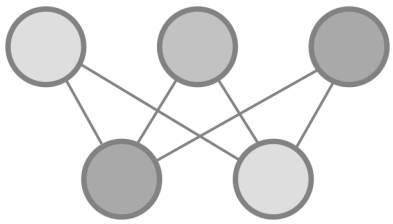
Umjetna inteligencija
Zavod za elektroniku, mikroelektroniku i inteligentne sustave
Fakultet elektrotehnike i računarstva

Umjetne neuronske mreže

prof.dr.sc. Bojana Dalbelo Bašić

mr.sc. Marko Čupić

mr.sc. Jan Šnajder



Zagreb, svibanj 2008.

1 Uvod u neuro-računarstvo 4

2 Umjetna neuronska mreža 7

2.1 Neuron: biološki i umjetni 7
2.2 Definicija i osobitosti umjetne neuronske mreže 8
2.3 Vrste umjetnih neurona 9
2.4 Postupak učenja mreže 12

3 ADALINE 15

3.1 Učenje ADALINE jedinice 15
3.2 Konvergencija LMS-a 22
3.3 Popravljanje svojstava konvergencije 27
3.4 ADALINE u punoj snazi 28

4 TLU perceptron 31

4.1 Pravilo perceptrona 31
4.2 Delta pravilo 32

5 Višeslojne neuronske mreže 35

5.1 Struktura mreže 35
5.2 Višeslojna mreža perceptrona 36
5.3 BACKPROPAGATION algoritam 40
5.4 Interpretacija skrivenog sloja 43

6 Primjer: raspoznavanje novčanica 44

Dodatak A. Java programi za gradijentni spust 46

Literatura 49

1 Uvod u neuro-računarstvo

Iako danas većinu podataka koje ne obrađujemo snagom vlastita uma obrađujemo digitalnim računalom, potpuno je pogrešna 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 iz sekunde u sekundu 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.

Području umjetne inteligencije inherentan je cilj ostvarenje imitiacije ljudskog mozga, odnosno njegovo pretakanje u umjetan oblik, jer ljudski mozak sadrži maksimum inteligencije koju poznajemo (odnosno ili danas ili 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 kocept 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 li 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 potizanja generalizacije.

Tablica 1.2. Usporedba karakteristika paradigmi

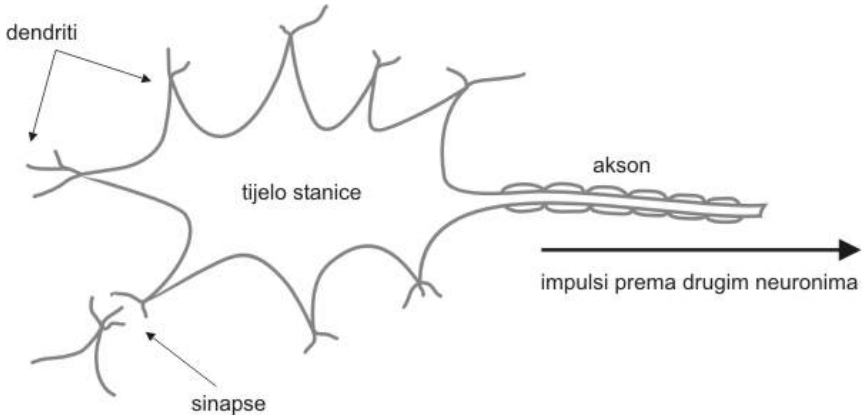
von Neumann	neuronska mreža
Računalu se unaprijed detaljno mora opisati algoritam u točnom slijedu koraka (program)	Neuronska mreža uči samostalno ili s učiteljem
Podaci moraju biti precizni – nejasni ili neizraziti podaci ne obrađuju se adekvatno	Podaci ne moraju biti precizni (gotovo uvijek su neprecizni)
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

2 Umjetna neuronska mreža

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 cijevčice koje prenose električke poruke) i niza završnih članaka. Slika 2.1 prikazuje građu neurona.

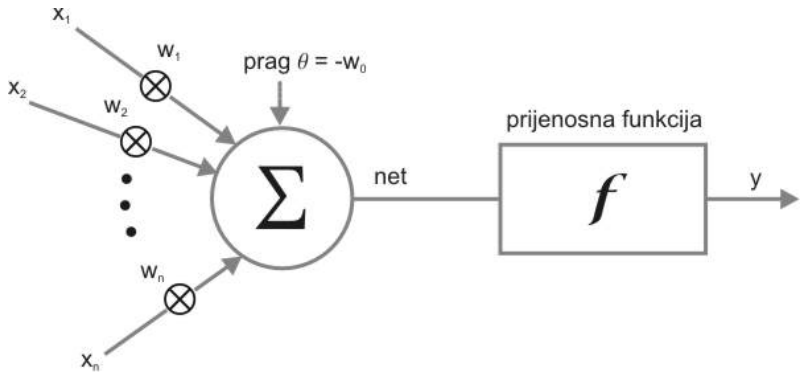
Tijelo stanice sadrži informaciju predstavljenu električkim potencijalom između unutrašnjeg i vanjskog dijela stanice (oko –70 mV u neutralnom stanju). Na sinapsama, spojnom sredstvu dvaju neurona kojim su pokriveni dendriti, primaju se informacije od drugih neurona u vidu post-sinaptičkog potencijala koji utječe na potencijal stanice povećavajući (hiperpolarizacija) ili smanjivajući 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 potenijala, proizvode i otpuštaju kemikalije, tzv. neurotransmitere. To zatim ponovno inicira niz opisanih događaja u daljnjim neuronima. Propagacija impulsa očigledno je jednosmjerna.



Slika 2.1. 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 2.2. U nastavku ćemo za pojam *umjetni neuron* ravnopravno koristiti i istovjetne pojmove: *procesni element* (PE), *čvor* ili *jedinica*.



Slika 2.2. Umjetni neuron

Ulazne signale, njih ukupno *n*, označavamo sa $x_1, x_2, \ldots x_n$. Težine označavamo sa $\omega_1, \omega_2, \ldots \omega_n$.

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

$$net = \omega_1 x_1 + \omega_2 x_2 + \ldots + \omega_n x_n - \theta \tag{2.1}$$

ali se zbog kompaktnosti često dogovorno uzima da je vrijednost praga $\theta = -\omega_0$ te se dodaje ulazni signal x_0 s fiksiranom vrijednošću 1, pa pišemo jednostavnije

$$net = \omega_0 x_0 + \omega_1 x_1 + \omega_2 x_2 + \ldots + \omega_n x_n = \sum_{i=0}^{n} \omega_i x_i \tag{2.2}$$

dok je izlaz *y* rezultat prijenosne funkcije

$$y = f(\sum_{i=1}^{n} \omega_i x_i) = f(net) \tag{2.3}$$

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 klimava jer uz mnoga učinjena pojednostavljena postoje još mnogi fenomeni živčanog sustava koji nisu modelirani umjetnim neuronskim

mrežama, kao što postoje i karakteristike umjetnih neronskih 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 nelineranih 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 konvencionalnih metoda koje pretpostavljaju normalnu raspodjelu obilježja u ulaznim podacima.
- Stvaraju vlastite odnose između podataka koji nisu zadani na ekplicitan simbolički način.
- Mogu raditi s velikim brojem varijabli ili parametara.
- Prilagodljive su okolini.
- Moguća je 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 obriza 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,
- obrada slike,
- obrada govora,
- problemi optimizacije,
- nelinearno upravljanje,
- obrada nepreciznih i nekompletnih podataka,
- simulacije i sl.

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.

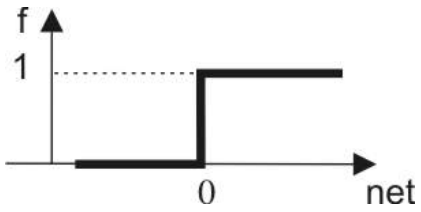
Najjednostavnija moguća aktivacijska funkcija je

$$f(net) = net \tag{2.4}$$

Takva je funkcija svojstvena modelu umjetnog neurona ADALINE (Adaptive Linear Element). Izlaz iz takve jedinice upravo je, dakle, težinska suma njegovih ulaza. Druga je

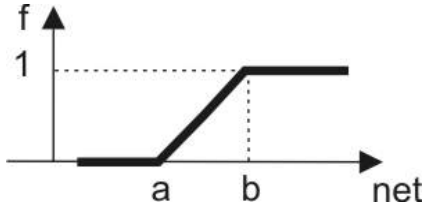
mogućnost korištenje funkcije skoka ili praga (engl. threshold function, hard-limiter) čime dobivamo procesnu jedinicu koja daje Booleov izlaz (engl. Threshold Logic Unit, TLU):

$$f(net) = \begin{cases} 0 & za\ net < 0 \\ 1 & inace \end{cases} \tag{2.5}$$



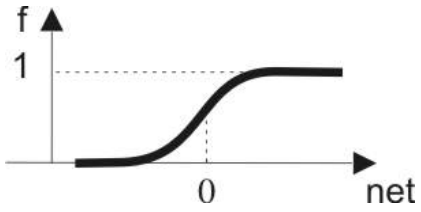
pri čemu znak nejednakosti može prema potrebi uključiti i jednakost. Prijenosna funkcija može biti definirana po dijelovima linearno:

$$f(net) = \begin{cases} 0 & za\ net \le a \\ net & za\ a < net < b \\ 1 & za\ net \ge b \end{cases} \tag{2.6}$$



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:

$$f(net) = \frac{1}{1 + e^{-a \cdot net}} \tag{2.7}$$



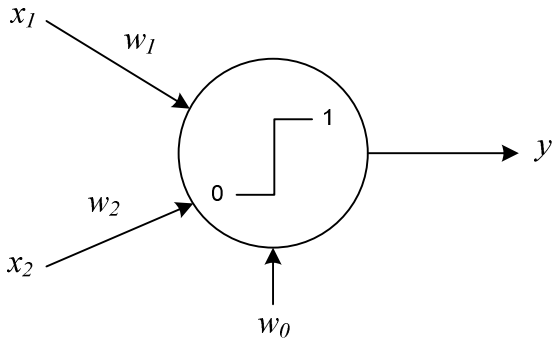
uz parametar *a* koji određuje nagib funkcije. Sigmoidalna funkcija ponekad se naziva i *logističkom funkcijom*.

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).

Primjer 1

Zadan je TLU perceptron prikazan slikom. Težine pojedinih ulaza su: w_0 = -1, w_1 = 2, w_2 = 1. Ako se na ulaz dovede uzorak $x = (x_1, x_2) = (-4, 2)$, odredite izlaz neurona.



Rješenje

Izračunajmo najprije koliko iznosi težinska suma *net* prema 2.2:

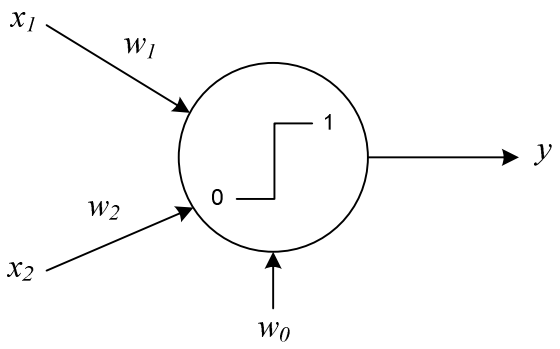
$$net = \sum_{i=0}^{2} \omega_i x_i = \omega_0 + \omega_1 x_1 + \omega_2 x_2 = -1 + 2 \cdot (-4) + 1 \cdot 2 = -7$$

Izlaz TLU perceptrona određen je izrazom 2.3 i prijenosnom funkcijom *f* prema 2.5. Slijedi:

$$y = f(net) = f(-7) = 0$$

Primjer 2

Zadan je TLU perceptron prikazan slikom. Težine pojedinih ulaza su: w_0 = -1.5, w_1 = 1, w_2 = 1. Promatrajte ulazni uzorak $x = (x_1, x_2)$ kao par dviju booleovih varijabli. Pri tome istinitosti odgovara vrijednost 1, a laži vrijednost 0. Odredite koju logičku funkciju u tom slučaju obavlja prikazani neuron.



Rješenje

Zadatak ćemo riješiti tako da izračunamo izlaz neurona za sve 4 mogućnosti.

x_1=0, x_2=0 $\quad y = f(\omega_0 + \omega_1 x_1 + \omega_2 x_2) = f(-1.5 + 1 \cdot 0 + 1 \cdot 0) = f(-1.5) = 0$

x_1=0, x_2=1 $\quad y = f(\omega_0 + \omega_1 x_1 + \omega_2 x_2) = f(-1.5 + 1 \cdot 0 + 1 \cdot 1) = f(-0.5) = 0$

x_1=1, x_2=0 $\quad y = f(\omega_0 + \omega_1 x_1 + \omega_2 x_2) = f(-1.5 + 1 \cdot 1 + 1 \cdot 0) = f(-0.5) = 0$

x_1=1, x_2=1 $\quad y = f(\omega_0 + \omega_1 x_1 + \omega_2 x_2) = f(-1.5 + 1 \cdot 1 + 1 \cdot 1) = f(0.5) = 1$

Kako je izlaz postao 1 (istinit) samo za slučaj kada su oba ulaza bila 1 (istinita), zaključujemo da se radi o Booleovoj funkciji I.

2.4 Postupak učenja mreže

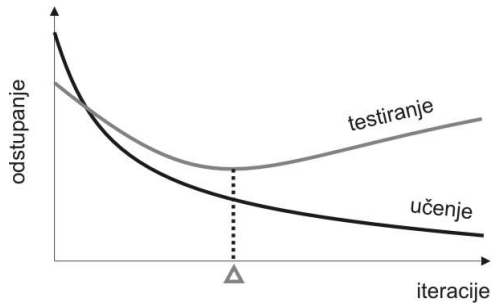
Jednostavnije neuronske mreže moguće je konstruirati tako da obavljaju određeni zadatak. Ovo će redom biti moguće za mreže koje se sastoje od TLU perceptrona, i koje obavljaju unaprijed zadanu logičku funkciju, jer u tom slučaju možemo pratiti što i kako točno mreža radi. U slučaju kada se koriste složenije prijenosne funkcije, poput sigmoidalne, ili dopušta rad s realnim brojevima, tipično se gubi zor nad načinom kako mreža obrađuje podatke. U tom slučaju uobičajeno se da se definira arhitektura mreže, i prije postupka obrade podatka obavi postupak *učenja* ili *treniranja*. Za razliku od konvencionalnih tehnika obrade podataka gdje je postupak obrade potrebno analitički razložiti na određeni broj algoritamskih koraka, kod ovog tipa neuronskih mreža 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 (uzoraka, iskustva) 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 2.3). Točnost i preciznost obrade podataka moguće je naposlijetku provjeriti nad trećim skupom primjera – skupom za provjeru.



Slika 2.3. Odstupanje 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.

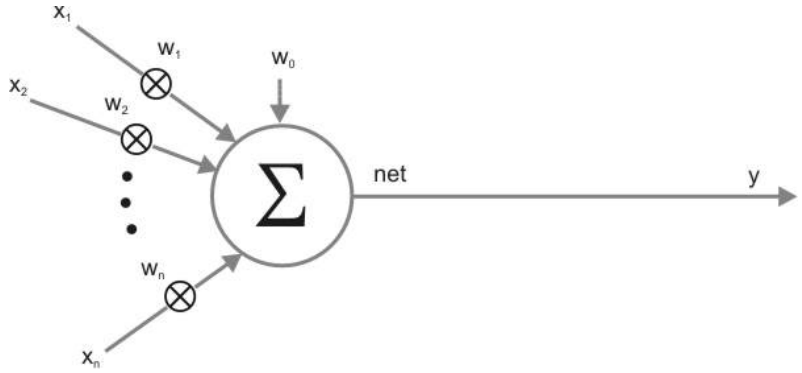
Neriješeni zadaci

1. Rješenje primjera 2 daje naslutiti kako će se uporabom samo TLU perceptrona moći ostvariti sve Booleove funkcije. Pokušajte konstruirati TLU perceptron koji obavlja logičku funkciju ILI, te TLU perceptron koji obavlja logičku funkciju NE. Uporabom takvih perceptrona izgradite mrežu koja računa vrijednost paritetnog bita nad binarnim vektorom $x = (x_1, x_2)$.

2. Promotrite TLU perceptron kojemu se na ulaz dovodi $x = (x_1, x_2)$. Težine pojedinih ulaza su: $w_0 = -2$, $w_1 = 4$, $w_2 = -1$. Ako je ulaz x_1 konstantno postavljen na -2, u kojem se rasponu mogu kretati vrijednosti na ulazu x_2, ako izlaz mora biti 1?

3 ADALINE

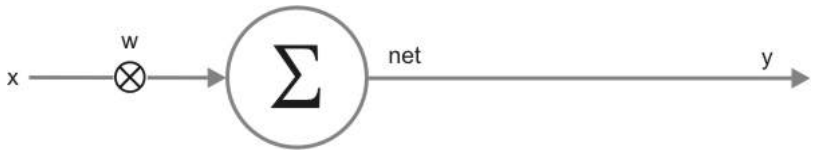
ADALINE je najjednostavniji PE kod kojega je nelinearna funkcija izbačena, tako da neuron zapravo računa težinsku sumu ulaza. ADALINE procesnu jedinicu prikazuje slika 3.1. Za ADALINE vrijedi: *y = net*.



Slika 3.1. ADALINE procesna jedinica

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. Evo jednostavnog primjera. 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: $d_i \approx w \cdot x_i$ (uočite da se ovo može realizirati kao ADALINE jedinica koja ima točno jedan ulaz, i njemu je pridružen težinski faktor *w*, kao na slici 3.2). Koristimo znak ≈ jer nismo sigurni da su sve točke baš na pravcu.



Slika 3.2. 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):

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

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

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

Međutim, vidjeti ćemo da će se u nastavku stvari dovoljno zakomplicirati 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:

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

Vidimo da gradijent 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:

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

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

$w(k+1) = w(k) + \eta \cdot \varepsilon(k) \cdot x(k)$	LMS pravilo

Primjer 3.1 Linearna regresija

Zadan je niz točaka za koje se pretpostavlja linearna ovisnost. Pronađite pravac koji aproksimira ovu ovisnost i prolazi kroz ishodište, analitički iterativno. Točke su (0,-0.1), (1,2.1), (2,3.9), (3,6.1), (4,8).

Analitičko se rješenje dobije uvrštavanjem u formulu:

$$w = \frac{\sum_{i=1}^{N} d_i \cdot x_i}{\sum_{i=1}^{N} x_i^2} = 2.0067$$

Dakle, linearna ovisnost koja vrijedi jest: $d_i \approx 2.0067 x_i$. Za iterativni postupak treba definirati kolika će biti stopa učenja (npr. 0.1), te koja je početna vrijednost faktora w (npr. 0). Tada možemo krenuti sa iterativnim izračunom. Java program A.1 demonstrira upravo ovaj slučaj. Njegovim pokretanjem dobija se slijedeći izlaz:

```
Koristim točku br 1: (0.0000,-0.1000).
w = 0.0000
w_novi = w + eta*grad = 0.0000 + 0.1000*-0.0000 = 0.0000
Koristim točku br 2: (1.0000,2.1000).
w = 0.0000
w_novi = w + eta*grad = 0.0000 + 0.1000*2.1000 = 0.2100
Koristim točku br 3: (2.0000,3.9000).
w = 0.2100
w_novi = w + eta*grad = 0.2100 + 0.1000*6.9600 = 0.9060
Koristim točku br 4: (3.0000,6.1000).
w = 0.9060
w_novi = w + eta*grad = 0.9060 + 0.1000*10.1460 = 1.9206
Koristim točku br 5: (4.0000,8.0000).
w = 1.9206
w_novi = w + eta*grad = 1.9206 + 0.1000*1.2704 = 2.0476
Koristim točku br 1: (0.0000,-0.1000).
w = 2.0476
w_novi = w + eta*grad = 2.0476 + 0.1000*-0.0000 = 2.0476
Koristim točku br 2: (1.0000,2.1000).
w = 2.0476
w_novi = w + eta*grad = 2.0476 + 0.1000*0.0524 = 2.0529
Koristim točku br 3: (2.0000,3.9000).
w = 2.0529
w_novi = w + eta*grad = 2.0529 + 0.1000*-0.4115 = 2.0117
Koristim točku br 4: (3.0000,6.1000).
w = 2.0117
w_novi = w + eta*grad = 2.0117 + 0.1000*0.1945 = 2.0312
Koristim točku br 5: (4.0000,8.0000).
```

```
w = 2.0312
w_novi = w + eta*grad = 2.0312 + 0.1000*-0.4988 = 1.9813
Koristim točku br 1: (0.0000,-0.1000).
w = 1.9813
w_novi = w + eta*grad = 1.9813 + 0.1000*-0.0000 = 1.9813
Koristim točku br 2: (1.0000,2.1000).
w = 1.9813
w_novi = w + eta*grad = 1.9813 + 0.1000*0.1187 = 1.9932
Koristim točku br 3: (2.0000,3.9000).
w = 1.9932
w_novi = w + eta*grad = 1.9932 + 0.1000*-0.1727 = 1.9759
Koristim točku br 4: (3.0000,6.1000).
w = 1.9759
w_novi = w + eta*grad = 1.9759 + 0.1000*0.5169 = 2.0276
Koristim točku br 5: (4.0000,8.0000).
w = 2.0276
w_novi = w + eta*grad = 2.0276 + 0.1000*-0.4414 = 1.9834
Koristim točku br 1: (0.0000,-0.1000).
w = 1.9834
w_novi = w + eta*grad = 1.9834 + 0.1000*-0.0000 = 1.9834
Koristim točku br 2: (1.0000,2.1000).
w = 1.9834
w_novi = w + eta*grad = 1.9834 + 0.1000*0.1166 = 1.9951
Koristim točku br 3: (2.0000,3.9000).
w = 1.9951
w_novi = w + eta*grad = 1.9951 + 0.1000*-0.1804 = 1.9771
Koristim točku br 4: (3.0000,6.1000).
w = 1.9771
w_novi = w + eta*grad = 1.9771 + 0.1000*0.5065 = 2.0277
Koristim točku br 5: (4.0000,8.0000).
w = 2.0277
w_novi = w + eta*grad = 2.0277 + 0.1000*-0.4433 = 1.9834
```

Kao što ispis pokazuje, rješenje je negdje oko broja *2*, što je se savršeno poklapa sa analitičkim izračunom.

Pogledajmo primjer 3.1. Zašto se faktor *w* kod iterativnog izračuna stalno mijenja kada stigne oko broja *2*? Da bismo na ovo mogli odgovoriti, moramo se prisjetiti kako smo došli do LMS formule. Izračun gradijenta zadavao nam je probleme pa smo se odlučili na jednostavnu aproksimaciju gradijenta 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. Ovo dakako rezultira pojavom koju najbolje demonstrira predočeni primjer.

Vratimo se sada na izvod gradijenta i zaboravimo na aproksimiranje. Pravi izraz za gradijent glasi:

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

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

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

Faktor *1/N* ispred sume apsorbiran[1] je u konstantu η koja je ionako proizvoljna. Riješimo sada 3.2. Uočite da upravo izvedena formula gradijent određuje na temelju svih točaka, što rezultira učenjem po epohama.

Primjer 3.2. Iterativna minimizacija s određivanjem gradijenta po epohama

Iterativnim postupkom i upravo izvedenom formulom riješite zadatak iz primjera 3.1.

Za rješavanje je zadužena mala modifikacije programa A.2 koji je rješavao prethodni primjer (A.1). Uočite da je vrijednost stope učenja smanjena na petinu (sada iznosi 0.02). Program daje slijedeći ispis:

```
Epoha 1
--------------------
w = 0.0000
Koristim tocku br 1: (0.0000,-0.1000).
  Trenutni gradijent je: -0.0000
  Ukupni gradijent je: 0.0000
Koristim tocku br 2: (1.0000,2.1000).
  Trenutni gradijent je: 2.1000
  Ukupni gradijent je: 2.1000
Koristim tocku br 3: (2.0000,3.9000).
  Trenutni gradijent je: 7.8000
  Ukupni gradijent je: 9.9000
Koristim tocku br 4: (3.0000,6.1000).
  Trenutni gradijent je: 18.3000
  Ukupni gradijent je: 28.2000
Koristim tocku br 5: (4.0000,8.0000).
  Trenutni gradijent je: 32.0000
  Ukupni gradijent je: 60.2000
w_novi = w + eta*grad = 0.0000 + 0.0200*60.2000 = 1.2040

Epoha 2
--------------------
w = 1.2040
Koristim tocku br 1: (0.0000,-0.1000).
  Trenutni gradijent je: -0.0000
  Ukupni gradijent je: 0.0000
```

[1] U nastavku ćemo ipak vidjeti da će faktor *1/N* iskrsnuti tamo gdje mu se najmanje nadamo, ali za sada, zbog kompaktnosti zapisa ostajemo pri toj apsorbciji.

```
Koristim tocku br 2: (1.0000,2.1000).
  Trenutni gradijent je: 0.8960
  Ukupni gradijent je: 0.8960
Koristim tocku br 3: (2.0000,3.9000).
  Trenutni gradijent je: 2.9840
  Ukupni gradijent je: 3.8800
Koristim tocku br 4: (3.0000,6.1000).
  Trenutni gradijent je: 7.4640
  Ukupni gradijent je: 11.3440
Koristim tocku br 5: (4.0000,8.0000).
  Trenutni gradijent je: 12.7360
  Ukupni gradijent je: 24.0800
w_novi = w + eta*grad = 1.2040 + 0.0200*24.0800 = 1.6856

Epoha 3
--------------------
w = 1.6856
Koristim tocku br 1: (0.0000,-0.1000).
  Trenutni gradijent je: -0.0000
  Ukupni gradijent je: 0.0000
Koristim tocku br 2: (1.0000,2.1000).
  Trenutni gradijent je: 0.4144
  Ukupni gradijent je: 0.4144
Koristim tocku br 3: (2.0000,3.9000).
  Trenutni gradijent je: 1.0576
  Ukupni gradijent je: 1.4720
Koristim tocku br 4: (3.0000,6.1000).
  Trenutni gradijent je: 3.1296
  Ukupni gradijent je: 4.6016
Koristim tocku br 5: (4.0000,8.0000).
  Trenutni gradijent je: 5.0304
  Ukupni gradijent je: 9.6320
w_novi = w + eta*grad = 1.6856 + 0.0200*9.6320 = 1.8782

Epoha 4
--------------------
w = 1.8782
Koristim tocku br 1: (0.0000,-0.1000).
  Trenutni gradijent je: -0.0000
  Ukupni gradijent je: 0.0000
Koristim tocku br 2: (1.0000,2.1000).
  Trenutni gradijent je: 0.2218
  Ukupni gradijent je: 0.2218
Koristim tocku br 3: (2.0000,3.9000).
  Trenutni gradijent je: 0.2870
  Ukupni gradijent je: 0.5088
Koristim tocku br 4: (3.0000,6.1000).
  Trenutni gradijent je: 1.3958
  Ukupni gradijent je: 1.9046
Koristim tocku br 5: (4.0000,8.0000).
  Trenutni gradijent je: 1.9482
  Ukupni gradijent je: 3.8528
w_novi = w + eta*grad = 1.8782 + 0.0200*3.8528 = 1.9553

Epoha 5
```

```
--------------------
w = 1.9553
Koristim tocku br 1: (0.0000,-0.1000).
  Trenutni gradijent je: -0.0000
  Ukupni gradijent je: 0.0000
Koristim tocku br 2: (1.0000,2.1000).
  Trenutni gradijent je: 0.1447
  Ukupni gradijent je: 0.1447
Koristim tocku br 3: (2.0000,3.9000).
  Trenutni gradijent je: -0.0212
  Ukupni gradijent je: 0.1235
Koristim tocku br 4: (3.0000,6.1000).
  Trenutni gradijent je: 0.7023
  Ukupni gradijent je: 0.8259
Koristim tocku br 5: (4.0000,8.0000).
  Trenutni gradijent je: 0.7153
  Ukupni gradijent je: 1.5411
w_novi = w + eta*grad = 1.9553 + 0.0200*1.5411 = 1.9861

Epoha 6
--------------------
w = 1.9861
Koristim tocku br 1: (0.0000,-0.1000).
  Trenutni gradijent je: -0.0000
  Ukupni gradijent je: 0.0000
Koristim tocku br 2: (1.0000,2.1000).
  Trenutni gradijent je: 0.1139
  Ukupni gradijent je: 0.1139
Koristim tocku br 3: (2.0000,3.9000).
  Trenutni gradijent je: -0.1445
  Ukupni gradijent je: -0.0306
Koristim tocku br 4: (3.0000,6.1000).
  Trenutni gradijent je: 0.4249
  Ukupni gradijent je: 0.3943
Koristim tocku br 5: (4.0000,8.0000).
  Trenutni gradijent je: 0.2221
  Ukupni gradijent je: 0.6164
w_novi = w + eta*grad = 1.9861 + 0.0200*0.6164 = 1.9984
```

Prethodni primjer pokazuje da stabilan i monoton porast faktora ka svojoj ciljnoj vrijednosti. Budući da se ovdje gradijent računa na temelju svih točaka, nema nikakvog šuma zbog aproksimacije i minimizacija ide upravo prema očekivanjima. 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 – može biti presudno.

Grupno se učenje može dobiti i uporabom klasične LMS formule, samo što tijekom jedne epohe (dakle 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 usrednji se te šum nestane što opet rezultira pravim gradijentom.

3.2 Konvergencija LMS-a

U nastavku ćemo se pozabaviti pitanjima vezanim za konvergenciju LMS-a. Naime, već smo došli do formule za LMS:

$$w(k+1) = w(k) + \eta \cdot \varepsilon(k) \cdot x(k)$$

Pitanje koje se postavlja je kada (i da li uopće) ova formula konvergira? Pa krenimo od korijena problema. Do formule smo došli vođeni težnjom za minimizacijom pogreške:

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

Analitičkim rješavanjem minimuma ove funkcije došli smo do:

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

Uvrštavanjem se dalje dobije:

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

uz

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

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

Gradijent funkcije pogreške možemo napisati i ovako:

$$\nabla J = \lambda\left(w - w^*\right)$$

pa uvrštavanjem dobivamo iterativnu formulu za koeficijente analognu LMSu:

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

Od lijeve i desne strane oduzmimo w^*:

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

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

$$\begin{aligned}
w(1) - w^* &= \left(1 - \eta \cdot \lambda\right) \cdot \left(w(0) - w^*\right) \\
w(2) - w^* &= \left(1 - \eta \cdot \lambda\right) \cdot \left(w(1) - w^*\right) = \\
&= \left(1 - \eta \cdot \lambda\right) \cdot \left(\left(1 - \eta \cdot \lambda\right) \cdot \left(w(0) - w^*\right) + w^* - w^*\right) = \\
&= \left(1 - \eta \cdot \lambda\right)^2 \cdot \left(w(0) - w^*\right) \\
&\ldots \\
w(k) - w^* &= \left(1 - \eta \cdot \lambda\right)^k \cdot \left(w(0) - w^*\right)
\end{aligned}$$

Odavde je rješenje očito:

$$w(k) = \left(1 - \eta \cdot \lambda\right)^k \cdot \left(w(0) - w^*\right) + w^*$$

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

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

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

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

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

Sada vidimo 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 veće, zatim 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}.

Primjer 3.3. Divergencija algoritma

Da se pojava divergencije doista javlja, možemo se uvjeriti vrlo jednostavno. Ponovimo primjer 3.1 uz jednu minornu modifikaciju: stavimo stopu učenja na 0.5. Rezultat je slijedeći:

```
Koristim tocku br 1: (0.0000,-0.1000).
w = 0.0000
w_novi = w + eta*grad = 0.0000 + 0.5000*-0.0000 = 0.0000
Koristim tocku br 2: (1.0000,2.1000).
w = 0.0000
w_novi = w + eta*grad = 0.0000 + 0.5000*2.1000 = 1.0500
Koristim tocku br 3: (2.0000,3.9000).
w = 1.0500
w_novi = w + eta*grad = 1.0500 + 0.5000*3.6000 = 2.8500
Koristim tocku br 4: (3.0000,6.1000).
w = 2.8500
w_novi = w + eta*grad = 2.8500 + 0.5000*-7.3500 = -0.8250
Koristim tocku br 5: (4.0000,8.0000).
w = -0.8250
w_novi = w + eta*grad = -0.8250 + 0.5000*45.2000 = 21.7750
Koristim tocku br 1: (0.0000,-0.1000).
w = 21.7750
w_novi = w + eta*grad = 21.7750 + 0.5000*-0.0000 = 21.7750
Koristim tocku br 2: (1.0000,2.1000).
w = 21.7750
w_novi = w + eta*grad = 21.7750 + 0.5000*-19.6750 = 11.9375
Koristim tocku br 3: (2.0000,3.9000).
w = 11.9375
w_novi = w + eta*grad = 11.9375 + 0.5000*-39.9500 = -8.0375
Koristim tocku br 4: (3.0000,6.1000).
w = -8.0375
w_novi = w + eta*grad = -8.0375 + 0.5000*90.6375 = 37.2812
Koristim tocku br 5: (4.0000,8.0000).
w = 37.2812
```

```
w_novi = w + eta*grad = 37.2812 + 0.5000*-564.5000 = -244.9687
Koristim tocku br 1: (0.0000,-0.1000).
w = -244.9687
w_novi = w + eta*grad = -244.9687 + 0.5000*-0.0000 = -244.9687
Koristim tocku br 2: (1.0000,2.1000).
w = -244.9687
w_novi = w + eta*grad = -244.9687 + 0.5000*247.0687 = -121.4344
Koristim tocku br 3: (2.0000,3.9000).
w = -121.4344
w_novi = w + eta*grad = -121.4344 + 0.5000*493.5375 = 125.3344
Koristim tocku br 4: (3.0000,6.1000).
w = 125.3344
w_novi = w + eta*grad = 125.3344 + 0.5000*-1109.7094 = -429.5203
Koristim tocku br 5: (4.0000,8.0000).
w = -429.5203
w_novi = w + eta*grad = -429.5203 + 0.5000*6904.3250 = 3022.6422
Koristim tocku br 1: (0.0000,-0.1000).
w = 3022.6422
w_novi = w + eta*grad = 3022.6422 + 0.5000*-0.0000 = 3022.6422
Koristim tocku br 2: (1.0000,2.1000).
w = 3022.6422
w_novi = w + eta*grad = 3022.6422 + 0.5000*-3020.5422 = 1512.3711
Koristim tocku br 3: (2.0000,3.9000).
w = 1512.3711
w_novi = w + eta*grad = 1512.3711 + 0.5000*-6041.6844 = -1508.4711
Koristim tocku br 4: (3.0000,6.1000).
w = -1508.4711
w_novi = w + eta*grad = -1508.4711 + 0.5000*13594.5398 = 5288.7988
Koristim tocku br 5: (4.0000,8.0000).
w = 5288.7988
w_novi = w + eta*grad = 5288.7988 + 0.5000*-84588.7812 = -
37005.5918
```

Uočite kako se faktor w mijenja od lošeg na još goreg. Naime, analizom skupa ulaznih podataka za primjer 3.1 dobiva se λ=6 i η_{max}=0.3333. Maksimalno brza konvergencija trebala bi biti za η=0.1666, a za vrijednosti η između 0.1666 i 0.3333 algoritam bi trebao generirati alternirajući niz.

Primjer 3.4. Alterniranje algoritma

Pojava alterniranja dobiva se uz npr. η=0.2 (vidi opasku na kraju primjera 3.3). Dobiva se:

```
Koristim tocku br 1: (0.0000,-0.1000).
w = 0.0000
w_novi = w + eta*grad = 0.0000 + 0.2000*-0.0000 = 0.0000
Koristim tocku br 2: (1.0000,2.1000).
w = 0.0000
w_novi = w + eta*grad = 0.0000 + 0.2000*2.1000 = 0.4200
Koristim tocku br 3: (2.0000,3.9000).
w = 0.4200
```

```
w_novi = w + eta*grad = 0.4200 + 0.2000*6.1200 = 1.6440
Koristim tocku br 4: (3.0000,6.1000).
w = 1.6440
w_novi = w + eta*grad = 1.6440 + 0.2000*3.5040 = 2.3448
Koristim tocku br 5: (4.0000,8.0000).
w = 2.3448
w_novi = w + eta*grad = 2.3448 + 0.2000*-5.5168 = 1.2414
Koristim tocku br 1: (0.0000,-0.1000).
w = 1.2414
w_novi = w + eta*grad = 1.2414 + 0.2000*-0.0000 = 1.2414
Koristim tocku br 2: (1.0000,2.1000).
w = 1.2414
w_novi = w + eta*grad = 1.2414 + 0.2000*0.8586 = 1.4132
Koristim tocku br 3: (2.0000,3.9000).
w = 1.4132
w_novi = w + eta*grad = 1.4132 + 0.2000*2.1474 = 1.8426
Koristim tocku br 4: (3.0000,6.1000).
w = 1.8426
w_novi = w + eta*grad = 1.8426 + 0.2000*1.7163 = 2.1859
Koristim tocku br 5: (4.0000,8.0000).
w = 2.1859
w_novi = w + eta*grad = 2.1859 + 0.2000*-2.9743 = 1.5910
Koristim tocku br 1: (0.0000,-0.1000).
w = 1.5910
w_novi = w + eta*grad = 1.5910 + 0.2000*-0.0000 = 1.5910
Koristim tocku br 2: (1.0000,2.1000).
w = 1.5910
w_novi = w + eta*grad = 1.5910 + 0.2000*0.5090 = 1.6928
Koristim tocku br 3: (2.0000,3.9000).
w = 1.6928
w_novi = w + eta*grad = 1.6928 + 0.2000*1.0287 = 1.8986
Koristim tocku br 4: (3.0000,6.1000).
w = 1.8986
w_novi = w + eta*grad = 1.8986 + 0.2000*1.2129 = 2.1411
Koristim tocku br 5: (4.0000,8.0000).
w = 2.1411
w_novi = w + eta*grad = 2.1411 + 0.2000*-2.2584 = 1.6895
Koristim tocku br 1: (0.0000,-0.1000).
w = 1.6895
w_novi = w + eta*grad = 1.6895 + 0.2000*-0.0000 = 1.6895
Koristim tocku br 2: (1.0000,2.1000).
w = 1.6895
w_novi = w + eta*grad = 1.6895 + 0.2000*0.4105 = 1.7716
Koristim tocku br 3: (2.0000,3.9000).
w = 1.7716
w_novi = w + eta*grad = 1.7716 + 0.2000*0.7137 = 1.9143
Koristim tocku br 4: (3.0000,6.1000).
w = 1.9143
w_novi = w + eta*grad = 1.9143 + 0.2000*1.0712 = 2.1285
Koristim tocku br 5: (4.0000,8.0000).
w = 2.1285
w_novi = w + eta*grad = 2.1285 + 0.2000*-2.0568 = 1.7172
```

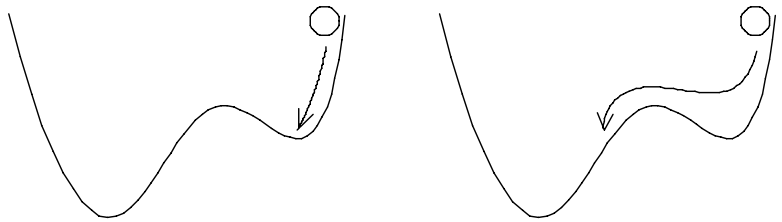
Analizom vrijednosti koje poprima *w* lako se može uočiti da niz konvergira prema broju blizu broja *2*, ali na način da se pojavljuju i veći, i manji brojevi. Zapravo, pažljivom analizom (primjer 3.1) može se vidjeti da je algoritam i tada alternirao iako je stopa učenja prema našim proračunima trebala dati monoton rast. O čemu se tu radi? Kada god radimo sa LMS-om, koristimo aproksimaciju gradijenta što unosi šum. Tada niti naš proračun ne vrijedi – zapravo vrijedi ako uzmemo faktor sigurnosti! Naime, obično se za stopu učenja uzima stopa koja je barem deset puta manja od maksimalne kako bi se osigurala konvergencija niza unatoč prisutnom šumu.

Ovo što smo upravo izveli, 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*. (Sjetite se kako smo *1/N* apsorbirali u izrazu za η kako bi nam formula ostala elegantnijom). Kao što vidite na ovom mjestu, ljepota se plaća. A to je ujedno i razlog onom misterioznom smanjivanju stope učenja pri prelasku iz primjera 3.1 u primjer 3.2 za iznos *5*. Naime, prilikom nastajanja programa koji rješava 3.2 stopa učenja izvorno je ostala nepromijenjena – rezultat je, dakako, čuđenje zašto sada algoritam divergira… Eto, i na to treba misliti.

Prije no što krenemo na ozbiljne stvari, dok smo još u ovim matematičkim vodama, pogledajmo neke probleme i doskočice temi zvanoj konvergencija.

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. Naime, 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 3.3).



a) bez momenta inercije b) sa momentom inercije

Slika 3.3. Učenje uz moment inercije

Trenutni položaj na plohi pogreške možemo zamisliti kao položaj loptice. Ukoliko ta loptica nema inercije, tada će se ona spustiti u prvi lokalni minimum i u njemu će ostati.

No ukoliko loptica ima moment inercije, lako je zaključiti da će se loptica po inerciji nastaviti gibati i ukoliko je lokalni minimum dovoljno malen, loptica će ga napustiti i nastaviti gibanje prema globalnom minimumu. Ovakvo razmišljanje dovodi nas do modificirane LMS formule:

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

Pri tome se γ naziva moment inercije, i obično je između 0 i 1.

3.4 ADALINE u punoj snazi

U uvodnom dijelu poglavlja uveli smo pojam ADALINE jedinice, ali smo zbog jednostavnosti daljnju analizu proveli za ADALINE sa samo jednim ulazom (slika 3.2). Sada ćemo razmotriti kako se ovaj isti element ponaša kada dopustimo višestruke ulaze, kao što to pokazuje slika 3.1. 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 dodaje još jedan stupanj slobode. U tom slučaju ADALINE računa funkciju:

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

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

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

pri čemu je $x_0=1$ (uvijek; x_0 nije stvarni ulaz ADALINE-a). Formula je identična formuli 2.2. 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:

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

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:

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

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

Pogreška za sve uzorke tada je definirana izrazom:

$$J = \frac{1}{2N} \sum_{i=1}^{N} \left(d_i - \sum_{k=0}^{D} w_k \cdot x_{i,k} \right)^2 \tag{3.1}$$

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 $D+1$ komponentni stupčasti vektor:

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

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

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

Sada se vidi da za izračun gradijenta trebamo sve ulazne uzorke. Međutim, ako opet primijenimo Widrowovu ideju koja je toliko revolucionarizirala ovo područje, a to je da gradijent aproksimiramo u i-toj iteraciji samo pomoću i-tog ulaznog uzorka, prethodna formula prelazi u:

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

Tada gradijent u i-tom koraku možemo zapisati kao:

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

Te automatski slijedi iterativna formula gradijentnog spusta:

$$\vec{w}(k+1) = \vec{w}(k) - \eta \cdot \nabla J(k) = \vec{w}(k) + \eta \cdot \varepsilon(k) \cdot \vec{X}(k) \qquad \text{LMS formula}$$

pri čemu je $\vec{X}(k)$ vektor ulaznog uzorka proširen jedinicom:

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

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.

4 TLU perceptron

TLU (Threshold Logic Unit) jest procesni element koji kao nelinearnost koristi step funkciju. Izlaz ovog perceptrona je, prema 2.5:

$$y = \begin{cases} 1, \text{ako je } net < 0 \\ 0, \text{ako je } net \geq 0 \end{cases}$$

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.

4.1 Pravilo perceptrona

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

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

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.

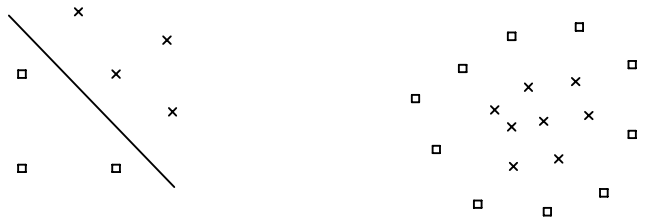
Ovo matematički možemo zapisati na slijedeći način.

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

Tablica 4.1. Pseudo kod za "Perceptron pravilo"

```
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 zaredom
```

Već smo pokazali da je ADALINE linearni element. Kako je TLU perceptron izveden iz ADALINE perceptrona, i 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 4.1a, dok razrede koji nisu linearno odvojivi prikazuje 4.1b. Postupak učenja pravilom perceptrona konvergirati će ispravnom rješenju ako i samo ako su razredi međusobno linearno odvojivi. U suprotnom postupak nikada neće završiti.



a) linearno odvojivi razredi b) linearno neodvojivi razredi

Slika 4.1. Linearna odvojivost razreda

Ovdje se javlja dosta veliki problem. Pretpostavimo da smo započeli učenje perceptrona, i nakon što je postupak trajao dosta dugo, što napraviti? Da li nastaviti postupak učenja (a moguće je da smo učili tako dugo jer uzorci nisu linearno razdvojivi, a tada i nema smisla dalje učiti) ili prekinuti postupak učenja (a opet je moguće da je postupak trajao toliko dugo jer imamo vrlo sporu konvergenciju, ali bi daljnje učenje ipak jednom došlo do dobrog rezultata). Koristeći ovu metodu učenja tu nema pomoći. No postoje druge metode učenja.

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:

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

problem možemo promatrati i ovako: zadana je funkcija od D varijabli na slijedeći način

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

Funkciju f možemo nazvati klasifikacijskom funkcijom; to je funkcija koja uzorku pridružuje njegov razred. No to je upravo 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, a to znamo jer smo već upoznali kako naučiti ADALINE (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 pogrešku svakom iteracijom 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.

Tablica 4.2. 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 zaredom
```

Y(i) 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 3. Delta pravilo može se implementirati i kao postupak grupnog učenja na način prikazan u tablici 4.3.

Tablica 4.3 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
```

5 Višeslojne neuronske mreže

5.1 Struktura mreže

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

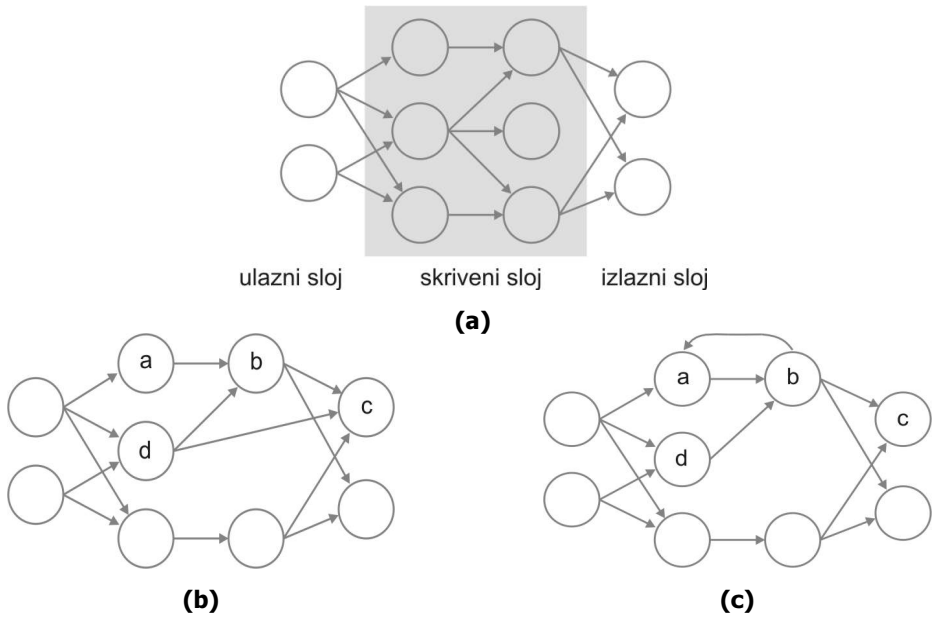
- aciklička (engl. feedforward net) mreža,
- mreža s povratnom vezom (engl. recurrent net),
- lateralno povezana mreža (rešetkasta),
- hibridne mreže.

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 podrazumjevamo podatake organizirane u vektor konkretnih vrijednosti. Struktura mreže obično se zadaje kao n-torka u zapisu $n_1 \times n_2 \times .. \times n_n$ 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).

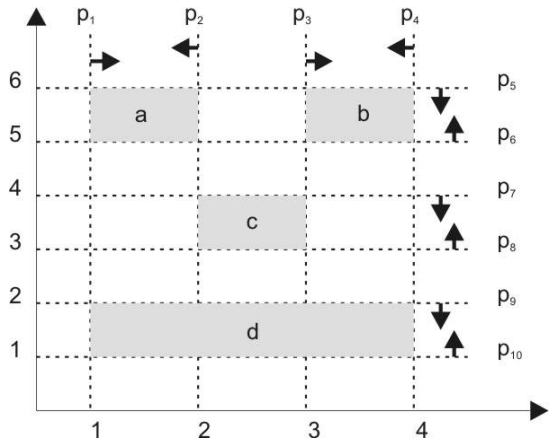
Spomenute strukture mreže prikazane su na slici 5.1.



Slika 5.1. Neke strukture mreža: (a) aciklička slojevita 2×3×3×2 mreža, (b) aciklička koja ne ispunjava uvjet slojevitosti radi povezanosti neurona b-c-d, (c) mreža s povratnom vezom b-a

5.2 Višeslojna mreža perceptrona

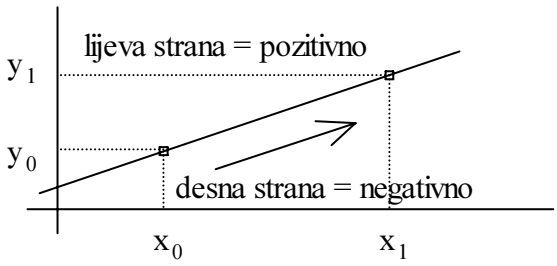
Uzevši u obzir ograničnu primjenjivost perceptrona na predočavanje samo linearne funkcije, potrebno je za predočavanje složenijih odnosa koristiti neuronsku mrežu koja se sastoji od više međusobno povezanih perceptrona. Pogledajmo jednostavan primjer kako se može projektirati neuronska mreža za klasifikaciju uzoraka pomoću TLU perceptrona. Ovakva mreža može se vrlo jednostavno projektirati ako unaprijed znamo raspodjelu uzoraka (slika 5.2).



Slika 5.2. 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 hiperravnina (odnosno pravac u 2D prostoru). Prvo moramo pogledati kako možemo sami odrediti težinske faktore jednog TLU perceptrona kada znamo kako izgleda granica. Prema slici 5.3, 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 5.3. Granica područja TLU perceptrona

Raspišemo li jednadžbu pravca kroz dvije točke

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

dobit ćemo

$$(y - y_0)(x_1 - x_0) - (y_1 - y_0)(x - x_0) = 0$$
$$\dots$$
$$-x(y_1 - y_0) + y(x_1 - x_0) + x_0(y_1 - y_0) - y_0(x_1 - x_0) = 0$$

Izjednačavanjem s težinskom sumom koju računa perceptron:

$$w_1 x + w_2 y + w_0 = 0$$

Možemo pročitati težinske faktore:

$$w_1 = -(y_1 - y_0)$$
$$w_2 = (x_1 - x_0)$$
$$w_0 = x_0(y_1 - y_0) - y_0(x_1 - x_0)$$

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 kada smo ovo ustanovili, krenimo na definiranje mreže.

Područje 'a' na slici 5.2 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:

$$w_1 = -(y_1 - y_0) = -(5-6) = 1$$
$$w_2 = (x_1 - x_0) = 1-1 = 0$$
$$w_0 = x_0(y_1 - y_0) - y_0(x_1 - x_0) = -1$$

te jednadžba pravca glasi:

$$p_1 \ldots\ w_1 x + w_2 y + w_0 = 0 \Rightarrow 1 \cdot x + 0 \cdot y - 1 = 0$$

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.

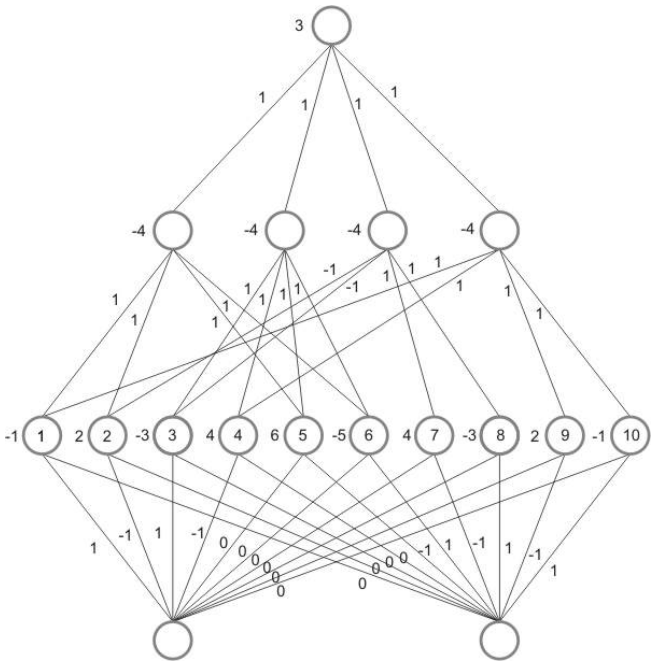
$$p_1 \ldots \quad 1 \cdot x + 0 \cdot y - 1 = 0$$
$$p_2 \ldots \quad -1 \cdot x + 0 \cdot y + 2 = 0$$
$$p_3 \ldots \quad 1 \cdot x + 0 \cdot y - 3 = 0$$
$$p_4 \ldots \quad -1 \cdot x + 0 \cdot y + 4 = 0$$
$$p_5 \ldots \quad 0 \cdot x - 1 \cdot y + 6 = 0$$
$$p_6 \ldots \quad 0 \cdot x + 1 \cdot y - 5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

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. kaže *1* tek kada svi kažu *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 riješava 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 riješava 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. Posao ćemo završiti 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. Dakle, treba reći *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 5.4. Mreža za klasifikaciju sastavljena od TLU perceptrona

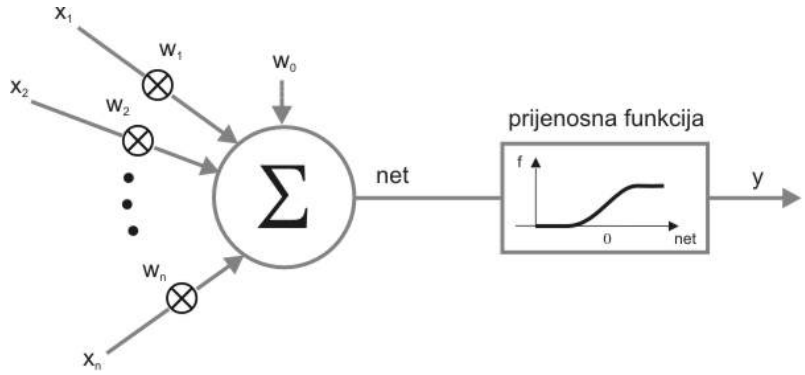
Konstruirana mreža prikazna je na slici 5.4. Provjerimo je li konstrukcija mreže valjana. Dovedimo na ulaz mreže uzorak (2.5, 3.5). To je uzorak koji pripada području 'c' pa mu stoga treba dodijeliti razred *1*.

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 smo mrežu konstruirali ispravno, jer je primjer ispravno klasificiran. Slično se može ponoviti za neki drugi primjer i promatrati odziv mreže.

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 (2.7). Stoga će višeslojna neuronska mreža sa sigmoidalnom funkcijom kao prijenosnom funcijom procesnih elemenata biti u stanju predstaviti nelinearne odnose ulaznih i izlaznih podataka. Procesni element prikazan je na slici 5.5. No kako naučiti takvu višeslojnu mrežu? Učinkovita i popularna metoda učenja višeslojnih mreža jest *algoritam sa širenjem pogreške unazad* - BACKPROPAGATION algoritam.



Slika 5.5. Sigmoidalna jedinica

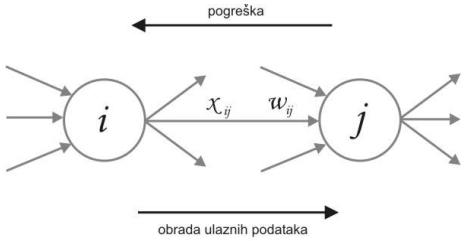
Algoritam koristi metodu gradijentnog spusta kako bi minimizirao nastalu pogrešku. Kod višeslojne mreže izlazni sloj može sačinjavati veći broj neurona, te je potrebno proširiti definiciju pogreške (3.1) za višestruke izlaze:

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 \tag{5.1}$$

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 paraboličke 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 inercije – opisana je u poglavlju 3.3.



Slika 5.6. Povezani neuroni

Algoritam BACKPROPAGATION dan je u tablici 5.1. 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), ω_{ij} je odgovarajuća težina, δ_n je pogreška izlaza jedinice n. Veličine su skicirane na slici 5.6. Algoritam kao parametre uzima skup za učenje D, stopu učenja η, 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 ispunen uvijet 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 proslijeđuju 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.

Tablica 5.1. Stohastički BACKPROPAGATION algoritam

Algoritam BACKPROPAGATION (D, η, n_i, n_o, n_h)

Inicijaliziraj težinske faktora na malene slučajno generirane vrijednosti,

npr. [-0.5, 0.5]

Dok nije ispunjen uvjet zaustavljanja čini

Za svaki $(\underline{x}, \underline{t})$ iz D čini

// Proslijedi ulazne vrijednosti unaprijed kroz mrežu:

Izračunaj izlaz o_u za svaku jedinicu u za ulaz $\underline{x}$

// Proslijedi grešku unazad kroz mrežu:

Za svaki izlazni čvor k izračunaj pogrešku δ_k

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k) \qquad \text{(T5.1)}$$

Za svaku skrivenu jedinicu h izračunaj pogrešku δ_h

$$\delta_h \leftarrow o_h(1-o_h)\sum_{s\in Downstream(h)}\omega_{hs}\delta_s \qquad \text{(T5.2)}$$

Ugodi svaki težinski faktor ω_{ij}

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

gdje je

$$\Delta\omega_{ij} = \eta\delta_j x_{ij} \qquad \text{(T5.3)}$$

Zato što primjeri za učenje određuju ciljne vrijednosti samo izlaznog sloja neurona, poznata nam je jedino pogreška izlaznog sloja (5.1). Kako ugađati težinske faktore neurona u skrivenom sloju? BACKPROPAGATION algoritam računa pogrešku bilo kojeg skrivenog neurona h tako da zbraja pogreške δ_s svih onih neurona s na koje utječe izlaz neurona h, uz dodatno množenje težinskim faktorom ω_{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 (T5.2) možemo napisati jednostavnije kao

$$\delta_h \leftarrow o_h(1-o_h)\sum_{s\in sloj\ m+1}\omega_{hs}\delta_s \qquad (5.2)$$

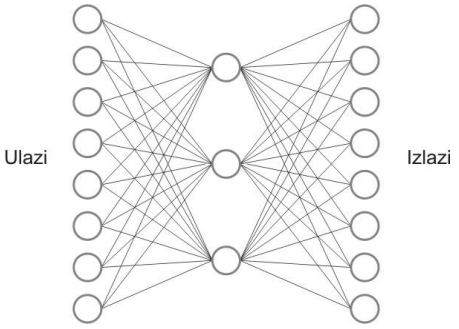
gdje *m* sloj u kojem se nalazi jedinica *h,* a *m+1* je idući sloj u smjeru izlaznog sloja.

Računajući pogrešku svakog neurona, algoritam propagira pogrešku od izlaznog sloja ka ulaznome, dakle unzad 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 proslijeđuju unazad. Mreža, međutim, opravdano može zadržati naziv *aciklička* (engl. feedforward) jer pod tim nazivom podrazumijevamo smjer kretanja samo ulaznih podataka.

U tablici 5.1 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.

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.



(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)

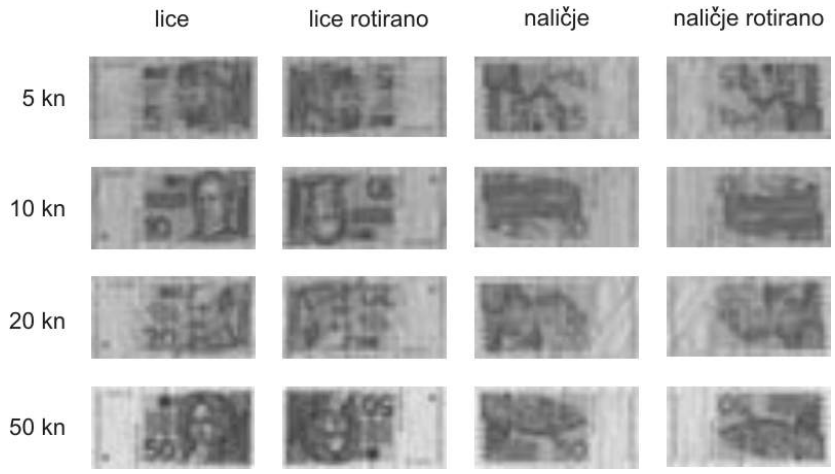
Slika 5.7. Mreža 8×3×8: (a) struktura, (b) ulazne vrijednosti, izlazne vrijednosti i izlazi neurona skrivenog sloja nakon učenja

Kao primjer neka posluži umjetna neuronska mreža sa slike 5.7a. 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 5.7b, izlazi neurona skrivenog sloja nakon jedne epohe učenja mreže

odgovaraju nakon zaokruživanja na cjelobrojnu vrijednost upravo binarnom kôdu za osam različitih ulaznih vrijednosti.

6 Primjer: raspoznavanje novčanica

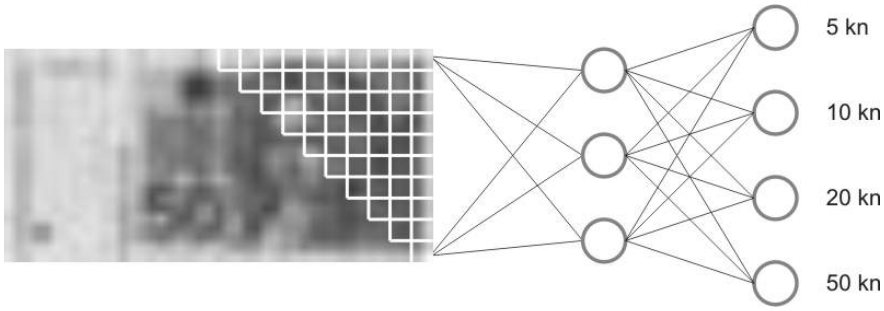
Teorijski pregled umjetnih neuronskih mreža zaključit ćemo jednim primjerom implementacije umjetne neuronske mreže za raspoznavanje papirnatih novčanica. Zadatak mreže jest klasificiranje pet različitih vrsta novčanica neovisno o orijentaciji. Skup primjera za učenje sačinjavaju novčanice od 5, 10, 20 i 50 kuna, svaka u četiri moguće varijante ovisno o orijentaciji (slika 6.1). Novčanice su odgovarajućim filtrom uzorkovane na rezoluciju 45×22 slikovna elementa nakon čega je provedena transformacija u crno-bijelu sliku. (Koristili smo pretvorbu iz RGB modela u YUV model iako je do zadovoljavajućih rezultata moguće doći i drugim metodama, npr. srednjom vrijednosti RGB komponenti).



Slika 6.1. Skup primjera za učenje

Mreža je strukture 990×3×4. Ulazni sloj od 990 neurona sačinjavaju slikovni elementi. 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. Pri tome na razliku između dvaju najvećih izlaznih vrijednosti možemo gledati kao na mjeru pouzdanosti klasifikacije. Slika 6.2 prikazuje organizaciju sustava za raspoznavanje.

Neuronska mreža trenirana je sa stopom učenja η=0.02 i momentom inercije γ=0.02. Svakih 2500 epoha rad mreže provjeren je nad skupom primjera za testiranje.



Slika 6.2. Umjetna neuronska mreža za klasifikaciju novčanica

Dodatak A. Java programi za gradijentni spust

Java program A.1. Iterativni postupak traženja faktora *w* (adaline1)

```
import MojiBrojevi;

public class adaline1 {

 public static void main(String argv[]) {
  double eta;
  double x[] = new double[5];
  double d[] = new double[5];
  double w, w0;
  int k, i;
  double grad = 0;

  MojiBrojevi.Decimals = 4;          // Ispis sa 4 decimale
  MojiBrojevi.ForceDecimals = true; // Prikazi sve (npr. 3.5000)

  eta = 0.1;
  x[0] = 0; d[0] = -0.1;
  x[1] = 1; d[1] =  2.1;
  x[2] = 2; d[2] =  3.9;
  x[3] = 3; d[3] =  6.1;
  x[4] = 4; d[4] =  8.0;

  w0 = 0;
  for( k = 0; k<20; k++ ) {
   i = k % 5;
   grad = (d[i]-w0*x[i])*x[i];
   w = w0 + eta*grad;
   System.out.println("Koristim tocku br "+(i+1)+": ("+

MojiBrojevi.ConvertDouble(x[i])+","+MojiBrojevi.ConvertDouble(d[i]
)
    +").");
   System.out.println("w = "+MojiBrojevi.ConvertDouble(w0));
   System.out.println("w_novi = w + eta*grad = "
    +MojiBrojevi.ConvertDouble(w0)+"                                +
"+MojiBrojevi.ConvertDouble(eta)
    +"*"+MojiBrojevi.ConvertDouble(grad)+" = "
    +MojiBrojevi.ConvertDouble(w));
   w0 = w;
  }
 }

}
```

Java program A.2. Iterativno određivanje koeficijenta s izračunom gradijenta po epohama

```
import MojiBrojevi;

public class adaline2 {

 public static void main(String argv[]) {
  double eta;
  double x[] = new double[5];
  double d[] = new double[5];
  double w, w0;
  int k, i;
  double grad = 0, grad_i = 0;

  MojiBrojevi.Decimals = 4;          // Ispis sa 4 decimale
  MojiBrojevi.ForceDecimals = true; // Prikazi sve (npr. 3.5000)

  eta = 0.02;
  x[0] = 0; d[0] = -0.1;
  x[1] = 1; d[1] =  2.1;
  x[2] = 2; d[2] =  3.9;
  x[3] = 3; d[3] =  6.1;
  x[4] = 4; d[4] =  8.0;

  w0 = 0;
  for( k = 0; k<6; k++ ) {
   System.out.println("Epoha "+(k+1));
   System.out.println("--------------------");
   System.out.println("w = "+MojiBrojevi.ConvertDouble(w0));
   grad = 0;
   for( i = 0; i<5; i++ ) {
    grad_i = (d[i]-w0*x[i])*x[i];
    grad = grad + grad_i;
    System.out.println("Koristim tocku br "+(i+1)+": ("+

MojiBrojevi.ConvertDouble(x[i])+","+MojiBrojevi.ConvertDouble(d[i]
)
    +").");
    System.out.println("  Trenutni gradijent je: "
     +MojiBrojevi.ConvertDouble(grad_i));
    System.out.println("  Ukupni gradijent je: "
     +MojiBrojevi.ConvertDouble(grad));
   }
   w = w0 + eta*grad;
   System.out.println("w_novi = w + eta*grad = "
    +MojiBrojevi.ConvertDouble(w0)+"                                    +
"+MojiBrojevi.ConvertDouble(eta)
    +"*"+MojiBrojevi.ConvertDouble(grad)+" = "
    +MojiBrojevi.ConvertDouble(w));
   w0 = w;
  }
 }
}
```

Java program A.3 Zajednički dio koji koriste primjeri (A.1 i A.2)

```
public class MojiBrojevi {
  public static String ConvertDouble( double d, int decimals ) {
   if( decimals == 0 ) d = Math.rint(d);
   else {
    d = Math.rint(d*Math.pow(10,decimals))/Math.pow(10,decimals);
   }
   String s = new String(Double.toString(d));
   if( s.indexOf('E') != -1 ) return s;
   StringBuffer sb = new StringBuffer();
   int i;
   for(i=0; i<s.length();i++) {
    sb.append(s.charAt(i));
    if( s.charAt(i) == '.' ) { i++; break; }
   }
   for(; i<s.length();i++) {
    if(decimals==0) break;
    sb.append(s.charAt(i));
    decimals--;
   }
   if( decimals > 0 && ForceDecimals ) {
    for(i=0; i<decimals;i++) {
     sb.append('0');
    }
   }
   return new String(sb);
  }
  public static int Decimals = 6;
  public static String ConvertDouble( double d ) {
    return ConvertDouble( d, Decimals );
  }
  public static boolean ForceDecimals = false;
}
```

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. D. 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] *AuotmataTheory.* Encyclopaedia Britannica Inc., 2001 CD-ROM Edition.