Univerzitet u Sarajevu
Elektrotehnički Fakultet

Diplomski rad

Agentske metode učenja u okruženju za razvoj inteligentnih agenata

Neuronske mreže u vizuelizaciji CT snimaka

Student: Dženan Zukić
Mentor: V. Prof Dr. Zikrija Avdagić, dipl. el. ing.
Sarajevo, Septembar 2007.

Sadržaj

Postavka diplomskog rada

Agentske metode učenja u okruženju za razvoj inteligentnih agenata

Neuronske mreže u vizuelizaciji CT snimaka

U okviru teoretskog istraživanja, istražiti šta su neuronske mreže, kako rade, za šta se koriste, koje vrste neuronskih mreža postoje. Fokusirati se na višeslojne perceptronske mreže, kao najjednostavniju i najrasprostranjeniju vrstu, te na najrasprostranjeniji algoritam za njihovo treniranje, algoritam prostiranja grešaka unazad (backpropagation). Dobro bi bilo napraviti i vlastitu programsku implementaciju.

U okviru praktične primjene neuronskih mreža, naći pogodno polje primjene i primjeniti neuronske mreže na rješavanje nekog specifičnog problema.

V. prof. Dr. Zikrija Avdagić, dipl. el. ing.

Sažetak/Abstract

Sažetak

Neuronske mreže su najrasprotranjeniji sistemi mašinskog učenja. Početci neuronskih mreža sežu do 1943., ali značajni napori na njihovom razvoju su uloženi 60-tih godina 20-tog vijeka. Od tada do danas je nastalo mnoštvo vrsta neuronskih mreža, a neke od popularnijih su: višeslojne perceptronske mreže, mreže sa funkcijom na osnovu udaljenosti i rekurzivne mreže.

Najčešći tip neuronske mreže je višeslojna perceptronska mreže, koja se trenira algoritmom prostiranja greške unazad. Ove mreže su univerzalni aproksimatori funkcija, jer svaku realnu kontinualnu funkciju mogo aproksimirati sa proizvoljnom preciznošću (ako imaju dovoljan broj skrivenih neurona).

Za medicinsku vizuelizaciju CT snimaka, najvažniji zadatak je prikazati interesantni dio snimka, npr. srčane krvne žile, ali tako da nisu zaklonjeni drugim organima na snimku (na primjer, plućima). Ovi snimci sadrže tkiva koja je teško izolirati i vizelizirati sa jednodimenzionalnim prenosnim funkcijama baziranim samo na koeficijentu prigušenja rendgenskih zraka.

Višedimenzionalne prenosne funckije omogućavaju daleko veću preciznost klasifikacije podataka, zbog čega je lakše razlikovati jedno tkivo od drugog. Nažalost, kreiranje 2D prenosnih funkcija može biti poprilično složen zadatak, koji se općenito rješava metodom pokušaja i pogeške.

Višeslojna perceptronska mreža je istrenirana za kreiranje početne postavke prenosne funkcije, za CT snimke srca. Ulazi ove mreže su pixeli smanjenog 2D histograma snimka, a izlazi su parametri filtera koji čine prenosnu funkciju.

Abstract

Neural networks are most widespread machine learning systems. Initial efforts of their development begun around middle of 20th century and still continue today. During that time, number of different types was tried out, with more or less success. Some of the most popular are: multi-layer perceptron, radial basis function networks and recursive networks.

Most widespread type is multi-layer perceptron (MLP), also called feed-forward backpropagation network, due to the fact it is trained with backpropagation algorithm. This type of network is universal function approximator, because it can arbitrarily precisely approximate any continuous real function, with enough neurons in hidden layer(s).

For medical volume visualization, one of the most important tasks is to reveal clinically relevant details from the CT study, i.e. the coronary arteries, without obscuring them with less significant parts. These volume datasets contain different materials which are difficult to extract and visualize with 1D transfer functions based solely on the attenuation coefficient.

Multi-dimensional transfer functions allow a much more precise classification of data which makes it easier to separate different surfaces from each other. Unfortunately, setting up multi-dimensional transfer functions can become a fairly complex task, generally accomplished by trial and error.

MLP neural network was trained to make initial setup of the transfer function for heart CT scan type. Inputs to the network are pixels of downscaled 2D histogram of the scan, and its outputs are filter parameters for the transfer function.

Predgovor

Ovaj diplomski rad je nastao kroz projekat saradnje Elektrotehničkog fakulteta Univerziteta Sarajevo i Univerziteta Paderborn, odsjek za Informatiku. Projekt je finansijski podržao DAAD (Njemački servis za akademsku razmjenu). U projektu su učestvovala 3 studenta sa odsjeka za Računarstvo i Informatiku ETF-a Sarajevo. U sklopu projekta smo bili u Njemačkoj 3 mjeseca.

U Paderbornu sam uradio većinu svog diplomskog rada (dobar dio teoretskog istraživanja, sav razvoj softvera osim Java implementacije kao i nešto teksta). Prvobitno je bilo zamišljeno da neuronske mreže primjenim na nekom od projekata katedre za razvoj softvera (Softwaretechnik – Software Engineering), koji je bio naš "domaćin" tokom boravka u Njemačkoj. Međutim, projekat koji je sa jednim specifičnim problemom odlično poslužio za primjenu neuronskih mreža je bio softver za vizuelizaciju na katedri za računarsku grafiku.

Softver za vizuelizaciju VolumeStudio, namjenjen vizuelizaciji CT snimaka (Computed Tomography – Kompjuterska tomografija) ljudskog tijela u medicinske svrhe, primarno dijagnozi oboljenja srca. Problem je bio kod postavljanja filtera za prikaz, vidjeti poglavlje Softver za vizuelizaciju.

Tokom boravka u Paderbornu, osmislio sam i razvio neuronsku mrežu koja pozicionira filtere na osnovu dvodimenzionalnog histograma snimka. Korištena je troslojna feed-forward mreža sa logističkom sigmoidnom (logsig) funkcijom aktivacije.

Mašinsko učenje

Učenje je dosta širok pojam pa ga je zbog toga teško precizno opisati. Definicije uglavnom koriste fraze kao "sticanje znanja, razumijevanja ili vještine proučavanjem, slijeđenjem uputstava ili kroz iskustvo" i "izmjena pretežitog ponašanja kroz iskustvo". Postoje mnoge paralele između mašinskog i životinjskog/ljudskog učenja. Razlog je jednostavan: mnoge tehnike mašinskog učenja su zasnovane na teorijama životinjskog učenja (i učenja kod ljudi) koje razvijaju psiholozi.

Mašine uče, uopšteno rečeno, kad god promjene svoju strukturu, program ili podatke tako da poboljšaju očekivane performanse u budućnosti. Neke od ovih promjena, kao na primjer zapisivanje sloga u bazi podataka, se obično i ne nazivaju učenje. Ali, ako se uspješnost softvera za prepoznavanje govora poboljša nakon nekoliko "istreniranih" rečenica, tada opravdano kažemo da je "mašina naučila". Mašinsko učenje se obično odnosi na promjene u sistemima u oblasti vještačke inteligencije.

Zašto mašinsko učenje

Postavlja se pitanje zašto bi se neko uopšte bavio mašinskim učenjem. Dostignuća u oblasti mašinskog učenja mogu pomoći u shvatanju kako životinje i ljudi uče, a pored toga postoji i nekoliko inžinjerskih razloga:

Neke zadatke ne možemo dobro opisati osim pomoću primjera, tj. možemo navesti primjere ulaza i odgovarajućeg izlaza, ali ne i tačne relacije između ulaza i izlaza. U tom slučaju želimo da mašina učenjem na primjerima izmjeni svoju unutrašnju strukturu/podatke/program tako da njena prenosna funkcija što bolje odgovara implicitnoj relaciji sadržanoj u trenažnom skupu primjera.

Moguće je da se u velikoj hrpi podataka nalaze važne relacije. Mašinsko učenje se često može koristiti za pronalaženje ovih relacija. To je poznato kao "data mining" - "rudarenje za podacima".

Dizajneri (ljudi) često naprave mašine koje ne rade tačno onako kako to neko želi. Određene karakteristike radnog okruženja možda nisu poznate u vrijeme dizajniranja. A i radno okruženje se stalno mijenja. Mašinsko učenje može pomoći da se mašine (bar djelimično) prilagođavaju radnom okruženju u cilju postizanja željenog ponašanja.

Količina dostupnih podataka o nekom problemu može biti prevelika da bi ga čovjek eksplicitno izmodelirao/isprogramirao. Mašine koje nauče ovo znanje mogu s vremenom da ga prikupe i više nego što bi ljudi bili voljni čak i da zapišu.

Ljudsko znanje se proširuje i produbljuje. Stalni redizajn sistema vještačke inteligencije je nepraktičan, ali bi se pomoću mašinskog učenja moglo da "stane u kraj" barem dijelu problema.

Izvori mašinskog učenja

Radovi u mašinskom učenju se međusobno približavaju (konvergiraju), a potječu iz nekoliko glavnih izvora:

Statistika: interpolacija i ekstrapolacija vrijednosti funkcija na osnovu nekoliko poznatih tačaka i slični problemi. Može se smatrati mašinskim učenjem jer se procjene baziraju na uzorcima podataka iz problemskog okruženja.

Modeli mozga: nelinearni elementi sa težiranim ulazima su predloženi kao jednostavni model biološkog neurona, i osnova su za **neuronske mreže**. Psihologe zanima koliko dobro ovi modeli aproksimiraju živi mozak (sa aspekta učenja). Ova oblast se ponekad zove i *konekcionizam*,

računanje u stilu mozga ili *pod-simbolsko procesiranje*.

Adaptivna teorija upravljanja: upravljanje procesom koji ima nepoznate parametre koji se u toku rada moraju procjenjivati. Ovi parametri se često i mijenjaju u toku rada, i upravljanje procesom mora voditi brigu i o tome. Primjer su neki aspekti upravljanja robotom na bazi senzorskih ulaza.

Psihološki modeli: između ostalog drvo odlučivanja i semantičke mreže.

Vještačka inteligencija: od početka su se istraživanja u ovoj oblasti bavila mašinskim učenjem.

Evolucijski modeli: modeliranje programa u analogiji sa teorijom evolucije. Najpoznatiji su *genetski algoritmi* i *genetsko programiranje*.

Šta se uči?

Pored ostalih kriterija, postoji i pitanje "šta se treba (na)učiti". Najčešće su to **funkcije**, i njima se i bavim u ovom radu. Takođe je moguće učiti i druge računske strukture:

1. Logički programi i skupovi pravila
2. Konačni automati
3. Gramatike
4. Sistemi za rješavanje problema

Učenje funkcija

Postoji takozvano supervizijsko i samostalno učenje. Samostalno učenje je ograničeno na klasifikaciju uzoraka u podskupove, na neki prikladan način. Supervizijsko učenje je mnogo korisnije, mogu se učiti nepoznate funkcije ili aproksimirati poznate (kompleksne) funkcije.

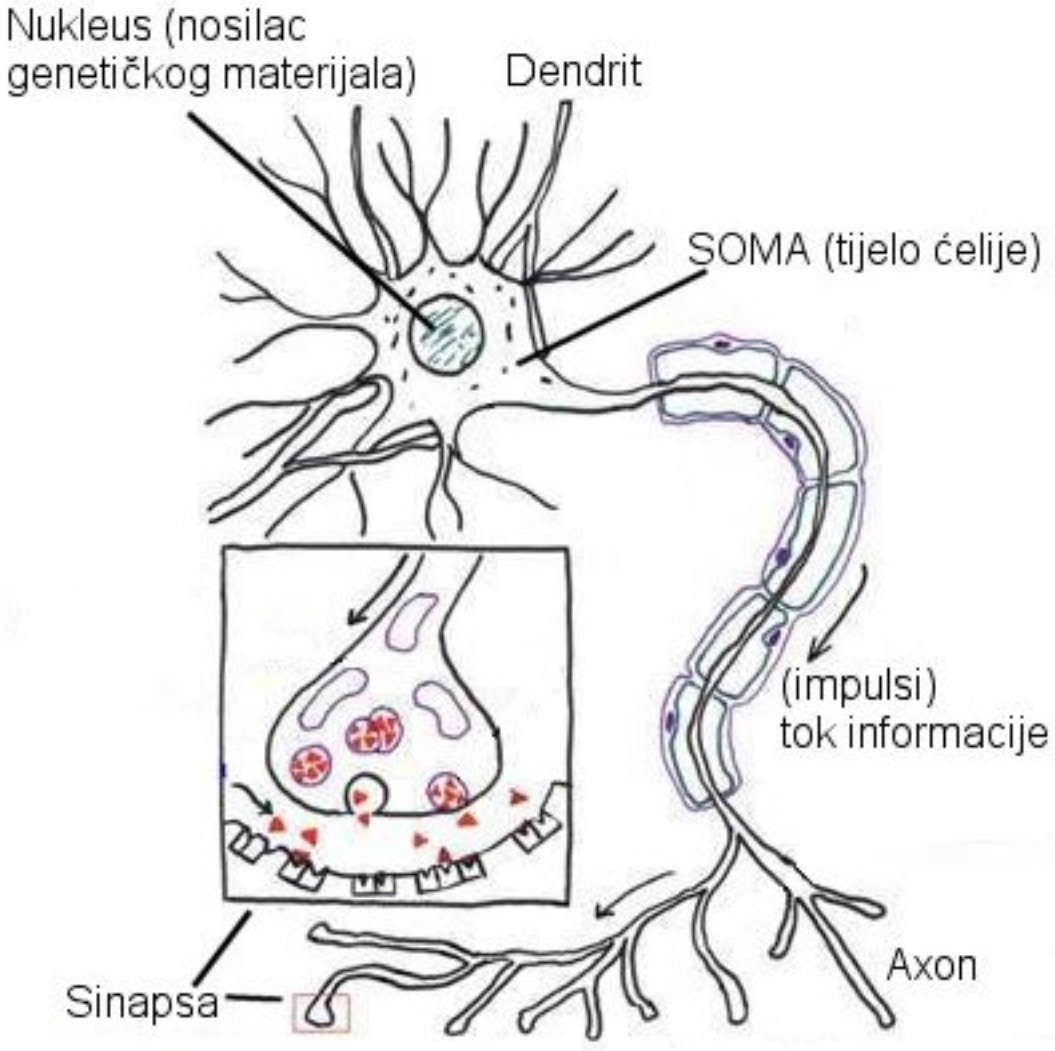
Kod nepoznatih funkcija imamo nekoliko poznatih uzoraka (tačaka, vektora), na osnovu kojih pokušavamo konstruisati funkciju koja će se što bolje slagati sa nepoznatom u poznatim tačkama (i nadamo se da će se dobro slagati i u ostalim tačkama nepoznate funkcije).

Kod funkcija koje treba aproksimirati, mi biramo uzorke kako nam najviše odgovara: minimalan broj koji rezultuje odgovarajućom aproksimacijom, takav raspored da imamo što uniformniju preciznost na domenu, itd…

Neuronske mreže

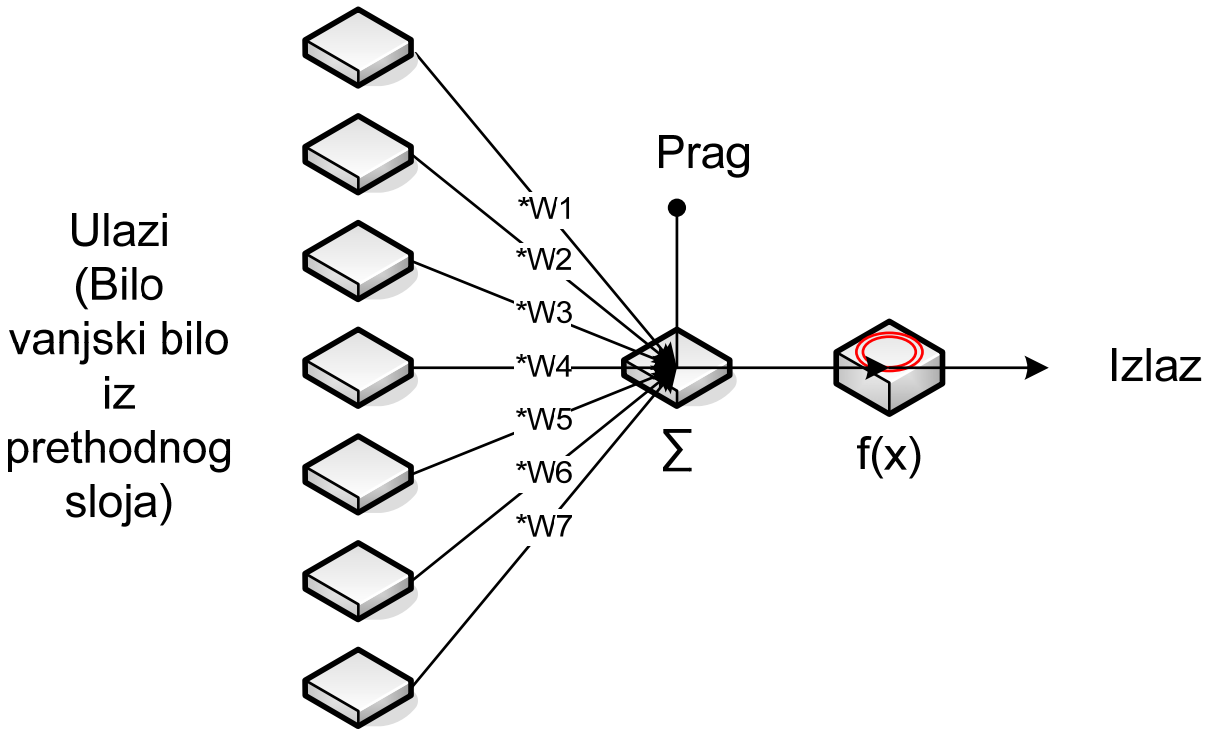
U ovom radu se pod neuronska mreža podrazumijeva vještačka neuronska mreža. Vještačke neuronske mreže su inspirisane prirodnim neuronskim mrežama, tj. nervnim sistemima živih bića. Na Slika 1 je dat izgled jednog prirodnog neurona.

Prirodni neuron ulazne signale dobija preko dendrita, obično kraćih nervnih završetaka koji primljene signale dovode do tijela ćelije. Tijelo ćelije vrši neku funkciju nad njima i proizvodi izlazni signal, koji se putem dugog nervnog završetka axon-a dostavljaju do sinapsi. Sinapse su veze sa drugim neuronima. Ono što je interesantno za sinapse je da one imaju određeno pojačanje (odnosno slabljenje) signala.



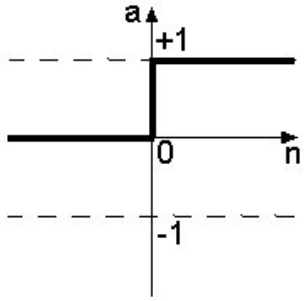
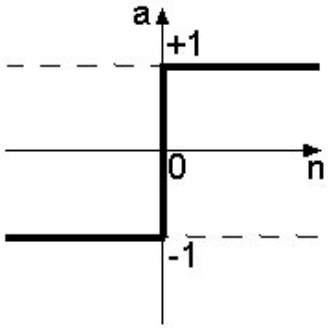
Slika 1 Prirodni neuron

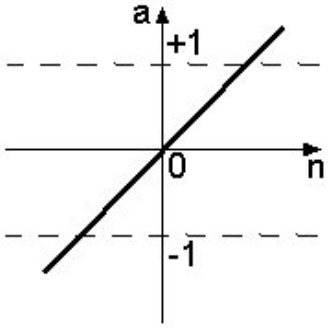
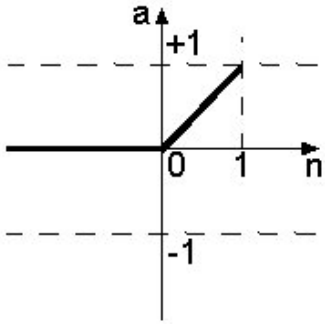
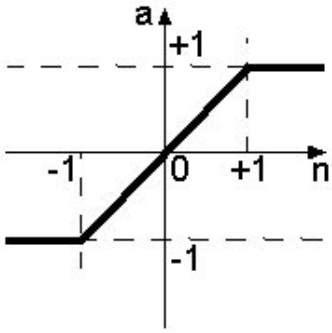
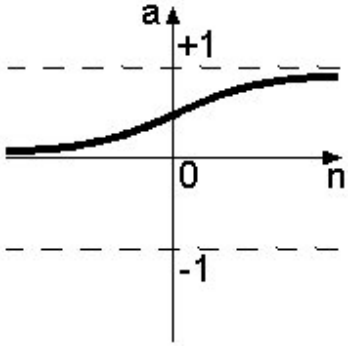
Neuronska mreža je skupina neurona povezana težiranim vezama. Ovo nije definicija, jer je pojam neuronske mreže takođe dosta opširan, pa je teško dati preciznu definiciju koja bi obuhvatila sve postojeće neuronske mreže.

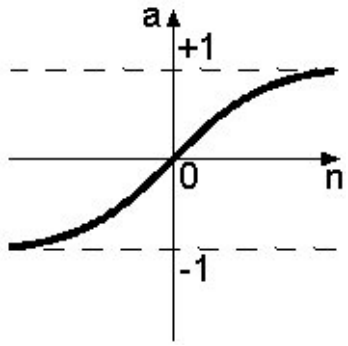
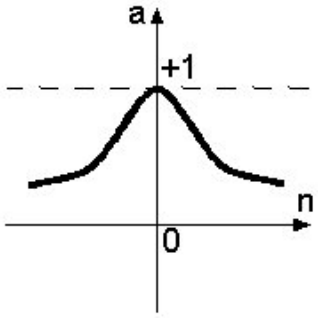


Slika 2 Model vještačkog neurona

Ono što je zajedničko za sve neuronske mreže je neuron. Tipična struktura neurona je data na Slika 2. Neuron ima nekoliko ulaza (općenito to može biti mješavina vanjskih ulaza i izlaza iz nekih drugih neurona). Svaki ulaz ima svoju specifičnu težinu. Ulazi, pomnoženi svojim težinama, se sabiraju (uključujući i prag ako postoji), a zatim se taj zbir propušta kroz funkciju aktivacije. Rezultat funkcije aktivacije je ujedno i izlaz iz neurona.

Češće korištene funkcije aktivacije		
Naziv i opis funkcije	Grafik funkcije	Formula funkcije
Funkcija koraka (engl. *Step function*) se rijetko koristi u današnjim neuronskim mrežama budući da nije u mogućnosti aproksimirati većinu kontinuiranih funkcija iz realnog svijeta.		izlaz = { 0 ako je *ulaz* ≤ T ; 1 ako je *ulaz* ≥ T T – prag funkcije, na slici je T=0
Signum funkcija je poseban oblik funkcije koraka. Upotrebljavala se u prvoj neuronskoj mreži, perceptonu. Rijetko se koristi u današnjim mrežama iz istog razloga kao i funkcija koraka.		izlaz = { 1 ako je *ulaz* > 0 ; 0 ako je *ulaz* = 0 ; -1 ako je *ulaz* < 0

Linearna funkcija		izlaz = g·ulaz
Linearna funkcija sa pragom je poseban oblik linearne funkcije. Prema ovoj funkciji, neuron ima vrijednost različitu od nule samo ako njegov ulaz dostigne vrijednost praga T.		izlaz = { 0 ako je *ulaz* ≤ T; ulaz-T ako je *ulaz* > T T – prag funkcije, na slici je T=0
Zasićena linearna funkcija		izlaz = { -1 ako je *ulaz* < -1; ulaz ako je -1<ulaz<1; 1 ako je *ulaz* > 1
Logistička (sigmoidna) funkcija se upotrebljava u *backpropagation* i *Hopfield*-ovoj neuronskoj mreži i zajedno s hiperboličko-tangentnom funkcijom je jedna od najčešće korištenih funkcija. Funkcija rezultira kontinuiranim vrijednostima u intervalu [0,1].		izlaz = $\frac{1}{1+e^{g\cdot ulaz}}$ g = 1/T – doprinos funkcije određuje zaobljenost funkcije oko nule; T – prag funkcije
Hiperboličko-tangentna funkcija je poseban oblik sigmoidne funkcije. Hiperboličko-tangentna i sigmoidna funkcija imaju slične grafove, s razlikom da je rezultat hiperboličko-tangentne funkcije u intervalu [-1,1]. Granične aktivacione funkcije, kakve su logistička i hiperboličko-		izlaz = $\frac{e^{u}-e^{-u}}{e^{u}+e^{-u}}$ u = g·ulaz

tangentna, su naročito korisne kada su ciljne vrijednosti ograničene		
Radijalna funkcija se koristi u radijalnim mrežama.		$izlaz_i = e^{-ulazi^2}$

U ovom poglavlju ću opisati i neke poznatije neuronske mreže.

Perceptron

Prva neuronska "mreža" je jedan sloj perceptrona. Perceptron su '40-tih godina prošlog vijeka opisali Warren McCulloch i Walter Pitts.

Perceptron je čvor, koji ima težirane ulaze, a funkcija aktivacije je obična step-funkcija, koja daje vrijednost 1 ako je zbir težiranih ulaza veći od praga, a -1 ako je manji od praga (prag je obično 0). Pojedinačni perceptron je mogao ispravno klasificirati samo linearno odvojive tačke prostora. Za treniranje perceptrona se koristi delta pravilo.

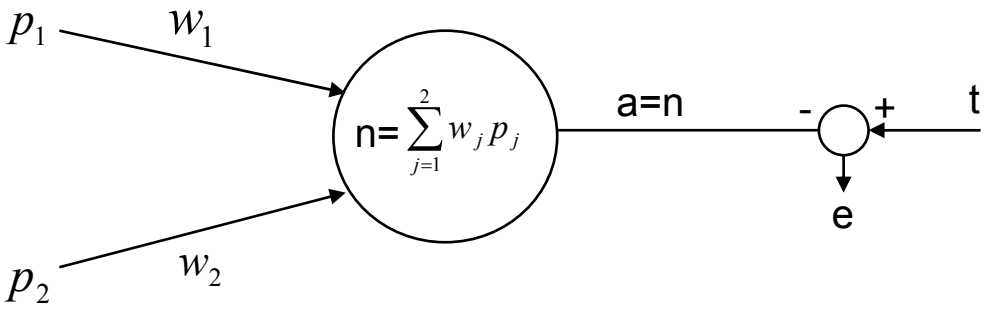
$w(j)' = w(j) + \alpha(\delta - y) * x(j)$, gdje je $w(j)$ j-ta težina prije učenja, $w(j)'$ je težina poslije učenja, α je brzina učenja ($0 < \alpha < 1$), δ je željeni izlaz, y je stvarni izlaz, $x(j)$ je j-ti ulaz. Pošto je funkcija aktivacije step-funkcija (funkcija koraka), izlaz je ili jednak željenom izlazu ili suprotan od željenog izlaza. Učenje se dešava samo u slučaju greške.

Umjesto step-funkcije, kao funkcija aktivacije se može koristiti i neka kontinualna funkcija, a čest izbor je logistička (logsig) funkcija. Perceptron sa logsig aktivacijskom funkcijom je opisan u poglavlju „Višeslojne perceptronske mreže", koje su ujedno i centralna tema ovog rada.

Adaline

Adaline je skraćenica od **Ada**ptive **Li**near **Ne**uron (prilagodljivi linearni neuron), razvili su ga Bernard Widrow i Ted Hoff na Univerzitetu Stanford 1960-te. Ovaj neuron je baziran na perceptronu, s tim da mu je funkcija aktivacije linearna (y=W*X+b). Ima praktičnu primjenu u teoriji upravljanja.

Interesantno je izdvojiti Widrow-Hoff pravilo za učenje linearnog neurona. Objasnit ćemo ga ne primjeru neurona sa 2 ulaza.



$$n = \sum_{j=1}^{2} w_j p_j$$

$$e = t - a$$

$$e^2 = (t-a)^2$$

$$e^2 = (t-n)^2 = (t - \sum_{j=1}^{2} w_j p_j)^2$$

$$e^2 = [t - w_1 p_1 - w_2 p_2]^2$$

$$e^2 = t^2 + w_1^1 p_1^1 + w_2^2 p_2^2 - 2tw_1 p_1 - 2tw_2 p_2 + 2w_1 p_1 w_2 p_2$$

Grupišemo li uz w_1 imat ćemo:

$$e^2 = w_1^2 \left[p_1^2\right] + w_1 \left[-2p_1 (t - w_2 p_2)\right] + (t - w_2 p_2)^2$$

Grupišemo li uz w_2 imat ćemo:

$$e^2 = w_2^2 \left[p_2^2\right] + w_2 \left[-2p_2 (t - w_1 p_1)\right] + (t - w_1 p_1)^2$$

Potrebno je pronaći minimalnu kvadratnu grešku koja se dobije kada su parcijalni izvodi kvadratne greške u odnosu na težine w_1 i w_2 jednaki nulama.

$$\frac{\partial e^2}{\partial w_j} = -2(t-a)p_j$$

$$\frac{\partial e^2}{\partial w_1} = -2[t - w_1 p_1 - w_2 p_2] * p_1 = 0$$

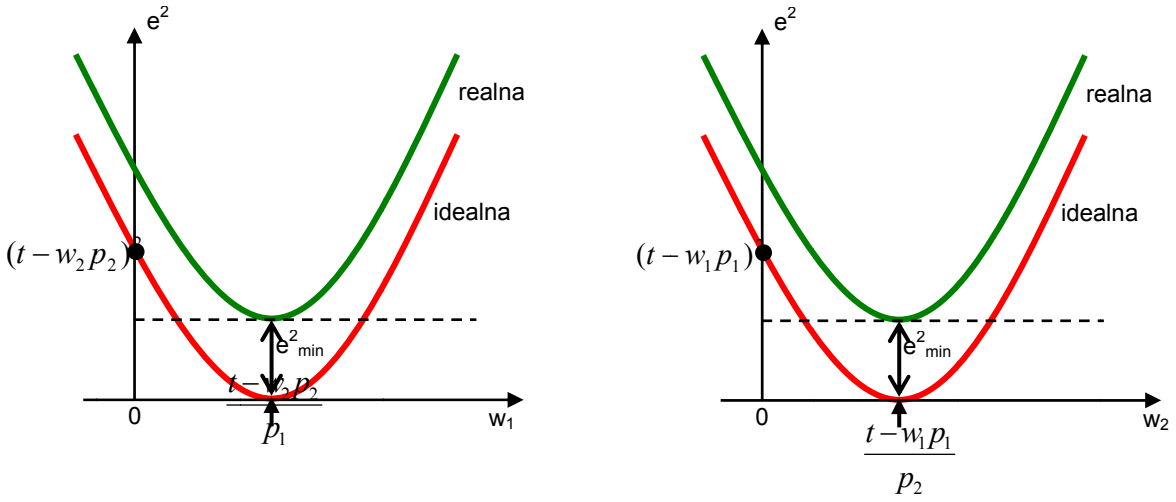
$$\frac{\partial e^2}{\partial w_2} = -2[t - w_1 p_1 - w_2 p_2] * p_2 = 0$$

Kako p_1 i p_2 ne mogu biti jednaki nulama (za dokazivanje ne bi imalo smisla), a p_1 i p_2 su ujedno i jedina razlika tada veličine u zagradama moraju biti jednake nulama:

$$t - w_1 p_1 - w_2 p_2 = 0$$

$$w_1 = \frac{t - w_2 p_2}{p_1} \qquad w_2 = \frac{t - w_1 p_1}{p_2}$$

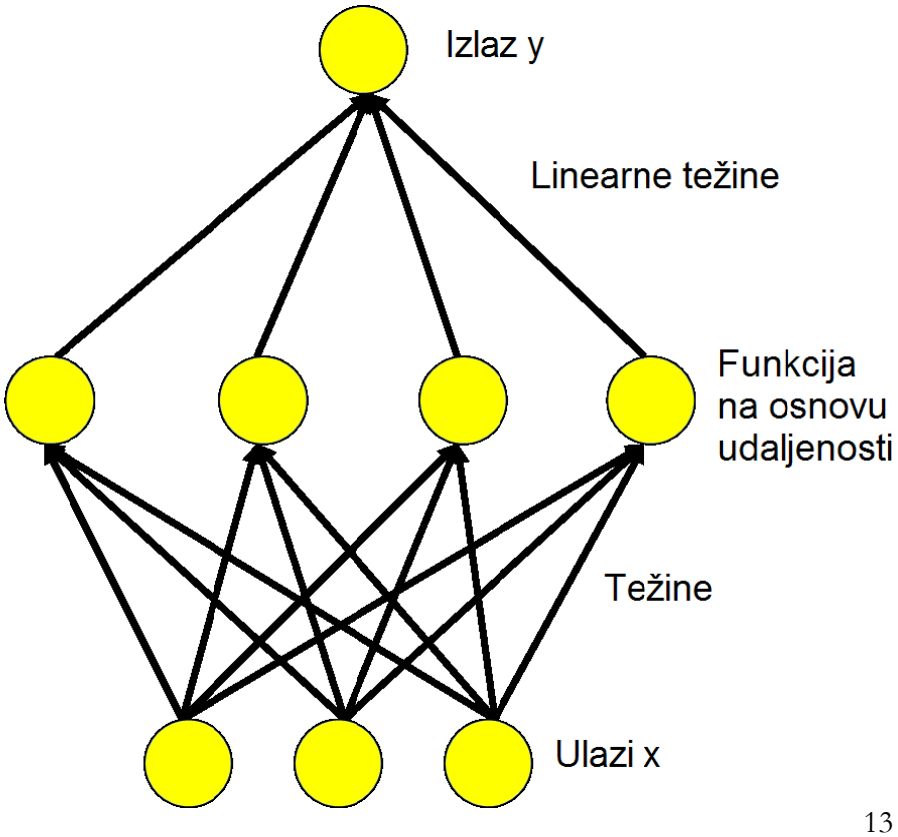
Zadnje dvije jednakosti sada rezultiraju da minimalna kvadratna greška bude jednaka 0:



RBF mreže

Mreže sa funkcijom na osnovu udaljenosti (Radial Basis Function – RBF) najčešće imaju 3 sloja: ulazni sloj, skriveni sloj sa nelinearnom funkcijom aktivacije koja izlaz daje na osnovu udaljenosti ulaza u odnosu na neku tačku ulaznog prostora. Izlaz ove mreže je prema tome:

$y(\boldsymbol{x}) = \sum_{i=1}^{N} a_i \rho(||\boldsymbol{x} - \boldsymbol{c}_i||)$, N-broj neurona u skrivenom sloju, $\boldsymbol{c}_i$-vektor centra za i-ti neuron, a_i-težine linearnog izlaznog neurona. Norma udaljenosti je obično Euklidska udaljenost, a bazna funkcija ρ je obično Gauss-ova.



RBF mreže su univerzalni aproksimatori na kompaktnom podskupu R^n. To jest, RBF mreža sa dovoljno skrivenih neurona može sa proizvoljnom preciznošću aproksimirati proizvoljnu neprekidnu funkciju.

Kod RBF mreža, 3

vrste parametara se moraju prilagodit okruženju primjene: vektori centra $\boldsymbol{c}_i$, težine linearnog izlaznog neurona a_i i parametri širine funkcija sa radijalnom osnovom β_i (Gauss-ova funkcija sa centrom u 0 se može pisati: $g(x) = e^{-\beta x^2}$).

Normalizacija

RBF mreže mogu biti normalizovane, što je korisno ako se radi sa stohastičkim protokom podataka. Izlaz mreže se u tom slučaju opisuje formulom:

$$y(x) = \frac{\sum_{i=1}^{N} a_i \rho(||\boldsymbol{x} - \boldsymbol{c}_i||)}{\sum_{i=1}^{N} \rho(||\boldsymbol{x} - \boldsymbol{c}_i||)}$$

Interpolacija

RBF mreže se mogu koristiti za interpolaciju funkcija $f: R^n \to R$, ako su poznate vrijednosti u konačnom broju tačaka $x_i,\ i = 1..$N. RBF mreža se konstruiše tako da je broj neurona RBF funkcijom jedan broju poznatih tačaka, te da su centri radijalnih funkcija upravo te tačke u kojima je poznata vrijednost funkcije f. Težine se mogu dobiti iz matrične jednačine:

$$\begin{bmatrix} g_{11} & \cdots & g_{1N} \\ \vdots & \ddots & \vdots \\ g_{N1} & \cdots & g_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix},$$ gdje su $b_i = f(x_i),\ g_{ij} = \rho(||x_i - x_j||)$. Ako su x_i međusobno različiti, tada je matrica G ne-singularna i težine se dobijaju kao: $w = G^{-1}b$.

Aproksimacija funkcija

Aproksimacija je komplikovanija od interpolacije, i obično se obavlja treniranjem u dvije faze. Prva faza se fokusira na centre i širine radijalnih funkcija, a druga na težine linearnog izlaznog neurona.

Rekurzivne mreže

Za razliku od većine tipova mreža, rekurzivne mreže podatke ne prosljeđuju samo od ulaza ka izlazu, već mogu proslijediti podatke i ka nekom od ranijih koraka u procesiranju.

Jednostavna rekurzivna mreža

Jednostavna rekurzivna mreža, takođe zvana Elmannova mreža po svom izumitelju Jeffu Elmannu je troslojna mreža. Pored standardne strukture ulaznog, skrivenog i izlaznog sloja ona ima i povratne sprege iz skrivenog sloja u ulazni sloj (tako da u ulaznom sloju postoje čvorovi čija je vrijednost kopija vrijednosti nekog od skrivenih neurona iz prethodnog koraka). Na ovaj način, mreža ima svoje interno stanje što joj omogućuje predviđanje sekvenci, što je van mogućnosti običnih neuronskih mreža.

U potpuno povezanoj rekurzivnoj neuronskoj mreži, svaki neuron kao ulaze ima izlaze svih drugih neurona u mreži. To znači da neuroni nisu organizovani u slojeve. Obično samo podskup neurona dobija i vanjske ulaze, a drugi podskup svoje izlaze šalje i van mreže (koji time postaju izlazi mreže).

Hopfieldova mreža

Hopfieldova mreža je svoj naziv dobila po izumitelju Johnu Hopfieldu. Hopfieldove mreže služe kao memorije adresibilne po sadržaju (content addressable memory). Konvergencija ka lokalnom minimumu je garantovana, ali nije garantovana konvergencija ka jednom od pohranjenih

uzoraka.

Čvorovi („neuroni“) u hopfieldovoj mreži su binarni pragovi, tj. Svaki čvor može biti u jednom od dva stanja, zavisno da li ulaz premašuje prag. Stanja mogu biti 1 i 0 ili 1 i -1, pa pojedinačni čvor je u osnovi perceptron.

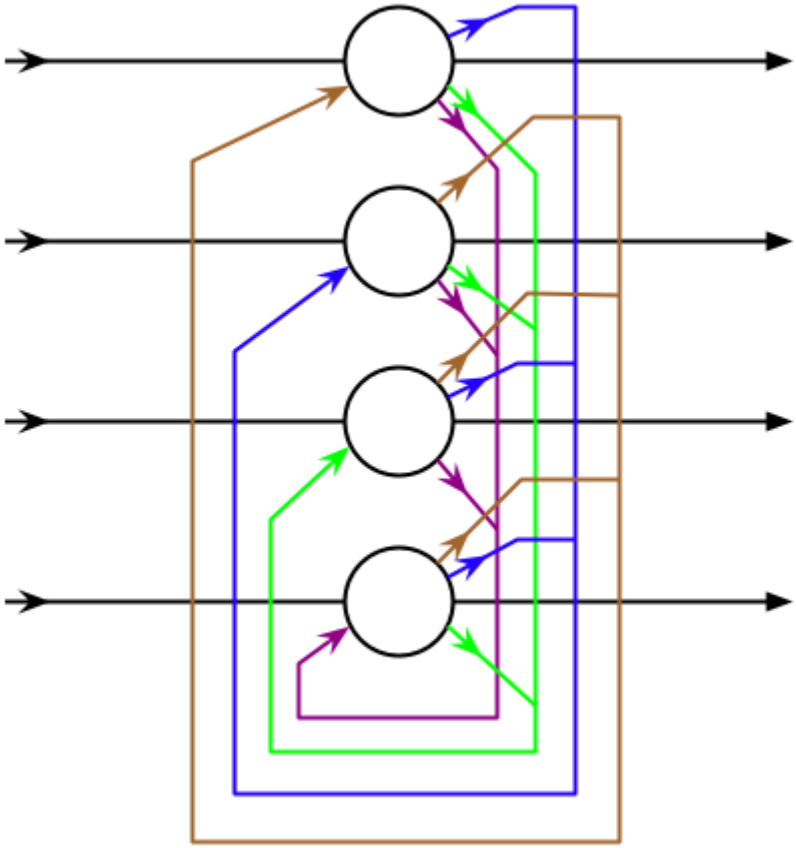
Veze u hopfieldovoj mreži tipično imaju slijedeća dva zahtjeva:

$w_{ii} = 0, \forall i$-čvorovi nisu povezani sami sa sobom i

$w_{ij} = w_{ji} \forall i, j$-sve veze su simetrične.

Hopfieldove mreže imaju skalarnu vrijednost povezanu sa svakim stanjem, koje se naziva „energija“, koja se dobija po formuli:

$$E = -\frac{1}{2}\sum_{i<j} w_{ij} y_i y_j + \sum_i \theta_i y_i$$



Slika 3 Hopfieldova mreža sa 4 čvora

Ova vrijednost se zove „energija“, jer će prilikom izračunavanja vrijednosti u mreži (update stanja)slučajnim redoslijedom mreže konvergirati ka stanju koje predstavlja lokalni minimum u ovoj funkciji energije. Kao posljedica, svako stanje koje predstavlja lokalni minimum u funkciji energije je ujedno i stabilno stanje u mreži.

Ova mreža se simulira („vrti“) tako što se u svakom koraku slučajno izabere jedan čvor mreže, a zatim se njegovo stanje (vrijednost na izlazu) izračuna u zavisnosti od toga da li je težirana suma ulaznih vrijednosti veća od praga u tom čvoru.

Treniranje hopfieldove mreže se sastoji u smanjivanju energije onih stanja koje mreža treba da zapamti. Na ovaj način mreža služi kao memorija adresibilna po sadržaju, tj. ako na ulaz dovedemo stanje čiji je jedan dio ujedno i dio stabilnog stanja, mreža će konvergirati ka tom stabilnom stanju. Na primjer, ako je mreže sa 5 čvorova istrenirana tako da je stanje (1,0,1,0,1) stanje sa energetskim minimumom, a na ulaz dovedemo stanje (1,0,0,0,1), nakon određenog broja koraka simulacije mreža će završiti (konvergirat će) u stabilnom stanju (1,0,1,0,1), koje zatim možemo očitati sa izlaza.

Dakle, hopfieldova mreža je ispravno istrenirana ako su stanja koja mreža treba zapamtiti ujedno i lokalni minimumi funkcije energije.

Mreža sa eho-stanjem

Mreža sa eho stanjem je rekurzivna neuronska mreža sa rijetko povezanim skrivenim slojem.

Tipično je prisutno oko 1% mogućih veza. Postojanje veza i njihove težine u skrivenom sloju mreže su slučajne i fiksne. Treniraju se samo težine izlaznog sloja. Dobra je za (re)produkciju vremenskih uzoraka.

Iako je ponašanje ove mreže nelinearno, jedini podesivi parametri su težine izlaznog sloja. Zbog toga je greška (u odnosu na vektor težina) kvadratna funkcija, i određivanje težina se može diferenciranjem može lagano svesti na rješavanje sistema linearnih jednačina.

Stohastičke neuronske mreže

Stohastičke neuronske mreže se razlikuju od „običnih“ po tome što unose slučajne varijacije u mrežu. Sa probablističkog stanovišta, te varijacije se mogu posmatrati kao jedan oblik statističkog uzorkovanja, što je slično Monte Karlo uzorkovanju.

Boltzmannova mašina

Boltzmannova (Boltzmann) mašina se može smatrati hopfieldovom mrežom u kojoj postoji šum. Izumili su je Geoff Hinton i Terry Sejnowski 1985. godine. Boltzmannova mašina je važna zato što je to jedna od prvih neuronskih mreža koje su pokazale moć učenja skrivenih (latentnih) varijabli.

Učenje boltzmannove mašine je ispočetka bilo sporo za simulaciju, ali algoritam kontrastne divergencije koji je izmislio Geoff Hinton oko 2000. godine omogućava modelima kao što je boltzmannova mašina da uče znatno brže.

Modularne neuronske mreže

Biološke studije su pokazale da ljudski mozak ne funkcioniše kao jedna ogromna neuronska mreža, nego kao više manjih mreža. Ova spoznaja je potakla razvoj koncepta modularnih neuronskih mreža, kod kojih više malih neuronskih mreža radi zajedno ili se takmiči kako bi se riješio neki problem.

Komitet mašina

Komitet mašina (Committee of Machines – CoM) je skup različitih neuronskih mreža koje zajedno „glasaju“ za svaki dati uzorak (većina pobjeđuje u slučaju klasifikacije, a prosjek se uzima u slučaju kontinualnog izlaza). U opštem slučaju, ovo daje bolje rezultate od drugih tipova neuronskih mreža.

U mnogim slučajevima, počevši od iste arhitekture, ali sa različitim slučajno izabranim početnim težinama, kao rezultat se dobijaju znatno različite mreže. Metod „komiteta mašina“ tu varijaciju u dobroj mjeri stabilizuje.

Komitet mašina je sličan algoritmu „bootstrap aggregation“, opštoj metodi učenje, s tim da komitet mašina ne bira uzorke za treniranje pojedinačnih neuronskih mreža kao podskup čitavog trening skupa, nego su varijacije između pojedinih mreža posljedica različitih početnih težina.

Asocijativne neuronske mreže

Asocijativne neuronske mreže (Associative Neural Network – ASNN) su nadgradnja komiteta mašina, jer ne koriste prosjek pojedinačnih mreža. ASNN koriste tehniku k-tog najbližeg susjeda. Koristi se veza između odgovora koje daju pojedinačne mreže kao mjera udaljenosti među analiziranim uzorcima za tehniku k-tog najbližeg susjeda. Ovo ispravlja „predrasude“ skupa neuronskih mreža.

ASNN ima memoriju koja se može podudariti sa trenažnim skupom. Ako novi podatci postanu

dostupni, ASNN trenutačno poboljša svoje sposobnosti predviđanja i pruža aproksimaciju podataka (samostalno učenje podataka) bez potrebe da se pojedinačne mreže iz skupa ponovo treniraju. Ova metoda je demonstrirana na http://www.vcclab.org/lab/asnn, gdje se može upotrebljavati ili skinuti.

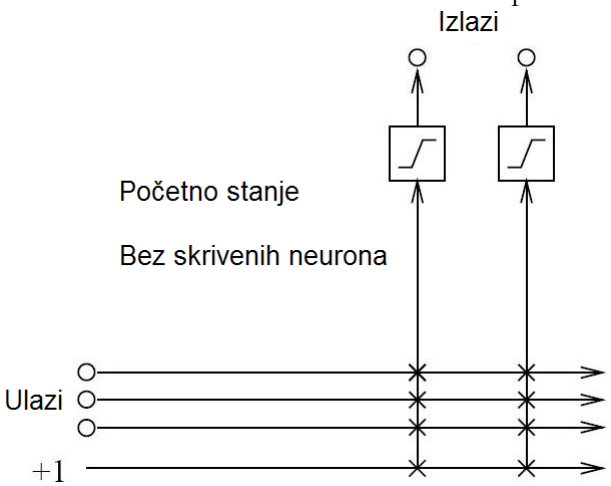
Ostali tipovi mreža

Instantno trenirane mreže

Težine skrivenog i izlaznog sloja ovih mreža se dobijaju direktnim mapiranjem iz vektora trenažnih podataka. Obično rade sa binarnim podacima, ali postoje i verzije za kontinualne podatke koje zahtijevaju nešto dodatnog procesiranja. Najpoznatija mreža ovog tipa je Willshaw-ova.

Kaskadne

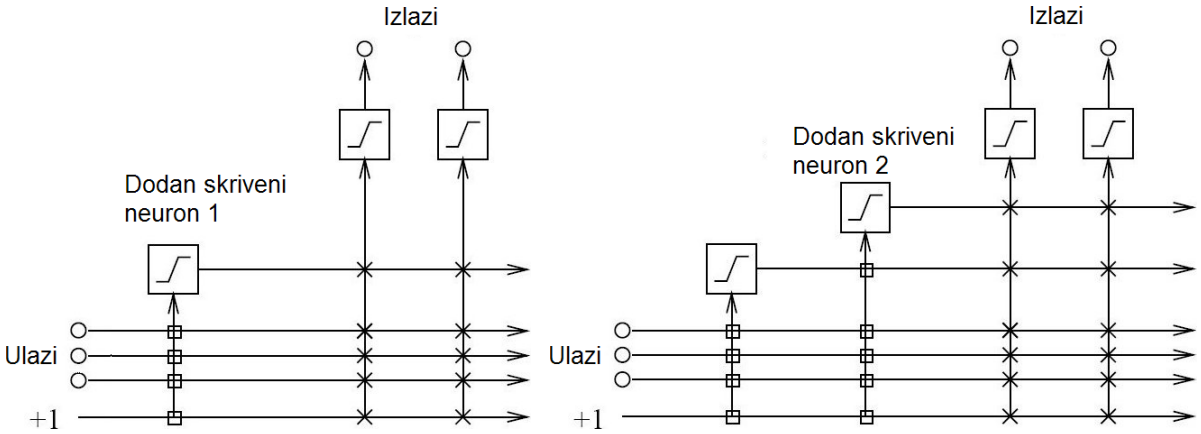
Neuronske mreže sa kaskadnom uzajamnom vezom (cascade correlation) su „samo-organizirajuće“ mreže. Mreža počinje samo od ulaznih i izlaznih neurona. Tokom procesa treniranja, iz skupa mogućih neurona, bira se jedan i dodaje u skriveni sloj.



Nakon što se doda neuron u skriveni sloj, njegovi težinski koeficijenti (sa ulazne strane) su fiksni, tj. ne mogu se mjenjati (na priloženim ilustracijama fiksirane težine su predstavljene kvadratićima, a promjenjive su predstavljene iksićima). Mreža se zove kaskadna zato što izlazi svih ranijih skrivenih neurona ulaze u najnoviji skriveni neuron.

Kako se novi neuroni dodaju u mrežu, algoritam učenja pokušava da maksimizira značaj koji novi neuron ima na grešku mreže koju pokušavamo smanjiti.

Ulazni „sloj“ samo distribuira ulaze skrivenim i izlaznim neuronima. Izlazni sloj je (preko težinskih koeficijenata) povezan sa svim neuronima izlaznog i skrivenog sloja. Funkcija aktivacije u izlaznom sloju je sigmoidna ako se radi klasifikacija, a linearna ako je problem računski.



Prednosti kaskadnih mreža su:

1. Pošto su samo-organizirajuće (povećavaju skriveni sloj po potrebi), ne moramo razmišljati o

parametrima veličine mreže.

2. Treniranje je veoma brzo – često i 100 puta brže od obične višeslojne perceptronske mreže. Zbog ovoga su kaskadne mreže pogodne za velike trenažne skupove.
3. Kaskadne mreže su tipično male, često sa samo desetak skrivenih neurona.
4. Način treniranja kaskadnih mreža je dosta robustan, i dobri rezultati se dobijaju čak i bez podešavanja parametara učenja.
5. Kaskadne mreže imaju manju vjerovatnoću zapadanja u lokalne minimume od višeslojne perceptronske mreže.

Kaskadne mreže nisu bez mana:

1. Kaskadne mreže veoma često su previše trenirane dostupnim podacima (overfitting), pa zbog toga imaju lošu preciznost na „neviđenim" podacima.
2. Nisu naročito precizne na problemima male i srednje veličine.

Neuro-fuzzy

Neuro-fuzzy (šala: neuro-čupave) mreže nastaju ugrađivanjem Fuzzy inferentnog sistema (FIS – fuzzy inference system) u tijelo neuronske mreže. Zavisno od tipa FIS-a, postoji više slojeva koji simuliraju procese fuzzy inferentne mašine, kao što su fuzzifikacija, inferencija, spajanje i defuzzifikacija. Ugrađivanje fuzzy inferentnog sistema u neuronsku mrežu ima prednost korištenja metoda za treniranje neuronskih mreža kako bi se pronašli parametri fuzzy sistema.

Teoretska svojstva neuronskih mreža

Kapacitet

Vještačke neuronske mreže imaju osobinu nazvanu kapacitet, koja grubo odgovara mogućnosti mreže da modelira bilo koju funkciju. Kapacitet je povezan sa količinom informacija koje se mogu pohraniti u mreži, a djelimično oslikava i kompleksnost mreže.

Konvergencija

Uopšteno se ne može ništa reći o konvergenciji treniranja neuronskih mreža, jer ona zavisi od mnoštva faktora. Prvo, možda postoji mnogo lokalnih minimuma, a to zavisi od funkcije cijene i modela. Drugo, za metodu optimizacije možde nije garantovana konvergencija ako je početno stanje daleko od lokalnog minimuma. I treće, za veliki broj parametara ili veliku količinu podataka, neke metode postaju nepraktične.

Generalno, prilikom praktične primjene nije preporučljivo previše se oslanjati na teoretske garancije o konvergenciji.

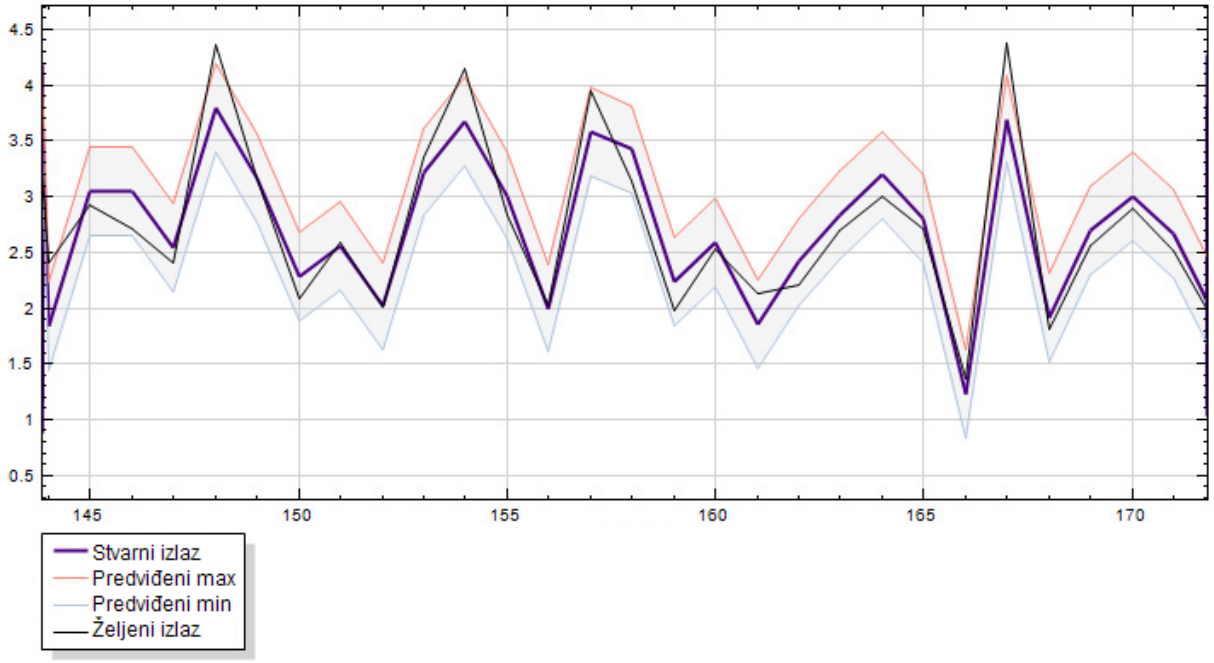
Generalizacija i statistika

U primjenama gdje je cilj napraviti sistem koji daje dobre rezultate i u slučajevima sa kojima se nije dotad susreo (koji dobro generalizira), pojavio se problem pretjeranog treniranja (overfitting). Ovo se dešava u prekompleksnim ili previše specificiranim sistemima, kada kapacitet mreže znatno nadilazi potrebe slobodnih parametara.

Postoje dva pravca razmišljanja za otklanjanje ovog problema. Prvi je upotreba validacije na neviđenim primjerima i slične tehnike, kako bi se optimalno zadali parametri mreže sa ciljem minimiziranja greške generalizacije. Drugi je korištenje nekog oblika regularizacije. Ovaj koncept je prirodan u probablističkom okruženju, gdje se regularizacija može postići davanjem veće „a

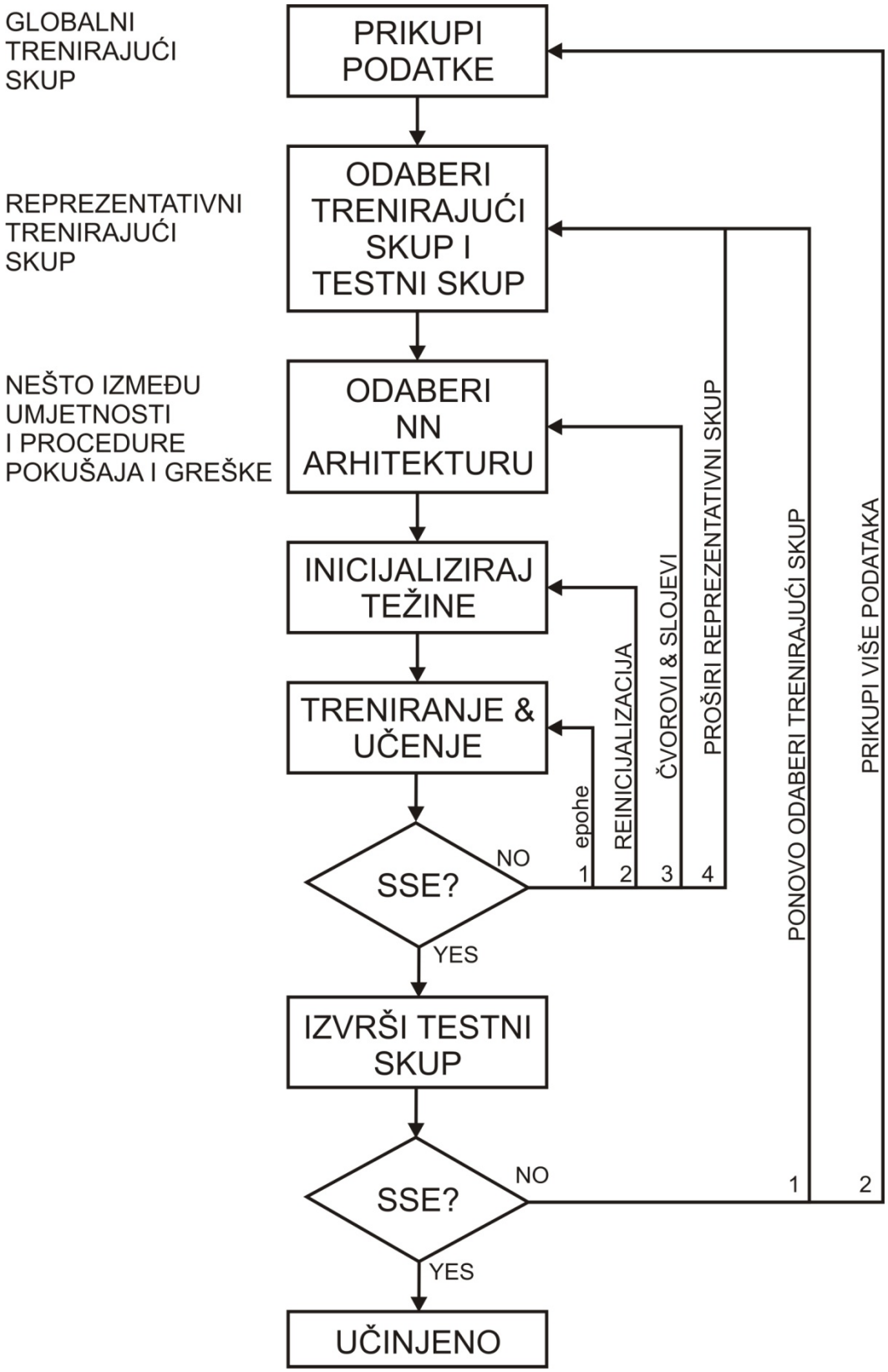
priori“ vjerovatnoće jednostavnijim modelima. Regularizacija se također javlja i u teoriji statističkog učenja, gdje je cilj minimizirati dvije veličine: „empirijski rizik“ i „strukturni rizik“. Ovi „rizici“ grubo odgovaraju grešci na trenažnom skupu i predviđenoj grešci na neviđenim podacima koja će nastati zbog pretjeranog treniranja.

Neuronske mreže trenirane u supervizijskom modu, koje koriste srednju kvadratnu grešku (MSE – mean square error) kao funkciju cijene, mogu koristiti formalne statističke metode radi određivanja pouzdanosti treniranog modela. MSE na skupu za validaciju se može koristiti kao procjena varijanse. To se dalje može koristiti da se odredi interval pouzdanosti izlaza iz mreže, pretpostavljajući normalnu raspodjelu. Procjena pouzdanosti napravljena na ovaj način je



statistički validna sve dok je raspodjela vjerovatnoća izlaza ista i dok se mreža ne modifikuje.

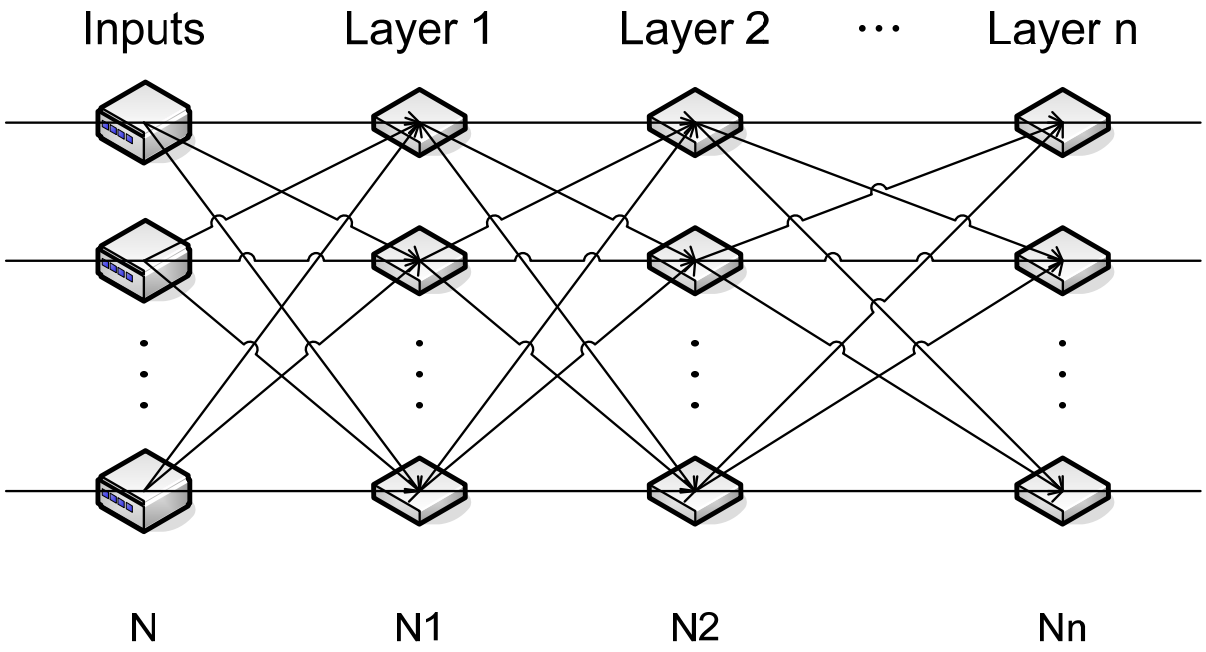
Šema primjene neuronskih mreža



Višeslojne perceptronske mreže

Višeslojne mreže sa prostiranjem signala unaprijed (feed-forward), još nazvane i višeslojne perceptronske mreže (multi-layer perceptron – MLP), su najpopularnija vrsta neuronskih mreža. Razloga za to je više:

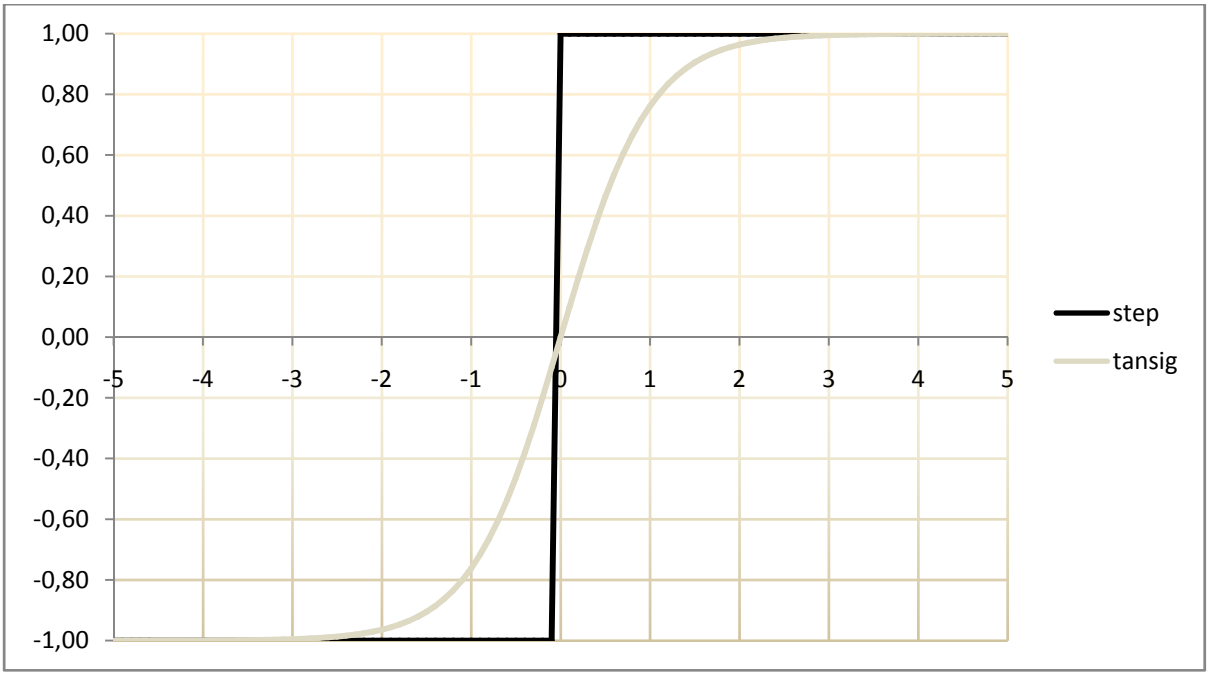
1. One su univerzalni aproksimatori (mreže od 3 i više slojeva), tj. svaku kontinualnu funkciju mogu aproksimirati sa proizvoljnom preciznošću (naravno, sa dovoljnim brojem neurona).
2. Dosta su jednostavne. Ove mreže mogu imati različite funkcije aktivacije u svakom sloju, ali se najčešće koristi samo jedna funkcija aktivacije u čitavoj mreži.
3. Imaju dosta standardizovan algoritam učenja, prostiranje greške unazad (backpropagation).
4. Iako učenje može biti sporo, normalni rad je veoma brz (brzina odziva je deterministička).



Slika 4 Opšta šema višeslojne feed-forward mreže Ulazni sloj distribuira ulaze (i možda ih direktno podvrgava funkciji aktivacije). Izlazi izlaznog sloja su izlazi iz mreže. Broj neurona u svakom sloju je nezavisan.

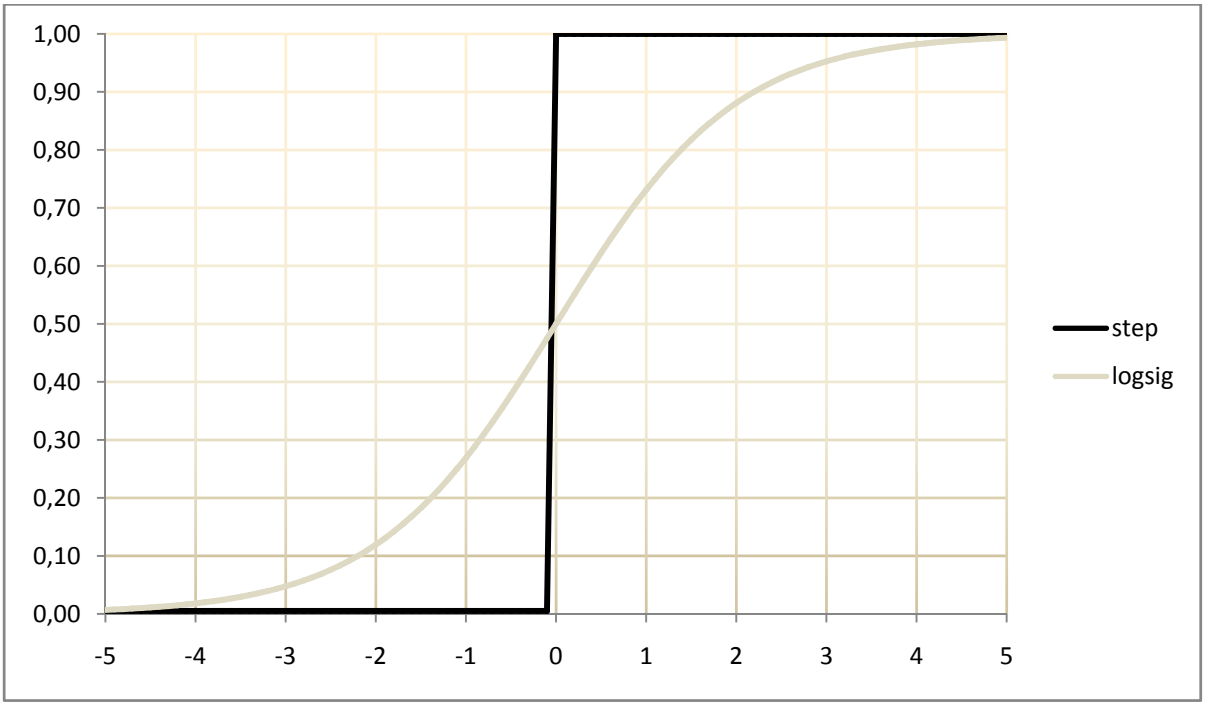
Jedan perceptron je u stanju ispravno klasificirati samo linearno odvojive probleme. Mreža sa više od 1 sloja perceptrona je u stanju ispravno klasificirati i linearno neodvojive probleme, ali to radi pomoću pravih linija ako je ulaz 2-dimenzionalan, pomoću ravni ako je ulaz 3-dimenzionalan, odnosno pomoću hiper-ravni ako ulaz ima više od 3 dimenzije. Uz to, treniranje takve mreže (višeslojne mreže perceptrona koji su imali step-funkciju kao funkciju aktivacije) je predstavljao problem.

Predstavljen je algoritam prostiranja greške unazad (nešto kasnije naveden u ovom radu), ali za njegovu upotrebu funkcija aktivacije mora biti diferencijabilna. Rješenje je da se step funkcija zamijeni nekom sličnom, ali diferencijabilnom funkcijom.



Slika 5 Poređenje step {-1,1} i tansig funkcije

Predložene funkcije su date na Slika 5 i Slika 6, zajedno sa poređenjem sa odgovarajućim step-funkcijama (funkcijama koraka). Funkcije su imenovane asocirajući ih sa tangensom odnosno logaritmom.



Slika 6 Poređenje step {0,1} i logsig funkcije

Ove funkcije, pored toga što su diferencijabilne i kontinualne, imaju i prednost što se izvod funkcije može izračunati pomoću vrijednosti funkcije:

1. $y' = 1 - y^2$, ako je u pitanju tansig funkcija ($y = \frac{e^{2x}-1}{e^{2x}+1}$)
2. $y' = y * (1 - y)$, ako je u pitanju logsig funkcija ($y = \frac{1}{1+e^{-x}}$)

Ova činjenica dodatno pojednostavljuje (a i ubrzava izvođenje) neuronske mreže sa sigmoidnim aktivacijskim funkcijama.

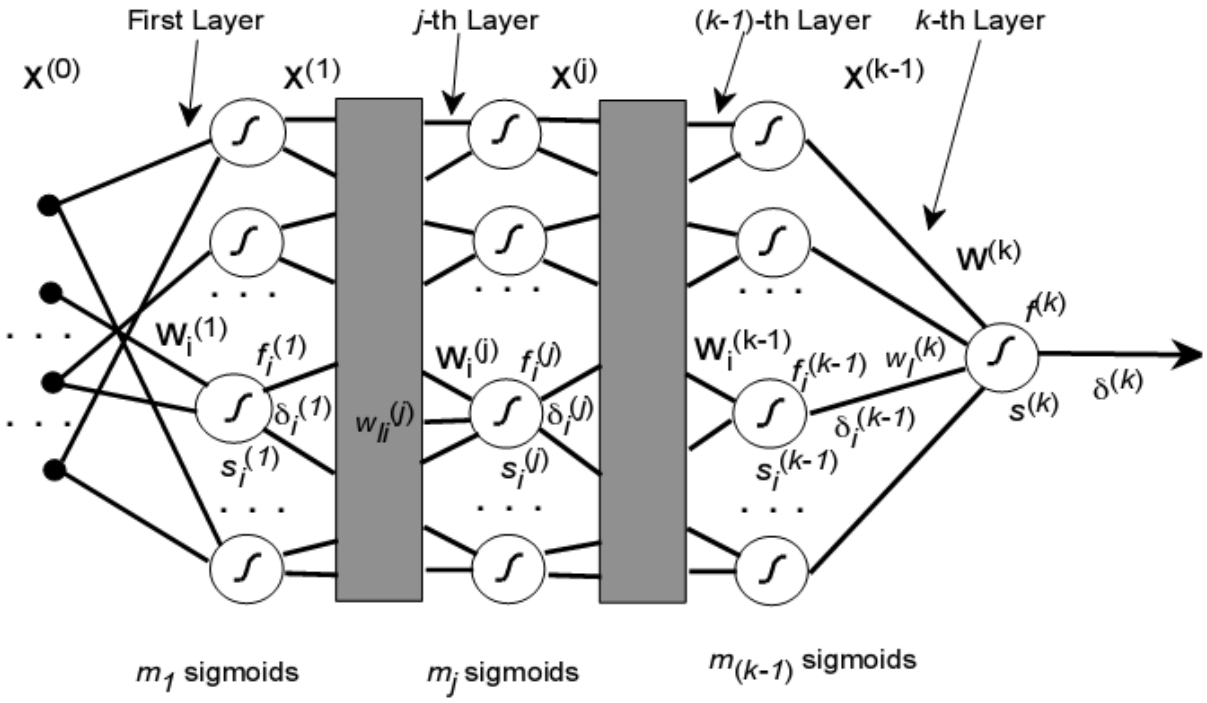
Backpropagation

Algoritam prostiranja greške unazad, poznatiji pod originalnim engleskim nazivom „backpropagation“, je daleko najzastupljeniji mehanizam treniranja višeslojnih perceptrona, a i drugih vrsta mreža koje ih na neki način uključuju, kao npr. jednostavna rekurzivna (Elmannova) mreža.

Uopšteno gledajući, backpropagation je metoda „spuštanja nizbrdo“, idući u pravcu koji najviše snižava grešku (u pravcu negativnog gradijenta). Takođe, backpropagation pokušava da eliminiše grešku sa minimalnom promjenom težinskih faktora (jer se taj isti rezultat može postići na više načina).

Notacija

Algoritam prostiranja grešaka unazad objasniću na sljedećoj mreži, sa iterativnim, a ne grupnim (batch) pristupom:



Ova mreža ima samo jedan izlaz, ali se algoritam lahko prilagođava za više izlaza.

Indeks sloja se vodi u gornjem desnom uglu, npr. $\boldsymbol{X}^j$ su izlazi iz j-tog sloja. Ulazi su označeni sa $\boldsymbol{X}^0$, a izlaz sa f, f_i^j – izlaz iz i-tog neurona j-tog sloja. Indeks neurona se vodi u donjem desnom uglu: δ_i^j – delta u j-tom sloju i-tom neuronu. $w_{1,i}^j$ – težinski faktor na ulazu i-tog neurona u j-tom sloju za 1 neuron u j-1-vom sloju. $\boldsymbol{W}_i^j$ – vektor težina ulaza i-tog neurona u j-tom sloju. U prvom sloju ima m_1 neurona, m_j – broj neurona u j-tom sloju. k – broj slojeva. Težirana suma ulaza u i-ti neuron j-tog sloja: $s_i^j = \boldsymbol{X}^{j-1} * \boldsymbol{W}_i^j$. Greška (koju pokušavamo minimizirati): $e = (d - f)$, d – željeni izlaz.

Matematski pristup

Za kretanje u pravcu negativnog gradijenta, trebaće nam vrijednost tog gradijenta:

$$\frac{\partial e}{\partial \boldsymbol{W}_i^j} = \left[\frac{\partial e}{\partial w_{1,i}^j}, \dots, \frac{\partial e}{\partial w_{l,i}^j}, \dots, \frac{\partial e}{\partial w_{m_{j-1},i}^j}\right]$$

Pošto je ovisnost e o $\boldsymbol{W}_i^j$ isključivo kroz s_i^j, možemo da koristimo pravilo o izvodu složene funkcije:

$$\frac{\partial e}{\partial \boldsymbol{W}_i^j} = \frac{\partial e}{\partial s_i^j}\frac{\partial s_i^j}{\partial \boldsymbol{W}_i^j}$$

Pošto je $s_i^j = \boldsymbol{X}^{j-1} * \boldsymbol{W}_i^j$, $\frac{\partial s_i^j}{\partial \boldsymbol{w}_i^j} = \boldsymbol{X}^{j-1}$, nakon zamjene imamo:

$$\frac{\partial e}{\partial \boldsymbol{W}_i^j} = \frac{\partial e}{\partial s_i^j}\boldsymbol{X}^{j-1}$$

Pošto je $\frac{\partial e}{\partial s_i^j} = -2(d-f)\frac{\partial f}{\partial s_i^j}$, imamo:

$$\frac{\partial e}{\partial s_i^j} = -2(d-f)\frac{\partial f}{\partial s_i^j}\boldsymbol{X}^{j-1}$$

Pošto je izraz $(d-f)\frac{\partial f}{\partial s_i^j}$ dosta važan u izvođenju, označićemo ga sa δ_i^j. Svaka delta nam govori koliko je kvadratna greška izlaza osjetljiva na promjene u ulazima svake funkcije aktivacije. Pošto ćemo mijenjati težinske vektore u pravcu negativnog gradijenta greške po njima, osnovno pravilo za izmjenu težina će biti:

$$\boldsymbol{W}_i^j \leftarrow \boldsymbol{W}_i^j + c_i^j \delta_i^j \boldsymbol{X}^{j-1}$$

c_i^j je konstanta učenja za ovaj težinski vektor. Obično su sve konstante učenja u pojedinim neuronima međusobno jednake, i zovu se konstanta učenja mreže. Tada imamo:

$$\boldsymbol{W}_i^j \leftarrow \boldsymbol{W}_i^j + c\delta_i^j \boldsymbol{X}^{j-1}$$

Izmjene težina u izlaznom sloju

Prvo moramo izračunati deltu u izlaznom sloju (sloju k):

$$\delta^k = (d - f^k)\frac{\partial f^k}{\partial s_i^k}$$

Koristeći se korisnom osobinom sigmoidne (logsig) funkcije aktivacije $\frac{\partial f}{\partial s} = f(1-f)$ imamo:

$$\delta^k = (d - f^k)\, f^k(1 - f^k)$$

Specijalizirajući uopšteno pravilo izmjene težina, za izlazni sloj imamo:

$\boldsymbol{W}^k \leftarrow \boldsymbol{W}^k + c\delta^k \boldsymbol{X}^{k-1}$, uvrštavajući izraz za deltu kod sigmoidne funkcije imamo:

$$\boldsymbol{W}^k \leftarrow \boldsymbol{W}^k + c(d - f^k)\, f^k(1 - f^k)\boldsymbol{X}^{k-1}$$

Jedina razlika između ove formule i delta pravila i Widrow-Hoff pravila je član $f^k(1-f^k)$, koji

predstavlja izvod.

Izmjene težina u skrivenim slojevima

Krećemo od izraza: $\delta_i^j = (d-f)\frac{\partial f}{\partial s_i^j}$. Koristeći pravilo o izvodu složene funkcije imamo:

$$\delta_i^j = (d-f)\left[\frac{\partial f}{\partial s_1^{j+1}}\frac{\partial s_1^{j+1}}{\partial s_i^j} + \cdots + \frac{\partial f}{\partial s_l^{j+1}}\frac{\partial s_l^{j+1}}{\partial s_i^j} + \cdots + \frac{\partial f}{\partial s_{m_{j+1}}^{j+1}}\frac{\partial s_{m_{j+1}}^{j+1}}{\partial s_i^j}\right]$$

$$\delta_i^j = (d-f)\sum_{l=1}^{m_{j+1}}\frac{\partial f}{\partial s_l^{j+1}}\frac{\partial s_l^{j+1}}{\partial s_i^j} = \sum_{l=1}^{m_{j+1}}\delta_l^{j+1}\frac{\partial s_l^{j+1}}{\partial s_i^j}$$

Jedino još preostaje da se izračuna $\frac{\partial s_l^{j+1}}{\partial s_i^j}$. Da bi to uradili, prvo pišemo:

$$s_l^{j+1} = \boldsymbol{X}^j * \boldsymbol{W}_l^{j+1} = \sum_{v=1}^{m_{j+1}} f_v^j w_{vl}^{j+1}$$

Dalje, pošto težine ne zavise od s-ova:

$$\frac{\partial s_l^{j+1}}{\partial s_i^j} = \frac{\partial \sum_{v=1}^{m_{j+1}} f_v^j w_{vl}^{j+1}}{\partial s_i^j} = \sum_{v=1}^{m_{j+1}} w_{vl}^{j+1}\frac{\partial f_v^j}{\partial s_i^j}$$

Pošto f_v^j zavisi od s_i^j samo ako je $v = i$, u ostalim slučajevima je $\frac{\partial f_v^j}{\partial s_i^j} = 0$. Za $v = i$, imamo:

$\frac{\partial f_v^j}{\partial s_i^j} = f_v^j(1 - f_v^j)$. Dalje imamo:

$$\delta_i^j = f_v^j(1-f_v^j)\sum_{l=1}^{m_{j+1}}\delta_l^{j+1}w_{il}^{j+1}$$

Interesantno je primijetiti da δ_i^j ne zavisi od funkcije greške direktno. Rekurzivna je po deltama, ali za izlazni sloj delta se direktno računa iz greške: $\delta^k = (d - f^k)\, f^k(1 - f^k)$. Interesantno je analizirati finalnu formulu za δ_i^j u skrivenim slojevima. Direktno zavisi od izvoda u datom neuronu, i težirane sume grešaka u svim neuronima u koje ide izlaz tog neurona.

Finalna formula za izmjenu (update) težina u skrivenim slojevima je ujedno i ona generička:

$$\boldsymbol{W}_i^j \leftarrow \boldsymbol{W}_i^j + c\delta_i^j \boldsymbol{X}^{j-1}$$

Ovako napisano, algoritam izgleda komplikovano. Međutim, dosta je logičan.

Prvo: faktor kojim skaliramo izmjenu težina je konstanta učenja c*vektor ulaza. Konstanta učenja (obično oko 0,1 do 0,3) osigurava stabilnost algoritma, tj. osigurava da algoritam ne reaguje na greške prenaglo. Što ulazi više odstupaju od neke srednje vrijednosti (0), to je taj trenažni uzorak značajni za algoritam.

Drugo: delte skrivenog sloja su rekurzivno ovisne od delti u narednom sloju, i na kraju sve ovise od delti izlaznog sloja. Delte izlaznog sloja direktno ovise o grešci $(d - f^k)$, koja daje znak deltama izlaznog sloja. Takođe, sve delte ovise i od starih težina, pa se niz veze sa niskim

težinskim koeficijentom prenosi vrlo malo greške (kao što se u normalnim operacijama prenosi vrlo malo ulaza). Takođe naopako, delta ovisi od svih delti neurona u koje šalje svoje izlaze, a ulazi ovise od svih izlaza koji ulaze u neuron! Sve ovo je razlog da se algoritam zove prostiranje grešaka unazad (naopako), tj. Backpropagation.

Implementacija u Javi

Kada je definisan projekat, prvobitno je bio ugrubo zamišljen kao vještačka inteligencija u javi, pa sam, pored ostalih stvari koje smo uradili dok smo bili ovdje, napravio svoju implementaciju neuronske mreže u programskom jeziku Java.

I pored dosta postojećih implementacija, za svoju sam se odlučio iz više razloga:

1. Ako neki algoritam zaista želiš da dobro naučiš, najbolje je da ga sam implementiraš.
2. Dugo vremena prije toga nisam programirao ništa komplikovano, pa sam računao da će mi dobro doći vježba.
3. Sa Javom se nisam prije toga susretao, tako da bi mi izvođenje tog osrednje teškog zadatka dobro došlo i kao upoznavanje sa jezikom Java (koja je po sintaksi i koncepciji dosta slična jeziku C++).

Pored implementacije same neuronske mreže (koja je u Javi implementirana kao klasa u zasebnoj datoteci „FFNN.java“), da bih tokom razvoja mogao testirati tu klasu napravio sam i jedan mali testni alat „tester.java“. Alat čita datoteku „TestVectors.txt“, učitavajući broj trenažnih i testnih vektora, te same vektore. Zatim inicijalizira i trenira mrežu trenažnim vektorima, a zatim je testira testnim vektorima i ispisuje rezultate.

FFNN.java

```
import java.util.*;
import java.lang.Math;
import java.io.*;

public class FFNN extends Object implements Serializable
{

private static final long serialVersionUID = 301019830215L;

// network architecture and general parameters
private int numInputs;
private int numLayers[];
private int numOutputs;// can be calculated as numLayers[numLayers.length-
1];

public double learnRate;

// network data
private double weights[][][];// connection weights
private double outs[][];// neurons' outputs
private double thresholds[][];
private double deltas[][];
private double wDeltas[][][];
private double tDeltas[][];

private double cSSE;// current error

private void init ()
// initializes neural network
```

```
{
  // TODO: dynamic learn rate
  learnRate = 0.2;

  // outs are computed outputs of neurons
  outs = new double[numLayers.length][];
  for (int i = 0; i<numLayers.length; i++)
    outs[i] = new double[numLayers[i]];

  // deltas are error deltas
  deltas = new double[numLayers.length][];
  for (int i = 0; i<numLayers.length; i++)
    deltas[i] = new double[numLayers[i]];

  // tDeltas are threshold deltas
  tDeltas = new double[numLayers.length][];
  for (int i = 0; i<numLayers.length; i++)
    tDeltas[i] = new double[numLayers[i]];

  // thresholds are sum offsets in neurons
  thresholds = new double[numLayers.length][];
  for (int i = 0; i<numLayers.length; i++)
  {
    thresholds[i] = new double[numLayers[i]];
    for (int k = 0; k<thresholds[i].length; k++)
      thresholds[i][k] = 0.5-(Math.random());
  }

  // wDeltas are weight deltas calculated in back propagation
  wDeltas = new double[numLayers.length][][];
  for (int i = 1; i<numLayers.length; i++)
  {
    wDeltas[i] = new double[numLayers[i]][];
    for (int k = 0; k<wDeltas[i].length; k++)
      wDeltas[i][k] = new double[numLayers[i-1]];
  }

  // connection weights multiply outs of previos layer before summation
  weights = new double[numLayers.length][][];
  for (int i = 1; i<numLayers.length; i++)
  {
    weights[i] = new double[numLayers[i]][];
    for (int k = 0; k<weights[i].length; k++)
    {
      weights[i][k] = new double[numLayers[i-1]];
      for (int p = 0; p<weights[i][k].length; p++)
        weights[i][k][p] = 0.5-(Math.random());
    }
  }

}// end of init

// constructors

public FFNN (int NumberOfInputs, int NumberOfOutputs, int
NumberOfCellsInHiddenLayer)
// network with one hidden layer
{
  numLayers = new int[3];
  numLayers[0] = NumberOfInputs;
  numLayers[1] = NumberOfCellsInHiddenLayer;
```

```
  numLayers[2] = NumberOfOutputs;
  numInputs = NumberOfInputs;
  numOutputs = NumberOfOutputs;
  init();
}

public FFNN (int[] NumbersOfCellsInLayers)
{
  numLayers = NumbersOfCellsInLayers;
  numInputs = numLayers[0];
  numOutputs = numLayers[numLayers.length-1];
  init();
}

public FFNN (Vector NumbersOfCellsInHiddenLayers)
{
  numLayers = new int[NumbersOfCellsInHiddenLayers.size()];
  for (int i = 0; i<NumbersOfCellsInHiddenLayers.size(); i++)
    numLayers[i] = ((Integer)
NumbersOfCellsInHiddenLayers.elementAt(i)).intValue();
  numInputs = numLayers[0];
  numOutputs = numLayers[numLayers.length-1];
  init();
}

// activation funcion
public static double logistic (double sum)
{
  return 1.0/(1+Math.exp(-1.0*sum));
}

private void ForwardPass (double data[])
// processes the vector
{
  double sum;
  for (int k = 0; k<numLayers[0]; k++)
    outs[0][k] = logistic(data[k]+thresholds[0][k]);

  for (int i = 1; i<numLayers.length; i++)
  {
    for (int k = 0; k<numLayers[i]; k++)
    {
      sum = 0;
      for (int p = 0; p<numLayers[i-1]; p++)
        sum += outs[i-1][p]*weights[i][k][p];
      outs[i][k] = logistic(sum);
    }
  }
}// end of ForwardPass

private void CalcDeltas (double trainVector[])
{
  double wdSum;
  int i = numLayers.length-1;

  // error deltas for output layer
  for (int k = 0; k<numLayers[i]; k++)
  {
    deltas[i][k] = outs[i][k]*(1-outs[i][k])*(trainVector[numInputs+k]-
outs[i][k]);
    tDeltas[i][k] += deltas[i][k]*learnRate;
```

```
    cSSE += (trainVector[numInputs+k]-
outs[i][k])*(trainVector[numInputs+k]-outs[i][k]);
  }
  // error deltas for all other layers
  for (i = numLayers.length-2; i>=0; i--)
  {
    for (int k = 0; k<numLayers[i]; k++)
    {
      wdSum = 0;
      for (int p = 0; p<numLayers[i+1]; p++)
        wdSum += deltas[i+1][p]*weights[i+1][p][k];

      deltas[i][k] = outs[i][k]*(1-outs[i][k])*(wdSum);
      tDeltas[i][k] += deltas[i][k]*learnRate;
    }
  }

  // calculate weight deltas
  for (i = 1; i<numLayers.length; i++)
    for (int k = 0; k<weights[i].length; k++)
    {
      for (int p = 0; p<weights[i][k].length; p++)
        wDeltas[i][k][p] += learnRate*deltas[i][k]*outs[i-1][p];
    }
}// end of CalcDeltas

public double[] Test1 (double TestInput[])
{
  double result[] = new double[numOutputs];
  ForwardPass(TestInput);
  for (int i = 0; i<numOutputs; i++)
    result[i] = outs[numLayers.length-1][i];
  return result;
}

public double TrainOnceBatch (Vector TrainInputs)
// returns SumSquaredError
{
  cSSE = 0;
  // reset deltas before batch
  for (int i = 0; i<numLayers.length; i++)
    for (int k = 0; k<tDeltas[i].length; k++)
      tDeltas[i][k] = 0;

  for (int i = 1; i<numLayers.length; i++)
    for (int k = 0; k<wDeltas[i].length; k++)
      for (int p = 0; p<wDeltas[i][k].length; p++)
        wDeltas[i][k][p] = 0;

  for (int i = 0; i<TrainInputs.size(); i++)
  {
    ForwardPass((double[]) TrainInputs.elementAt(i));
    CalcDeltas((double[]) TrainInputs.elementAt(i));
  }

  // now update weights and thresholds
  for (int i = 0; i<numLayers.length; i++)
    for (int k = 0; k<tDeltas[i].length; k++)
      thresholds[i][k] += tDeltas[i][k];

  for (int i = 1; i<numLayers.length; i++)
```

```
    for (int k = 0; k<weights[i].length; k++)
      for (int p = 0; p<weights[i][k].length; p++)
        weights[i][k][p] += wDeltas[i][k][p];

  return cSSE;
}// TrainOnceBatch

public double TrainBatch (Vector TrainInputs, int NumberOfRepeats)
// returns RMSerror
{
  double sseSum = 0;
  for (int i = 0; i<NumberOfRepeats; i++)
    sseSum += TrainOnceBatch(TrainInputs);
  return Math.sqrt(sseSum/(TrainInputs.size()*numOutputs));
}

public double TrainBatch (Vector TrainInputs, double GoalRMSError, int
MaxSteps)
// returns RMSerror
{
  int i = 0;
  double err;
  do
  {
    i++;
    err = TrainOnceBatch(TrainInputs);
  }
  while ((err>GoalRMSError)&&(i<MaxSteps));
  return err;
}

public void displayThresholds (PrintStream ps)
{
  ps.println("Thresholds:");
  for (int i = 0; i<numLayers.length; i++)
  {
    ps.print("Layer ");
    ps.print(i+1);
    ps.print(": ");
    for (int k = 0; k<thresholds[i].length; k++)
    {
      ps.print(thresholds[i][k]);
      ps.print(' ');
    }
    ps.println();
  }
}

public void UnTrain ()
{
  for (int i = 0; i<numLayers.length; i++)
    for (int k = 0; k<weights[i].length; k++)
    {
      thresholds[i][k] = 0.5-(Math.random());
      for (int p = 0; p<weights[i][k].length; p++)
        weights[i][k][p] = 0.5-(Math.random());
    }
}// UnTrain
}
```

tester.java

```
import java.util.*;
import java.io.*;
import java.text.DecimalFormat;
import java.text.NumberFormat;

public class tester
{
/**
 * @param args
 * @throws IOException
 */
public static void main (String[] args) throws IOException
{
  FFNN net = null;
  Vector trainVectors, testVectors;
  int inputs, outputs, trains, tests; // number of ...
  DataInputStream in = new DataInputStream(new
FileInputStream("TestVectors.txt"));
  StringTokenizer tok = null; // for separation of numbers on input line

  // now reading "globals"
  String l = in.readLine();
  tok = new StringTokenizer(l);
  inputs = Integer.parseInt(tok.nextToken());
  outputs = Integer.parseInt(tok.nextToken());
  trains = Integer.parseInt(tok.nextToken());
  tests = Integer.parseInt(tok.nextToken());
  trainVectors = new Vector();
  testVectors = new Vector();

  // reading train vectors first
  for (int i = 0; i<trains; i++)
  {
    l = in.readLine();
    tok = new StringTokenizer(l);
    double v[] = new double[inputs+outputs];
    for (int k = 0; k<inputs+outputs; k++)
      v[k] = Double.parseDouble(tok.nextToken());
    trainVectors.add(v);
  }

  // reading test vectors second
  for (int i = 0; i<tests; i++)
  {
    l = in.readLine();
    tok = new StringTokenizer(l);
    double v[] = new double[inputs+outputs];
    for (int k = 0; k<inputs+outputs; k++)
      v[k] = Double.parseDouble(tok.nextToken());
    testVectors.add(v);
  }

  int BrojCelija[] = {inputs, 3*inputs, 2*inputs, outputs};
  net = new FFNN(BrojCelija);

  System.out.println("Train vectors:");
  for (int i = 0; i<trains; i++)
  {
    for (int k = 0; k<inputs+outputs; k++)
```

```
    {
      System.out.print(((double[]) (trainVectors.elementAt(i)))[k]);
      System.out.print(' ');
    }
    System.out.println();
    // net.TrainOnce((double[]) trainVectors.elementAt(i));
  }

  double rms = net.TrainBatch(trainVectors, 0.001, 50000);

  System.out.println("\nTest3:");
  for (int i = 0; i<tests; i++)
  {
    double outs[];
    for (int k = 0; k<inputs+outputs; k++)
    {
      System.out.print(((double[]) (testVectors.elementAt(i)))[k]);
      System.out.print(' ');
    }
    System.out.println();

    outs = net.Test1((double[]) testVectors.elementAt(i));
    for (int k = 0; k<inputs; k++)
    {
      System.out.print(((double[]) (testVectors.elementAt(i)))[k]);
      System.out.print(' ');
    }
    for (int k = 0; k<outputs; k++)
    {
      System.out.print(outs[k]);
      System.out.print(' ');
    }
    System.out.println('\n');
  }

  NumberFormat formatter = new DecimalFormat("0.000000");
  String s = formatter.format(rms);

  System.out.println("RMSerror: "+s);

  // net.displayThresholds(System.out);
}
}
```

TestVectors.txt

```
2 1 10 12
0.10 0.10 0.20
0.00 0.00 0.00
0.20 0.20 0.40
0.50 0.50 1.00
0.10 0.50 0.60
0.00 0.20 0.20
0.20 0.00 0.20
0.15 0.50 0.65
0.10 0.20 0.30
0.20 0.10 0.30
0.10 0.10 0.20
0.00 0.00 0.00
0.20 0.20 0.40
0.50 0.50 1.00
```

```
0.10 0.50 0.60
0.00 0.20 0.20
0.20 0.00 0.20
0.15 0.50 0.65
0.10 0.20 0.30
0.20 0.10 0.30
0.15 0.15 0.30
0.50 0.00 0.50
```

Rezultati (izlaz iz testera)

```
Train vectors:
0.1 0.1 0.2
0.0 0.0 0.0
0.2 0.2 0.4
0.5 0.5 1.0
0.1 0.5 0.6
0.0 0.2 0.2
0.2 0.0 0.2
0.15 0.5 0.65
0.1 0.2 0.3
0.2 0.1 0.3

Test3:
0.1 0.1 0.2
0.1 0.1 0.14561504562210992

0.0 0.0 0.0
0.0 0.0 0.04809177073276895

0.2 0.2 0.4
0.2 0.2 0.28031828155767463

0.5 0.5 1.0
0.5 0.5 0.9687942009872802

0.1 0.5 0.6
0.1 0.5 0.5536247210305157

0.0 0.2 0.2
0.0 0.2 0.15305689120350932

0.2 0.0 0.2
0.2 0.0 0.15610419028575576

0.15 0.5 0.65
0.15 0.5 0.565481776449661

0.1 0.2 0.3
0.1 0.2 0.2182765980669442

0.2 0.1 0.3
0.2 0.1 0.21018859556875588

0.15 0.15 0.3
0.15 0.15 0.21058728787743505

0.5 0.0 0.5
0.5 0.0 0.7970743578770905

RMSerror: 0.042843
```

Softver za vizuelizaciju

Uvod

Za vizuelizaciju i analizu CT snimaka, glavna prednost direktnog renderiranja zapremine je potencijal 3D strukture nama interesantnog tkiva, umjesto samo malog dijela podataka koji su vidljivi iz 2D presjeka. To pomaže korisniku da odredi relativne prostorne pozicije sastavnih komponenti, i olakšava otkrivanje i razumijevanje složenih pojava, kao što je stenoza srca (stanjenje krvnih žila koje srce napajaju krvlju), a za potrebe dijagnoze i planiranja operacija.

Drugi pristup koji se često koristi za ovu namjenu je projekcija maksimalnog intenziteta (maximum intensity projection – MIP), koji vrši emitovanje zrake[1] i pronalazi maksimalnu vrijednost duž svake zrake, umjesto slaganja vrijednosti kao što to radi direktno renderiranje zapremine. Pošto ova metoda zanemaruje prostornu dubinu iz pravca kamere, tkiva sa većom gustinom[2] izgledaju kao da se nalaze ispred svih ostalih, bez obzira da li se zaista nalaze ispred ili iza tkiva sa manjom gustoćom.

Jedan od osnovnih zahtjeva dobrog sistema zapreminske vizuelizacije je sofisticiran način odvajanja podataka koji su od interesa, i istovremeno uklanjanje nebitnih dijelova. Dosta napretka je ostvareno u razvoju prenosnih funkcija kao klasifikatora osobina (vrste tkiva) za zapreminske podatke. Uloga prenosnih funkcija je da daju vizuelne atribute (boju i providnost) interesantnim tkivima.

Najjednostavniji pristup je da se različitim gustoćama daju različite boje. Ali ovaj pristup u većini slučajeva nije u stanju da ukloni nepotrebne dijelove zapremine, kao što su druge anatomske strukture slične (ili iste) gustoće, a čije se granice često preklapaju. Zbog toga ovaj pristup (sa 1D prenosnim funkcijama), sam za sebe, i nije naročito koristan.

Umjesto klasificiranja zapreminskih podataka samo po skalarnoj gustoći pojedinačnih voxela[3], višedimenzionalne prenosne funkcije rješavaju ovaj problem dodavanjem dodatnih dimenzija domeni[4] prenosne funkcije. Koristeći klasifikaciju zasnovanu na kombinaciji svojstava, različita tkiva se mogu prikazivati odvojeno jedno od drugog. VolumeStudio koristi 2D prenosne funkcije za klasifikaciju tkiva. Prva dimenzija je gustoća voxela, a druga dimenzija je izvod vrijednosti gustoće u tom voxelu.

Nažalost, kreiranje 2D prenosnih funkcija može biti poprilično komplikovano i dugotrajno. Prva stvar, korisnik mora da barata sa još jednom dimenzijom. Drugo, korisnik mora zadati prenosnu funkciju u poprilično apstraktnom domenu, a zatim pogledati rezultate na renderiranoj slici (što je prostorni domen). Situaciju pogoršava činjenica da male promjene ovih parametara često prave znatne i nepredvidive promjene u vizuelizaciji. Zbog svega toga, kako bi se izbjeglo kreiranje prenosnih funkcija isključivo metodom pokušaj-pogreška, neophodno je najprije identificirati tkiva u domenu prenosne funkcije, a što zahtjeva dobro poznavanje vrijednosti podataka i njihove veze sa određenim tkivom. Takođe, ručno dodjeljivanje optičkih svojstava je ne-standardizovano i znatno ovisi od ličnog ukusa korisnika. Ovo može izazvati pogrešno tumačenje ako vizuelizaciju kasnije gleda treća osoba. Ovo često dovodi do rezultata koji su teško ponovljivi i zbog toga nisu pogodni za kliničku praksu. I na kraju, radiolozi[5] imaju

[1] Zraka se „emituje“ za svaku tačku prikaza, sa pozicije virtuelne kamere

[2] Gustina se ovdje odnosi na nepropusnost materijala na Rentgenske zrake (X-zrake), a brojna vrijednost je tzv. Hounsfield-ova vrijednost (-1000=zrak, 0=voda, +1000=kost)

[3] Voxel=volume element, jedinični element zapremine

[4] Domena je matematski pojam, skup svih vrijednosti nad kojima je funkcija definisana

[5] Radiolog je specijalista za dijagnozu i terapiju pomoću uređa koji se oslanjaju na radijaciju (kao što je CT i rentgen)

ograničenu količinu vremena za dijagnozu pojedinačnog pacijenta, što čini 2D prenosne funkcije nepraktičnim bez prikladnog upućivanja u proces ili polu-automatskog generiranja prenosnih funkcija.

Primjena neuronskih mreža u VolumeStudiu je dovoljno fleksibilna za mnoge vrste snimaka, ali moj fokus je bio (kao i cijelog tima koji ga razvija) na srcu, zbog dobre saradnje Univerziteta u Paderbornu sa Klinikom za srce i dijabetes Njemačke savezne države „Nordrhein-Westfalen". Konkretno, imali smo na raspolaganju dvadesetak snimaka srca i desetak snimaka svih drugih vrsta.

Direktno renderiranje zapremine

Za prikaz zapremine opisane skalarima, kao što je CT snimak, postoje različiti pristupi.

Korištenje teksturnih tehnika se pokazalo superiornim, jer kombinuje visok kvalitet slike i razumne brzine prikaza (framerate). Ovi pristupi koriste prednosti hardverske podrške za bilinearnu i trilinearnu interpolaciju koju pružaju moderne grafičke kartice, što čini visok kvalitet vizuelizacije dostupnim na obični PC kompjuterima. Kod ovih pristupa, kompletan skup podataka je smješten u teksturnu memoriju grafičke kartice. Zatim se vrši uzorkovanje koristeći hardversku interpolaciju.

Pristupi bazirani na 2D teksturama koriste 3 kopije podataka o zapremini u teksturnoj memoriji grafičke kartice. Svaka kopija sadrži fiksan broj presjeka duže jedne od ortogonalnih osa, a koji se adresiraju zavisno od trenutnog gledišta. Nakon bilinearne interpolacije, vrijednosti presjeka se propuštaju kroz lookup tabelu, renderiraju kao poligon u ravni i stapaju u ravan ekrana. Ova metoda često pati od vizuelnih grešaka koje uzrokuju fiksan broj presjeka i njihova statična pozicija duž glavnih osa. Alternativno, hardverske ekstenzije se mogu koristiti za međupresjek duž pravca gledanja sa ciljem postizanja većeg kvaliteta prikaza.

Moderne grafičke kartice podržavaju mapiranje 3D tekstura, koje dozvoljava pohranjivanje čitavog skupa podataka kao jedne 3D teksture u teksturnoj memoriji. Zbog toga je moguće uzorkovanje napraviti presjeke duž ose gledanja, koristeći trilinearnu interpolaciju. Ovaj pristup izbjegava vizuelne greške (artifakte) koji se javljaju kod metoda sa 2D teksturama prilikom prelaza sa jedne ortogonalne ose na drugu. 3D tekstura dozvoljava proizvoljnu frekvenciju uzorkovanja, što rezultira boljim sveukupnim kvalitetom slike. Takođe, potrebna je samo jedna kopija podataka, čime se snižavaju memorijski zahtjevi na teksturnu memoriju grafičke kartice.

Prenosne funkcije

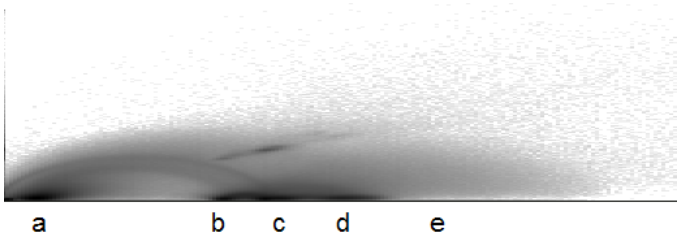
Vrijednost zapreminskog renderiranja u medicini, nauci i drugim poljima primjene ne zavisi samo od postignutog vizuelnog kvaliteta, jer je najvažniji zadatak naći dobru tehniku klasifikacije koja izdvaja interesantne karakteristike a potiskuje nebitne dijelove. Kao što je već rečeno, ta klasifikacija se može postići pomoću prenosnih funkcija koje vrijednostima iz skupa podataka daju optička svojstva kao što su boja i providnost.

2D prenosne funkcije ne klasificiraju zapreminske podatke samo na osnovu brojčane vrijednosti, nego na osnovu kombinacije osobina i zbog toga su u stanju bolje izolirati granične vrijednosti među strukturama od jednodimenzionalnih prenosnih funkcija. Razlog za ovo je što 2 različita tkiva mogu imati istu brojčanu (Hounsfieldovu) vrijednost, te ih jednodimenzionalne prenosne funkcije nisu u stanju razlikovati.

Kniss i ostali su predstavili metodu za ručno pravljenje višedimenzionalnih prenosnih funkcija baziranih na vrijednosti i njenim izvodima. Izvod je koristan kao dodatni kriterij za klasificiranje, pošto on razlikuje homogene oblasti u unutrašnjosti tkiva i granične oblasti na prelazima između tkiva. Naime, za voxele u unutrašnjosti tkiva, svi susjedni voxeli imaju približno iste vrijednosti,

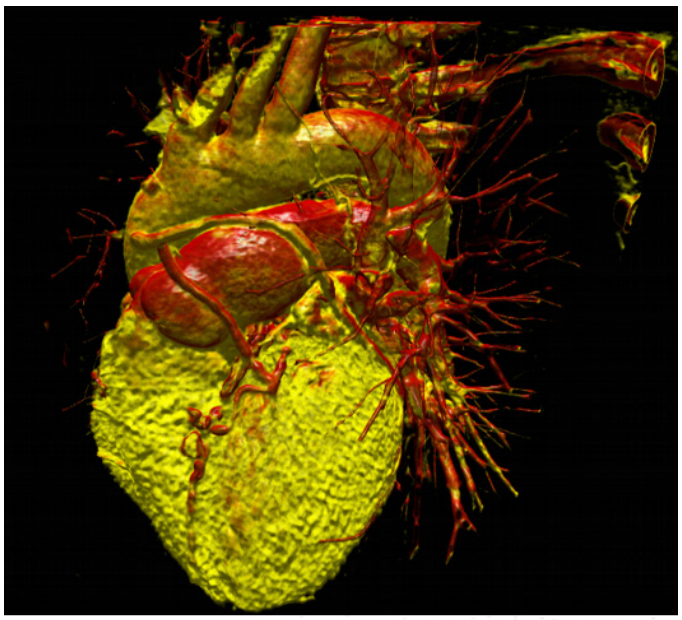
pa izvod tog voxela je blizu nule, a za voxel na graničnoj površi, susjedni voxeli sa jedne strane imaju slične vrijednosti a sa druge strane (unutar drugog tkiva) različite, pa je izvod daleko iznad nule (po apsolutnoj vrijednosti). Takođe, izvodi se mogu koristiti kod primjene osvjetljenja u procesu vizuelizacije, što poboljšava osjećaj dubine.

Ručno kreiranje se obavlja u vizuelnom editoru koji prikazuje distribuciju uređenih parova vrijednosti i njenog izvoda na zajedničkom histogramu. Koeficijent prigušenja rendgenskih zraka, odnosno „gustina" je okarakterisana Housfieldovom vrijednošću i prostire se duž x-ose. Izvod ove vrijednosti (odnosno razlika u odnosu na susjedne tačke zapremine, tj. voxele) se nalazi na y-osi.



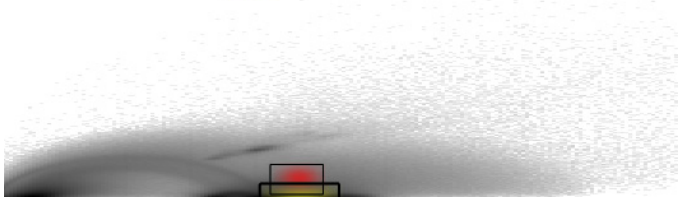
Homogene regije izgledaju kao elipsaste tačke na dnu histograma, pošto je njihov izvod blizu nule. 5 osnovnih regija na ovom snimku srca su: a) zrak, b) meka tkiva kao što su koža i masnoće, c) mišići, d) krv (koja sadrži kontrastni agens[1]) i e) kosti. Lukovi predstavljaju prelaze između ovih glavnih tkiva.

Da se napravi nova prenosna funkcija, postavljaju se filteri na histogram (dva pravougaonika u ovom slučaju). Svaki filter dodjeljuje boju i providnost voxelima u zapremini, koji odgovaraju uređenom paru paru u histogramu koje filter pokriva. Veličina i pozicija filtera se može mjenjati, kao i boja i distribucija providnosti. Implementirane su Gaussova i sinusna raspodjela. Ako je postavljeno više filtera, njihova boja i providnost se stapaju.



Opisano kreiranje se vrši interaktivno, i na osnovu grafičkog prikaza (u prostornom domenu) se može odlučiti da li je trenutna postavka filtera dobra. Pomoću ove „povratne veze" se takođe može odrediti koji dijelovi histograma odgovaraju kojim tkivima u skupu zapreminskih podataka.

Na slici desno je prikazan vizualiziran CT snimak srca, zajedno sa prenosnom funkcijom pomoću koje je nastao. Prenosna funkcija se sastoji od 2 filtera sa Gaussovom raspodjelom providnosti. Žuti filter je između regija c) i d), i predstavlja srčani mišić i krvne sudove. Crveni filter se nalazi iznad njega (djelimično se preklapaju) i pojačava kontrast između srčanog mišića i krvnih sudova, bojeći u crveno granice kontrastnog agensa (odnosno krvi).

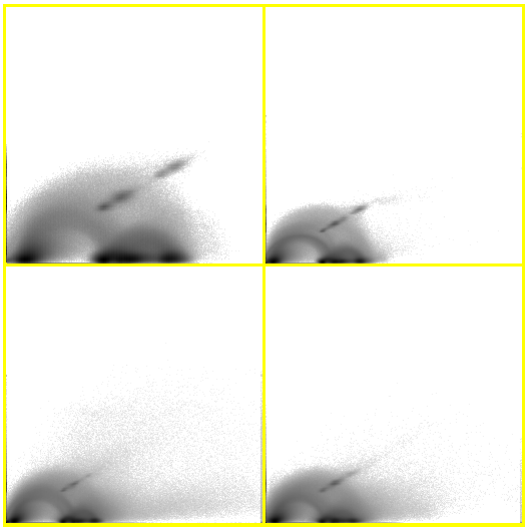


[1] Kontrasni agens je hemikalija (ili mješavina hemikalija) koja više prigušuje rentgenske zrake (krv ima gustoću tek nešto veću od vode koje u tijelu ima dosta), a dodaje se krvlju prilikom snimanja srca kako bi se dobio što kontrastniji snimak. Pošto se veoma brzo razgrađuje, vrijeme unutar kojeg se može napraviti kvalitetan snimak je tek nekoliko sekundi

Za iskusnog korisnika, poznavanje histograma, odnosno poznajući šta koji dio histograma predstavlja, olakšava kreiranje prenosnih funkcija. Ali i sa tim iskustvom, to je i dalje vremenski zahtjevan iterativni proces. Korisnik mora istraživati zapreminski skup podataka (snimak) postavljajući filtere i pomjerajući ih po interesantnim mjestima na histogramu. Kada se identificiraju interesantna tkiva, parametri filtera se moraju optimizirati (oblik, boja, distribucija providnosti) sa ciljem postizanja što bolje vizuelizacije.

Primjena neuronskih mreža u VolumeStudio-u

Poznajući osobine 2D histograma i prenosnih funkcija sa jedne, i poznavanja neuronskih mreža sa druge strane, osmisliti rješenje problema i nije tako teško.



Pošto histogrami srca imaju karakterističnu raspodjelu uređenih parova, tj. karakterističan oblik, ta činjenica se može iskoristiti kao dodatna informacija prilikom kreiranja prenosne funkcije. Na slici lijevo su dati histogrami 4 CT snimka srca. Ono od čega najviše zavisi kreiranje prenosne funkcije su pozicija i veličina elipsastih tačaka na dnu histograma, pošto one predstavljaju glavna tkiva, a na primjeru iz prethodnog naslova je objašnjeno da filtere treba postaviti između dijelova c) i d) (srčanog mišića i krvi sa kontrastnim agensom).

Ako se histogram dovede na ulaz neuronske mreže, može se iskoristiti za treniranje neuronske mreže da vrši prepoznavanje uzoraka (oblika), tj. da odredi pozicije regija c) i d).

Naravno, treniranje mreže se sastoji od davanja kombinacija histograma i željenih parametara filtera. To treniranje, koje je vremenski zahtjevno, se ne mora raditi često (u najboljem slučaju samo jednom), i može se uraditi van kliničkog okruženja, tj. u razvojnom okruženju (ali sa realnim kliničkim snimcima). Istrenirana mreža je u stanju da ispravno pozicionira filtere samo na osnovu histograma.

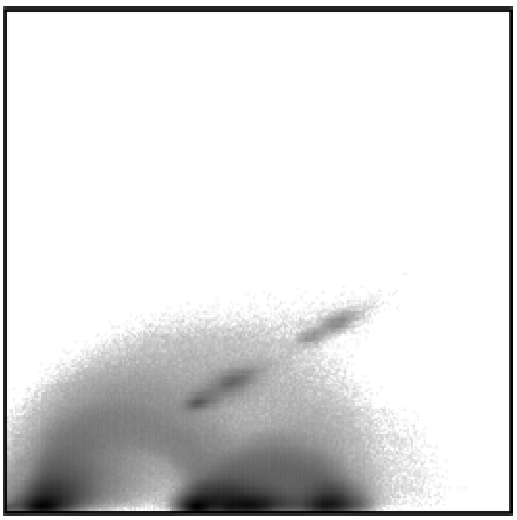
Parametri filtera koje smo odabrali da mreža daje kao izlaz su pozicija i veličina. Parametri koje mreža ne tretira su providnost (alfa parametar) i parametri Gaussove distribucije providnosti unutar filtera (sigma i granica providnosti).

2D histogram je slika od 256*256 pixela sivih nijansi. Kada bi svaka tačka histograma bio jedan ulaz, to bi zahtjevalo relativno mnogo memorije za operisanje (16MB samo za pohranu težina u slučaju 64 neurona u drugom sloju). Takođe, sa ovolikom mrežom treniranje bi sigurno bilo sporo, i njene mogućnosti generalizacije (koje su u ovom slučaju vrlo bitne) upitne.

Zbog toga se broj ulaza treba malo smanjiti. Prvo što sam uradio je odsjecanje gornjeg dijela histograma, pošto on skoro nikad ne sadrži uređene parove vrijednosti i izvoda. Za snimke srca (na koje smo bili fokusirani), odsjecanje gornje polovine je dovoljno, a sa stanovišta programske implementacije dosta jednostavno. Druga stvar je skaliranje naniže (na manju rezoluciju), i ovdje sam se odlučio za skaliranje sa brojem koji je stepen od 2, (2,4,8,...) i izabrao sam 4 kao broj koji dovoljno smanjuje sliku, a na slici se i dalje mogu identificirati potrebne karakteristike. Imali smo 256*256, odsjecanje pola ostavlja 256*128, a zatim skaliranje sa 4 daje 64*32. Dakle, broj ulaza smo smanjili sa 65536 na prihvatljivih 2048.

Dodatna prednost ovakvog pretprocesiranja je što se „izglađuju“ dijelovi histograma koji se nalaze van glavnih karakteristika (koja predstavljaju glavna tkiva). Ti uređeni parovi predstavljaju

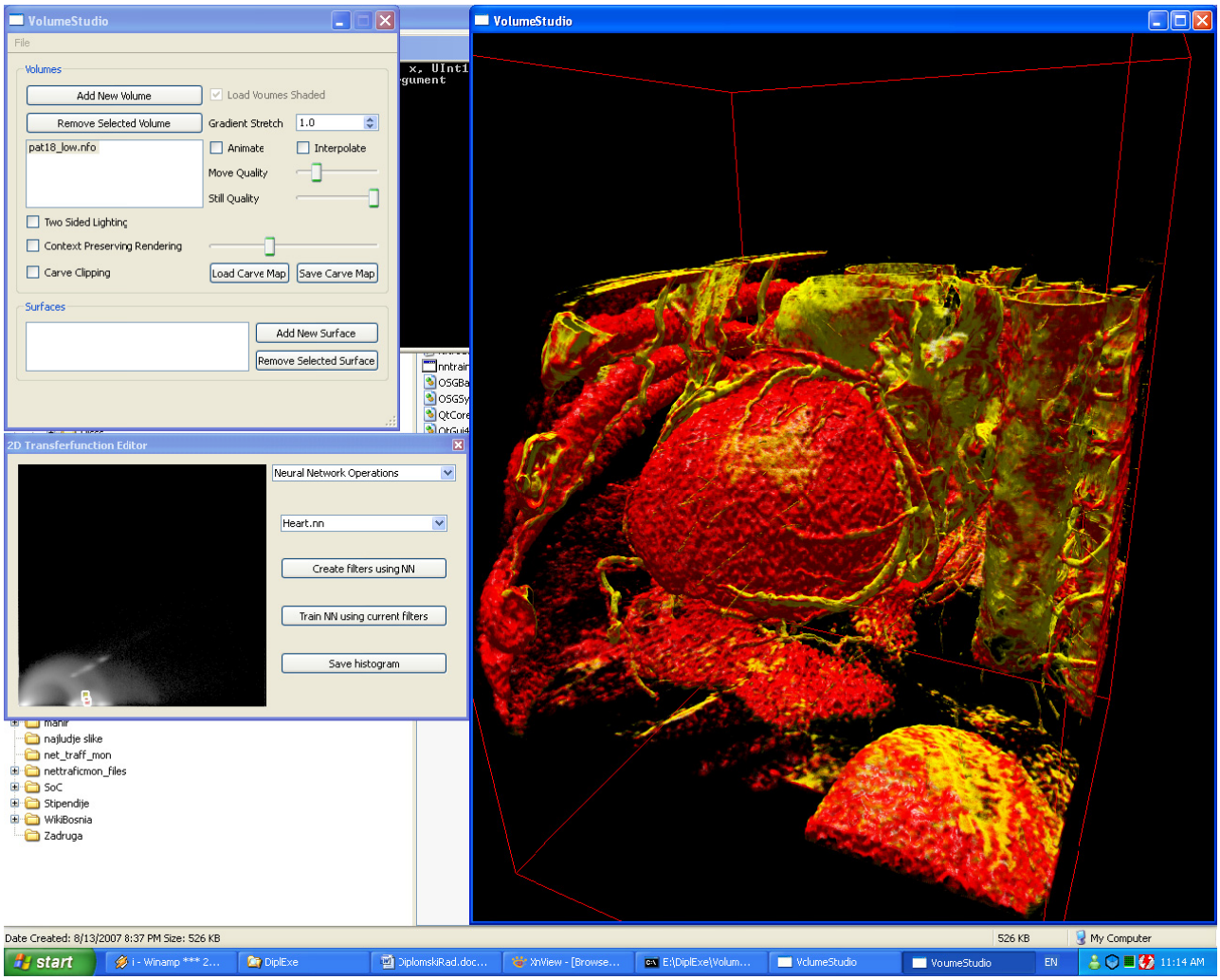
samo nekolicinu voxela, i neuronskoj mreži su praktično šum koji smeta prepoznavanju (jer se u svakom snimku nalaze na različitom mjestu). Ovo skaliranje dakle doprinosi brzini učenja, donekle olakšava učenje i, što je najvažnije, poboljšava generalizaciju.



Pošto se obično koriste dva Gauss filtera za vizuelizaciju srca, odlučio sam se da i neuronska mreža postavlja dva filtera. Pošto za svaki filter postoje 4 glavna parametra x i y koordinate i veličina, to neuronska mreža daje 8 izlaza (xpos1, ypos1, xsize1, ysize1, xpos2, ypos2, xsize2, ysize2).

To nam ostavlja neku varijabilnost jedino u skrivenim slojevima. Od početka razvoja rješenja, koristio sam 3-slojnu perceptronsku mrežu (1 skriveni sloj) sa sigmoidnim funkcijama aktivacije u svim slojevima i 64 neurona u skrivenom sloju.

Inače čitav softver je pisan u višeplatformskom okruženju, korišteni su jezik C++ za logiku, OpenGL za 3D prikaz, i QT prozorske biblioteke za korisnički interfejs. Zbog pretpostavke da ćemo raditi u Javi, ja sam ranije napravio implementaciju neuronske mreže u Javi, koja je sad



Slika 7 Slika ekrana (screenshot) za vrijeme rada u VolumeStudiu. Prikazan je izgled sa automatski napravljenom prenosnom funkcijom.

praktično beskorisna. Suočio sam se sa izborom: portanje tog koda iz Jave u C++, ili korištenje neke gotove biblioteke za neuronske mreže. Odabrao sam ovo drugo, iz razloga što je neka raširenija biblioteka vjerovatno bez grešaka (bugova), a i našao sam biblioteku koja mi konceptom i interfejsom izuzetno dobro leži. Ta biblioteka je Lightweight Neural Network++, inače razvijana i testirana na linuxu. Ta činjenica je uzrokovala jedan bug, oko kojeg sam izgubio dosta vremena a tiče se načina na koji se označava kraj linije na linuxu i windowsu. Osim tog problema, biblioteka se pokazala odlično.

Kada sam otprilike finalizirao dodatak VolumeStudiu koji se tiče neuronskih mreža, odlučio sam da malo eksperimentišem sa arhitekturom mreže. Prvo sam smanjio broj skrivenih neurona sa 64 na 32, a zatim i na 16, bez primjetnog padanja kvalitete rezultata (vizuelnih i MSE[1] kod treniranja). Zadržao sam taj broj neurona u skrivenom sloju, jer jednostavnija mreže bolje generalizuje. Broj od 3 sloja neurona u mreži nisam mijenjao, čisto jer nije bilo potrebe za tim.

Na završetku projekta, VolumeStudio je imao dodatak za neuronske mreže koje omogućavaju do-treniravanje postojećih neuronskih mreža (kao i upotreba tog dodatka za kreiranje prenosnih funkcija), kao i dobro istreniranu neuronsku mrežu za srce. Pored toga, napravio sam nezavisan mali alat koji može kreirati neuronsku mrežu prema zadatim parametrima, pomoću kojeg se mogu kreirati (a ja sam ih i kreirao) neuronske mreže za druge vrste snimaka, kao npr. Snimak glave, ili čitavog tijela. Drugih tipova snimaka nisam imao mnogo na raspolaganju, tako da sa njima nisam mogao eksperimentisati.

[1] MSE, Mean Square Error je srednja kvadratna greška (srednja vrijednost po trenažnim uzorcima sume kvadrata grešaka pojedinačnih izlaza)

Zaključak

Ono što bi se moglo još uraditi na poboljšanju dodatka za neuronske mreže je automatsko određivanje tipa snimka (da li je to snimak srca, glave, čitavog tijela ili, možda, samo ruke), i pozivanje odgovarajuće neuronske mreže (u trenutnoj implementaciji korisnik ručno odabire koju neuronsku mrežu da koristi). Također bi bilo interesantno isprobati kakvi bi bili rezultati sa skaliranjem histograma sa 2 umjesto sa 4, ali jednostavno nisam više imao vremena.

Profesorica Gitta Domik, šefica katedre za računarsku grafiku na univerzitetu u Paderbornu, rezultate koje daje mreža je ocijenila kao dobre, te pokazala interes za nastavak saradnje i sa mnom lično i sa mojim mentorom prof. Avdagićem.

Literatura

[1] "Fuzzy-Neuro-Genetika", Zikrija Avdagić, Grafo Art, 2002

[2] "Introduction to machine learning", Nils J. Nilsson, Draft, 1997

[3] "Information Theory, Inference and Learning Algorithms", David J.C. MacKay, Cambridge University Press 2003

[4] Wikipedia: http://en.wikipedia.org/wiki/Artificial_neural_network

[5] "Constructing Intelligent Agents using Java, 2nd edition", J.&J. Bigus, Wiley 2001

[6] J. Kniss, G. Kindlmann, C. Hansen. Multi-Dimensional Transfer Functions for Interactive Volume Rendering. IEEE Transactions on Visualization and Computer Graphics, 8(3): 270-285, July, 2002