Univerzitet u Kragujevcu

Seminarski rad

iz predmeta Teorijske osnove veštačke inteligencije

Tema:

Mašinsko učenje: Konenkcionizam

student: Željko Simić 3vi/2023
mentor: Tatjana Stojanović

1 Uvod

Inspirisani pristupima obuke po neuronskoj (poznatiji kao paralelno distribuirana procesiranja ili sistemi konekcionizma) iliti biološkoj prirodi, udaljavajući se od korišćenja simboličkog obrađivanja[1] pri rešavanju problema, već pripajajući inteligenciju kako se razvija sistem jednostavnih interagujućih komponenti (bioloških ili veštačkih neurona) kroz proces obuke ili adaptacije po kom veze medju komponentama su prilagodjene. Paralelno procesiranje se ogleda u tome što svi neuronu unutar kolekcije ili sloju procesuiraju njihove ulaze simultano i nezavisno. Ovi sistemi takođe streme da se obesmisle zato što infocmacije i procesuiranja su raspoređena naspram čitave čvorove i slojeve mreže. U konekzionističkom modelu tu je jak reprezentacioni karakter obostrano i za pravljenje ulaznih parametara kao i za interpretaciju izlaznih parametara. Za pravljenje neuronske mreže, na primer, dizajner mora napraviti šemu za enkodiranje obrazaca iz realnog sveta u numeričke mere u mreži. Izbor enkodiranja šeme može igrati važnu ulogu u potencijalno rezultujućem uspehu ili neuspehu obuke mreže. U konekcionističkim sistemima procesuriranje je paralelno i distribuirano bez manipulacija simbola kao takvih (imutabilno). Obrasci u skupa oblasti vrednosti su enkodirani kao numerički vektori. Vezu među komponentama su reprezentovane po numeričkim vrednostima.

Konačno, transformacija obrazaca je rezultat numeričkih operacija, uobičajeno, proivoda matrica. Ovi "dizajnerske odluke" za ahritekturu konekcionizma ustanovljavaju induktivnu bazu sistema. Algoritmi i arhitekture koje implementiraju ove tehnike su uobičajeno obučene ili uslovljene pre nego što bivaju posebno programirane.

Glavni cilj ovog pristupa su povoljno dizajniranje mreže arhitekture i obuka algoritama koja može često obuhvatiti neobičnosti realnog sveta, kao i anomalije, bez posebnog programiranja da bi ih prepoznale. Sve sada pomenuto će se kroz ovaj rad istaći. Zadaci za koje neuronski/konekcionističkim pristupi odgovaraju:

- klasifikacija - odlučivanje za kategorizovanje ili grupisanje za koje ulazna vrednost pripada;
- prepoznavanje obrazaca - identifikacija struktura ili obrazaca u podacima;
- memorijski odaziv - uključivanje problema sadržajnog adresiranja memorije;
- predvidjanje - kao što je identifikacija zaraza prema simptomima, uzrocima efekata;
- optimizacija - okončavanje sa najboljom organizacijom ograničavanja;
- alterniranje šumova, odvajanja signala iz pozadine, faktorisanje redudantnih komponenti signala.

Metodologije ovog dela lekcije rade najbolje sa zadacima koji su zahtevni za iskazivanje simboličkih modela. Ovo je tipično uključivanje zadataka u koje oblast problema zahteva veštine zasnovane na percepticiji, nedostaci su jasno definisani sintaksom. Osnove konekcionističkih mreža predstavljaju neuronski inspirisane modele obuke iz istorijske tačke gledišta. Predstavljene obuke neuronske mreže, uključuju "mehanički" neuron, da opiše neke istorijski važne rane radove, uključujući neuron **McCulloch-Pitts**-a (1943). Generacije razvijanja oblasti paradigmi mrežnih obuka zadnjih 60 godina prilaže važan uvid u tekuće stanje ove oblasti.

*Z*a perceptron obuku će se obrazložiti istorijska predstava zajedno sa uvodom o obuci perceptronske obuke, kao i *delta pravilo*. Ističe se primer primene perceptrona kao klasifikatora.

Za *backpropagation obuku* će se istaći neuronske mreže sa skrivenim slojevima, kao i pravilo backpropagation obuke. Ove inovacije će biti obrazložene sa ciljem za veštačke neuronske mreže da prevaziđu probleme za rane sisteme koji su se pojavljivali pri generalizaciji naspram tačaka podataka koji nisu linearno separabilni. Backpropagation je algoritam za "okrivljivanje" za netačne odazive čvorova višeslojnog sistema sa neprekidnim pragom aktivacije.

Za *takmičarsko obučavanje* se predstavlja model koji je zavijen od strane Kohonena (1984) i Hecht-Nielsen-a (1987). U ovim modelima, mrežne težine vektora su u službi reprezentovanja obrazaca pre

nego snaga veza. "Pobednik uzima sve" algoritam za obuku bira čvorove čiji obrazac težine je najviše sličniji ulaznom vektoru i prilagova ga njemu da bi ga namestio da još više liči ulaznom vektoru. Predstavlja nenadgledajuće učenje u kom pobeda je jednostavno identifikovala čvor čija tekuća težina vektora je najviše bliska po sličnosti ulaznom vektoru. Kombinacija Kohonena sa Grossbergovim (1982) slojevima u samoj mreži pruža zanimljiv model za stimulativno-odazivajuće obuke zvane *obuka counterpropagation (suprotne propagacije).*

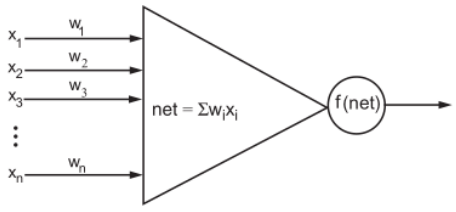
Za *Hebovu obuku slučajnostima* mi predstavljamo Hebov (1949) model za podsticajuću obuku. Hebb je izložio stav da svaki put neuron doprinosi mreži svakim okidanjem drugog neurona, tkao jačina putanje među neuronima je šira. Hebovu obuku su modelira po jednostavnim algoritmima za prilagođavanje težina veza. Mi predstavljamo obostranu nenadgledanu i nadgledanu verziju za Hebbovu obuku. Mi predstavljamo linearni veznik, Hebovog zasnovanog modela za dobavku obrazaca iz memorije.

Za *Atraktivne mreže ili "Uspomene"* predstavljena vrlo važna familija mreža pod nazivom atraktor mreže. Mreže pohranjuju povratnu spregu po petlji signala unutar mreže. Izlaz mreže je sagledan kao stanje mreže nad dostizanjem ekvilibrijuma (poklapanjem obostranog toka pružanja signala). Težine mreže su konstruisane tako da skup atraktora bude izgrađen. Ulazni obrasci unutar atraktoru okupljajućem čvorištu (basin) dostižu ekvilibrijum samo u tom određenom atraktoru. Aktraktori mogu da služe za skladištenje obrazaca u memoriju. Dati ulazni obrazac, mi dobavljamo ili najbliži uskladišteni obrazac u mreži ili obrazac asociran sa najbližim uskladištenim obrascem. Prva vrsta memorije naziva se autoasociativnom, a druga heteroasocijativnom. Džon Hopfild ('82.), teoretski fizičar, definisao je klasu mreža atraktora čija konvergnecija može biti rastumačena po minimalizaciji energije. Hopfildove mreže mogu biti korišćene da rešavaju problem za constraint satisfaction, kao što može biti i problem putujućeg trgovca po mapiranju optimizacionom funkcijom u energetsku funkciju.

2 Temelji konekcionističkih mreža

2.1 Rana istorija

Konekcionističke ahritekture su često uvažene za skorašnji razvitak, moguće je pratiti trag njihovih porekla od ranih radova u računarskoj nauci, psihologiji, filozofiji. Fon Nojman je bio oduševljen ćelijskim automatima, neuronski inspirisanim pristupima proračuna. Rani radovi o obuci neurona su bili pod uticajem psiholoških teorija dresiranja životinja, pogotovo Hebb-u. U ovoj lekciji će se istaći osnovne komponente obuke neuronskih mreža, predstaviti važan istorijski rad u polju.



Slika 1: Veštački neuron

Osnova neuronskih mreža je veštački neuron (prikazan na slici 1) koji je sadržan od:

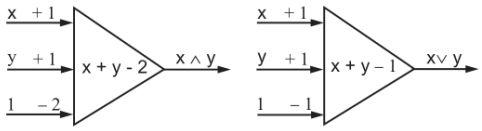
- Ulaznih signala x_i - Ovi signali dolaze iz okruženja ili aktivacija drugih neurona. Drugačiji modeli variraju po dovoljenom opsegu skupa ulaznih vrednosti. Uobičajeno su diskretne vrednosti, iz skupa $0, 1$ ili $-1, 1$ ili $\mathbb{R}$;
- Skup relano vrednovanih težina w_i - Težine opisuju jačine veza;

- Aktivacioni nivo $\sum w_i x_i$ - Neuronski aktivacioni nivo je određen kumulativnom jačinom sopstvenih ulaznih signala gde svaki ulazni signal je skaliran po težini veze w_i na liniji ulaza.
- Funkcija aktivacionog praga f - ova funkcija računa neuronski konačno ili izlazno stanje po određivanju koliko daleko neuronski aktivacioni nivo je ispod ili iznad vrednosti praga. Namenjena je da proizvede stanja uključenosti ili isključenosti više neurona.

  Uz ova navedena svojstva pojedinačnih neurona, neuronska mreža je takođe karakterizovana globalnim svojstvima kao što su:

- Mrežna topologija - obrazac veza među pojedinačnim neuronima. Topologija je primarni izvor mrežnog induktivnog bias-a;
- Šema enkodiranja - uključuje interpretaciju pozicioniranih podataka u mreži i rezultuje njihovo procesuiranje.

weight



Slika 2: McCulloch-Pitts neuroni pri računanju logičkih funkcija AND i OR

Najraniji primer neuronskog računa je kroz McCulloch-Pitts neuron. Ulazi neurona su ili pobuđujuće(+1) ili umirjuće(-1). Neuron pri $\sum w_i x_i \geqslant 0$ vraća 1, inače -1. Po slici 2 neuroni prikaza McCulloch-Pitts za računanje logičkih funkcija. AND neuron ima 3 ulaza x, y i bias (ima konstantnu vrednost +1). Podaci ulaza i bias imaju težine +1, +1, -2, respektivno. Za bilo koju vrednost x i y neuron računa vrednost $x + y - 2 < 0$, vraća se -1, inače 1.

x	y	x + y - 2	Output
1	1	0	1
1	0	-1	-1
0	1	-1	-1
0	0	-2	-1

Tabela 1: McCulloch-Pitts model za logički AND

Tabela 1 ilustruje neronsko sračunavanje $x \wedge y$. U sličnom smislu bi se računala težinska suma ulaza podataka za OR neuron kome je vrednost veća od 0 ukoliko ne budu oba x i y jednaki -1. McCulloch i Pitts demonstriraju moć neuronskog računa, zanimanje u pristupu je počelo da buja sa razvitkom algoritama praktične obuke. Rani modeli obuke uticali su na Hebb-a koji je imao stav da obuka nastaje u mozgu kroz modifikaciju sinapsi. Hebb je teoretisao da ponovno okidanje putem sinapsi uvećava senzitivnost i budući oslonac ka okidanju. Određeni stimulus (podsticaj) ponovno ishodovan aktivnosti u grupi ćelija, ove ćelije su međusobno jako vezane. U budućnosti, sličan stimuli će težiti da pobudi iste neuronske putanje, rezultujući u prepoznavanju stimuli-a.

Hebb-ov model obuke radi čisto po podsticavanju korišćenih putanja i ignorisajući kočenje, kažnjavanje pri greškama, osipanje. Moderni psiholozi su pristupili Hebb-ovom modelu, ali nisu uspeli da proizvedu opšte rezultate bez uključivanja kočnih sistema. U sledećoj lekciji vrši se proširenje neuronskog modela McCulloch-Pitts-a dodavanjem slojeva neuronskih mehanizama i algoritama za njihovo interagovanje. Prva verzija je zvana perceptron.

3 Perceptron obuka

3.1 Algoritam

Frenk Rozenblat ('58., '62.) istakao je algoritam obuke za tipove jednoslojne mreže zvanog perceptron. Signalnom propagacijom, perceptron je bio sličan McCulloch-Pitts neuronu. Ulazne vrednosti i nivoi aktivacije perceptrona su bile ili -1 ili 1. Težine su bile realne vrednosti.

Ulazne vrednosti i nivoi aktivacije perceptrona su ili -1 ili 1; težine su vrednovane kao realne. Nivo aktivacije perceptrona dat je po sumiranju težinskih ulaznih vrednosti. Perceptron koristi jednostavni teže-ograničavajući funkciju praga, gde aktivacija iznad praga rezultuje da mu vrednost izlaza bude 1 i inače -1. Perceptron računa izlaznu vrednost kao:

$$\begin{cases} 1, & \sum x_i w_i \geqslant t \\ -1, & \text{inače} \end{cases}$$

Perceptron koristi jednostavan oblik nadgledane obuke. Posle pokušaja da reši problem instance, učitelj prilaže tačan rezultat. Perceptron menja svoje težine u cilju da redukuje grešku. Prateće pravilo je korišćeno. Neka c bude konstanta čija veličina odlučuje stopu brzine obuke i d gde je ona rezultujuća izlazna vrednosti. Nameštanje za težine na i-toj komponenti ulaznih vektora, Δw_i je data kao:

$$\Delta w_i = c(d - sign(\sum(x_i w_i)))x_i$$

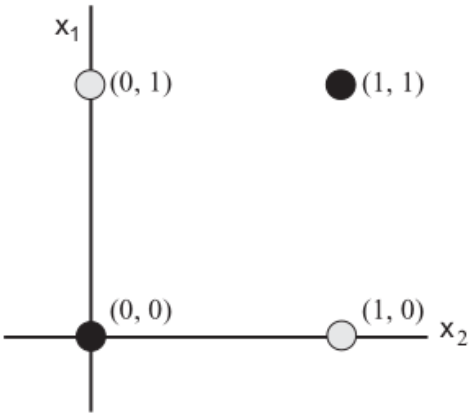
$sign(\sum(x_i w_i))$ je izlazna vrednost perceptrona (+1 ili -1). Razlika između željenih izlaznih i stvarnih izlaznih vrednosti je ili 0, 2, -2. Pritom za svaku kompoentu vektora ulaza važi:

- Ako željena izlazna i stvarni izlazna vrednost su jednake, ne raditi ništa.
- Ako stvarna izlazna vrednost je -1, a trebalo je da bude 1, povećava se težina na i-toj liniji za $2cx_i$.
- Ako stvarna izlazna vrednost je 1, a tebalo je da bude -1, smanjuje se težina na i-toj liniji za $2cx_i$

Ova procedura ima efekat proizvodnje skupa težina takvog da bude namenjen minimizaciji prosečne greške nad čitavim skupom za obuku. Ako postoji skup težina koji pruža tačan izlaz za svaki član skupa za obuku, procedura obuke perceptrona će ga obukom usvojiti (Minsky i Papert '69). Nils Nilson ('65.) i ostali su analiziriali ograničenja modela perceptrona. Demonstrirali su da perceptroni nisu mogli da reše određene teške klase problema, tj. probleme tamo gde podatkovne tačke nisu linearno separabilne. Pojačanjem modela perceptrona, uz višeslojni perceptron, sagledan za taj period Marvin Minski i Seymor Papert, u njihovim knjigama Perceptrons ('69.), diskutovali su da problem linearne separabilnosti ne može da prevaziđe ni jedan oblik mreža perceptrona. Kao primer je data nelinearno separabilna klasifikacija koja radi po principu ekskluzivne disjunkcije (XOR).

Uočimo perceptron sa 2 ulaza x_1, x_2, 2 težine, w_1, w_2 i aktivacioni prag t. U službi da obučimo ovu funkciju, mreža mora naći dodelu težina tako da zadovolji sledeće nejednačine, po slici 3:

- $w_1 * 1 + w_2 * 1 < t$, za $x_1 = 1, x_2 = 1, output = 0$
- $w_1 * 1 + w_2 * 0 > t$, za $x_1 = 1, x_2 = 0, output = 1$
- $w_1 * 0 + w_2 * 1 > t$, za $x_1 = 0, x_2 = 1, output = 1$
- $w_1 * 0 + w_2 * 0 < t$ ili $t > 0$, za $x_1 = 0, x_2 = 0, output = 0$



Slika 3: XOR problem. Nema 2D prave koja može podeliti (0,1) i (1,0) podatkovne tačke sa (0,0) i (1,1)

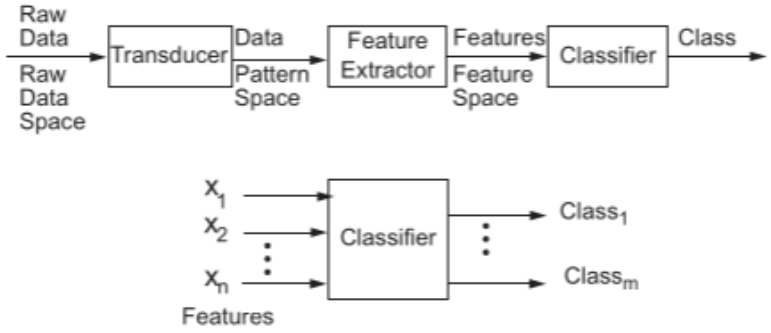
Red jednačina nad w_1, w_2, i t nemaju rešenja, dokazujući da perceptron koji rešava XOR je nemoguć, ali zato višeslojne mreže mogu potencijalno da naprave takvo rešenje za XOR problem. Algoritma obuke perceptrona samo radi u slučaju za jednoslojnu mrežu. Ono što čini XOR nemogućim za perceptron je to da 2 klase se razaznaju kao da nisu linearno separabilne. Ovo može da bude viđeno na slici tamo. Nemoguće je nacrtati pravu u 2D prostoru koja odvaja podatkovne tačke {(0,0), (1,1)} od {(0,1), (1,0)}. Možemo misliti o skupu vrednosti podataka za mrežu kao definisanje prostora. Svaki parametar ulaznih podataka oslanja za 1 dimenziju, sa svakom ulzanom vrednošću koja definiše tačku u prostoru. U XOR primeru, 4 ulazne vrednosti, indeksirane sa x_1, x_2 koordinatam grade podatkovne tačke po slici 3. Problem obuke binarnom klasifikacijom pri obuci instancama odnosi se na podelu ovih tačaka u 2 grupe. Za n-dimenzionalni prostor, klasifikacija je linearno separabilna ako klase mogu biti podeljene na n-1 dimenzionu hiperravan (u 2D n-dimenzionalna hiperravan je linija, u 3D ravan, itd.). Kao rezultujuće ograničenje linearne separabilnosti, istraga se usmerila na posao simboličko-zasnovane arhitekture, usporavajući progress u konekcionističkoj metodlogiji. Pojedini poslovi 80-tih, 90-tih pokazalo pokazali su rešivost problema (Ackey et '85, Hinton and Sejnowski 1986, 1987).

Uskoro će se diskutovati o backpropagation modelu, proširenju obuke perceptrona koji radi za višeslojne mreže. Pre sagledavanja backpropagation modela, obraća se pažnja na primer perceptrona koji vrši klasifikaciju. Obuka perceptrona će biti dovršena definisanjem generalizacije delta pravila. Generalizacija algoritma obuke perceptrona koji je korišćen za nekolicinu arhitektura neuronskih mreža, uključujući i backpropagation.

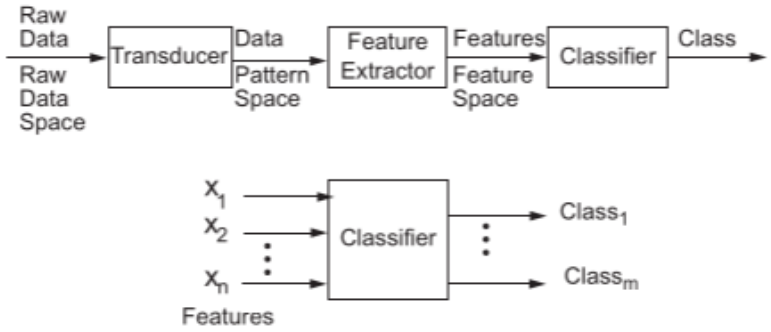
3.2 Primer: Korišćenje mreže perceptrona za klasifikaciju

Slika 4 prikazuje klasifikatorni problem. Sirovi podaci iz prostora mogućih tačaka su odabrane i probličene u novi podatak/obrazac prostor. U tom novom prostoru obrasca karakteristike su identifikovane, pa konačno pojave po karakteristikama su klasifikovane. Primer bi mogao da budu zvučni talas snimljen na digitalnom aparatu za snimanje zvuka. Odavde akustični signali su prevedeni u skupinu parametara amplituda i učestalosti. Konačno klasifikatorni sistem može prepoznati ove obrasce karakteristika kao oglašen govor određene osobe. Još jedan primer je hvatanje informacija medicinske opreme za testiranje, kao što su srčani defibliratori. Karakteristike nađene u ovom prostru obrazaca mogu da budu korišćeni za klasifikaciju skupine simptoma po različitim kategorijama zaraza.

U primeru za klasifikaciju pretvaranje forme podataka i ekstraktor karakteristika po slici 5. prevodi problem informacija u parametre 2D Dekarkovog prostora. Slika 6 predstavlja analizu perceptrona dveju karakteristika ispoljen na tabeli 2.



Slika 4: Celi klasifikatorni sistem



Slika 5: Celi klasifikatorni sistem

x_1	x_2	Output
1.0	1.0	1
9.4	6.4	-1
2.5	2.1	1
8.0	7.7	-1
0.5	2.2	1
7.9	8.4	-1
7.0	7.0	-1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	-1

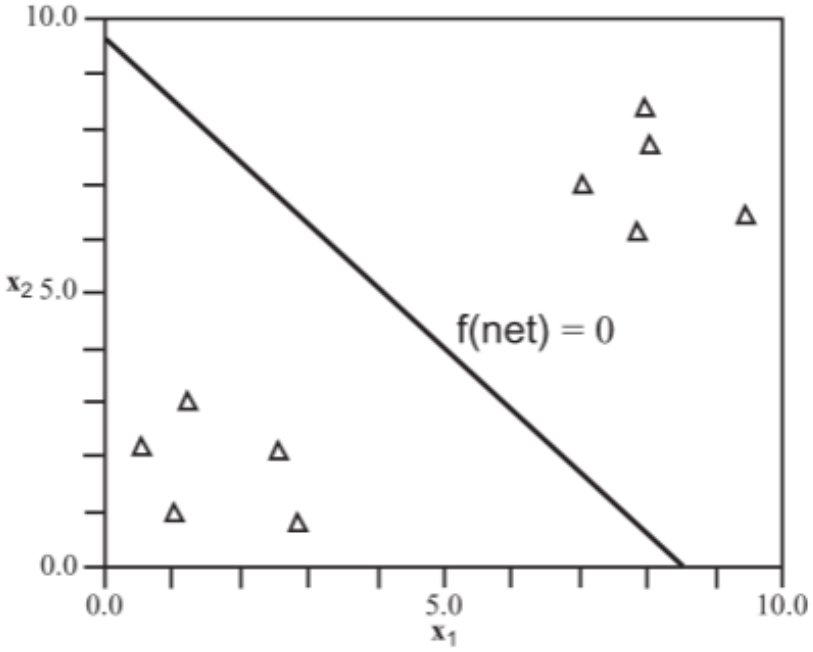
Tabela 2: skup podataka za klasifikaciju perceptrona

Prve dve kolone tabele 2 predstavljaju podatkovne tačke nad kojim mreža je obučena. 3. kolona prikazuje klasifikaciju, +1 ili -1, korišćena kao povratna informacija mrežne obuke.

Slika 6 je grafik skupa podataka namenjenog za obuku pri rešavanju problema, prikazivanje linearne separabilnosti klasa podataka pravljenih kada obučena mreža radi nad svakom tačkom podataka. Diskutuje se prva generalna teorija klasifikacije. Svaki od podataka se grupiše klasifikatorom koji identifikuje reprezentacije regiona u višedimenzionog prostora. Svaka klasa R_i ima razdvajajuću funkciju g_i merenjem članstva u regionu. Unutar regiona R_i, i-ta razdvajajuća funkcija ima najveće vrednosti:

$$g_i(x) > g_j(x) \forall j, 1 < j < n$$

U prostom primeru iz tabele 2, 2 ulazna parametra proizvode 2 očigledna regiona ili klasa u prostoru, jedan reprezentovan sa 1 i drugi sa -1. Bitan specijalni slučaj razdvajajućih funkcija je jedan od kojih evaluira klase članstva utemeljenim nad distancama iz neke centralne tačke regiona.

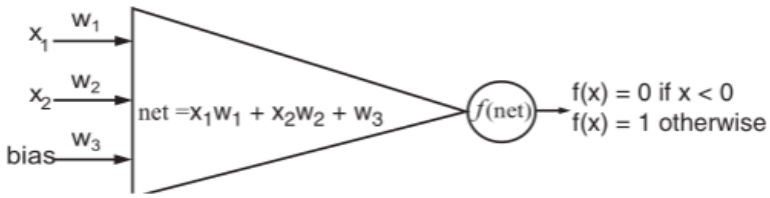


Slika 6: 2D plot tačaka podataka iz tabele 2 gde perceptron sa slike 3 ustanovljava linearnu separabilnost

Klasifikacija utemeljena nad ovom razdvajajućom funkcijom zove se *klasifikacija minimalne razdaljine*. Ovaj stav pokazuje da ako su klase linearno separabilne onda se vrši klasifikacija minimalne razdaljine. Ako region R_i i R_j su susedni, tako ta 2 regiona (na slici 6.) imaju granični region gde za njega razdvajajuća funkcija jednači:

$$g_i(x) = g_j(x), g_i(x) - g_j(x) = 0$$

Ako klase su linearno separabilne, kao po slici 6., razdvajajuća funkcija koja razdvaja regione je prava, $g_i(x) - g_j(x)$ je linearna. Otkad prava je presečna tačka jednake razdaljenosti 2 fiksirane tačke, razdvajajuće funkcije $g_i(x)$ i $g_j(x)$, koje su funkcije minimalne razdaljine, merene po centru Dekartovog koordinatinatnog sistema za svaki region posebno.



Slika 7: Perceptron mreže za primer podataka sa tabele 2, funkcija praga aktivacije je linearna i bipolarna

Perceptron na slici 7. će sračunavati linearnu funkciju. 2 ulazna parametra i bias sa konstantnom vrednošću 1. Perceptron izračunava $f(net) = f(w_1 * x_1 + w_2 * x_2 + w_3 * 1)$, gde f(x) je znak od x. f(x) biva +1, rastumačen je kao 1 klasa, a kada je -1, tada x je neka druga klasa. Prag aktivacije sa vrednošću +1 ili -1 su nazivane da su linearno bipolarne. Bias uspostavlja pomeraj praga aktivacione

funkcije po apcisi (x-osi). Proširenje ovog pomeraja je zarad prilagođavanja obuke težinom w_3 pri treniranju. Moramo primeniti instance iz skupa podataka (podatkovne tačke) iz tabele 2. pri obuke perceptrona sa slike 7. Nagađamo da inicijalizacije su nasumičnih težina $\begin{bmatrix} 0.75 \\ 0.50 \\ -0.60 \end{bmatrix}$ i koristimo algoritam obuke perceptrona AND sa slike 2. "superskripte", npr. 1 u $f(net)^1$ tumači se kao tekući broj iteracije algoritma. Počinje se uzimanjem prve instance iz tabele 2:

$$f(net)^1 = f(.75 * 1 + .5 * 1 - .6 * 1) = f(.65) = 1$$

Počev od f(net)$^1 = 1$, tačna izlazna vrednost se ne prilagođava težini. Pritom je $W^2 = W^1$. Za drugu instancu:

$$f(net)^2 = f(.75 * 9.4 + .5 * 6.4 - .6 * 1) = f(9.65) = 1$$

Ovog puta imamo -1 koje primenjujemo kao pravilo obuke, opisano po:

$$\boxed{W^t = W^{t-1} + c(d^{t-1} - sign(W^{t-1} * X^{t-1}))X^{t-1}}$$

c je konstanta obuke, X ulazni vektor i W težinski vektor, t iteracija mreže. d^{t-1} je željen rezultat iteracije t-1, a u ovom slučaju t = 2. Mrežni izlaz za t=2 je 1. Pritom, razlika između željenog i stvarnog izlaza mreže, $d^2 = sign(W^2 * X^2)$ je -2. Čvrsto ograničeni linearno bipolarni perceptron, povećavanje obuke će uvek biti ili +2c ili inače -2c pomnoženo vektorom obuke. Prepuštamo konstanti obuke da bude mala pozitivna realna brojka, 0.2. Potom, ažuriramo težinski vektor:

$$W^3 = W^2 + 0.2(-1 - 1)X^2 = \begin{bmatrix} 0.75 \\ 0.50 \\ -0.60 \end{bmatrix} - 0.4 \begin{bmatrix} 9.4 \\ 6.4 \\ 1.0 \end{bmatrix} = \begin{bmatrix} -3.01 \\ -2.06 \\ -1.00 \end{bmatrix}$$

Tako i za 3. instancu:

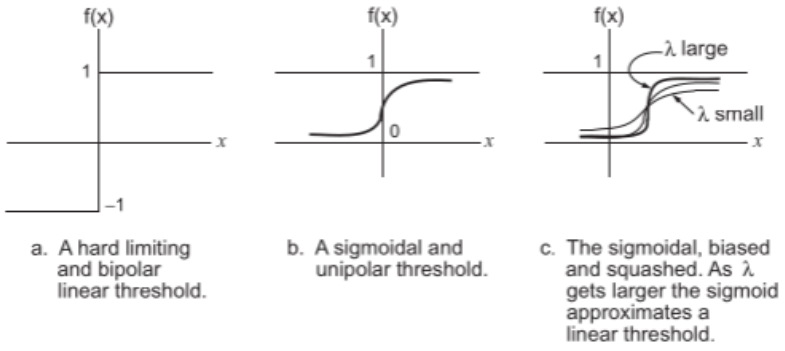
$$f(net)^3 = f(-3.01 * 2.5 - 2.06 * 2.1 - 1.0 * 1) = f(-12.84) = -1$$

$$W^4 = W^3 + 0.2(1 - (-1))X^3 = \begin{bmatrix} -3.01 \\ -2.06 \\ -1.00 \end{bmatrix} + 0.4 \begin{bmatrix} 2.5 \\ 2.1 \\ 1.00 \end{bmatrix} = \begin{bmatrix} -2.01 \\ 1.22 \\ -0.60 \end{bmatrix}$$

Nakon 10 iteracija za perceptron linearna separabilnost sa slike 6. je izvedena. Nakon ponovljene obuke nad skupom podataka, ukupno u 500 iteracija, težinski vektor teži $\begin{bmatrix} -1.3 \\ -1.1 \\ 10.9 \end{bmatrix}$. Zanima nas linija koja razdvaja dve klase. One presečne tačke gde $g_i(x) - g_j(x) = 0$ i izlaz mreže je 0, po formuli $output = w_1x_1 + w_2x_2 + w_3$. Ishod je dobije da linija koja odvaja 2 klase je definisana po linearnoj jednačini: $-1.3x_1 - 1.1x_2 + 10.9 = 0$

3.3 Uopšteno delta pravilo

Direktan način za uopštavanje mreže mreže perceptrona je da se zameni teže ograničavajuća funkicija aktivacionog praga sa drugim funkcijama aktivacije. Na primer, neprekidnom aktivacionom funkcijom pruža se mogućnost algoritmu sofisticiranija obuka po prihvatanju profinjenije usitnjenosti u merljivosti grešaka. Slikom 8. prikazuje se prag aktivacione funkcije: linearna bipolarna funkcija praga aktivacije na slici 8. pod a., sličan onom koji je korišćen za perceptron, a i većini sigmoidnih funkcija. Sigmoidna funkcija je nazvana tako zato što je oblika nalik slovu "S" kao što je na slici prikazana pod b. Uobičajena sigmoidalna aktivaciona funkcija je nazivana i logističkom funkcijom data jednačinom: $f(net) = \frac{1}{1+e^{-\lambda * net}}$, gde je $net = \sum x_i w_i$ Prethodno istaknutom funkcijom, x_i je ulaz linije i, w_i je



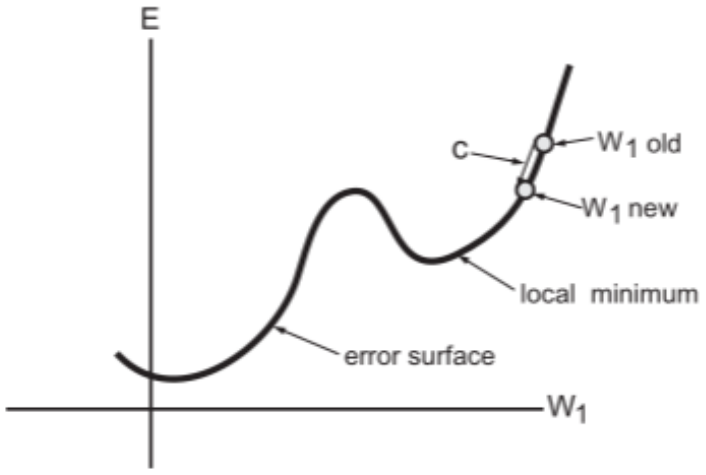
a. A hard limiting and bipolar linear threshold.

b. A sigmoidal and unipolar threshold.

c. The sigmoidal, biased and squashed. As λ gets larger the sigmoid approximates a linear threshold.

Slika 8: Pragovi aktivacione funkcije

težina linije i, i λ je parametar potisnuća korišćen za profinjeno-prilagođavanje sigmoidalne krive. Pri većem λ sigmoidalna funkcija je nalik linearnoj funkciji praga aktivacije nad $0,1$ neuređenim skupom. Kako je bliža 1 ona prostaje prava. Ovi grafici pragova aktivacije ulaznih vrednosti predstavljaju nivoe aktivacija neurona naspram skalirane aktivacije ili izlaza neurona. Sigmoidalna aktivaciona funkcija je neprekidna, koja dozvoljava preciznu merljivost greške. Kao i kog čvrsto ograničene funkcije praga aktivacije, sigmoidalna aktivaciona funkcija mapira većinu vrednosti oblasti unutar regiona bliskom 0 ili 1. Postoji i region rapidnog-kontinualnog preoblikovanja između 0 i 1. U smislu, da se aproksimativno sračunava prag aktivacije takvim ponašanjem da pruža neprekidnu izlaznu funkciju. Primena λ u podešavanju eksponenta koeficijentom pravca oblika sigmoida u regionu preoblikovanja. Težinski bias pomera prag aktivacije nad apcisom (x-osom). Istorijski razvoj mreža sa kontinualnim aktivacionim funkcijama podstakli su obuku umanjenjem greške. Widrow-Hoff ('60.) pravilo obuke je nezavisno od aktivacionih funkcija, minimalizovalo kvadriranu grešku među željene izlazne vrednosti i aktivacije mreže $net_i = WX_i$. Jedno od najvažnih pravila obuke za kontinualne aktivacione funkcije je delta



Slika 9: Pragovi aktivacione funkcije

pravilo (Rumelhart '86.). Intuitivno, delta pravilo je zasnovano na ideji površi greške, predstavljeno na slici 9. Površ greške reprezentuje kumulativnu grešku nad skupom podataka kao funkcijom težina

meže. Svaka moguća konfiguracija težine mreže je reprezentovana po tački n-dimenzione površi greške. Datom konfiguracija težine, uspostavlja se algoritam obuke zarad nalaženja smera na površi kojom se rapidno umanjuje greška. Ovaj priustup je nazvan obukom gradijentnog spusta zato što se gradijent smatra merom koeficijenta pravca, kao funkcije usmerenja, po tački na površi. Za korišćenje delta pravila mreža mora primeniti aktivacionu funkciju koja je neprekidna, a i diferencijabilna. Logistička formula je predstavila ovo svojstvo, pa formula za delta pravilo obuke za težinsko prilagođavanje na j-ti ulaz i-tog čvora je:

$$c(d_i - O_i)f'(net_i)x_j$$

c je konstanta koja uspostavlja stopu brzine obuke, d_i je željena, O_i je stvarna vrednost i-tog čvora. x_j j-ti ulaz za čvor i. Pokazujemo sada izvod ovih formula. Srednja kvadratna greška je nađena sumiranjem kvadratnih grešaka za svaki čvor: $Error = \frac{1}{2}\sum_i (d_i - O_i)^2$. Kvadriranje se vrši zarad izbegavanja potiranja oduzimanjem računatih grešaka.

Ovde smo ustanovili da za čvor izlaznog sloja može se ustanoviti da opšti slučaj kada predstavljamo mrežu sa skrivenim slojem po budućoj lekciji Backpropagation-a želimo da izmerimo stopu promene greške mreže u odnosu na izlaz svakog čvora. Stičemo taj ishod parcijalnim izvodom, koji nam pruža stopu izmene funkcije više promenljivih u odnosu na određenu promenljivu. Parcijalni izvod ukupne greške u odnosu na svaki izlaz jedinice i je:

$$\frac{\partial Error}{\partial O_i} = \frac{\partial \frac{1}{2}\sum (d_i - O_i)^2}{\partial O_i} = \frac{\partial \frac{1}{2}(d_i - O_i)^2}{\partial O_i}$$

Drugo uprošćenje je moguće zato što smo uočili čvor na izlaznom sloju, kom greška ne utiče na druge čvorove. Preuzimajući izvod ovog kvantifikovanja dobija se:

$$\frac{\partial \frac{1}{2}(d_i - O_i)^2}{\partial O_i} = -(d_i - O_i)$$

Uzima se stopa izmene greške mreže kao funkcije greške u težini na čvoru i. Da bi dobili izmenu određene težine, w_k, oslanjamo se na primenu parcijalnog izvoda, ovog puta se uzima parcijalni izvod za grešku svakog čvora u odnosu na težine w_k za svaki čvor posebno. Proširenje desnom stranom jednakosti dobijeno je izvođenje složene funkcije za parcijalni izvod.

$$\frac{\partial Error}{\partial w_k} = \frac{\partial Error}{\partial O_i} * \frac{\partial O_i}{\partial w_k}$$

A pritom kombinujemo:

$$\frac{\partial Error}{\partial w_k} = -(d_i - O_i) * \frac{\partial O_i}{\partial w_k}$$

Uzima se najuticajniji faktor zdesna, percijalni izvod stvarnog izlaza i-tog čvora uzet u odnosu na svaku težinu čvora. Formula za izlaze čvora i po funkciji njegovih težina je: $Oi = f(W_i X_i)$, gde $W_i X_i = net_i$. Pošto je f diferencijabilna:

$$\frac{\partial O_i}{\partial w_k} = x_k * f'(W_i X_i) = f'(net_i) * x_k$$

Smenom prethodne jednačine:

$$\frac{\partial Error}{\partial w_k} = -(d_i - O_i)f'(net_i)x_k$$

Minimalizacijom greške zahteva se da težine se promene u smeru negativne komponente gradijenta.

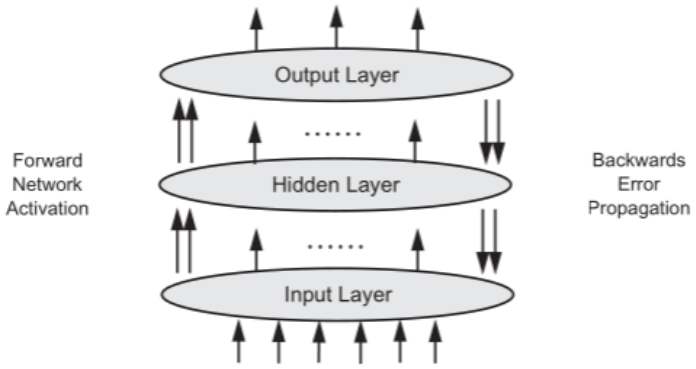
$$\Delta w_k = -c\frac{\partial Error}{\partial w_k} = -c(-(d_i - O_i)f'(net_i)x_k) = c(d_i - O_i)f'(net_i)x_k$$

Osmotrili smo delta pravilo kao metodu pentranja uzbrdo[4]. Svakim korakom pokušava se minimizovati mera lokalne greške primenom izvoda nalažeći koeficijent pravca prostora greške u regionu koji je lokalan, koliko toliko. Delta pravilo je time ranjivo problemo razaznavanja lokalnog i globalnog minimuma prostora greške. Konstanta obuke c, pridaje vežnu ulogu obavljanja delta pravila, daljom analizom ilustovano na slici 9. Vrednost c određuje koliko vrednost težina pomerena u obuci jedne iteracije. Većim c-om brže se dostiže optimalna vrednost težina. Prevelikom vrednošću c, algoritam prekomerno naciljava minimum i oscilira oko optimalnih težina. Manjom vrednošću c takve problematike se izbegavaju, ali ne dozvoljavaju sistemu da se obučava brzo. Optimalna vrednost stope brzine obuke, ustpostavljena faktorom momentuma (Zurada '92.) ističe se kao parametar prilagođen za određene primene kroz eksperimente. Delta pravilo ne prevazilazi ogranizenja jednoslojne mreže, njena izopšten oblik je centar funkcionisanja algoritma backpropagation-a, algoritma za obuku višeslojne mreže.

4 Backpropagation obuka

4.1 Izvođenje backpropagation obuke

Jednoslojne mreže su bile ograničene za pojedine klasifikacije. U sledeće 2 lekcije se obrazlaže prevazilaženje ovih ograničenja. A moguće je i istaći da je Tjuring-kompletna. Predstavljeno je opšte delta pravilo, kojim se rešava problematika dizajniranja algoritma obuke.



Slika 10: Backpropagation u konekcionističkoj mreži sa skrivenim slojem

Neuroni u višeslojnoj mreži kao na slici 10. su povezani po slojevima sa jedinicama u sloju n prosleđujući njihove aktivacije samo neurona u sloju n+1. Višeslojno signalno procesiranje naznačuje da greške duboko u mreži se proširuju i generacijski razvijaju na kompleksne neočekivane načine kroz sledujuće slojeve. Analiza izvorne greške na izlaznom sloju je kompleksna. Backpropagation pruža algoritamsko uspostavljanje krivice i povoljno podešavanje težina. Pristup kojim je backpropagation algoritam uveden počevši od izlaznog sloja, propagirajući grešku unazad kroz skrivene slojeve. Kada analizirana obuka sa delta pravilom se obavlja uočava se da sve informacije potrebne ažuriranju težina nekog neurona su lokalne za taj neuron, osim za količinu greške. Za izlazne čvorove, lakše sračunata je razlika među željenom i stvarnom izlaznom vrednošću. Za čvorove skrivenog sloja, uočljivo je teže ustanoviti grešku koju je uzrokovao neuron. Aktivaciona funkcija za backpropagation je uobičajeno logistička funkcija $f(net) = \frac{1}{1+e^{-\lambda * net}}$, gde je $net = \sum x_i w_i$.

Ova funkcija je korišćena zbog 4 razloga:

1. Sigmoidni oblik;

2. Neprekidna funkcija koja ima izvod svuda.
3. Pošto je vrednost izvod koji je najveći gde sigmoidna funkcija je brzopleto promenljiva, dodela najveće greške istaknute u ime čvorova čija aktivacija je najmanje upadljiva.
4. Izvod lagodno sračunat oduzimanjem i množenjem: $f(net) = \frac{1}{1+e^{-\lambda * net}} = \lambda(f(net) * (1 - f(net)))$.

Backpropagation obuka primenjuje opšte delta pravilo. Korišćenjem istog pristupa gradijentnog sputa. Za čvorove u skrivenom sloju uviđa se doprinos grešci na izlaznom sloju. Formule sračunate prilagođavanjem tečina $w_k i$ na putanje od k-tog do i-tog čvora obuke backpropagation-a su:

1. $\Delta w_k i = -c(d_i - O_i)O_i(1 - O_i)x_k$, za čvorove na izlaznom sloju,
2. $\Delta w_k i = -cO_i(1 - O_i)\sum_j(-delta_j * w_{i,j})x_k$, za čvorove skrivenih slojeva.

U drugoj formuli j označava indeks čvorova u sledećem sloju gde i-ti signali razilaze tako da:

$$delta_j = -\frac{\partial Error}{\partial net_j} = (d_i - O_i)O_i(1 - O_i)$$

.

Prikazuje se izvođenje ovih formula. Vrši se diferenciranje 1), formula za usklađenu težinu čvorova izlaznog sloja. Kao pre, šta želimo je stopa promene greške mreže kao promenu funkcije u k-toj težini, w_k čvora i. Ophodimo se prema situaciji tako što izvodom delta pravila dobija se:

$$\frac{\partial Error}{\partial w_k} = -((d_i - O_i)f'(net_i)x_k)$$

Pošto f može biti bilo koja funkcija, ovo se bira da bude logistička funkcija aktivacije, pa tada:

$$f'(net) = f'(\frac{1}{1 + e^{-\lambda * net}}) = f(net)(1 - f(net))$$

Odaziv po $f(net_i)$ je prosto O_i. Smenom prethodne jednačine, imamo:

$$\frac{\partial Error}{\partial w_k} = -(d_i - O_i)O_i(1 - O_i)x_k$$

Kako minimalizacija greške zahteva da promene težina budu smerovi negativne komponente gradijenta, množimo formulu sa -c, pa dobijamo prilagodjavanje težina za i-ti čvor izlaznog sloja: $\Delta w_k = c(d_i - O_i)O_i(1 - Oi)x_k$ Potom vršimo izvođenje prilagodjavanja težina za skrivene čvorove. Zarad razjašnjenja, početno nudimo jedan skriveni sloj. Uzimamo jedan čvor, tako i skirven sloj i analiziramo doprinos ukupne greške mreže. U početku uočavamo doprinos za i-ti čvor za grešku na čvoru j iz izlaznog sloja. Sumiraju se doprinosi putem svih čvorova dostupnih na izlaznom sloju. Konačno, opisujemo doprinos k-tog ulaza po težini na čvoru i za grešku mreže.

Slika 11. prikazuje ovu situaciju, imamo prvi pogled na parcijalni izvod greške mreže u odnosu na izlaz čvora i na skrivenom sloju. Dobijamo to primenom pravila ulačavanja:

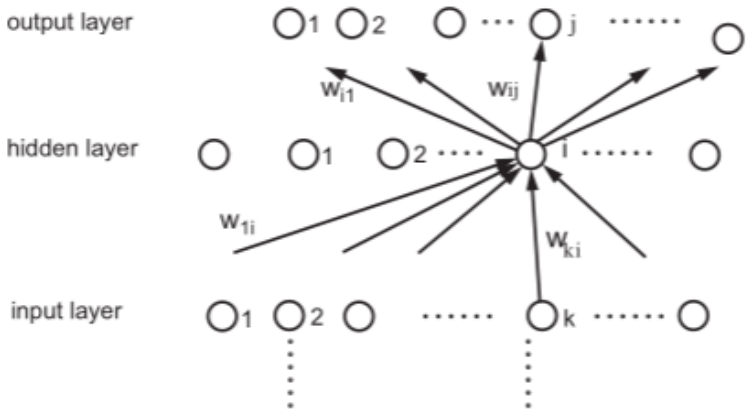
$$\frac{\partial Error}{\partial O_i} = \frac{\partial Error}{\partial net_j} * \frac{\partial net_j}{\partial O_i}$$

Negativan prvi izraz desno $\frac{\partial Error}{\partial net_j}$ se sada ogleda kao $delta_j$. Dobija se:

$$\frac{\partial Error}{\partial O_i} = -delta_j \frac{\partial net_j}{\partial O_i}$$

Odaziv čvora j:

$$net_j = \sum w_{i,j} O_i$$



Slika 11: $\sum_j -delta_j w_{i,j}$ je ukupan doprinos čvora i izlazu greške. Izvođenjem dobija se prilagodjavanje $w_{k,i}$

$$\frac{\partial net_j}{\partial O_i} = w_{i,j}$$

$$\frac{\partial Error}{\partial O_i} = -delta_j w_{i,j}$$

Sumiraju se sve veze čvora i za izlazni sloj:

$$\frac{\partial Error}{\partial O_i} = \sum_j -delta_j w_{i,j}$$

Udeljuje se senzitivnost greške mreže izlaznog čvora i na skrivenom sloju. Određuje se vrednost $delta_i$, senzitivnost greške mreže kod net nivoa aktivacije skrivenog čvora i. Time pruža senzitivnost greške mreže nadolazećih težina čvora i, a ponovo se radi izvod složene funkcije:

$$-delta_i = \frac{\partial Error}{\partial net_i} = \frac{\partial Error}{\partial Oi} * \frac{\partial O_i}{\partial net_i}$$

Pošto se koristi logistička aktivaciona funkcija:

$$\frac{\partial O_i}{\partial net_i} = O_i(1 - O_i)$$

$$-delta_i = O_i(1 - O_i) \sum_j -delta_j w_{i,j}$$

Izvodom složene funkcije dobija se:

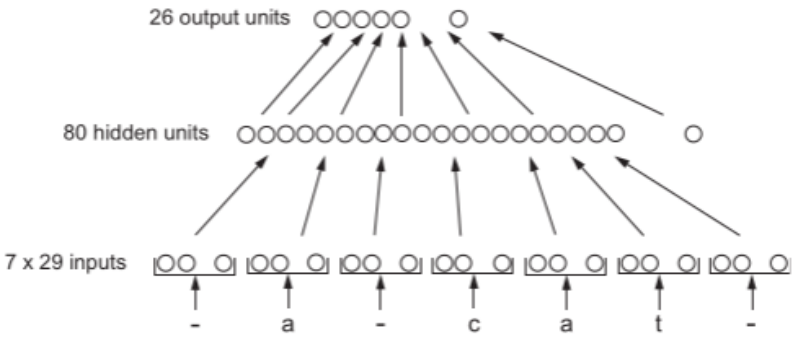
$$\frac{\partial Error}{\partial w_{k,i}} = \frac{\partial Error}{\partial net_i} \frac{\partial net_i}{\partial w_{k,i}} = -delta_i \frac{\partial net_i}{\partial w_{k,i}} = -delta_i x_k$$

$$\frac{\partial Error}{\partial w_{k,i}} = O_i(1 - O_i) \sum_j (-delta_j w_{i,j}) x_k$$

Za mreže sa više od 1 skrivenog sloja, iste procedure su primenjene rekurentno pri propagaciji greške iz skrivenog sloja n u skriven sloj n-1. Iako daje rešenje za problem obuke višeslojnih mreža back-propagation je besmilen bez sopstvenih poteškoća. Kao što pri pentranju uzbrdo, može konvergirati u lokalni minimum, po slici 9. Backpropagation može biti skup za sračunavanje, pogotovo kada mreža konvergira polako.

4.2 Backpropagation primer 1 : NETtalk

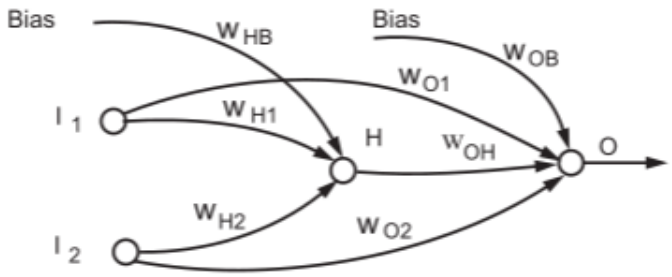
Zanimljiv primer rešenja neuronskih mreža pri problemu teže obuke (Sejnovski i Rozenberg '87.). NETtalk je obučen da izgovara engleski tekst. Ovo može biti težak zadatak za posebno namešten pristup simbolima, npr. po sistemu zasnovanom na pravilu pošto je engleski izgovor visoko neregularan. Obučen da čita niske teksta i vrati glas i asocirani stres. Glasovni izgovor se smatra kao jedinica zvuka u jeziku; stres je povezan sa bučnošću zvuka. Pošto izgovor jednog slova zavisi od njegovog konteksta i slova vezanog za njega. NETtalk 7 karakternih prozora. Kako tekst se pomera kroz prozor NETtalk vraća glasovni izgovor/stres par za svako slovo.



Slika 12: Mrežna topologija NETtalk-a

Slika 12. prikazuje arhitekturu NETtalk-a. Sadržana od 3 sloja jedinica. Ulazna jedinica odaziva se na prozor sedam karaktera 7 teksta. Svaka pozicija prozora reprezentira 29 ulaznih jedinica, od kojih za svako slovo alfabeta, 3 interpunkcijska znaka i blanko karaktera. Slovo svakoj poziciji aktivira odgovarajuću jedinicu. Izlazna jedinica enkodira glasovne izgovore koristeći 21 različitu karakteristiku ljudske artikulacije. Ostalih 5 jedinica enkodiraju stres i zavisnost sloga. NETtalk ima 80 skrivenih jedinica, 26 izlaznih vrednosti, 18629 veza. Poredi probane izgovore do tačnog izgovora i onda prilagođava težine backpropagation-a. Primer ilustruje broj zanimljivih karakteristika neuronskih mreža, većina njih reflektuje prirodu poučavanja ljudi. Npr., obuka kao procenat tačnih odziva, obavlja poslove rapidno na početku, potom usporava kako se procenat tačnosti uvećava. Težine se ažuriraju nasumično tako da budu otporni na nesuglasice, razlažu se dosledno pri ažuriranju težina. Smatra se da ponovno učenje u oštećenoj mreži je visoko efikasno. Višeslojne mreže imaju takođe zanimljiv koncept skrivene slojeve, omogućavajući mreži da generalizuje. Bilo koji algoritam obuke mora obučavati se nad generalizacijama koja su primenljiva nad neobazirućim instancama u oblasti problema. NETtalk ima manje neurona u skrivenom sloju nego u ulaznom. To znači da mu je namena enkodiranje informacija pri obuci obrazaca, gde neki oblik aptstrakcija zauzima mesto. Za prekraćeno enkodiranje sledi da za raznolike obrasce na ulaznom sloju je moguće mapirati na identične obrasce skrivenog sloja. Takva redukcija je generalizacija. NETtalk uči efektivno, neobavezno ono zahteva veći broj instanci za obuku, kao i višestruko pregledanje skupa podataka namenjenog za obuku. Neki od empirijski testirajućih poređenja backpropagation-om i ID3-em[5] za ovaj problem, ustanovilo se da algoritmi su imali jednake rezultate, iako su njihovi postupci pri obuci i pristupu podacima vrlo različiti. Vršeno je deljenje celog skupa podataka na onaj koji je namenjen za obuku i onaj za testiranje. Zajedno ID3 i NETtalk su bili u mogućnosti da izgovore 60% podataka test skupa podataka na 500 primera. ID3 je zahteva samo jednostruko pristupanje skupu podataka za obuku, dok NETtalk višestruko ponavljanje (100) pristupa elementima skupa za obuku. Uočava se ovime da simboličko i konekcionističko obučavanje je složenije nego što prvobitno liči.

4.3 Backpropagation primer 2: XOR - ekskluzivna disjunkcija



Slika 13: Mreža backpropagation-a rešava XOR problem.

e. In fact there is nothing unique Preslikavanja istinitonosnih vrednosti XOR-a:

$$(0,0) \rightarrow 0;$$

$$(0,1) \rightarrow 1;$$

$$(1,0) \rightarrow 1;$$

$$(1,1) \rightarrow 0;$$

Kroz ukupno 1400 ciklusa obuka korišćenjem 4 instanci proizvedene su sledeće vrednosti za težinske parametre.

$$W_{H,1} = -7.0, W_{H,2} = -7.0, W_{H,B} = 2.6, W_{O,B} = 7.0, W_{O,1} = -5.0, W_{O,2} = -4.0, W_{O,H} = -11.0$$

Za ulazne vrednosti (0, 0), izlaz skrivenog čvora je:

$$f(0 * (-7.0) + 0 * (-7.0) + 1 * 2.6) = f(2.6) \rightarrow 1$$

Izlaz izlaznog čvora (0,0) je:

$$f(0 * (-5.0) + 0 * (-4.0) + 1 * (-11.0) + 1 * (7.0)) = f(-4.0) \rightarrow 0$$

Za ulazne vrednosti (1, 0), izlaz skrivenog čvora je:

$$f(1 * (-7.0) + 0 * (-7.0) + 1 * 2.6) = f(-4.4) \rightarrow 0$$

Izlaz izlaznog čvora (1,0) je:

$$f(1 * (-5.0) + 0 * (-4.0) + 0 * (-11.0) + 1 * (7.0)) = f(2.0) \rightarrow 1$$

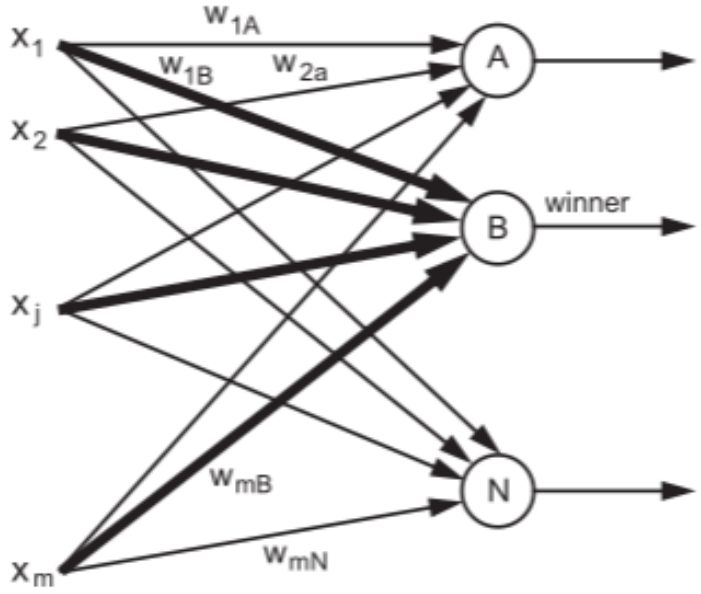
Tako isto i za ulaznu vrednost (0,1). Za ulazne vrednosti (1, 1), izlaz skrivenog čvora je:

$$f(1 * (-7.0) + 1 * (-7.0) + 1 * 2.6) = f(-11.4) \rightarrow 0$$

Izlaz izlaznog čvora (1,1) je:

$$f(1 * (-5.0) + 1 * (-4.0) + 0 * (-11.0) + 1 * (7.0)) = f(-2.0) \rightarrow 0$$

Moguće je uočiti da feedforward mreža sa backpropagation obukom nelinearnu separabilnost ovih tačaka instanci.



Slika 14: Sloj čvorova za primenu "Pobednik uzima sve" algoritma. Stari ulazi vektora podržavaju pobednički čvor.

5 Takmičarsko obučavanje

5.1 "Pobednik uzima sve" algoritam klasifikacije za obuku

Ovaj algoritam (Kohonen '84., Hecht-Nielsen '87.) radi sa jednim čvorom u sloju čvorova koji se odazove sa najjačim ulaznim obrascem. Vrši se takmičenje među čvorovima mreže, prikazan na slici 14. Ulazni vektor X = $\begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_m \end{bmatrix}$ prosleđen mreži čvorova A, B,..., N. Dijagram pokazuje da je čvor B pobednik takmičenja (on ima signalni izlaz 1). Obuka ovog algoritma je nenadgledana u odabiru pobednika po testiranju "maksimalne aktivacije". Težinski vektor pobednika je podstaknut dovođenjem njegovih komponenata bliže ulaznom vektoru. Težine W pobedničkog čvora i komponente X ulaznog vektora imaju uvećavanje za:

$$\Delta W_t = c(X_{t-1} - W_{t-1})$$

gde je c mala pozitivna konstanta obuke koja uobičajeno se smanjuje kako protiče obuka. Pobednički težinski vektor je prilagođen dodavanjem ΔW_t.

Podsticaj uvećan ili umanjen za svaku komponentu pobedničkog težinskog vektora po podeoku razlike $x_i - w_i$. Efekat je poklopiti pobednički čvor bliže ulaznom vektoru, a i po težinskog vektoru ulaza. Ovaj algoritam ne mora da sračunava direktno nivoe aktivacije da bi našao najjači odaziv. Za čvor i sa normalizovanim težinskim vektorom W_i, aktivacionim nivoom, W_iX je funkcija Euklidske razdraljenosti među W_1 i X. Sračunavanje se vrši po:

$$\|X - W_i\| = \sqrt{(X - W_i)^2} = \sqrt{X^2 - 2XW_i + W_i^2}$$

Iz ove jednačine moguće je uočiti da skup normalizovanih težinskih vektora sa najmanjom Euklidskom distancom, će biti onaj sa maksimalnom vrednošću aktivacije, WX. U više slučajeva lakše je odrediti pobednika po Euklidskoj razdaljini nego porediti nivoe aktivacija na normalizovanim težinskim vektorima. Uzima se u obzir Kohonenovo pravilo obuke pri algoritmu "pobednik uzima sve" zbog više razloga:

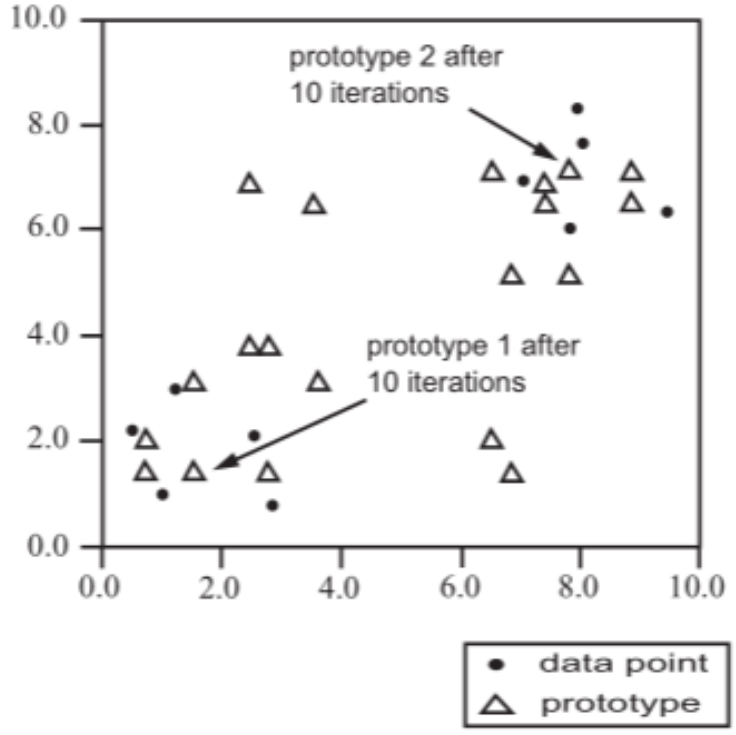
1. Uočava se klasifikatorni metod i poredi sa klasifikacijom perceptrona.

2. Moguće je kobinovati sa ostalim arhitekturama mreža koje pružaju sofisticiranije modele obuke.

Sagledamo, takođe, Kohonenovo prototip obučavanje outstarom sa nadgledanim obučavanjem mreže. Ovaj hibrid, prvobitno objavljen od Robert Hecht-Nielsen ('87., '90.) pod nazivom **suprotna propagacija mreže**. Pre toga, "pobednik uzima sve" algoritam ima mane koje treba ispoljiti. Parametar za "savest" je podešen i ažuriran svakom iteracijom da bi suzbio da pojedinačni čvorovi pobeđuju previše često. Osigurava se da za sve mrežne čvorove postepeno utiču na reprezentaciju prostora obrazaca. Nekim algoritmima identifikovan je pobednik koji uzima sve, skup bliskih čvorova odabrani su i težine za svaki blago su povećani. Sa druge strane moguć je pristup blagom podsticanju susednih čvorova pobednika. Težine su prvobitno nasumičnih vrednosti i potom normalizovane tokom metoda obučavanja (Zurada '92.). Hecht-Neilsen prikazuje kako ovaj tip algoritama može se jednačiti sa k-means analizom skupa podataka.

5.2 Kohonenova mreža za prototipsko obučavanje

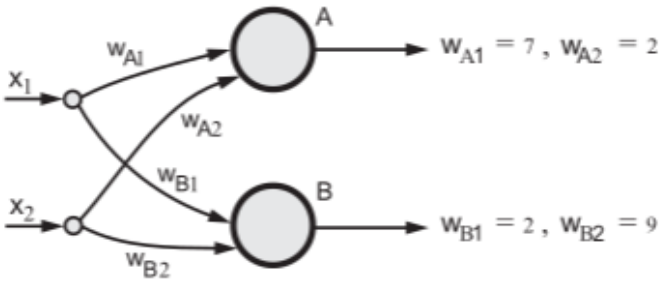
Klasifikacija podataka i uloga prototipova u obuci su razlog česte zabrinutosti psiholozima, lingvistima, informatičarima (Witgneštajn '53., Roš '78., Lakof '87.). Postoje simbolički predstavljene klasifikacije, kao i algoritmi verovatonosne klasterizacije (COBWEB, CLUSTER/2[6]). U konekcionističkim modelima, demonstriramo klasifikacije zasnovane na perceptronima gde pokazujemo Kohonenov algoritam "pobednika koji sve uzima". Primena Kohonenovog sloja, nenadgledanog, koji generiše



Slika 15: Primena Kohonenovog sloja, nenadgledanog, koji generiše niz prototipova predstavljenih u tabeli 2.

niz prototipova predstavljenih u tabeli 2 prikazana na slici 15. Nadređen nad ovim tačkama instanci koji su niz prototipova izgrađenih tokom obuke mreže. Algoritam obuka perceptrona konvergira

nakon svih brojnih iteracija rezultatu konfiguracije težina definisanjem sporednog sračunavanja Euklidskog "centra" svakog klastera. Centri služe da klaster pri perceptronskoj klasifikaciji prototipa klase. Kohonenova obuka je nenadgledana sa prototipovima nasumično generisanim i profinjenim do mere gde izrazito reprezentuju klastere podataka. Kako algoritam obavlja posao, konstanta obuke je napredujući umanjena zarad manjeg stresa prototipova pri pojavi novih ulaznih vektora. Kohonenova obuka (nalik CLUSTER/2) ima jak induktivni bias u brojnih poželjnih prototipova koji su izrazito identifikovani prvobitno radom algoritma i neprekidno vršeći profinjavanje. Ovo omogućuje dizajneru algoritma mreže da identifikuje određen broj prototipova za reprezentaciju više klastera podataka. Suprotna propagacija pruža dalje upravljanje određenim brojem prototipova.



Slika 16: Arhitektura Kohonenovog zasnovanog obučavanja mreže po tabeli 2 i klasifikaciji slike 15

Slika 16. je mreža Kohonenovog obučavanja za klasifikaciju tabele 2. Podaci predstavljeni u 2D koordinatnom sistemu, tako da prototipovi reprezentuju klastere podataka koji bi takođe bili uređeni parovi. Biraju se 2 prototipova, 1 je onaj koji predstavlja svaki od nekog klastera podataka. Nasumično se inicijalizuje čvor A na (7, 2) i B na (2, 9). Nasumična inicijalizacija radi samo pri probleimima uzorkovanja kao u ovom primeru. Alternativa je postavljanje težinskih vektora tako da jednače reprezentaciji svakog od klastera. Pobednički čvor će uzeti težinski vektor najbližeg ulaznom vektoru. Težinski vektor za pobednički čvor će biti podstaknut pomerajući ga još bliže ulaznim podacima, dok težine izgubljenih čvorova će ostati nepromenjene. Posebnim sračunavanjem Euklidske razdaljine ulazni vektor za svaki od prototipova neće imati potrebe za normalizacijom vektora.

Kohonenova obuka je nenadgledana i jednostavna mera razdaljine prototipa i tačke instance pruža mogućnost odabira pobednika. Klasifikacija će biti "nađena" u smislu samostalno organizovane mreže. Kohonenova obuka bira instance nasumičnim redosledom pri analizi, ovde će se instance uzimati tabelom 2 odozgo nadole. Tako za tačku (1, 1) se meri rastojanje za svaki prototip:

$$\|(1,1) - (7,2)\| = (1-7)^2 + (1-2)^2 = 37$$

$$\|(1,1) - (2,9)\| = (1-2)^2 + (1-9)^2 = 65$$

Čvor A (7,2) je pobednik pošto je najbliži (1,1). Sada se podstiče pobednički čvor, konstanta obuke c postavljena je na 0.5.

$$W^2 = W^1 + c(X^1 - W^1)$$

$$= (7,2) + .5((1,1) - (7,2)) = (7,2) + .5((1-7),(1-2))$$

$$= (7,2) + (-3,-.5) = (4,1.5)$$

U drugoj iteraciji obuka se vrši za instancu (9.4, 6.4):

$$\|(9.4,6.4) - (4,1.5)\| = (9.4-4)^2 + (6.4-1.5)^2 = 53.17$$

$$\|(9.4,6.4) - (2,9)\| = (9.4-2)^2 + (6.4-9)^2 = 60.15$$

Ponovo, čvor A je pobednik. Pri trećoj iteraciji:

$$W^3 = W^2 + c(X^2 - W^2) = (4, 1.5) + .5((9.4, 6.4) - (4, 1.5)) = (4, 1.5) + (2.7, 2.5) = (6.7, 4)$$

Za tačku (2.5, 2.1) pri trećoj iteraciji imamo:

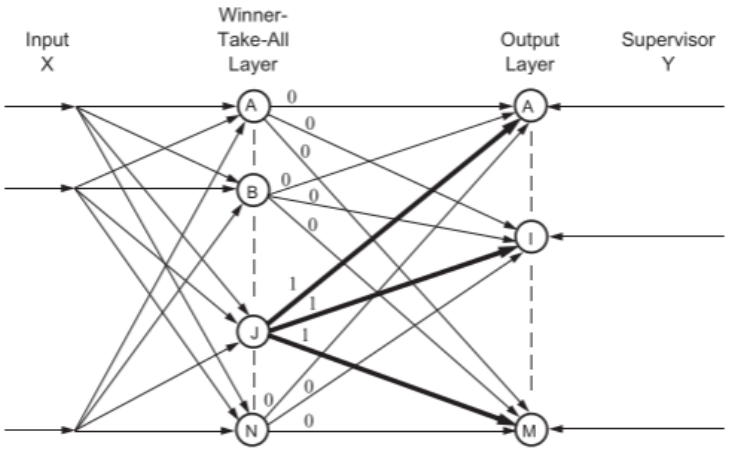
$$\|(2.5, 2.1) - (6.7, 4)\| = (2.5 - 6.7)^2 + (2.1 - 4)^2 = 21.25$$

$$\|(2.5, 2.1) - (2, 9)\| = (2.5 - 2)^2 + (2.1 - 9)^2 = 47.86$$

Čvor A odnosi ponovnu pobedu i sračunava se njegov novi težinski vektor. Slika 15. pokazuje razvijanje prototipa kroz 10 iteracija. Algoritam gradi podatke obarane nasumično po tabeli 2, prikazan prototip će biti različit od onog skorije izgrađenog. Napredak prototipa može biti primicanje do centra više klastera podataka. Pošto je ovo nenadgledani "pobednik uzima sve" podstičući algoritam, gradi se kao skup razvijajućih i posebno reprezentovanih prototipova više klastera podataka. Brojna istraživanja pored Zurade('92.) i Hecht-Nielsena('90.) ispticu da Kohonenova nenadgledana klasifikacija podataka je prosto ista kao k-means analiza. Uočava se, zajedno sa Grossberg-om, ili outstar-om, proširenje Kohonenove "pobednik uzima sve" analize, biće algoritam koji dopušta moćniji odabir prototipa.

5.3 Outstar mreže i suprotna propagacija

Do ovog stanovišta uočili smo nenadgledanu klasterizaciju ulaznih podataka. Obuka zahteva manje a priori poznavanje oblasti problema. Postepeno detektujući svojstvene karakteristike podataka, kao i istoriju obuke, što dovodi do identifikacije klasa i otkriće razilaženja među njima. Jednom tačke instanci su klasterizovane po sličnosti vektorskih reprezentacija, učitelj može asistirati uspostavljajući ravnotežu ili imenovati klase. Ovo je deo nadgledane obuke, gde se uzimaju izlazni čvorovi "pobednik uzima sve" sloja mreže i koristi ih kao ulaze drugog sloja mreže- Posebno će se posdstaknuti odlučivanje izlaznog sloja. Nadgledana obuka, pa i podsticanje izlaza pruža, npr. mapiranje rezultat Kohonenove mreže u izlazni obrazac ili klasu. Grosberg sloj ('82., '88.) implementira algoritam outstar koji pruža ovu pogodnost. Kombinovana mreža pridružena Grosberogovim slojem nazvana je suprotno propagirajućom i istaknuta je prvobitno od Rober Hecht-Nielsen-a ('87., '90.). Naspram Kohonenovog sloja, sada se stavlja u obzir Grosbergov sloj. Slika 17. prikazuje čvorove za slojeve



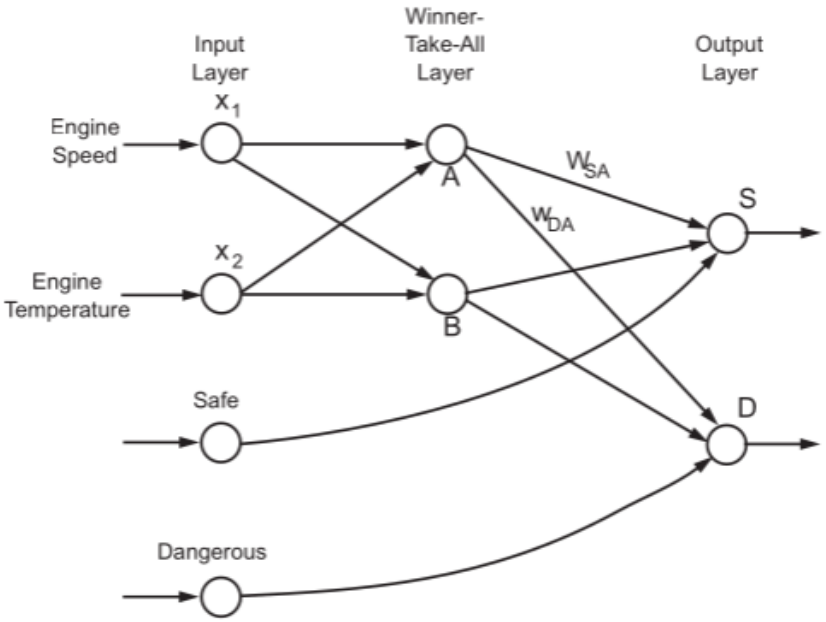
Slika 17: Outstar čvor J, pobednički za mrežu "pobednik uzima sve". Y vektor nadgleda odziv na izlaznom sloju u Grosbergovoj obuci. Outstar je podebljan na slici sa svim težinama vrednosti 1, ostale težine su 0.

A,B,...,N gde za jedan čvor J, odabran je pobednik. Obuka Grosberga je nadgledana u cilju da pruži

povratnu informaciju učitelju, kroz reprezentaciju vektora Y, podstičući težinu vezivanja J na čvor I u izlaznom sloju koji bi trebalo da okine. Sa outstar obukom, identifikujemo i uvećavamo težinu $w_{J,I}$ na spoljašnjim razilaženjima veza J do I. Obukom suprotnom propagacijom mreže vrši se prvo obuka Kohonenovog sloja. Kada pobednik se dobije, vrednost svih veza ispoljavaju se sa vrednostima 1, dok sve izlazne vrednosti njihovih suparnika ostaje 0. Čvor zajednosa svim čvorovima izlaznog sloja koji su vezivani, grade outstar. Obuka za Grosbergov sloj je zasnovan na outstar komponentama. Ako svaki klaster ulaznih vektora reprezentuje jednostruku klasu i traganjem za svim članovima klase da bi ih mapirali prema istim vrednostima izlaznog sloja, samim tim, ne zahtevajući iterativnu obuku. Želimo samo odrediti koji čvor je u sloju "pobednik uzima sve" vezan za koju klasu i tako utvrditi težine onih čvorova kojima su izlazni čvorovi zasnovani vezujući ih među klasama i željenih vrednosti izlaza. Npr. J-ta jedinica "pobednik uzima sve" je pobednička za sve elemente klastera gde je I = 1 kao željeni izlaz mreže, sa postavljenim $w_{J,I} = 1$ i $w_{J,K} = 0$ za sve težine na outstar-u J-tom. Ako je željeni izlaz za elemente klastera promenljiv, tada iterativna procedura, koristeći nadgledani vektor Y, prilagođava outstar težine. Rezultat ove procedure obuke je prosečna vrednost željenih izlaza za elemente određenog klastera. Jednačina kojom se vrši obuka nad težinama outstar veza pobedničkog čvora je:

$$W^{t+1} = W^t + c(Y - W^t)$$

Konstantu malog pozitivnog obučavanja je c, W_t je težina vektora outstar komponente i Y je željeni izlaz vektora. Uočava se da algoritam ovuke ima efekat uvećanja veza među čvora J iz Kohonenovog sloja i čvora I izlaznog sloja preciznije kada I je pobednički čvor sa izlazom 1 i željenim izlazom J koji je takođe 1. Ovo je primerak Hebbian obuke, način obuke kojim neuronska putanja je ojačana svaki put kako čvor doprinosti okidanju nekog drugog čvora. Naredno primenjujemo pravilo obuke mreže suprotne propagacije da bismo uočili klastere podataka tabele 2. Uvodi se koncept implementacije uslovljene obuke pri suprotnoj propagaciji. x_1 parametar tabele 2 reprezentuje sistem pogona motora. x_2 reprezentuje temperaturu motora. Oba parametra sistema su za uspostavljanje povoljnog stanja zarad proizvođenja tačaka instanci u opsegu od [0, 10]. Monitoring sistem podataka uzorkuje tačke instanci na regularnim intervalima. Kako brzina i temperatura su previsoke želimo istaći pažnju na oprez. Nudimo preimenovanje izlaznih vrednosti tabele 2 iz +1 u "bezbedno" i -1 u "opasno".



Slika 18: Mreža suprotnog propagiranja uočava klase tabele 2. Vrši se obuka outstar težina čvora A, $w_{S,A}$, $w_{D,A}$

Mreža suprotne propagacije prikazana je na slici 18. Kako znamo koje vrednosti unapred želimo da budu vrednosti čvorova pobednika Kohonenove mreže mapiraju se na izlazni sloj Grosbergove mreže, gde unapred podešavamo te vrednosti. Demonstracijom outstar obuke, obuku mreže vršimo primenom malo pre istaknute formule. Ako izvedemo (arbitrarnu) odluku da čvor S izlaznog sloja treba signalizirati bezbednu situiranost, a čvor D opasnu, time outstar težine čvora A izlaznog sloja Kohonenove mreže bivaju $\begin{bmatrix}1\\0\end{bmatrix}$, oustar težina za B bi trebalo biti $\begin{bmatrix}0\\1\end{bmatrix}$. Uoči simetriji situacija, možemo ispoljiti obučivanje outstar-a samo za čvor A. Kohonenova mreža biva stabilizirana pre mreže Grosberga pri obuci. Demonstrira se Kohonenova konvergencija. Ulazni vektori za obuku outstar-a čvora A koji je oblika $\begin{bmatrix}x_1\\x_2\\1\\0\end{bmatrix}$. x_1 i x_2 su vrednosti tabele 2 koji su klasterizacija Kohonenovog izlaznog čvora A i poslednje 2 komponente ističu da kada je čvor A pobednik Kohonen mreže, stanje biva *true* za bezbednost, $false$ za opasnost kao na slici 16. Inicijalizuje se outstar težine čvora A na $\begin{bmatrix}0\\0\end{bmatrix}$ i primenom .2 konstantom obuke:

$$W^1 = \begin{bmatrix}0\\0\end{bmatrix} + .2(\begin{bmatrix}1\\0\end{bmatrix} - \begin{bmatrix}0\\0\end{bmatrix}) = \begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}.2\\0\end{bmatrix} = \begin{bmatrix}.2\\0\end{bmatrix}$$

$$W^2 = \begin{bmatrix}.2\\0\end{bmatrix} + .2[\begin{bmatrix}1\\0\end{bmatrix} - \begin{bmatrix}.2\\0\end{bmatrix}] = \begin{bmatrix}.2\\0\end{bmatrix} + \begin{bmatrix}.16\\0\end{bmatrix} = \begin{bmatrix}.36\\0\end{bmatrix}$$

$$W^3 = \begin{bmatrix}.36\\0\end{bmatrix} + .2[\begin{bmatrix}1\\0\end{bmatrix} - \begin{bmatrix}.36\\0\end{bmatrix}] = \begin{bmatrix}.36\\0\end{bmatrix} + \begin{bmatrix}.13\\0\end{bmatrix} = \begin{bmatrix}.49\\0\end{bmatrix}$$

$$W^4 = \begin{bmatrix}.49\\0\end{bmatrix} + .2[\begin{bmatrix}1\\0\end{bmatrix} - \begin{bmatrix}.49\\0\end{bmatrix}] = \begin{bmatrix}.49\\0\end{bmatrix} + \begin{bmatrix}.10\\0\end{bmatrix} = \begin{bmatrix}.59\\0\end{bmatrix}$$

$$W^5 = \begin{bmatrix}.59\\0\end{bmatrix} + .2[\begin{bmatrix}1\\0\end{bmatrix} - \begin{bmatrix}.59\\0\end{bmatrix}] = \begin{bmatrix}.59\\0\end{bmatrix} + \begin{bmatrix}.08\\0\end{bmatrix} = \begin{bmatrix}.67\\0\end{bmatrix}$$

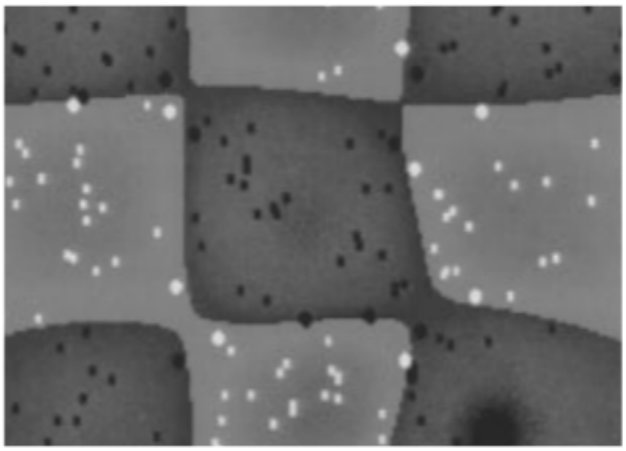
Moguće je uočiti da obukom težine su pomaknute bliže $\begin{bmatrix}1\\0\end{bmatrix}$. Elementi slučaja klastera vezanih sa čvorom A uvek se mapiraju na $\begin{bmatrix}1\\0\end{bmatrix}$, time smo koristili jednostavnu dodelu u algoritmu pre nego računanje proseka pri algoritamskoj obuci. Sada pokazujemo da dodela pruža povoljan odziv iz mreže suprotne propagacije. Ulazni vektor instance tabele 2 je primenjen na mrežu u slici 18. Dobijamo $\begin{bmatrix}1\\1\end{bmatrix}$ aktivaciju za outstar težine čvora A i $\begin{bmatrix}0\\0\end{bmatrix}$ za outstar čvora B. Skaralni proizvod aktivacija i težina čvora S za izlazni sloje $\begin{bmatrix}1\\0\end{bmatrix} * \begin{bmatrix}1\\0\end{bmatrix}$; čime se dobija aktivacija vrednosti 1 za S izlazni čvor. Sa težinama outstar-a čvora B obuka po $\begin{bmatrix}0\\1\end{bmatrix}$, aktivaciji za čvor D je $\begin{bmatrix}1\\0\end{bmatrix} * \begin{bmatrix}0\\1\end{bmatrix}$; i dobijamo željene vrednosti. Testiranje drugog reda tabele 2 rezultuje aktivaciju $\begin{bmatrix}0\\0\end{bmatrix}$ za čvor A i $\begin{bmatrix}1\\1\end{bmatrix}$ za čvor B za sloj "pobednik uzima sve". Skalarni proizvod vrednosti obučenih težina rezultuju vrednošću 0 za S čvor i 1 za D vrednošću 1, ponovo kao očekivanom. I tako dalje, za druge redove tabele 2. Iz stanovišta kognitivnog sagledanja, dobija se vezničko tumačenje mreže suprotnog propagiranja. Uzimajući u obzir sliku 18. obuka na sloju Kohonena ustanovljena kao dopremljivač uslovljenog stimulus-a kako mreža se obučava obrascima događaja. Obuka na nivou Grosberga je veza čvora (neuslovljeni stimuli) zarad odziva. U našem

slučaju, sistem obučen je da širi upozorenje na opasnost kada podaci su skladni po odgovarajućem obrascu. Kako je odziv usvojen obukom, tada i bez neprekidnog obučavanja od strane učitelja sistem se odaziva pogodno naspram novih podataka. Drugo stanovište kognitivnog sagledanja suprotnog propagiranja je podsticanje memorijskih veza obrazaca fenomena. Ovo je slično gradnji tabeli za nagledanje odaziva naspram obrazaca podataka. Suprotna propagacija ima, za određene slučajeve, odgovarajućih prednosti naspram backpropagation-a. Zajedno su sposobni da vrše nelinearno separabilnu klasifikaciju. Omogućeno je time što na Kohonenovom sloju vrši se obrada, gde skup podataka je particionisan na klastere homogenih podataka, što rezultuje značajnom napretku naspram backpropagation-a neke stope brzine obuke pri izrazito posebnom particionizanju podataka na podeljene klastere zamenjujući proširenu pretragu potrebnu pri backpropagation-u mreže skrivenog sloja.

5.4 Mašine potpornih vektora - SVM (Harrison, Luger '02.)

Još jedan primer takmičarske obuke. Ovim pristupom primenjuju se statističke mere korišćene za ustanovljavanje minimalnog skupa tačaka instanci (potpornih vektora) kojim maksimalno vrše podele pozitivnih i negativnih instanci naučenih koncepata. Ovi potporni vektori, reprezentuju odabrane tačke instanci iz oba skupa i pozitivnih i negativnih instanci koncepata, posredno definišući hiperravan deleći dva skupa podataka. Npr. izvršavajući SVM algoritam identifikuje tačke $\begin{bmatrix}2.5\\2.1\end{bmatrix}$ i $\begin{bmatrix}1.2\\3.0\end{bmatrix}$ i potporne vektore pozitivnih instanci $\begin{bmatrix}7.0\\7.0\end{bmatrix}$ i potpornih vektora negativnih instanci $\begin{bmatrix}7.8\\6.1\end{bmatrix}$ po tabeli 2 i sliku 6. Obukom potpornih vektora ostale tačke instanci nisu više neophodne. Dovoljni su samo potporni vektori zarad definisanja razdvajajuće hiperravni. SVM je linearni klasifikator gde obuka potrpornih vektora je nadgledana. Pretpostavlja se da podaci za SVM su proizvedeni i identični iz ufiksiranih, pritom nepoznatih, raspodela podataka. Hiperravan posredno definiše potporne vektore deli pozitivne od negativnih instanci skupa podataka. Tačke instanci najbliže hiperravni su u margini odluke (Burges '98.). Svako dodavanje ili oduzimanje potpornog vektora menja razdor uz hiperravan. Nakon obuke, moguće je rekonstruisati hiperravan i klasifikovati nove podatke naspram samostalnih potpornih vektora. SVM algoritam klasifikuje elemente podataka koji računajući razdaljinu tačaka instanci iz razdvajajućih hiperravni kao problemu optimizacije. Uspešno kontrolisanje uvećane fleskibilnosti višestrukih prostora karakteristika, često transformisanih parametara instanci zahteva da bi bile obučene sofisticiranu teoriju o generalizaciji. Teorija mora pružiti precizan opis karakteristika koje imaju kontrolu do oblika dobrog generalizovanja. Po statistici, nedostaci su poznati kao obuka stope uniformnog konvergiranja. Postoji mera PAC obuke[2] koja može ustanoviti odvajanje broja primeraka zahtevanih pokrićem na određenu grešku granice. Generalizacija zadataka se vrši Bajesovom tehnikom kompresije podataka ili nekom drugom. SVM obuka po teoriji Vapnik i Červonenkis(VC) je često korišćena, a time definisana je i VC dimenzija maksimalnog broja tačaka obuke koji su podeljeni na 2 kategorije po skupu funkcija (Burges '98.). Teorija VC pruža raspodelu slobodnih razdvajanja generalizacija pouzdanih hipoteza (Cristianini, Shawe-Taylor 2000). SVM algoritam primenjuje VC teoriju zarad sračunavanja hiperravni i upravljanje nad marginom greške pri generalizaciji tačnosti, nazvanom kapacitetom funkcije. SVM-ovi koriste skalarni proizvod sličan meri mapiranih podataka iz prostora karakteristika. Skalarani proizvod rezultuje reprezentaciju mapiranog vektora linearne kombinacije težina nađene uz rešavanje kvadratnog programa (Scholkopf '98.). Kernel funkija je polinom, splajn, Gausova kriva, koja je korišćena za rešavanje problema distribucije. SVM-ovi sračunavaju distance zarad ispoljavanja klasifikacije elemenata podataka. Pravila odluke pravljeni SVM reprezentacijom statističkim regularnostima u podacima. Po obuci SVM, klasifikacija novih tačaka instance je stvar prostog upoređivanja sa potpornim vektorima. Kritične karakteristike ispoljavajući koncepte obuke su klasterizovane na jednoj strani hiperravni, gde one druge što opisuju negaciju na drugoj strani. Karakteristike koje ne prave raskol nisu uračunate. Za perceptron algoritam je važilo da linearna separabilnost je važna. SVM kao alternativa ustupa maksimizaciju margine odlučivanja i izdržljiva je pri rukovanju sa lošijim razdvajanjima pri preplitanju tačaka instanci. Moguće

je koristiti promenljivu tolerantnosti zarad opuštanja linearnih ograničenja, potom nalaženja mekih margina, sa vrednostima koje naznače nivo pouzdanosti razdvajanja. Kao rezultat potpornih vektora outliers-a (diskriminirajućih tačaka) mogu biti loše klasifikovane zarad generisanja hiperravni i tako rezultat margine odluge biva sužen pri šumu podataka. SVM-ovi mogu biti uopšteni naspram 2 problema kategoričke klasifikacije na obilaženje više klasa ponovnom primenom SVM-a za svaku kategoriju dobrobiti naspram drugih kategorija. SVM su najbolje usaglašeni za probleme numeričkih podataka pre nego kategoričkih; kao rezultat njihova primenljivost za većunu klasičnih kategorizujućih problema sa kvalitativnim razdvajanjima je ograničena. Snaga leži u njihovoj matematičkoj utemeljenosti: minimalizacija kvadratne funkcije pod ograničenjem linearne nejednačine. SVM su primenjene na većinu situacija obuka, uključujući klasifikaciju veb stranica. Kategorizaciom teksta, vrednovana težinom je pristunost pretrage ili ostalih srodnih reči. Svaki dokument ponaosob postaje ulazni podatak za SVM da bi kategorizovao zasnovanost informacije učestalosti reči (Harrison, Luger '02.), usresređujući se na detekciju ivica i opis oblika korišćenjem informacija grayscale ili intenziteta boje.



Slika 19: SVM obuka šahovska tabla po tačkama generisanim u skladu sa uniformnom raspodelom koristeći Gausove kernels-e. Tačke instanci podataka sa većim tačkama ustanovljenim za skup potpornih vektora, tamnije površine ukazuju na pouzdanost klasfikacija (Cristianini i Shawe-Taylor '00.).

Na slici 19. po Critanini-u, Shawe-Taylor-u ('00.), SVM obilazi razdor na šahovskoj tabli.

6 Hebova obuka slučajnosti

6.1 Uvod

Hebova teorija obuke je zasnovana na sagledanju bioloških sistema pri doprinosu neurona pri ispaljivanju prema drugom neuronu, time putanja ili veza između 2 neurona se smatra da je ojačana. Hebb ('49.) ustanovio: Kada akson ćelije A je blizak da nadraži ćeliju B ponovno ili neprestano zauzima se okidanjem, proces rasta ili metaboličke promene zauzima mesto u jednoj ili obe ćelije tako da uvećava efikasnost kako A ćelija okida na B ćeliju. Hebova obuka je zgodna pošto uspostavlja podsticanje zasnovano ponašanjem koncepcije na neuronskom nivou. Neuronsko psihološko istraživanje potvrdilo je Hebb-ovu ideju da privremeni vidik okidanja vezanih neurona modifikuju jačinu sinapsi, čak u većini složenije tehnike Hebovog prostog "uvećanja u efikasnosti", da makar aproksimativno bude što tačnija. Ova obuka pripada kategoriji slučajnosti pravila obuke koji ishoduju da težine ovih kategorija budu njihova lokanlna svojstva vremena i prostora. Hebova obuka korišćena u brojnim mrežnim arhitekturama. Primenjena u režimima nadgledanog i nenadgledanog obučavanja. Efekat ojačavanja veza između 2 neurona, kad jedan doprinosi ispaljivanju drugog, može biti simuliran matematički podešavajući

težine njihovih veza po konstantnom broju prilika pomnoženom sa znakom proizvoda izlaznih vrednosti. Neka budu i i j povezane tako da izlaz i-tog je ulaz j-tog. Definišemo prilagođavanja težina veza između ΔW, potom je znak $c * (o_i * o_j)$. Gde konstanta c kontroliše stopu brzine obuke. Po tabeli 3

O_i	O_j	$O_i * O_j$
+	+	+
+	-	-
-	+	-
-	-	+

Tabela 3: Znakovi i proizvodi znakova vrednosti izlaznih čvorova

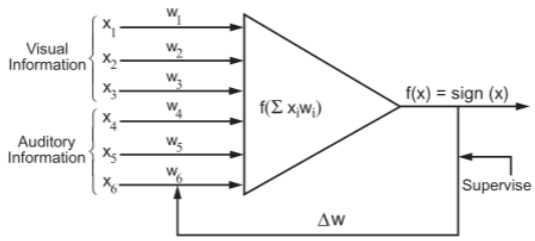
O_i je znak izlazne vrednosti i-tog i O_j je izlaza j-tog. Iz prve linije tabele oba znaka su pozitivna i njegovo prilagodjavanje težine ΔW je pozitivno. Efekat ojačavanja veze i i j, gde i-ti doprinosi j-tom okidanju. Za ostala 3 reda su drugačije okolnosi gde inkrement posledično ima odgovarajući znak. Ovaj mehanizam efekta podsticanja putanje među neuronima kada oni imaju slične signale, ili inače spečavaju protok.

6.2 Nenadgledano Hebbian obučavanje

Odazivu nenadgledane obuke nije zgodno reći šta je "tačan" izlaz; težine su modifikovane čisto kao funckija izlaznih i ulaznih vrednosti neurona. Obuka ove mreže pruža efekat ojačavanja mrežnog odziva po obrascima koji su već viđeni. Npr. tehnike Hebbiana mogu sprovesti odaziv obukom, arbitrarno odabranim stimulus-om mogu biti uslovljeni zarad željenog odaziva. Težina prilagođena ΔW za čvor i-ti u nenadgledanoj Hebbian obuci je:

$$\Delta W = c * f(X, W) * X$$

c je konstanta učenja, mali pozitivni broj, f(X,W) je i-ti izlaz i X je i-ti ulazni vektor. Sada pokazujemo mrežu kojom primenjujemo Hebbian obuku premeštanja odaziva iz primarnog neuslovljenog stimulus-a u uslovljeni stimulus. Ovo omogućuje da model vrste obučavanja pod uzorom na eksperimente Pavlov sindroma, gde istovremenim darivanjem hrane se ozvučava i zvonce, tako pseće curenje bale je sada prouzrokovano i tim ozvučenjem.



Slika 20: Primerak neurona za primenu hibridnog čvora Hebbian-a gde je obučavanje nadgledano

Na slici 20. postoje 2 sloja, ulazni sa 6 čvora i izlazni sa 1 čvorom. Izlazni čvor vraća vrednosti -1,+1 gde se ističe pobuđujuće okidanje ili umirujuće stanje. Neka je c = 0.2. Vrši se obuka obrasca [1, -1, 1, -1, 1, -1] konkatenacijom [1, -1, 1] za neuslovljeni stimulus i [-1, 1, -1] novi stimulus. Pretpostavlja se da mreža pozitivno deluje na neuslovljeni stimulus sa težinskim vektorom [1, -1, 1], a umirjuće deluje na novi stimulus. Neuslovljeni stimulus se poklapa sa ulaznim obrascem, dok umirujuć odaziv mreže je predstavljen kroz težinski vektor [0,0,0]. Konkatenacijom ova 2, dobija se prvobitni težinski vektor mreže [1, -1, 1, 0, 0, 0]. Vrši se obuka mreže po ulaznim obrascima, nadamo se sprovođenju

konfiguracija težina koje proizvode mrežni odziv po novom stimulus-u. A prva iteracija daje:

$$\begin{aligned} W * X &= (1 * 1) + (-1 * -1) + (1 * 1) + (0 * -1) + (0 * 1) + (0 * -1) \\ &= (1) + (1) + (1) = 3 \end{aligned}$$

$$f(3) = sign(3) = 1.$$

$$\begin{aligned} W^2 &= [1, -1, 1, 0, 0, 0] + .2 * (1) * [1, -1, 1, -1, 1, -1] \\ &= [1, -1, 1, 0, 0, 0] + [.2, -.2, .2, -.2, .2, -.2] \\ &= [1.2, -1.2, 1.2, -.2, .2, -.2] \end{aligned}$$

Time dobija se prilagodjena mreža po originalnim obrascima ulaza:

$$\begin{aligned} W * X &= (1.2 * 1) + (-1.2 * -1) + (1.2 * 1) + (-.2 * -1) + (.2 * 1) + (-.2 * -1) \\ &= (1.2) + (1.2) + (1.2) + (+.2) + (.2) + (.2) = 4.2 \end{aligned}$$

$$sign(4.2) = 1.$$

$$\begin{aligned} W^3 &= [1.2, -1.2, 1.2, -.2, .2, -.2] + .2 * (1) * [1, -1, 1, -1, 1 - 1] \\ &= [1.2, -1.2, 1.2, -.2, .2, -.2] + [.2, -.2, .2, -.2, .2, -.2] \\ &= [1.4, -1.4, 1.4, -.4, .4, -.4] \end{aligned}$$

Moguće je primetiti da proizvod vektora $W * X$ se povećava pozitivnim smerom, sa apsolutnom vrednošću svakog elemnta težinski vektor se uvećava za .2 svakim ciklusom obuke. Nakon 10 iteracija Hebbian obuke:

$$W^{13} = [3.4, -3.4, 3.4, -2.4, 2.4, -2.4]$$

Obučen težinski vektor koristimo za testiranje odziva mreže na dva odvojena obrasca. Videli bismo ako mreža nastavlja odgovarati na neuslovljeni stimulus pozitivno i važno je ako mreža zauzima pozitivan stav za nov, uslovljen stimulus. Testiranje mreže prvobitno neuslovljenog stimulus-a [1,-1,1]. Popunjavamo zadnja 3 arhumenta sa nasumičnim 1 i -1 dodelama. Npr. tako se dobije [1, -1, 1, 1, 1, -1]:

$$\begin{aligned} sign(W * X) &= sign((3.4 * 1) + (-3.4 * -1) + (3.4 * 1) + (-2.4 * 1) + (2.4 * 1) + (-2.4 * -1)) \\ &= sign(3.4 + 3.4 + 3.4 - 2.4 + 2.4 + 2.4) \\ &= sign(12.6) = +1 \end{aligned}$$

Drugo testiranje novim generisanim novim vektorom [1, -1, 1, 1, -1, -1]:

$$\begin{aligned} sign(W * X) &= sign((3.4 * 1) + (-3.4 * -1) + (3.4 * 1) + (-2.4 * 1) + (2.4 * -1) + (-2.4 * -1)) \\ &= sign(3.4 + 3.4 + 3.4 - 2.4 - 2.4 + 2.4) \\ &= sign(7.8) = +1. \end{aligned}$$

Ovim odgovorom je izmerena čista aktivacija kao ojačana ponovnim ispoljavanjem stimulus-u. Drugo testiranje novim generisanim novim vektorom [1, 1, 1, -1, 1, -1]:

$$\begin{aligned} sign(W * X) &= sign((3.4 * 1) + (-3.4 * -1) + (3.4 * 1) + (-2.4 * 1) + (2.4 * 1) + (-2.4 * -1)) \\ &= sign(3.4 - 3.4 + 3.4 + 2.4 + 2.4 + 2.4) \\ &= sign(10.6) = +1. \end{aligned}$$

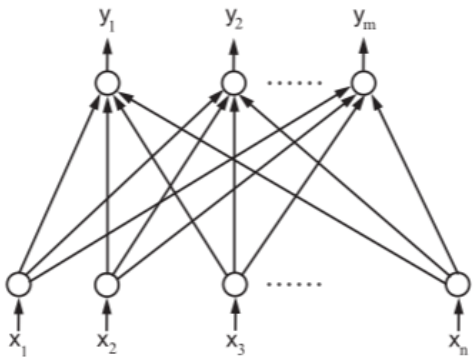
Sporednim stimulus-om obrazac je iznova prepoznat. Konačno, za vektor [1, -1, -1, 1, 1, -1] koji je parcijalno poremećen:

$$\begin{aligned} sign(W * X) &= sign((3.4 * 1) + (-3.4 * -1) + (3.4 * 1) + (-2.4 * 1) + (2.4 * 1) + (-2.4 * -1)) \\ &= sign(3.4 + 3.4 - 3.4 - 2.4 + 2.4 + 2.4) \\ &= sign(5.8) = +1. \end{aligned}$$

Takođe je izvršeno prepoznavanje. Šta je model Hebbian obuke proizveo? Napravila se veza među novog stimulus-a i starog odziva ponovnim ispoljavanjem starog i novo stimulusa đuture. Mreža se obučava da transferuje odziv po novom stimulus-u bez ikakvog nadgledanja. Time ojačava se senzitivnost i dozvoljava mreži da se odazove na isti način blago poremećenog stimuli-a. Ovo je dostignuto korišćenjem Hebbian slučajne obuke uvećavanjem snage mrežnog odziva svih obrazaca, povećava se efektom povećavanja snage mrežnog odziva svake komponente obrasca ponaosob.

6.3 Nadgledana Hebbian obuka

Pravilo Hebbian obuka je zasnovano na principu povećavanja jačine veza između neurona kako 1 neuron doprinosi okidanju drugog. Ovaj princip prilagodio se nadgledanom obučavanju zasnivanjem po prilagođavanju težina veza za željeni izlaz neurona, pre nego prilagođavanju težina stvarnog izlaza. Npr., ako mreža od neurona A do neurona B ima izlaz, a poželjan odziv neurona B je pozitivan, tada se povećava težina veze od A do B. Posmatramo uređene parove $\{< X_1, Y_1 >, < X_2, Y_2 >, ..., < X_t, Y_t >\}$, gde X_i, Y_i su vektori obrazaca koji su povezani. X_i ima dužinu n, a Y_i dužinu m.



Slika 21: Nadgledana Hebbian mreža obuke veza obrazaca

Posebno izrazito se dizajnira ova mreža za ovu okolnost. Po slici 21. Počinje se sa prvobitnim oblikom formule Hebbian obuke:

$$\Delta W = c * f(X, W) * X$$

f(X, W) je stvarna vrednost mrežnog izlaznog čvora. Nadgledajućoj obuci zamenjujemo joj nju sa željenim vektorom izlaza D, pa je:

$$\Delta W = c * D * X$$

Datim parom vektora $< X, Y >$ primenjuje se prabilo obuke za k-ti čvor izlaznog sloja:

$$\Delta W_{i,k} = c * d_k * x_i$$

gde $\Delta W_{i,k}$ je prilagodjavanje težina na i-tom ulazu k-tog čvora izlaznog sloja, dk je željeni izlaz k-tog čvora, x_i je i-ti element X. Primenjuje se formula za sve težine svih čvorova na izlaznom sloju.

$X = \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix}$ i vektor $Y = \begin{bmatrix} d_1 \\ d_2 \\ ... \\ d_m \end{bmatrix}$ izlazni. Primenom formula svako posebno prilagodjavanje težine kroz čitam izlazni sloj i prikupljanjem pojmova, dobijamo formulu:

$$\Delta W = c * Y * X$$

gde je $Y * X$ vektorski proizvod, prikazan matricom: $\begin{bmatrix} y_1 * x_1 & y_1 * x_2 & \dots & y_1 * x_m \\ y_2 * x_1 & y_2 * x_2 & \dots & y_2 * x_m \\ \dots & \dots & \dots & \dots \\ y_n * x_1 & y_n * x_2 & \dots & y_n * x_m \end{bmatrix}$.

Da bismo obučili mrežu čitavog skupa uparenih parova, petljom kružimo kroz parove, prilagodjavanjem težina svakog para $< X_i, Y_i >$ po:

$$W_{t+1} = W_t + c * Y_i * X_i$$

Za čitav skup obuke dobija se: $W_1 = W_0 + c(Y_1 * X_1 + Y_2 * X_2 + ... + Y_t * X_t)$, gde W_0 je prvobitna konfiguracija težina. Ako se inicijalizuje W_0 kao 0 vektor, i za konstant obuku važi c = 1, sledećom formulom dobija se dodeljivanje težina: $W = Y_1 * X_1 + Y_2 * X_2 + ... + Y_t * X_t$ Mapirajuća mreža ulaznih vektora ka izlaznim vektorima korišćenjem formule dodeljivanja težine vektora koja je nazvana linearnim veznikom. Imamo mreže linearnih veznika zasnovanih na pravilu Hebbian obuke. Primena je omogućena i bez izrazito posebne obuke. Analiziraju se svojstva linearnog veznika. Model skladišti višestruke veze matrici po težinskim vektorima. Omogućuje interakciju među skladištenim obrascima.

6.4 Asocijativna memorija i linearni veznik

Predlog od strane Tuevo Kohonena ('72.) i Džejms Andersona ('77.). Služi za skladištenje i oporavak memorijom. Uočava se različitost oblika memorijskog pohranjavanja, uključujući heteroasocijativne, autoasociativne, interpolativne modele. Analizira se linearni veznik mreže sa implementacijom interpolativne memorije zasnovane na Hebbian obuci. Konačno, krajnji razmatrani problemi će biti interferencija i crosstalk. Pogodno pri enkodiranju obrazaca u memoriji. Počinje se sagledanjem memorije uz obrazloženja definicija. Obrasci i vrednosti memorije prikazuju se kao vektori. Vektori karakteristika instanci skupa podataka izvedeni su od uspostavljenog bias-a. Veze skladištene u memoriji prikazane su kao $\{< X_1, Y_1 >, < X_2, Y_2 >, ..., < X_t, Y_t >\}$. Svakom paru vektora $< X_i, Y_i >$: X_i je ključ po kom se dobija Y_i obrazac. Tu su 3 tipa asocijativnih memorija:

1. Heteroasociativna: Ona mapira iz X u Y tako da ako arbitrarni vektor X je blizi vektoru X_i nego neki drugi primerak, tada je Y_i povratna vrednost.

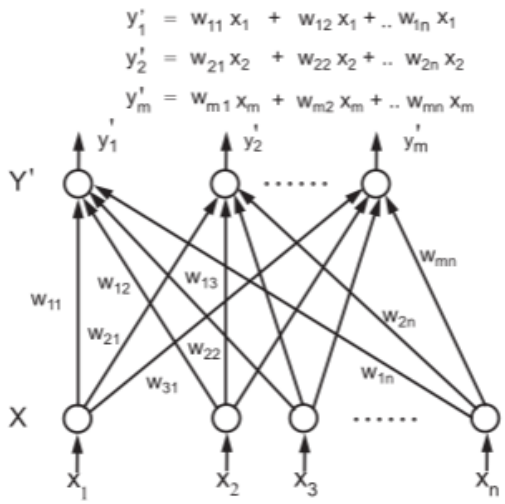
2. Autoasociativna: Mapiranje kao heteroasocijativna osim što je $X_i = Y_i$ za sve parove primeraka. Kako svaki obrazac X_i je u relaciji sa samim sobom, ovakav oblik memorije korišćen kada delimično ili poremećeno nadolazeći stimulus obrazac služi odzivu čitavog obrasca.

3. Interpolativna: mapiranje $\Phi : X \to Y$ takvo da $X = X_i + \Delta i$, izlaz je $\Phi(X) = \Phi(X_i + \Delta i) = Y_i + E$, a $E = \Phi(\Delta i)$. Ako ulazni vektor za primerak X_i se vezuje za, tada se pohranjuje Y_i. Ako se razlikuje od primerka vektora Δ gde $E = \Phi(\Delta)$.

Autoasociativne i heteroasocijativne memorije korišćene za dobijanje originalnih primeraka. Ustanovljena je memorija, u pravom smislu, po dobitku obraska koji je čitava kopija skladištenog obrasca. Želeli bismo da konstruišemo obrazac izlaza tako da bude različit obrascu skladištenom u memoriji na sistematizovan način, što funkcija intepolativne memorije.

Na slici 22. prikazan linearni veznik mreže koji implementira oblik interpolativne memorije. Po temelju Hebbian nadgledane obuke. Mreža inicijalizacija težina opisana je po jednačinama izvedenim kao i u prethodnoj lekciji:

$$W = Y_1 * X_1 + Y_2 * X_2 + ... + Y_t * X_t$$



Slika 22: Mreža linearnog veznika. Vektor X_i uveden kao ulaz i vezni vektor Y' kao proizvedeni izlaz. y'_i je linearna kombinacija x ulaza. Obuka svakog y'_i pohranjena je tačnim izlaznim signalima.

Time vršenje inicijalizacija težina je opisano po jednačini izvedenoj kao u prethodnoj lekciji gde je udeljena težina, mreža se poklapa jednim primerkom; inače vrši se interpolativno mapiranje. Uvode se koncepti i oznake zarad olakšavanja naše analize ophođenja ovih mreža. Prvobitno uvodi se metrika koja nam dozvoljava definisanje precizne distance među vektorima. Svi naši obrasci vektora u primeru su Hamming vektori, sastojanih od vrednosti iz -1, +1. Hamming rastojanje opisuje rastojanje 2 Hamming vektora. Definišemo Hamming prostor: $H_n = X = (x_1, x_2, ..., x_n)$, gde $\forall x_i \in \{-1, +1\}$. Hamming rastojanje definisano za bilo koja 2 vektora iz Hamming prostora po: $\|X, Y\|$ = brojnost komponenti po kojima se X i Y razlikuju. Daje se primer računanja Hamming rastojanja, za 4D Hamming prostor, među: (1, -1, -1, 1) i (1, 1, -1, 1) je 1 (-1, -1, -1, 1) i (1, 1, 1, -1) je 4 (1, -1, 1, -1) i (1, -1, 1, -1) je 0.

2 definicije sleduju:

1. Definiše se komplement Hammingovog vektora gde elementi se prebacuju na suprotnu vrednost, npr. (1, -1, -1, -1) u (-1, 1, 1, 1).

2. Definiše se ortonormiranost vektora. Vektori su ortonormirani jedinični, ortogonalni, normirani. 2 ortonormirana vektora, množe se skupa skalarnim proizvodom, vektorski proizvod svih njih im jednači 0. Ortonormirani skup vektora, kad X_i i X_j se izmnože dobije se 0, osim ako $i = j$: $X_i X_j = \delta_{i,j}$ gde $\delta_{i,j} = 1$ kada $i = j$, inače 0. Mreža linearnog veznika definiše iznad prateća 2 svojstva, sa $\Phi(X)$ predstavljajući mapiranje funkcije mreže.
   a) Ulazni obrazac X_i koji se tačno poklapa jednim od primeraka, izlaz mreže $\Phi(X_i)$ je Y_i, vezni primerak.
   b) Ulazni obrazac X_k, koji se ne poklapa nekim od primeraka, $\Phi(X_k)$ je Y_k izlaz mreže, tako da biva linearna interpolacija X_K. Preciznije $X_k = X_i + \Delta i$, gde X_i je primerak, a mreža vraća $Y_k = Y_i + E$, gde $E = \Phi(\Delta i)$. Za ulaz mreže X_i je 1 primerak, i vraća se pogodan primerak.
   $$\Phi(X_i) = W X_i$$
   , po definiciji aktivacione funkcije mreže.
   $$W = Y_1 X_1 + Y_2 X_2 + ... + Y_i X_i + ... + Y_n X_n$$

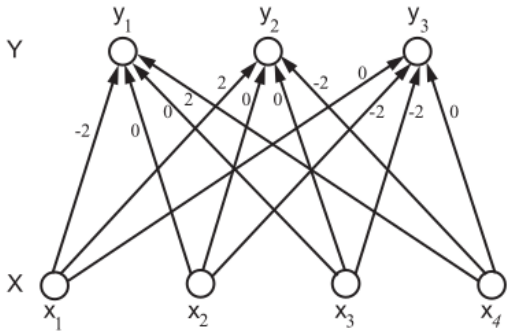
dobija se:

$$\begin{aligned}\Phi(X_i) &= (Y_1X_1 + Y_2X_2 + ... + Y_iX_i + ... + Y_nX_n)X_i \\ &= Y_1X_1X_i + Y_2X_2X_i + ... + Y_iX_iX_i + ... + Y_nX_nX_i \text{ (po distributivnosti)} \\ &= Y_1\delta_{1,i} + Y_2\delta_{2,i} + ... + Y_i\delta_{i,i} + ... + Y_n\delta_{n,i} \text{ (gde } X_iX_j = \delta_{i,j}) \\ &= Y_1 * 0 + Y_2 * 0 + ... + Y_i * 1 + ... + Y_n * 0 = Y_i. \\ &\textit{(po uslovu ortonormiranosti, } \delta_{i,j} = 1\textit{, gde } i = j\textit{, inače je 0)}\end{aligned}$$

Time omogućeno je pokazati da za X_k ne jednači se bilo koji primerkom, mreža izvodi interpolativno mapiranje. $X_k = X_i + \Delta i$, gde X_i je primerak,

$$\begin{aligned}\Phi(X_k) &= \Phi(X_i + \Delta_i) \\ &= Yi + E,\end{aligned}$$

gde Y_i je vektor vezan X_i-em i $E = \Phi(\Delta_i) = (Y_1X_1 + Y_2X_2 + ... + Y_nX_n)\Delta_i$



Slika 23: Mreža linearnog veznika. Težinska matrica sračunata korišćenjem prethodne lekcije.

Datim primerom procesuiranja linearnog veznika. Slika 23. nudi prost prikaz linearnog veznika mreže koji mapira vektor 4 elemenata X u vektor 3 elemenata Y. Kako radimo u Hamming prostoru, aktivaciona funkcija mreže f je znak funkcije korišćen ranije. Ako želimo skladištiti prateće veze 2 vektora $< X_1, Y_1 >, < X_2, Y_2 >$ tako da:

$$X_1 = [1, -1, -1, -1] \rightarrow Y_1 = [-1, 1, 1],$$

$$X_2 = [-1, -1, -1, 1] \rightarrow Y_2 = [1, -1, 1].$$

Korišćenjem formule inicijalizacije težina za linearne veznike, sa vektorskim proizvodom po prethodnoj lekciji: $W = Y_1X_1 + Y_2X_2 + ... + Y_nX_n$, Računa se $Y_1X_1 + Y_2X_2$, gde težina matrice za mrežu je:

$$W = \begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 0 & 0 & 2 \\ 2 & 0 & 0 & -2 \\ 0 & -2 & -2 & 0 \end{bmatrix}$$

Primenjuje se linearni veznik nad 1 primerkom. Počev od X = [1, -1, -1, -1] prvim primerkom dobije se vezujući Y:

$y_1 = (-2 * 1) + (0 * -1) + (0 * -1) + (2 * -1) = -4$, i $sign(-4) = -1$,

$y_2 = (2 * 1) + (0 * -1) + (0 * -1) + (-2 * -1) = 4$, i $sign(4) = 1$,

$y_3 = (0 * 1) + (-2 * -1) + (-2 * -1) + (0 * -1) = 4$, i $sign(4) = 1$.

Pošto $Y_1 = [-1, 1, 1]$, druga polovina parova primeraka je vraćena. Sledeći prikaz primera linearne interpolacije primerka je dat. Razmatra se vektor X [1, -1, 1, -1]:

$y_1 = (-2 * 1) + (0 * -1) + (0 * 1) + (2 * -1) = -4$, i $sign(-4) = -1$,
$y_2 = (2 * 1) + (0 * -1) + (0 * 1) + (-2 * -1) = 4$, i $sign(4) = 1$,
$y_3 = (0 * 1) + (-2 * -1) + (-2 * 1) + (0 * -1) = 0$, i $sign(0) = 1$.

Primeti se da $Y = [-1, 1, 1]$ nije primerak iz Y. Uočava se mapiranje koje osigurava zajedničke vrednosti koje imaju 2 Y primerka. [1, -1, 1, -1] X vektora ima Hamming rastojanje 1 za svaka 2 X primerka; izlazni vektor [-1, 1, 1] isto ima Hamming distancu 1 naspram svako drugog Y primerka. Zaključuje se nakon saglednja linearnih veznika. Željeno svojstvo linearnih veznika zavisi od zahteva da obrasci primeraka ustanove ortonormirani skup. Ovo je onemogućujući faktor na 2 načina:

1. Nema očiglednih mapiranja okolnosti sveta u ortonormirani vektor obrazaca.

2. Kapacitet obrazaca koji su skladišteni je ograničen po dimenzionalnosti prostora vektora. Ortonorminost nije ispunjena, interferencija između skladištenih obrazaca nastaje, ishodom fenomena zvanog crosstalk.

Sagleda se tkakođe kako linearni veznik ustupa vezu Y primeraka samo kada ulazni vektor se tačno poklopi sa primercima X vektora. Kada tu nema tačnog poklapanja na ulaznim obrascima, rezultat je intepolativno mapiranje. Upitno je da li interpolacija se nalazi u memoriji pod obavezno, time je potrebno implementirati funkciju stvarnog preuzimanja iz memorije, po aproksimaciji primeraka dobijenih tačnim obrascima koji su srodni. Potrebno je čvorište zgodnosti (atraktivnosti) za hvatanje vektora u okružujućem regionu.

7 Atraktivne mreže ili "Uspomene"

7.1 Uvod

Do ovog dela diskutovana je sve vreme je ispoljavana feedforward obuka, gde informacije su prikazane skupom ulaznih čvorova i signalom koji protiče napred kroz čvorove ili slojeve čvorova dok neki rezultat se ne dobije. Feedback mreže su važan deo konekcionističkih mreža. Arhitektura mreža je različita u smislu da izlazni signali čvora petljom kruze nazad, (ne)posredno, kao i ulaz prema tom čvoru. Feedback mreže se razlikuju naspram feedforward mreža na različite načine:

1. prisutnost feedback konekcija između čvorova,

2. vreme odlaganja, tj. nepožuriva propagacija signala,

3. izlaz mreže je stanje mreže do konvergnecije,

4. korisnost mreže zavisi od svojstava konvergiranja.

Kada feedback mreža dosegne iteraciju u kojoj se više ne menja, to stanje se smatra ekvilibrijumom. Stanje koje mreža doseže na ekvilibrijumu ustanovljena je da bude izlaz mreže. Ulazni obrazac inicializuje stanje mreže. Stanje mreže na ekvilibrijumu je obraza dopremljen iz memorije. BAM primerom ćemo sagledati mreže koje vrše implementaciju na heteroasocijativnoj memoriji, a za Hopfildove mreže koristimo autoasociativne memorije. Kognitivni aspekti ovih memorija da su oba važna i zanimljiva. Pružaju nam model za adresiranje sadržaja u memoriji. Ova vrsta asocijatora opisuje pridobijanje broja telefona, osećaja tuge prisećanja starih uspomena, ili čak prepoznavanje osobe delimično vidljivim licem. Istraživači su pokušavali da istraže većinu vezujućih aspekata za ovakvu memoriju u strukturama zasnovanim nad simbolima, uključujući semantiku mreže, okvira, sistema objekata, kao u gradnji upravljanja algoritama za pretragu prostora stanja[3]. Atraktorom je definisano stanje prema kom susedni region se razvija generacijski. Svaka atraktor mreža će imati region gde stanje mreže unutar regiona se razvijati do uloge atraktora. Region se naziva basin (sliv, pripajajuće čvorište). Atraktor može trajati u mreži jednostrukog stanja ili nizu stanja kroz koje mreža petljom kruži. Pokušaji da se razumu atraktori i njihovi basins-i matematički je izvelo polje funkciji energiji mreže (Hopfield '84.).

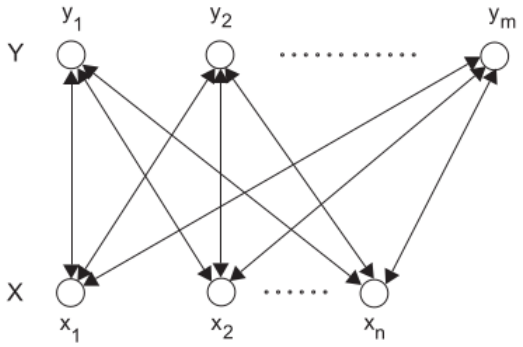
Feedback mreže sa funkcijom energije koje imaju svojstvo da svakom tranzicijom mreže se smanjuje ukupna energija mreže sa garantovanom konvergencijom. Atraktor mreže mogu se koristiti za implementaciju adresiranja sadržaja memorije po instaliranju željenih obrazaca kao atraktora u memoriji. Mogu, takođe rešiti optimizacione probleme, kao što su problem putujućeg trgovca, mapiranja među funkcijama cena u optimizacionim problemima i energiji mreže. Rešenje ovog problema je odrađeno tako što se koristi takozvana Hopfield-ova mreža.

7.2 BAM - Dvosmerna asocijativna memorija

BAM mreža prvobitno objavljena od Bart Kosko-a ('88.), sadržana 2 potpuno povezana slojva procesirajućih elemenata. Mogu takođe biti veze feedback (povratne sprege) za svaki čvor ponaosob.

BAM mapiranje n-dimenzionalnog ulaznog vektora X_n u m-dimenzionalni izlazni vektor Y_m predstavljen na slici 23. Kako veza među X i Y je povratna, daju se težine vezivanja sa obostranim tokom informacija. Kao kod težina linearnog veznika, težine BAM mreže poprimljene su unapred. Koriste se isti metodi za sračunavanje težina mreže kao i kod linearnog veznika. Vektori BAM arhitekture su dobijene Hamming vektorima. Dati N vektor parovi koji pripremaju skup primeraka koji su pogodni za skladištenje, gradi se matrica koja je izvedena u lekciji asocijativnih memorija i linearnih veznika:

$$W = Y_1X_1 + Y_2X_2 + ... + Y_iX_i + ... + Y_tX_t$$



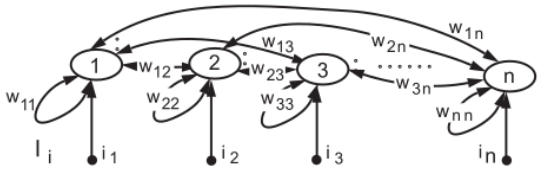
Slika 24: BAM mreža. Svaki čvor je povezan sa samim sobom.

Jednačina istaknuta po težinama X sloja i Y sloja, kao što je sagledano na slici 24. Npr. za $w_{3,2}$ je težina veze iz 2. jedinice X sloja na 3. jedinicu Y sloja. Pretpostavlja se da 2 čvora grade putanju između. To jest, težine veza čvorova na X i Y slojevima su identični u oba smera. Pritom, težinska matrica od X na Y je ona transponovana nad težinskom matricom W. BAM mreža može biti transofrmisana u autoasocijativnu mrežu po korišćenju inicijalizacije istih težina formula na skupu veza $< X1, X1 >, < X2, X2 >, \ldots$

Rezultovanjem identičnosti X i Y slojeva iz ove procedure moguće je eliminisati Y sloj, rezultujući da mreža liči na mrežu sa slike 25. BAM mreža je korišćena za dobavku obrazaca iz memorije pri inicijalizaciji X sloja sa ulaznim obrascima. Ako ulazni obrazac je šum ili nepotpuna verzija nekog primerka, BAM može često upotpuniti obraza i dobaviti veznički obrazac. Odaziv podataka sa BAM, dešava se naredno:

1. Primena prvobitnog para vektora (X, Y) za obrađivanje elemenata. X je obraza za koji želimo dobaviti primerak. Y je nasumično inicijalizovan.

2. Propagira se informacija od X sloja na Y sloj i ažuriraju se vrednosti na Y sloju.



Slika 25: Autoasociativna mreže sa ulaznim vektorom I_i. Pretpostavljamo da jedinstvene veze među čvorovima bivaju jedinstvene grane(veze), tako $w_{i,j} = w_{j,i}$ i težinska matrica je simetrična.

3. Šalje se ažurirana informacija Y u povratku prema X sloju, ažurirajući X jedinice.
4. Nastavlja se vršenje nadolazećih 2 koraka dok vektori se ne stabilizuju, to je moguće sve dok promene ne prestanu nad vrednostima vektora X i Y.

Algoritam koji je istaknut daje BAM-u svoj feedback tok, što je povratna kretnja prema ekvilibrijumu. Nadolazeći skup instrukcija bio bi započet obrascem Y nivoa vodeći se prema konvergiranju, odabira primeraka X vektora. Potpunom dvosmernošću: moguće je uzeti X vektor kao ulaz i uzeti Y vezu po konvergenciji, ili je moguće uzeti Y vektor ulaza i povratiti X vezom. Sledeći primer će dočarati bliže ovu okolnost.

Koristi se vektor dobavljanjem ostalih vektora para primerka. Hamming razdaljina merena je prema poređenju komponenti vektora, brojnošću svakog razlika elemenata. Zbog ograničenja ortonormiranosti, BAM konvergira tako da vektor postaje atraktor. Par stvari su uključene pri BAM konvergenciji:

- Ako previše primeraka je mapirano na metricu težina, primerci sami od sebe mogu biti previše zbliženi i prouzrokovati stanje pseudo-stabilnosti (imitacije stabilnosti) u mreži. Ovaj fenomen je takozvani crosstalk, nastaje u lokalnom minimumu prostora energije mreže.
- Detaljnijim procesiranjem BAM-a, proizvod množenja ulaznih vektora matrica težina sračunavaju sume proizvoda uparenih vektora koji rezultuju da u mreža liči na onu na slici 25. Ako se osvrnemo na primer iz lekcije o autoasocijativnoj mreži svaki element izlaznog vektora. Prosta funkija praga aktivacije preovdi rezultujući vektor natrag u Hamming prostor.

$$net(Y) = WX, \text{ ili } \forall Y_i \text{ kompoentu } net(Y_i) = \sum w_{i,j} * x_j,$$

sa sličnim odnosom po X sloju. Prag funkcije f za $net(Y)$ u periodu $t+1$ je direktno:

$$f(net^{t+1}) = \begin{cases} +1, & net > 0 \\ f(net^t), & net = 0 \\ -1, & net < 0 \end{cases}$$

Slikom 26. ističe se mala BAM mreža, prosta varijacija linearnog veznika. Mreža mapirana vektorom X sa 4 elemenata u 3 elemenata vektora Y i naopačke. Pretpostavlja se da pravimo 2 primerka parova vektora:

$$x_1 = [1, -1, -1, -1] \leftrightarrow y_1 = [1, 1, 1]$$
$$x_2 = [-1, -1, -1, 1] \leftrightarrow y_2 = [1, -1, 1]$$

Pravi se matrica težina koristeći primenu formula predstavljenom u prethodnoj lekciji:

$$\begin{aligned} W &= Y_1X_1^t + Y_2X_2^t + ... + Y_nX_n^t \\ &= \begin{bmatrix} 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -2 & -2 & 0 \\ 2 & 0 & 0 & -2 \\ 0 & -2 & -2 & 0 \end{bmatrix} \end{aligned}$$

Težinski vektor za mapiranje iz Y u X je $W^T = \begin{bmatrix} 0 & 2 & 0 \\ -2 & 0 & -2 \\ -2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix}$

Biraju se određeni vektori i testira se BAM veznik. Počev primerkom para, bira se X komponenta i gleda ako je dobijeno Y. Neka je $X = [1, -1, -1, -1]$:

$$Y_1 = (1 * 0) + (-1 * -2) + (-1 * -2) + (0 * -1) = 4, f(4) = 1,$$

$$Y_2 = (1 * 2) + (-1 * 0) + (-1 * 0) + (-1 * -2) = 4, f(4) = 1,$$

$$Y_3 = (1 * 0) + (-1 * -2) + (-1 * -2) + (-1 * 0) = 4, f(4) = 1$$

.

Sledeća polovina para primeraka dobija se kao povratna vrednost. Moguće je ručno proveriti da za Y vektor po ulaznom vektoru važi i proveriti da li je originalan vektor $X = [1, -1, -1, -1]$ vraćen.

Npr. moguće je uzeti u obzir X vektor [1, 1, 1, -1], sa Y nasumično inicijalizovanim. Mapira se X uz BAM mrežu:

$$Y_1 = (1 * 0) + (1 * -2) + (1 * -2) + (-1 * 0) = -4, f(4) = -1,$$

$$Y_2 = (1 * 2) + (1 * 0) + (1 * 0) + (-1 * -2) = 4, f(4) = 1,$$

$$Y_3 = (1 * 0) + (1 * -2) + (1 * -2) + (-1 * 0) = -4, f(4) = -1.$$

Ovo rezultuje da je funkcijom praga aktivacije dobijeno za [-4, 4, -4], [-1, 1, -1]. Mapiranje natrag na X daje:

$$X_1 = (-1 * 0) + (1 * 2) + (-1 * 0) = 2,$$

$$X_2 = (-1 * -2) + (1 * 0) + (-1 * -2) = 4,$$

$$X_3 = (-1 * -2) + (1 * 0) + (-1 * -2) = 4,$$

$$X_4 = (-1 * 0) + (1 * -2) + (-1 * 0) = -2.$$

Funkcija praga aktivacije primenjena ponovo, kao malo pre, daje originalni vektor [1, 1, 1, -1]. Početni vektor izvodi stabilan rezultat sa 1. prevodima, tako deluje da je otkriven još jedan par prototip primera. Primerom odabrane je komplement originalnog vektora primerka $< X2, Y2 >$. BAM mrežom kada par vektora je ustanovljen prototipom primeraka ističe se komplement tih parova. Ti prototipovi su.

$$X_3 = [-1, 1, 1, 1] \leftrightarrow Y_3 = [-1, -1, -1],$$

$$X_4 = [1, 1, 1, -1] \leftrightarrow Y_4 = [-1, 1, -1].$$

Neka odabirom vektora bliskog X primerku dobijen [1, -1, 1, -1]. Uočava se da Hamming rastojanje najbliža 4 X primerka je 1. Sledi nasumična inicijalizacija vektora Y na [-1, -1, -1]:

$$Y_{1,t+1} = (1 * 0) + (-1 * -2) + (1 * -2) + (-1 * 0) = 0,$$

$$Y_{2,t+1} = (1 * 2) + (-1 * 0) + (1 * 0) + (-1 * -2) = 4,$$

$$Y_{3,t+1} = (1 * 0) + (-1 * -2) + (1 * -2) + (-1 * 0) = 0.$$

Evaluacijom funkcije mreže $f(Y_{i,t+1}) = f(Y_{i,t})$ kada $y_{i,t+1} = 0$, po funkciji praga aktivacije pri kraju BAM procesiranja. Y je [-1,1,-1] tokom nasumične inicijalizacije prvog i trećeg parametra YT na -1. Sa Y ide se na X:

$$X_1 = (-1 * 0) + (1 * 2) + (-1 * 0) = 2,$$

$$X_2 = (-1 * -2) + (1 * 0) + (-1 * -2) = 4,$$

$$X_3 = (-1 * -2) + (1 * 0) + (-1 * -2) = 4,$$
$$X_4 = (-1 * 0) + (1 * -2) + (-1 * 0) = -2.$$

Funkcija praga aktivacije mapirana je na rezultat vektora X = [1,1,1,-1]. Ponovnim procesiranje povratkom na vektor Y:

$$Y_1 = (1 * 0) + (1 * -2) + (1 * -2) + (-1 * 0) = -4,$$
$$Y_2 = (1 * 2) + (1 * 0) + (1 * 0) + (-1 * -2) = 4,$$
$$Y_3 = (1 * 0) + (1 * -2) + (1 * -2) + (-1 * 0) = -4.$$

Funkcija praga aktivacije primenjena na [-4, 4, -4] ponovo rezultuje Y = [-1, 1, -1]. Ovaj vektor je identičan većini verzija Y, tako je mreža stabilna. Pokazano je da prolaskom kroz BAM mrežu, obrazac što je bliži X_4 on konvergira skladištenom primerku. Ovo bi bilo slično prepoznavanju lica ili skladištene slike sa nedostacima ili oštećenjima. Hamming rastojanje između originalnog X_4 vektora [1, -1, 1, -1] je bio 1. Vektor ustaljen u $< X_4, Y_4 >$ paru primerka. Moglo je inicijalno na početku se inicijalizovati i X ako bi to bilo nužno. Hecht-Nielsen ('90.) predstavlja zanimljivu analizu BAM mreže. Demonstrira ortonormirano svojstvo za mrežu linearnog veznika podržanom po BAM koji je previše ograničavajući. Ovo pruža argumen prikaza da zahtev građenja mreže je takav da su vektori linearno nezavisni, tako da, tu nema vektora koji nastaje linearnom kombinociom drugih vektora u prostoru primeraka.

7.3 Autoasociativna memorija and Hopfield mreže

Džon Hopfild, fizičar iz Kalifornijskog tehnološkog instituta, uzrokovao je najviše da konekcionističke arhitekture budu zapažene. Istraživao je svojstva konvergencija, koncepte minimalizacije energije. Dizajnirao je porodicu mreža zasnovanim na tim principima. Kao fizičar, razumeo je fizičke fenomene minimalizacije energetskih tačaka fizičkog sistema. Primer ovog pristupa je analiza simuliranog kaljenja za hlađenje metala. Prvo ćemo sagledati osnovne karateristike feedback vezničkih mreža. Mreža obrađuje signal kroz feedback putanje dok doseže stabilno stanje. Korišćenjem ovih arhitektura asocijativna memorija želeli bismo mrežu da ima 2 svojstva:

1. Počev od inicijalnog stanja poželjno je imati pouzdanost da će mreža konvergirati u neko stabilno stanje.

2. Poželjno je da stabilno stanje bude najbliskije ulaznom stanju po nekoj metrici rastojanja.

Moramo sagledati prvenstveno autoasociativnu mrežu koja je razvijena po istim principima kao BAM mreža. Razmatra se prethodna lekcija BAM mreža gde transformacija u autoasociativnu mrežu se vrši identičnim vektorima u X i Y pozicijama. Rezultat transformacije je simetrična kvadratna matrica težina. Slika 24. ističke primer. Težinska matrica za autoasociativnu mrežu skladišti skup vektora primeraka $X_1, X_2, \ldots, X_n$ je generisana po:

$$W = \sum X_i X_i^t, \forall i = 1, 2, \ldots, n.$$

Kada autoasociativnu memoriju iz heteroasocijativne izvodimo, težina čvorova x_i na x_j će biti identične isto tako od x_j na x_i, što omogućuje simetričnost. Pretpostavka samo zahteva da 2 obrađivana elemenata budu povezana jednom putanjom posedujući jednu težinu.Takođe je uočiti specialan slučaj neuronske pouzdanosti, tako da ni jedan čvor mreže biva direktno vezan samim sobom. U ovoj okolnosti glavna dijagonala težinske matrice su sve 0. Kako BAM radi težine matrice zasnovane na obrascima bivaju skladištene u memoriju. Razjašnjujemo prost primer. Razmatrajući 3 vektora skupa primeraka:

$$X_1 = [1, -1, 1, -1, 1],$$

$$X_2 = [-1, 1, 1, -1, -1],$$

$$X_3 = [1, 1, -1, 1, 1].$$

Sračunava se težina matrica koristeći $W = \sum X_i X_i^t,\ \forall i = 1, 2, 3$

$$W = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 \\ -1 & -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 \\ -1 & -1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & -1 & -1 & 1 & 3 \\ -1 & 3 & -1 & 1 & -1 \\ -1 & -1 & 3 & -3 & -1 \\ 1 & 1 & -3 & 3 & 1 \\ 3 & -1 & -1 & 1 & 3 \end{bmatrix}$$

Koristi se funkcija praga aktivacije:

$$f(net_{t+1}) = \begin{cases} +1, & net > 0 \\ f(net^t), & net = 0 \\ -1, & net < 0 \end{cases}$$

Prvo se testira primerak $X_3 = [1, 1, -1, 1, 1]$, da dobijemo:

$X3 * W = [7, 3, -9, 9, 7]$, i sa funkcijom praga aktivacije, [1, 1, -1, 1, 1]. Uviđa se momentalno samostalno stabilizovanje vektora. Ovo prikazuje da primerci su sami po sebi stabilna stanja atraktora. Sledeći testirajući vektor kojem Hamming rastojanje je 1 po primerku X_3. Mreža bi trebalo povratiti primerak. Ovo se jednači preuzimanju obrazaca memorije iz delimično oštećenih podataka. Biranjem $X = [1, 1, 1, 1, 1]$:

$$X * W = [5, 1, -3, 3, 5].$$

Korišćenjem funkcije praga aktivacije dobija se X_3 vector [1, -1, -1, 1, 1]. Naredno uzima se 3. primer, ovaj put vektor čije Hamming rastojanje je 2, pa i toliko udaljeno od najbližeg prototipa. Neka $X = [1, -1, -1, 1, -1]$. Proverljivo je da vektor za 2 udaljen od X_3, za 3 od X_1, za 4 od X_2. Otpočinjemo: $X * W = [3, -1, -5, 5, 3]$, gde prag aktivacije snabdeva [1, -1, -1, 1, 1]. Ovim ne ističe se ništa, ni tačka stabilnosti, time: $[1, -1, -1, 1, 1] * W = [9, -3, -7, 7, 9]$, gde $[1, -1, -1, 1, 1]$. Mreža je sada stabilna, ali bez ikakvih skladištenih uspomena. Nađen novi energetski minimum? Bližim posmatranjem ustanovljava se da novi vektro je komplement originalnog X_2 primerka [-1, 1, 1, -1, -1]. Ponovo, slučaj heteroasocijativne BAM mreže, naša autoasociativna mreža gradi atraktore za originalne primerke kao što su naspram njihovih komplemenata, gde u tom slučaju 6 atraktora za sve primerke. Sa ovog stanovišta u ovom tumačenju, sagledane su autoasociativne mreže zasnovane na modelu memorijskih linearnih veznika. Cilj, jedan od njih, Džon Hopfield-a je bio da se pruži opštija teorija autoasociativne mreže koja bi primenila jednoslojnu feedback mrežu ispunjavajući određen skup prostih ograničenja. Za ovu klasu jednoslojnih feedback mreža Hopfield je dokazao da bi uvek postojala pouzdana konvergencija funkcija energije mreže. Dalji Hopfield-ov cilj je bio da zameni diskretni period ažurirajućeg modela korišćenog prošli put sa onim koji bliže određuje neprekidnu obradu perioda stvarnih neurona. Uobičajen način za simuliranje neprekidno periodično asinhornog ažuriranja u Hopfield-ovim mrežama je da se ažuriraju čvorovi svaki posebno nego po sloju. Ovo je obavljeno korišćenjem procedure nasumičnog biranja za prikupljanja sledećeg čvora mreže koji je ažuriran jednako često. Struktura Hopfieldove mreže je identična autoasocijativnoj mreži iznad: jedan sloj čvorova potpuno povezanih (po slici 24.). Nivo aktivacije, aktivaciona funkcija, prag aktivacije rade kao i pre.

Za čvor i:

$$xinew = \begin{cases} +1, & \sum_j w_{i,j} x_j^{old} > T_i \\ x_i^{old}, & \sum_j w_{i,j} x_j^{old} = T_i \\ -1, & \sum_j w_{i,j} x_j^{old} < T_i \end{cases}$$

Data arhitektura, samo jedna udaljena ograničenost je zahtev za karakterizaciju Hopfildove mreže. Ako $w_{i,j}$ je težina konekcije u čvor i iz čvora j, definišemo Hopfield mrežu kao neku koja poštuje ograničenja težina:

$$w_{i,i} = 0, \qquad \forall i$$
$$w_{i,j} = w_{j,i}, \quad \forall i, j$$

Hopfield mreža nema tipično vezan sa njim metod obuke. Kao BAM njegove težine uobičajeno se sračunavaju unapred. Ophođenje Hopfield mreže je sada bolje razumljiv nego bilo koja druga klasa neurona osim perceptrona. Ovo je tako pošto ophođenje može okarakterisano u smislu utemeljenije funkcije energije otkrivene Hopfieldom:

$$H(X) = -\sum_i \sum_j w_{i,j} x_i^{new} x_j^{new} + 2 \sum_i T_i x_i$$

Prikazujemo funkciju energije kao takvu da ima svojstvo da svaka tranzicija mreže umanjuje ukupnu energiju mreže. Kako je H unapred određen minimum i svaki put njegovim umanjenjem umanjuje i konačnu količinu fiksnog minimuma, zaključuje se da iz bilo kog stanja mreža konvergira. Prvo pokazuje se arbitrarni prosirajući element k koji vrlo skoro ažuriran, k menja stanje akko H se umanjuje. Promena energije ΔH je:

$$\Delta H = H(X^{new} - X^{old}) \textit{ (Prošrenjem jednačine vrši se definisanje H, dobija se)}$$
$$\Delta H = -\sum_i \sum_j w_{i,j} x_i^{new} x_j^{new} - 2 \sum_i T_i x_i^{new} + \sum_i \sum_j w_{i,j} x_i^{old} x_j^{old} + 2 \sum T_i x_i^{old}$$

Gde je samo x_k izmenjen, $x_{i,new} = x_{i,old}$ za $i \neq k$. Što pokazuje da suma x_k ne sadrži međusobni odbitak svakog izlaza ponaosob. Preuređivanje i sakupljanje pojmova:

$$\Delta H = -2x_k^{new} \sum_j w_{k,j} x_j^{new} + 2T_k x_k^{new} + 2x_k^{old} \sum_j w_{k,j} x_j^{old} - 2T_k x_k^{old}.$$

Koristimo se pravilom $w_{i,i} = 0$ i $w_{i,j} = w_{j,i}$ we can finally rewrite this as:

$$\Delta H = 2(x_k^{old} - x_k^{new})[\sum_j w_{k,j} x_j^{old} - T_k].$$

Na 2 načina se pokazuje da je ΔH negativno:

1. pretpostavkom da x_k se menjao od -1 na +1. Tada izraz unutar uglastih srednjih zagrada biva pozitivan da x bude +1. Kako je $x_k^{old} - x_k^{new} = 2$, ΔH mora da bude negativan.

2. Pretpostavkom da x_k se menjao od 1 na -1, istim načinom razmišljanja, ΔH mora biti negativna. Ako x_k nema promenu stanja, $x_k^{old} - x_k^{new} = 0$, $\Delta H = 0$.

Dobijenim rezultatom, preko ma kakvog početnog stanja mreže ono konvergira obavezno. Nadalje, stanje mreže pri konvergenciji ono mora dostići lokalnu minimalnu energiju. Ako ne bi postojale, tada bi postojala tranzicija koja bi nadalje umanjila ukupnu energiju mreže i ažurirala algoritam odabila kako bi pod uticajem okolnosti odabralo čvor za ažuriranje. Predstavljeno je da Hopfield mreže imaju 2 svojstva kojim bi mreži se uspostavilo implementiranje asocijativne memorije. Ispostavlja se da Hopfield mreža ni nema, u opštem smislu, 2. željenu svojstvenost: ne konvergira stalno pri stabilnom stanju bližih inicijalnih stanja. Nema opšteg metoda popravljanja ovog problema. Ostaju par problema zasnovana na energetkom pristupu konekcionističkih mreža:

1. Energetsko stanje doseže potrebu da bude sistemski globalni minimum.
2. Hopfield mreže ne moraju konvergirati do najbližeg atraktora prema ulaznom vektoru. Ovo je neugogodna situacija pri implementaciji sadržajima adresibilnih memorija.
3. Primenom pri optimizaciji, nema opšteg metoda kreiranja mapiranja ograničenja u Hopfield funkciju energija.
4. Postoji ograničenje na brojnost minimuma energije koji je skladišten i preuzet iz mreže, što je važnije, nije moguće podesiti preciznije ovaj broj. Empirijsko testiranje ovih mreža pokazuje broj atraktora malih razlomaka brojeva čvorova u mreži.

Appendix

Implementacija backpropagation algoritma nad skupom podataka iris[7][8]

Daje se implementacija 1. Vrši se inicijalizacija korena posrednog pseudo-nasumičnog generisanja brojeva. Za svaku epohu se vrši praćenje tačnosti. Učitava se skup podataka iris[9] sastojanom od 4 karakteristike (features-a) 'sepal length in cm', 'sepal width in cm', 'petal length in cm', 'petal width in cm', a klasnim atributom nad kolonom 'class' višestrukih kategoričkih vrednosti ('Iris-Setosa', 'Iris-Versicolour', 'Iris-Virginica'). Vrši se kastovanje svih karakteristika u vrednost tipa pokretnog zareza (float) za svaku instancu. Enkodira se klasni atribut u celobrojnu vrednost (int). Uzima se kao binarizovano enkodirano u obzir. Nakon toga se vrši prilagođavanje vrednosti podataka tako da izvršava se normalizacija podataka na neklasifikatornim atributima skupa podataka čime se dostiže skaliranje vrednosti tako što se preslika u opsegu [0,1] kao vrednost, time su preraspodeljene za lakšu upotrebu. Definiše se brojnost međuevaluacionih podela cross-validaion-a i holdout estimation procena (folds-a), stopa brzine obuke, brojnost ponovnih ciklusa obuke (epoha), brojnost neurona u skrivenom sloju (`n_hidden`). Vrši se poziv implementirane funkcije `evaluate_algorithm(...)` na izvršavanje, sa pomenutim argumentima i callback funkcijama `back_propagation(...)` i `transfer_sigmoid(...)` kao aktivaciona funkcija - sigmoidna, tzv. logistička funkcija. Konačno se ispisuju po 3 folds-a tačnosti i središnja tačnost ukupne evaluacije.

```
seed(1)

accuracies = list()

dataset = load_csv()
for i in range(len(dataset[0])-1):
  str_column_to_float(dataset, i)
str_column_to_int(dataset, len(dataset[0])-1)
minmax = dataset_minmax(dataset)
normalize_dataset(dataset, minmax)
n_folds = 3
l_rate = 0.3
n_epoch = 500
n_hidden = 5
print('----------------------------------------')
scores = evaluate_algorithm(dataset,
                            back_propagation,
                            n_folds, l_rate, n_epoch, n_hidden,
                            transfer_sigmoid)
print('Scores (per fold): %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

Implementacija 1: Površni pregled programa

Daje se implementacija 2. Vršenje nalaženja para maksimalnih i minimalnih vrednosti po svakoj koloni.

```
def dataset_minmax(dataset):
    minmax = list()
    stats = [[min(column), max(column)] for column in zip(*dataset)]
    return stats
```

Implementacija 2: `dataset_minmax`

Daje se implementacija 3. Vrši se preskaliravanje kolona po opsegu $[0, 1]$:

```
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)-1):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])
```

Implementacija 3: `normalize_dataset`

Daje se implementacija 4. Radi zove se `cross_validaion_split(...)`, kojim se dobijaju foldovi nad kojim, osim 1 fold-a namenjenog za validaciju, se vrši validacija, testiranje predviđanja

`algoritm(...)` (tj. callback-om unet `back_propagation(...)`), poklapanja predviđanja vrednosti klasifikatornog atributa i hvata tačnost validacije fold-a `accuracy_metric(...)`.

```
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, fold, *args)
        actual = [row[-1] for row in fold]
        print(actual)
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
        print('- Training[%d] performed' % len(scores))
        print('----------------------------------------')
    return scores
```

Implementacija 4: `evaluate_algorithm`

Daje se implementacija 5. Vrši se podela skupa podataka po k folds-a.[10] Generišu se nasumično novi elementi izostavljenog dela, posledicom nasumičnih odabira očuvanja instanci, skupa podataka za svaki fold.

```
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for i in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split
```

Implementacija 5: `cross_validaion_split`

Za sada ću preskočiti `back_propagation(...)` i objasniti sledeću implementaciju 6.
`accuracy_metric(...)` - pozvanu od strane vršenja `evaluation_algorithm(...)`. I ovde se ističe broj poklapanja stvarnih vrednosti korišćenih kao skup podataka obuke i validacijom predviđenih odvojenih instanci posle crossvalidaion procesa.

```
def accuracy_metric(actual, predicted):
  correct = 0
  for i in range(len(actual)):
    if actual[i] == predicted[i]:
      correct += 1
  return correct / float(len(actual)) * 100.0
```

Implementacija 6: `accuracy_metric`

Daje se implementacija 7. koja je bila preskočena po redosledu, radi se backpropagation uz stohastički gradijentni spust (SGD[11]). Prima skup instanci fold-a namenjenog za teniranje i fold-a namenjenog za testiranje, sa svim parametrima, željenu funkciju aktivacije. Generiše se model neuronske mreže, ulaznim slojem po broju feature kolona, definisanim brojem neurona u skrivenom sloju, i izlaznim slojem po broju vrednosti klasnog atributa, a koja je ispunjena vrednostima po `initialize_network(...)`. Na standardni izlaz vrši se prikaz slojeva mreže. Potom, vrši obuka mreže uz `train_network(...)`. Potom se vrši predikcija uz `predict(...)`. Ispisuje na standarni izlaz za svaku instancu, redom, vrednost klase koja je predviđena.

```
def back_propagation(train, test, l_rate, n_epoch, n_hidden, transfer):
  n_inputs = len(train[0]) - 1
  n_outputs = len(set([row[-1] for row in train]))
  network = initialize_network(n_inputs, n_hidden, n_outputs)
  layerPrint=[]
  for i in range(len(network)):
    layerPrint.append(len(network[i]))
  print('network created: %d layer(s):' % len(network), layerPrint)
  train_network(network, train, test, l_rate, n_epoch, n_outputs, transfer)
  predictions = list()
  print ("perform predictions on %d set of inputs:" % len(test))
  for row in test:
    prediction = predict(network, row, transfer)
    predictions.append(prediction)
  print("pred =", predictions)
  return(predictions)
```

Implementacija 7: `back_propagation`

Daje se implementacija 8. generiše se svaki sloj ponaosob, za svaki neuron dodeljuju se nasumične vrednosti težina.

```
def initialize_network(n_inputs, n_hidden, n_outputs):
  network = list()
  hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(
    n_hidden)]
  network.append(hidden_layer)
  output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(
    n_outputs)]
  network.append(output_layer)
  return network
```

Implementacija 8: `initialize_network`

Implementacijom 9. se uspostavlja vršenje obuke date mreže naspram fold-a za obuku, fold-a za testiranje, kroz epohe, itd. Za svaki primerak ponaosob sledeće je ustanovljeno. Vrši se propagacija napred kroz mrežu `forward_propagate(...)` da bi se dobio rezultat izlaznog sloja, po pritom primenjenoj aktivacionoj funkiji. Hvataju se očekivane vrednosti po enkodiranju izlaznih predviđene vrednosti instance. Generise se zbir grešaka po

$$\sum^{|instance\ folda\ obuke|} \sum_{i} (expected_i - output_i)^2, \forall i = 1, \ldots, |expected|$$

Radi se provera grešaka unatrag zvanjem `backward_propagate_error(...)` (mreže suprotne propagacije). Potom se ažuriraju težine naspram mreže, primerka i stope brzine obuke uz zvanje `update_weights(...)`, potom se traže i tačnosti predviđanja uz `get_prediction_accuracy(...)` svakom epohom svakog primerka.

```
def train_network(network, train, test, l_rate, n_epoch, n_outputs, transfer):
  accuracy=[]
  for epoch in range(n_epoch):
    sum_error = 0
    for row in train:
      outputs = forward_propagate(network, row, transfer)
      expected = one_hot_encoding(n_outputs, row)
      sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])
      backward_propagate_error(network, expected, transfer)
      update_weights(network, row, l_rate)
      accuracy.append(get_prediction_accuracy(network, test, transfer))
    accuracies.append(accuracy)
```

Implementacija 9: `train_network`

Datom implementacijom 10. (pozivom `back_propagation`-a) uočava se da i ovde se radi `forward_propagate`. Potom se vraćaju pozicije liste izlaznih vrednosti, tj. vrednosti klasnog atributa.

```
def predict(network, row, transfer):
    outputs = forward_propagate(network, row, transfer)
    return outputs.index(max(outputs))
```

Implementacija 10: `predict`

Implementacijom 11. vrši se propagacija kroz mrežu i dobavljaju izlazne vrednosti, a svakim prolaskom kroz neuron, vrši se ustanovljavanje vrednosti aktivacije sa `activate` i `transfer(...)`.

```
def forward_propagate(network, row, transfer):
  inputs = row
  for layer in network:
    new_inputs = []
    for neuron in layer:
      activation = activate(neuron['weights'], inputs)
      neuron['output'] = transfer(activation, 0)
      new_inputs.append(neuron['output'])
    inputs = new_inputs
  return inputs
```

Implementacija 11: `forward_propagate`

Implementacijom 12. za neuron određuje se izlazna vrednost (klasa/labela) naspram ulaznih vrednosti (karakteristika/features-a) po određivanju nivoa aktivacije $n = \sum_i w_i x_i$. Sigmoidna (logistička) i tangentncijalno hiperbolička funkcija aktivacije naspram nivoa aktivacije i izvoda po ulaznoj vrednosti sagledanoj se uvažava.

```
def activate(weights, inputs):
  activation = weights[-1]
  for i in range(len(weights)-1):
    activation += weights[i] * inputs[i]
  return activation

def transfer_sigmoid(x, derivate):
    if derivate == 0:
        return 1.0 / (1.0 + exp(-x))
    else:
        return x * (1.0 - x)

def transfer_tanh(x, derivate):
    if derivate == 0:
        return tanh(x)
    else:
        return 1.0 - tanh(x)**2
```

Implementacija 12: `activate, transfer_sigmoid, transfer_tanh`

U implementaciji 13. za svaki sloj unatraške se sagleda svaki neuoron i njegov pomeraj delta i sagleda njegovo nepoklapanje sa očekivanim vrednostima neurona. Tako se prilagodi čitava mreža ovim proračunom.

```
def backward_propagate_error(network, expected, transfer):
    for idx_layer in reversed(range(len(network))):
        layer = network[idx_layer]
        errors = list()
        if idx_layer != len(network)-1:
            for idx_neuron_layer_N in range(len(layer)):
                error = 0.0
                for neuron_layer_M in network[idx_layer + 1]:
                    error += (neuron_layer_M['weights'][idx_neuron_layer_N] *
    neuron_layer_M['delta'])
                errors.append(error)
        else:
            for idx_neuron in range(len(layer)):
                neuron = layer[idx_neuron]
                errors.append(expected[idx_neuron] - neuron['output'])
        for idx_neuron in range(len(layer)):
            neuron = layer[idx_neuron]
            neuron['delta'] = errors[idx_neuron] * transfer(neuron['output'], 1)
```

Implementacija 13: `backward_propagate_error`

Implementacijom 14. vrši se obuka mreže po određenom stepenu obuke i primerku.

```
def update_weights(network, row, l_rate):
  for idx_layer in range(len(network)):
    inputs = row[:-1]
    if idx_layer != 0:
      inputs = [neuron['output'] for neuron in network[idx_layer - 1]]
    for neuron in network[idx_layer]:
      for idx_input in range(len(inputs)):
        neuron['weights'][idx_input] += l_rate * neuron['delta'] * inputs[idx_input]
      neuron['weights'][-1] += l_rate * neuron['delta'] * 1
```

Implementacija 14: `upadate_weights`

U implementaciji 15. dobaljaju se tačnosti po svakoj instanci skupa podataka, naspram ovde obavljenog predviđanja za svaku instancu.

```
def get_prediction_accuracy(network, train, transfer):
    predictions = list()
    for row in train:
        prediction = predict(network, row, transfer)
        predictions.append(prediction)
    expected_out = [row[-1] for row in train]
    accuracy = accuracy_metric(expected_out, predictions)
    return accuracy
```

Implementacija 15: `get_prediction_accuracy`

U implementaciji 16. po obuci mreže za izlazne vrednosti prolaskom feedforward-om vršilo se dodatno enkodiranje, da se ustanovi koja klasa za primerak je bila zastupljena.

```
def one_hot_encoding(n_outputs, row_in_dataset):
    expected = [0 for i in range(n_outputs)]
    expected[row_in_dataset[-1]] = 1
    return expected
```

Implementacija 16: `one_hot_encoding`

U implementaciji 17. vrši se prikupljanje iris skupa podataka kroz pandas.DataFrame, numpy array, sve do običnog python objekta liste, vrši se kastovanje elemenata klasnog atributa u float, vrši se enkodiranje vrednosti multiklasifikatornog atributa zarad korišćenja više neurona na izlaznom sloju.

```
def load_csv():
  dataset = load_iris(as_frame=True).frame.values.tolist()
  return dataset

def str_column_to_float(dataset, column):
  for row in dataset:
    row[column] = float(row[column])

def str_column_to_int(dataset, column):
  class_values = [row[column] for row in dataset]
  unique = set(class_values)
  lookup = dict()
  for i, value in enumerate(unique):
    lookup[value] = i
  for row in dataset:
    row[column] = lookup[row[column]]
  return lookup
```

Implementacija 17: `load_csv, str_column_to_float, str_column_to_int`

Ovo je izlaz nakon svih evaluacija, daje informaciju o napravljenom izlaznom od 5 i ulaznom sloju od 3 neurona. Data je vrednost kolone atributa pre i posle prolaska kroz mrežu, po predviđanju. Ovo je obavljeno za svaki fold ponaosob. Na kraju je data informacija o tačnosti obukom predviđanja za svaki sloj i prosečna vrednost tih vrednosti.

```
--------------------------------------
network created: 2 layer(s): [5, 3]
perform predictions on 50 set of inputs:
```

```
pred = [0, 2, 0, 1, 0, 2, 2, 2, 2, 1, 0, 2, 0, 2, 2, 0,
2, 1, 1, 0, 1, 0, 0, 0, 1, 2, 0, 2, 2, 1, 2, 0, 2, 2, 0,
 1, 0, 1, 1, 1, 1, 0, 1, 0, 2, 0, 2, 1, 1, 0]
[0, 2, 0, 1, 0, 2, 2, 2, 2, 1, 0, 2, 0, 2, 2, 0, 2, 1, 1, 0, 1, 0,
0, 0, 1, 1, 0, 2, 2, 1, 2, 0, 1, 2, 0,
1, 0, 2, 1, 1, 1, 0, 1, 0, 2, 0, 2, 1, 1, 0]
- Training[1] performed
--------------------------------------
network created: 2 layer(s): [5, 3]
perform predictions on 50 set of inputs:
pred = [1, 2, 2, 0, 0, 2, 2, 1, 0, 1, 2, 1, 2, 2, 0, 1, 1, 2, 2, 2,
 1, 2, 0, 2, 1, 1, 2, 0, 1, 2, 1, 0, 2,
 2, 0, 0, 2, 1, 1, 2, 0, 2, 0, 0, 2, 2, 1, 1, 1, 1]
[1, 2, 2, 0, 0, 2, 2, 1, 0, 1, 2, 1, 2, 2, 0, 1, 1, 2, 2, 2, 1, 2,
0, 2, 1, 1, 2, 0, 1, 2, 1, 0, 2, 2, 0, 0,
 2, 1, 1, 2, 0, 1, 0, 0, 2, 2, 1, 1, 1, 1]
- Training[2] performed
--------------------------------------
network created: 2 layer(s): [5, 3]
perform predictions on 50 set of inputs:
pred = [2, 0, 0, 1, 0, 0, 0, 2, 2, 0, 1, 2, 1, 2, 1, 2, 1, 0, 2, 1,
 2, 1, 1, 1, 2, 0, 1, 1, 0, 1, 0, 1, 1, 0, 2, 0,
  2, 0, 0, 0, 0, 2, 1, 0, 1, 0, 0, 1, 0, 1]
[2, 0, 0, 1, 0, 0, 0, 2, 2, 0, 1, 2, 1, 2, 1, 2, 1, 0, 2, 2, 2, 2,
 1, 1, 2, 0, 1, 1, 0, 1, 0, 1, 1, 0, 2, 0,
  2, 0, 0, 0, 0, 2, 1, 0, 1, 0, 0, 1, 0, 1]
- Training[3] performed
--------------------------------------
Scores (per fold): [94.0, 98.0, 96.0]
Mean Accuracy: 96.000%
```

Literatura

[1] Symbol-based AI, https://en.wikipedia.org/wiki/Symbolic_artificial_intelligence

[2] PAC learnibility, https://www.baeldung.com/cs/probably-aproximately-correct

[3] State space search, https://www.baeldung.com/cs/state-space-search

[4] Hill-Climbing, https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

[5] ID3, https://www.geeksforgeeks.org/iterative-dichotomiser-3-id3-algorithm-from-scratch/

[6] COBWEB, CLUSTER/2, https://cs.ccsu.edu/~markov/ccsu_courses/datamining-10.pdf

[7] Backpropagation neural network, https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/

[8] Neural network backpropagation from scratch in Python, https://github.com/HBevilacqua/neural_network_backprop_fromscratch/tree/master

[9] load_iris(), https://scikit-learn.org/stable/datasets/toy_dataset.html#iris-dataset

[10] Cross-validation: evaluating estimator performace, https://scikit-learn.org/stable/modules/cross_validation.html

[11] Deep Learning Optimizers: SGD with momentum, Adagrad, Adadelta, Adam optimizer, https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f