kombinacija se ostvaruje na način opisan slijedećim izrazom

$$Y_S(i,j) = g\left(Y_S(i,j) + \frac{I}{50}\left(Y_P(i,j) - Avg(Y_P)\right)\right) \tag{3.2}$$

gdje $Y_s(i, j)$ označava *Y* komponentu elementa u matrici uzorka novčanice u stupcu *j* i retku *i*, $Y_p(i, j)$ označava analogno *Y* komponentu iz pripremljene matrice, a *I* označava stupanj istrošenosti novčanice u intervalu [0, 100]. Funkcijom $Avg(Y_p)$ računa se prosječna vrijednost svjetline pripremljene matrice. Funkcija je definirana kao

$$Avg(Y) = \frac{\sum_{m}^{i=1}\sum_{n}^{j=1} Y(i,j)}{m \cdot n} \tag{3.3}$$

pri čemu su *n* i *m* broj redaka odnosno stupaca matrice. Kako su vrijednosti komponenti iz intervala [0, 255], potrebno ih je nakon kombiniranja svesti u taj isti interval, što čini funkcija $g(x)$ definirana na slijedeći način:

$$g(x) = \begin{cases} 0, & za\ x < 0 \\ 255, & za\ x > 255 \\ x, & inace \end{cases} \tag{3.4}$$

Djelovanje filtra prikazano je na slici 3.5.

Slika 3.5.

Generiranje istrošenosti novčanice

Poderanost novčanice generira se na sličan način. Ovdje se, naravno, koriste druge pripremljene matrice. Kombinacija tih matrica s matricom uzorka odvija se također na način opisan izrazom 3.2, uz dodatnu modifikaciju kromatskih komponenti:

$$U_S(i,j) = g\left(U_S(i,j) + \frac{I}{5}(Y_P(i,j) - Avg(Y_P))\right)$$
$$V_S(i,j) = g\left(V_S(i,j) - \frac{I}{8}(Y_P(i,j) - Avg(Y_P))\right) \quad (3.5)$$

Ovom modifikacijom dijelovi uzorka postaju žućkasto-smeđi. Slika 3.6. prikazuje uzorak nakon što je na njemu primijenjen filtar za poderanost.

Slika 3.6.

Generiranje poderanosti novčanice

Filtrom za izgužvanost postiže se efekt izgužvanosti papira kakva nastaje kada se novčanica smota. Na papiru tada zbog presavijenosti nastaju vertikalne pruge. Izraz koji kombinira matricu uzorka s pripremljenom matricom jednak je izrazu 3.2. Djelovanje tog filtra prikazano je na slici 3.7.

Slika 3.7.

Generiranje izgužvanosti novčanice

Filtar za generiranje nečistoća ostvaren je tako da na različitim mjestima matrice uzorka vrši kombinaciju dijela te matrice s pripremljenim matricama. Odabir pozicija na kojoj se vrši kombinacija učinjen je tako da su rubne pozicije vjerojatnije od onih u sredini. Kombinacija se vrši na slijedeći način:

$$
\begin{aligned}
Y_S(i,j) &= g\left(\frac{Y_S(i,j)}{2} - \frac{3}{4}\left(Avg(Y_P) - Y_P(i,j)\right) + 100\right) \\
U_S(i,j) &= g\left(\frac{U_P(i,j) + 0.5 \cdot U_S(i,j)}{2} + 20\right) \\
V_S(i,j) &= g\left(\frac{V_P(i,j) + 0.5 \cdot V_S(i,j)}{2} + 40\right)
\end{aligned}
\tag{3.6}
$$

Najprije se smanjuje važnost komponente svjetline Y uzorka novčanice za faktor 2, čime se ispod mrlji postiže efekt zamućenja. Zatim se svjetlina kombinira sa svjetlinom pripremljene matrice, i naposljetku se dodatno povećava za iznos 100. Kromatske komponente U i V težinski se zbrajaju i usmjeravaju ka smeđoj boji. Intenzitet I ne pojavljuje se u izrazu 3.6., i o njemu ovisi ukupan broj ovakvih kombinacija matrica, tj. broj mrlji. Rezultat primjene ovog filtra prikazuje slika 3.8.

Slika 3.8.

Generiranje nečistoća na novčanici

Na isti način, s drugačijim pripremljenim matricama, funkcionira i filtar za generiranje išaranosti novčanice. Rezultat je prikazan na slici 3.9.

Slika 3.9.

Generiranje išaranosti novčanice

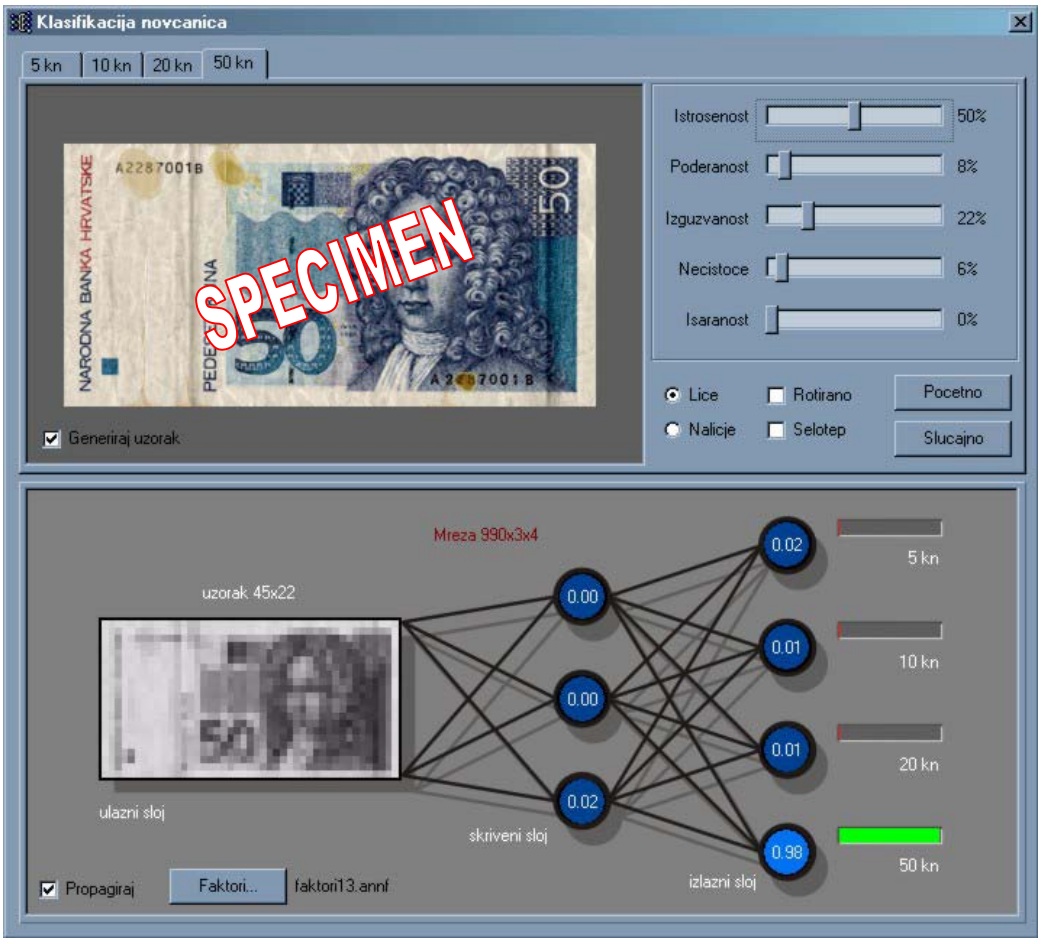
3.5. KORISNIČKO SUČELJE

Korisničko sučelje objedinjuje kod neuronske mreže, generiran pomoću alata FSIT, i generator umjetnih uzoraka. Izgled sučelja dan je na slici 3.10. U gornjem dijelu nalazi se generator uzoraka, a u donjem prikaz neuronske mreže koja se koristi za klasifikaciju.

Stupanj oštećenosti u opsegu od 0% do 100% korisnik odabire kliznim trakama. Ako je opcija *Generiraj uzorak* uključena, uzorak se odmah prosljeđuje generatoru uzoraka. Generator uzoraka primjenjuje filtre unaprijed određenim slijedom, te zatim vrši svu potrebnu pripremu uzorka, objašnjenu u poglavlju 3.2.2. (uzorkovanje i pretvorba boja). Skeniranje novčanica se ne provodi, jer one početno već postoje pohranjene u memoriji. Pohranjene novčanice preuzete su sa stranice Hrvatske narodne banke (http://www.hnb.hr). Za svaku je pojedinu novčanicu moguće nezavisno mijenjati stupanj oštećenja, a on se može postaviti i slučajno.

Slika 3.10.

Sučelje programa za generiranje umjetnih uzoraka i njihovu klasifikaciju



Prije prosljeđivanja generiranog uzorka neuronskoj mreži, potrebno je učitati težinske faktore mreže. Oni su pohranjeni u **.annf* datoteci za svakih 2500 epoha učenja mreže, kako je rečeno u poglavlju 3.2.3. Najbolji rad mreže postiže se, dakako, s onim faktorima koji minimiziraju pogrešku

na izlazu na skupu primjera za učenje i na skupu provjeru (vidi sliku 3.4.). Ti faktori pohranjeni su u datoteci *faktori_1_20.annf*. Rad mreže može se isprobati i na drugim pohranjenim faktorima (priloženima na CD-u), te tako promatrati kako se mijenjaju performanse mreže u zavisnosti o faktorima.

Kada su težinski faktori mreže učitani, odabirom opcije *Propagiraj* uzorak se prosljeđuje na ulaz neuronske mreže. S obzirom na to da je uzorak prošao prethodnu pripremu, mreža ga vidi u rezoluciji od 45x22 crno-bijela elementa, i uzorak je tako prikazan na njezinu ulazu. Zato jer je mreža razmjerno jednostavna, informacije se kroz nju propagiraju gotovo trenutačno, i tako na izlazu odmah dobivamo klasifikaciju uzorka. Ispisuje se i izlazna vrijednost pojedinih neurona, koji su tim svjetliji što je vrijednost bliža jedinici. Vrijednosti neurona izlaznog sloja prikazane su još i grafički.

4. ZAKLJUČAK

U ovom radu primjerom su obuhvaćene tri programske implementacije: simulator neuronskih mreža, generator C kôda neuronskih mreža opisanih u simulatoru i generator umjetnih uzoraka temeljenih na stvarnim uzorcima.

Kroz primjer je jasno prikazana robusnost neuronskih mreža, o čemu je govora bilo u teorijskom dijelu. Zadatak ostvarene neuronske mreže bio je raspoznavati predočene novčanice (Hrvatske kune) i klasificirati ih u jednu od četiri unaprijed definirana razreda. Razvijena mreža sastoji se od tri sloja: ulaznog sloja koji ima 990 neurona, skrivenog sloja koji ima 3 neurona i izlaznog sloja koji ima 4 neurona. Zamišljeno je da mreža radi na principu glasanja: svaki od izlaznih neurona zadužen je za jedan tip novčanice. Onaj neuron koji ima najveći izlaz određuje o kojoj se novčanici radi.

Prilikom učenja mreža je trenirana s dva primjerka svake novčanice, pri čemu je novčanica mogla biti postavljena u jedan od četiri vodoravna položaja, što ukupno daje 32 uzorka za učenje. Potrebno je naglasiti da su mreži prilikom učenja predočavane isključivo neoštećene novčanice (dakle bez nečistoća, išaranih dijelova i sl.).

Neuronska mreža koja je korištena za raspoznavanje novčanica najprije je opisana u sustavu FSIT. Sustav FSIT, osim što omogućava rad s modelima mekog računarstva (pa tako i s neuronskim mrežama), ima ugrađenu mogućnost generiranja optimiranog C kôda (pri čemu se optimira brzina izvođenja) koji obavlja funkciju opisane mreže. Prednosti tog postupka su značajno ubrzavanje, kako rada mreže, tako i samog postupka učenja koji je inače inherentno spor uslijed velikog broja izračuna koje je potrebno obaviti. Izvorni C kôd mreže generiran je sustavom FSIT. Tako dobivena mreža učena je s novčanicama koje smo preuzeli s Internet stranice Hrvatske narodne banke.

Kako bismo provjerili kako se mreža ponaša pri radu sa "stvarnim" novčanicama (dakle onima koje su neko vrijeme bile u opticaju i uslijed toga postale zgužvane, išarane, nečiste i sl.) napisan je program – *generator uzoraka* – koji neoštećenu novčanicu u željenoj mjeri oštećuje umjetnim postupcima. Prednost ovakvog generatora također leži u činjenici da često nije moguće unaprijed dobiti veliki broj uzoraka koji bi pokrivali sve interesantne primjere oštećenja novčanica. Provjeravali smo ponašanje mreže nad umjetno generiranim uzorcima. U skladu s našim očekivanjima i izloženom teorijom, mreža se pokazala izuzetno robusnom. Naime, tako dugo dok je stupanj takvih oštećenja bio unutar razumnih granica (odnosno dok je novčanica bila razmjerno prepoznatljiva), neuronska je mreža raspoznavanje obavljala s visokim stupnjem pouzdanosti.

LITERATURA

[1] T. M. Mitchell, *Machine Learning*. The McGraw-Hill Companies, Inc., 1997.

[2] R. S. Michalski, I. Bratko, M. Kubat, *Machine Learning And Data Mining*. John Wiley & Sons Ltd., 1998.

[3] P. Picton, *Neural Networks*. PALGRAVE, 1994.

[4] B. Dalbelo Bašić, Bilješke s predavanja. Fakultet elektrotehnike i računarstva, Zagreb, 2001.

[5] K. Gurney, "Computers and Symbols versus Nets and Neurons". Dept. Human Sciences, Brunel University, Uxbridge, 2001.

[6] K. Gurney, "Drawing things together – some perspectives". Dept. Human Sciences, Brunel University, Uxbridge, 2001.

[7] D. Mišljenčević, I. Maršić, *Umjetna inteligencija*. Školska knjiga, Zagreb, 1991.

[8] *Automata Theory*. Encyclopaedia Britannica Inc., 2001 CD-ROM Edition.

[9] D. Werner et al., *Taschenbuch der Informatik*, Fachbuchverlag Leipzig, 1995.

[10] J. Šribar, B. Motik, *Demistificirani C++*. Element, Zagreb, 2001.

[11] S.Haykin, Neural Networks, Comprehensive Foundation, Second Edotion, Prentice Hall, New Jarsey, 1999.

[12] J. C. Principe, N.R. Euliano, W. C. Lefebvre, *Neural and Adaptive Systems - Fundamentals Through Simulations*, John Wilwy & Sons, 2000.

[13] G. Booch, *Object-oriented analysis and design with applications*, Second edition, Addison Wesley, 1994.

[14] J. Cowell, *Books Essential Java 2 Fast : How to Develop Applications and Applets With Java 2 (Essential Series)*, Springer Verlag, ISBN: 1852330716, August 1999.

[15] Internet stranice Hrvatske narodne banke, http://www.hnb.hr, travanj 2002.

PRILOG

Na CD-u prilažemo:

- **FSIT**

  Za pokretanje programa potrebno je u web-pregledniku otvoriti datoteku *simulator/fsitprog.html*. Preglednik mora podržavati *Java 2.0*.

- **Klasifikacija novčanica**

  Program je potrebno instalirati pokretanjem datoteke *ann/setup.exe*. Program se zatim pokreće iz *Start* izbornika.