UNIVERZITET U BEOGRADU

MATEMATIČKI FAKULTET

VEŠTAČKE NEURONSKE MREŽE U ISTRAŽIVANJU PODATAKA: PREGLED I PRIMENA

MASTER RAD

Mentor:
dr. Nenad Mitić

Kandidat:
Sava Gavran
(1036/2012)

Beograd, januar 2016.

SADRŽAJ:

1. Uvod

"Verujem da će se na kraju veka korišćenje reči kao i opšte mišljenje obrazovanih toliko promeniti da će se moći pričati o tome kako mašine razmišljaju bez očekivanja kontradikcije."
~ Alan Turing, otac računarske nauke

U istraživanju podataka i kognitivnim naukama, veštačke neuronske mreže su familija statističkih modela učenja inspirisana biološkim neuronskim mrežama. Koriste se u svrhu aproksimacije funkcija koje mogu zavisiti od velike količine ulaznih podataka, a koje su u principu nepoznate. Veštačke neuronske mreže su sistemi međusobno povezanih neurona, koji šalju poruke jedni drugima. Veze između ovih neurona imaju numeričke težine koje mogu biti podložne promenama u zavisnosti od iskustva, što neuronske mreže čini adaptivnim i sposobnim za učenje.

Ljudski mozak sadrži otprilike 100 milijardi neurona. Svaki neuron sadrži telo ćelije, a telo ćelije čini centralnu masu iste. Telo ima određen broj veza-ulaza tj. dendrita, koji dovode informacije ka telu ćelije i jedan akson, preko kojeg se prenosi rezultat rada tj. izlaz ćelije. U računaru, ekvivalent neurona bio bi nanoskopski prekidački uređaj koji se zove tranzistor. Moderni mikroprocesori sadrže preko dve milijarde tranzistora, spakovanih u integrisano kolo površine od oko 25 kvadratnih milimetara. Ovde se poređenje mozga i računara završava, jer se oni u potpunosti razlikuju. Prava razlika je u „razmišljanju" računara u odnosu na ljudski mozak. Naime, računari i mozgovi "misle" na drugačiji način. Tranzistori u računaru su umreženi na relativno jednostavan način, u serijske lance (svaki je povezan sa dva ili tri druga tranzistora, formirajući osnovna logička kola), dok su neuroni u ljudskom mozgu gusto povezani na složene, paralelne načine (svaki je povezan sa približno deset hiljada suseda).

Suštinska strukturalna razlika računara i mozgova je osnov razlike u načinu na koji misle. Računari su savršeno kreirani da skladište ogromne količine (za njih) beznačajnih podataka i da ih preuređuju na brojne različite načine, u zavisnosti od preciznih instrukcija koje im bivaju prosleđene unapred, u vidu programa. Ljudski mozgovi, sa druge strane, uče sporo,

često mesecima ili godinama kako bi u potpunosti dodelili smisao veoma složenim pojavama i stvarima. Za razliku od računara, mogu spontano da spoje informacije na nove načine; prepoznaju iskonske šablone, ostvariju nove veze i spoznaju već naučene stvari u potpuno novom svetlu.

Osnovna ideja veštačke neuronske mreže je simulacija velike količine gusto napakovanih, međusobno povezanih nervih ćelija u okviru računara, tako da je omogućeno učenje pojmova, prepoznavanje šablona i donošenje odluka na način koji je sličan čovekovom. Neuronska mreža se ne programira eksplicitno da uči, ona to radi sama, isto kao i mozak. Suštinski, veštačke neuronske mreže su softverske simulacije, napravljene programirajući obične računare koji rade u uobičajenom režimu sa svojim tranzistorima i serijski povezanim logičkim kolima, tako da se ponašaju kao da su napravljene od milijardu međusobno povezanih ćelija mozga koje rade paralelno. Niko do sada nije pokušao da napravi računar tako da tranzistore umreži na način na koji su umreženi neuroni u mozgu. Drugim rečima, veštačka neuronska mreža se razlikuje od ljudskog mozga na isti način na koji se računarski model vremenskih prilika razlikuje od pravih oblaka, pahuljica ili svetlosnih zraka. Računarske simulacije su samo kolekcija algebarskih promenljivih i matematičkih jednačina koje ih spajaju. One ništa ne znače računarima, već ljudima koji ih programiraju. U daljem tekstu, svako pominjanje neuronskih mreža, odnosiće se upravo na veštačke neuronske mreže.

Tipična neuronska mreža ima od nekolicine do stotinu, hiljadu, ili pak milion veštačkih neurona tj. jedinica, umreženih u serije slojeva, gde je svaki neuron povezan sa oba sloja sa obe strane. Neki od njih, koji se nalaze u početnom sloju, tj. sloju ulaza, dizajnirani su tako da dobijaju različite oblike informacija iz spoljašnjeg sveta. Jedinice koje se nalaze na suprotnoj strani mreže, u krajnjem sloju, tj. sloju izlaza, signaliziraju način na koji mreža reaguje na naučene informacije. Između leži jedan ili više skrivenih slojeva, koji zajedno čine većinu veštačkog mozga. Većina neuronskih mreža su potpuno povezane, što znači da je svaki neuron iz skrivenog nivoa kao i svaki neuron iz izlaznog sloja, spojen sa svakim neuronom iz slojeva sa obe strane. Ove veze su predstavljene brojem koji se definiše kao težina. Težina može da bude ili pozitivna (ukoliko jedan neuron pobuđuje drugog) ili negativna (ukoliko jedan neuron inhibira drugog). Što je ova težina veća, veći uticaj jedan neuron ima na drugog. Ovo odgovara načinu na koji prave neuronske ćelije pobuđuju jedne druge, kroz sinapse.

Tok informacija kroz neuronsku mrežu odvija se na dva načina. Kada uči, tj. dok biva trenirana ili radi u uobičajenom režimu (nakon što se istrenira), šabloni informacija ulaze u mrežu putem ulaznog sloja, koji potom dalje pobuđuje slojeve skrivenih jedinica, da bi na kraju završili u izlaznom sloju. Ovakav uobičajeni dizajn zove se mreža propagacije unapred (*eng. Feedforward network*).

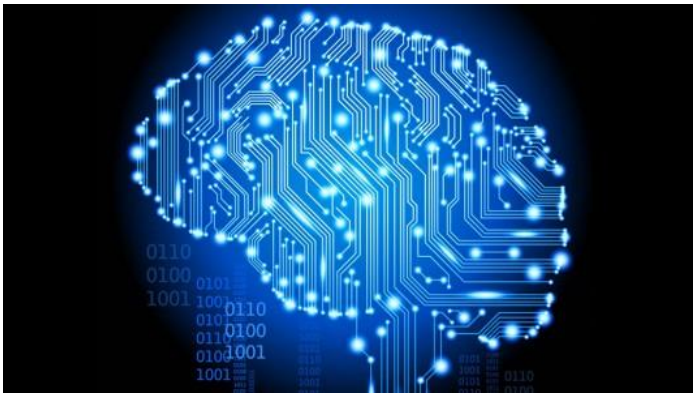
Svaka jedinica dobija svoj ulaz od jedinica iz prethodnog sloja, nakon čega se ulazne vrednosti množe sa težinama veza kroz koje su povezani. Svaka jedinica sumira sve ulazne vrednosti koje dobija na ovakav način. Ako je suma veća od određene vrednosti, koja se obično zove prag (*eng. Threshold*), jedinica se aktivira i pobuđuje jedinice sa kojima je povezana sa desne strane.

Kako bi neuronska mreža naučila da prepoznaje i klasifikuje pojmove, mora da postoji povratna informacija; kao kad deca uče tako što im se ukaže da je to što rade dobro ili loše. Svi ljudi koriste povratne informacije, u svakom trenutku. Mozak čoveka koji prvi put igra kuglanje posmatra koliko brzo se kreće kugla i putanja kojom ide. Evidentira koliko mu je bilo blizu da poruši sve čunjeve. Sledeći put kad se baci kugla, mozak se seća šta je pogrešno proračunao i ispravlja pokrete tela u skladu sa time, u nadi da će kugla biti bolje bačena. Povratna informacija se koristi kako bi se poredio željeni ishod sa ishodom koji se stvarno desio. Shvati se razlika između ova dva ishoda i ta novonaučena informacija doprinosi da se sledećom iteracijom stvarni ishod bliži željenom ishodu. Što je veća razlika između željenog i stvarnog ishoda, to je drastičnije potrebno menjati pokrete.

Veštačke neuronske mreže uče na isti način. Obično, kroz proces koji se zove propagacija unazad (*eng. Backpropagation*). Ovaj proces podrazumeva poređenje izlaza koji mreža generiše sa izlazom koji je trebalo da generiše. Razlika između ovih izlaza se koristi pri ispravljanju težina veza u mreži. Najpre se ispravljaju težine bliže izlaznom sloju, krećući se unatrag, sve do ulaznog sloja. Propagacija unazad, kroz neko vreme omogućava mreži da uči, redukujući razliku između stvarnih i željenih rezultata, sve dok se ove dve vrednosti ne poklope.

Kada se mreža istrenira sa dovoljnom količinom podataka, uvode se novi podaci, kod kojih nedostaje atribut po kome se vrši klasifikacija, kako bi se testirala njena uspešnost. Primera radi, istrenirana je mreža tako da prepoznaje slike određenih predmeta-stolica i stolova. Nakon toga, mreži se daje slika nekog trećeg pojma, npr. fotelje. U zavisnosti od toga kako je istrenirana, mreža će pokušati da kategoriše nove primere kao ili stolicu ili kao sto, uopštavajuću na osnovu prethodnih iskustava, isto kao i čovek. Ovo doduše, ne znači da neuronska mreža može da gleda slike nameštaja i u trenutku odreaguje na smislen način, tj. ne ponaša se kao osoba.

Ulazi neuronske mreže su brojevi, tako da je svaka jedinica ulaza ili isključena ili uključena. Ako postoji pet ulaznih jedinica, mreža se može izložiti informacijama o pet različitih karakteristika stolica koristeći binarne da/ne odgovore. Pitanja bi bila npr. Da li ima leđa? Da li ima ploču? Da li ima tapacirung? Da li može udobno da se sedi duže vreme? Da li može veća količina stvari da se stavi na površinu? Tipična stolica bi bila predstavljena sa Da, Ne, Da, Da, Ne ili binarno 10110, dok bi tipični sto bio predstavljen sa Ne, Da, Ne, Ne, Da ili 01001. Tokom faze učenja, mreža samo posmatra veliku količinu brojeva kao što su 10110 i 01001 i uči da neki od njih znače stolica (što bi na primer bio izlaz 1), dok neki drugi znače sto (izlaz 0).



Slika 1.1. Veštački mozak (ilustracija)

U radu [1] napravljen je računarski model za neuronske mreže zasnovane na matematici i algoritmima koji koriste logiku praga (*eng. Threshold Logic*). Ovaj rad je uveo prvi matematički model neuronske mreže, koristeći elektronska kola. Jedinice ovog modela, jednostavni formalizovani neuroni, još uvek su standardna referenca u polju veštačkih neuronskih mreža. Neuron ovog modela se naziva Mekkuloh-Pits (McCulloch-Pitts) neuron. Ovaj model prouzrokuje podelu istraživanja neuronskih mreža na dva različita prilaza. Jedan prilaz se fokusira na biološke procese u mozgu, a drugi se fokusira na primenu neuronskih mreža u veštačkoj inteligenciji.

Psiholog Donald Heb (Donald Hebb) je 1949. Godine, u radu [2] napravio model učenja zasnovan na mehanizmu neuralne plastičnosti (neural plasticity), koncept koji je fundamentalno esencijalan za način na koji ljudi uče. Ovaj model je danas poznat kao Hebovsko učenje (Hebbian learning). Ovaj rad je ukazivao na to da veze između neurona jačaju svaki put kada se koriste. Ako se dva neurona pobude u isto vreme, po Hebu, njihova veza jača. Hebovsko učenje važi za jedno tipično nenadgledano učenje i njegove varijante nastale kasnije, bili su rani modeli za ideju dugoročnog potenciranja (*eng. Long Term Potentiation*). Istraživači su počeli da koriste ove ideje kod računarskih modela 1948. godine sa Tjuringovim B-tip mašinama.

Kako su računari bivali sve napredniji 50-tih godina, konačno je bila moguća simulacija hipotetičke neuronske mreže. Prve korake u ovom smeru je učinio Natanjel Ročester (Nathanial Rochester) iz IBM laboratorija. Nažalost, prvi pokušaj nije uspeo.

Farli (Farley) i Klark (Clark) 1953. godine, prvi su koristili kalkulatore, za simulaciju Hebovske mreže na MIT-u. Frenk Rozenblat (Frank Rosenblatt) 1958. godine napravio je algoritam za prepoznavanje šablona, zasnovan na dvoslojnoj mreži koristeći sabiranje i oduzimanje. Uz matematičku notaciju, Rozenblat je takođe opisao kolo koje nije u okviru običnog perceptrona, kao što je ekskluzivno *ili* kolo, čije matematičko izračunavanje nije moglo da se obradi sve dok se nije pojavio algoritam propagacije unazad, napravljen od strane Pola Verbosa (Paul Werbos) 1975. godine.

Godine 1959. Bernard Vidrov (Bernard Widrow) i Marcijan Hof (Marcian Hoff) sa Stanforda razvijaju modele “ADALINE” i “MADALINE”. Ova imena su akronimi od Multiple

ADAptive LINear Elements. ADALINE je razvijen u cilju prepoznavanja binarnih šablona tako je da prilikom čitanja bitova sa telefonske linije, moguće predvideti sledeći bit. MADALINE je bila prva neuronska mreža primenjena na problem iz realnog sveta, koristeći adaptivne filtere koji eleminišu ehoe na telefonskim linijama.

Godine 1962, Vidrov i Hof razvijaju proceduru učenja, koja ispituje vrednosti, pre nego što se težina promeni, po pravilu: Promena težine = (Vrednost pre primene težine) * (Greška / (Broj jedinica ulaza)). Zasnovana je na ideji da dok jedan aktivan perceptron možda ima veliku grešku, mogu se ispraviti vrednosti težina, tako da se greška distribuira na celu mrežu, ili makar na susedne perceptrone. Primenjujući ovo pravilo, i dalje se nailazi na grešku, ukoliko je vrednost pre težine 0, iako će verovatno samu sebe ispraviti. Ukoliko je greška distribuirana na sve težine, onda se može reći da je eleminisana.

Iako su bila uspešna u polju neuronskih mreža, neuralna istraživanja su zapostavljena, a tradicionalna fon Nojmanova arhitektura je preuzela računarsku scenu. Ironično, fon Nojman lično predložio je imitacije neuralnih funkcija, koristeći releje telegrafa ili vakuumske cevi.

U istom periodu, napisan je članak, koji je sugerisao da je nemoguće proširiti jednoslojne neuronske mreže u višeslojne. Dodatno, većina ljudi u ovom polju je koristila funkcije učenja, koje su u osnovi imale nedostatke, jer nisu bile diferencijabilne u svim tačkama. Kao rezultat, finansiranja istraživanja su drastično smanjena. Rani uspesi su prouzrokovali preuveličavanje potencijala neuronskih mreža, posebno uzimajući u obzir tehničke mogućnosti tog vremena. Ljude je obuzeo strah od naučno fantastičnog scenarija gde roboti preovladavaju, a autori su polemisali o efektu koji bi takozvane mašine koje razmišljaju imale na ljude. O ovim temama se i dan danas diskutuje.

Istraživanje neuronskih mreža je stagniralo nakon publikacije istraživanja Marvina Minskog (Marvin Minsky) i Sejmura Paperta (Seymour Papert) u domenu mašinskog učenja. Oni su otkrili dva ključna problema kod računara koji su obrađivali neuronske mreže. Prvi problem bio je to što jednoslojna neuronska mreža nije u mogućnosti da simulira ekskluzivno *ili* kolo. Drugi problem bio je što računari tada nisu imali dovoljno procesorske snage, koje su iziskivale velike neuronske mreže. Značajni napredak ostvaren je kada se došlo do algoritma propagacije unazad, koji je efektivno rešio problem ekskluzivnog *ili* kola.

Godine 1982, zainteresovanost za neuronske mreže je obnovljena. Džon Hopfild (John Hopfield) iz instituta Caltech, predstavio je rad [3] američkoj nacionalnoj akademiji nauka. Njegov pristup problemu bio je da napravi korisnije mašine, koristeći dvosmerne veze. Do tada su veze između neurona bile samo u jednom smeru. Iste godine, Rajli (Reilly) i Kuper (Cooper) koriste hibridnu mrežu, sazdanu od više slojeva, gde svaki sloj koristi drugačiju strategiju rešavanja problema. Godine 1982. održana je Američko-Japanska konferencija na temu kooperativnih/kompetativnih neuronskih mreža. Japan je najavio novu petu generaciju računara kroz neuronske mreže. Američki novinari su izrazili brigu što Amerika zaostaje u toj oblasti.

Prva generacija je koristila prekidače i žice, druga tranzistore, treća integrisana kola i programske jezike višeg nivoa, a četvrta generatore koda. Peta generacija računara podrazumeva veštačku inteligenciju. Iz ovih razloga, više se ulagalo, a samim tim bilo je i više istraživanja u Americi.

Godine 1986. kod neuronskih mreža sa više slojeva, problem je bio kako proširiti Vidrov-Hof (Widrow-Hoff) pravilo na više slojeva. Tri različite grupe istraživača, od kojih je u jednoj bio Dejvid Rumelhard (David Rumelhard), bivši član Stanfordove katedre za psihologiju, došle su do iste ideje, koja se danas zove propagacija unazad, zato što distribuira greške u prepoznavanju kroz celu mrežu. Hibridne mreže su koristile samo dva sloja, dok mreže sa propagacijom unazad mogu da koriste više slojeva. Zbog toga mreže sa propagacijom unazad sporo uče i potrebno im je hiljade iteracija za obuku.

Paralelno izvršavanje sredinom osamdesetih godina prošlog veka, postalo je popularno pod nazivom konekcionizam (*eng. Connectionism*). Skripta Dejvid Rumelharta i Džejms Meklelanda (James McClelland) 1986. Godine, obezbedila je kompletan pregled korišćenja konekcionizma kod računara koji simuliraju neuronske procese. Neuronske mreže, korišćene u veštačkoj inteligenciji, bile su posmatrane kao pojednostavljeni modeli neuroloških procesa u mozgu, iako je relacija između ovog modela i biološke arhitekture mozga diskutabilna.

Mašine podržavajučih vektora i ostale jednostavnije metode kao što su linearni klasifikatori, postepeno su postale popularnije od neuronskih mreža u domenu istraživanja podataka i mašinskog učenja. Tek sa početkom novog milenijuma, neuronskim mrežama raste popularnost, zahvaljujući novim visoko-performansnim algoritmima koji su pomerili granice praktičnih primena. U krugovima naučnika počinje da kruži termin dubokog učenja (*eng. Deep Learning*), koji je samo sinonim za neuronske mreže i predstavlja neki oblik ponovnog brendiranja nečeg što već postoji, zarad širenja interesovanja i dobijanja na popularnosti.

Zbog novodobijene popularnosti, počinju da se organizuju takmičenja neuronskih mreža. Između 2009. i 2012. godine, rekurentne neuronske mreže i mreže propagacije unapred, razvijene u razvojnoj grupi Jurgen Šmidhubera (Juergen Schmidhuber) u švajcarskoj laboratoriji veštačke inteligencije IDSIA, bile su prve na osam internacionalnih takmičenja u prepoznavanju šablona i mašinskom učenju. Dvosmerna i multidimenzionalna duga kratkoročna memorija (*eng. Long short term memory*) autora Aleksa Grejvsa (Alex Graves) osvojila je tri takmičenja 2009. godine na ICDARu (*eng. International Conference on Document Analysis and Recognition*) u prepoznavanju rukom pisanih tekstova.

Implementacije zasnovane na korišćenju grafičkog procesora od strane Den Kiresen-a (Dan Ciresan) i kolega sa IDSIA, pobedile su na više takmičenja u prepoznavanju šablona uključujući: IJCNN 2011 takmičenje u prepoznavanju saobraćajnih znakova, ISBI 2012 segmentacija neuronskih struktura u stekovima elektronske mikroskopije (Segmentation of Neuronal Structures in Electron Microscopy Stacks challenge), i druge. Njihove neuronske mreže su bile prvi veštački prepoznavači šablona, koji su mogli da dostignu performance koje mogu da pariraju čoveku, ili pak da imaju i nadljudske mogućnosti kod prepozavanja saobraćajnih znakova i rukopisa.

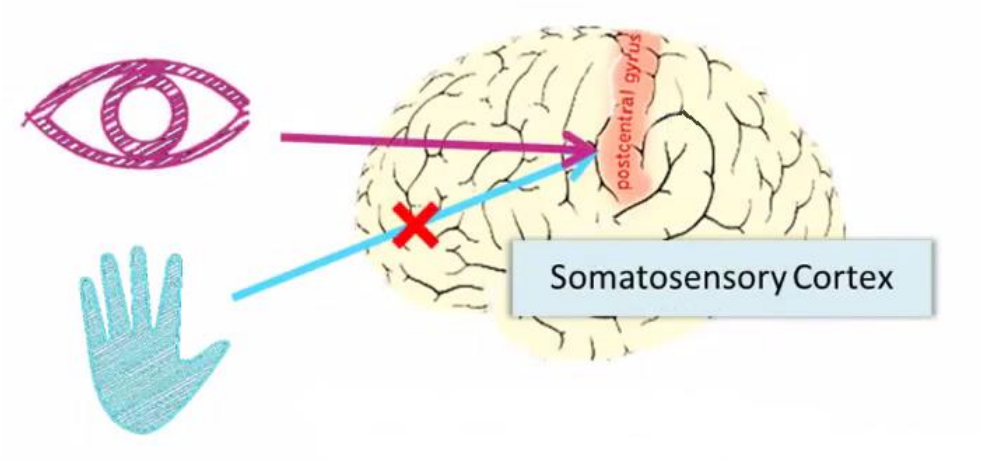
Fundamentalna ideja iza neuronskih mreža je da ukoliko je moguće da radi u prirodi, mora da može da radi i u računaru. Pored razvoja algoritama, budućnost neuronskih mreža zavisi i od razvoja hardvera. Slično kao napredne mašine za igranje šaha, brze efikasne neuronske mreže zavise od hardvera koji je specijalno prilagođen za njihovu upotrebu. Zbog ograničenja procesora, neuronskim mrežama su potrebne nedelje da se istreniraju. Neke kompanije pokušavaju da naprave tzv. silikonski kompajler, koji bi generisao specifičan tip integrisanog kola, koji je optimizovan za neuronske mreže. Digitalni, analogni i optički čipovi, su razne vrste integrisanih kola koja se razvijaju. Iako se možda smatra da su analogni signali stvar prošlosti, neuroni u mozgu zapravo rade više kao analogni signali, nego kao digitalni. Dok digitalni signali imaju dva stanja (1 ili 0), analogni signali variraju u vrednosti između minimalnih i maksimalnih vrednosti. Biće potrebno neko vreme da optički čipovi počnu da se koriste u komercijalne svrhe.

2. Principi funkcionisanja

Algoritmi neuronskih mreža su nastali kao pokušaj da se oponaša način na koji ljudski mozak uči, tj. način na koji ljudski mozak klasifikuje i spoznaje stvari. Računske operacije koje obavljaju neuronske mreže su zahtevne, stoga je tek relativno skoro napredak tehnologije omogućio njihovo intenzivnije korišćenje. Oponašanje mozga je jako zahtevan proces, s obzirom na širok dijapazon mogućnosti i količine podataka koje mozak svakodnevno obrađuje.

Postoji hipoteza koja tvrdi da svi procesi koje mozak obavlja, nisu sačinjeni od velikog broja raznih programa, već od jednog algoritma učenja. Ovo jeste samo hipoteza ali postoje određeni naučni dokazi koji idu u prilog ispravnosti ove hipoteze. Naučnici su uspeli da na životinjama prespoje komunikacione kanale u mozgu na sledeći način: neuronske veze između ušiju i centra za sluh, koje omogućavaju životinjama (i ljudima) da čuju, prespojili su sa nervnim kanalima očiju, tako da električni impulsi koje oko proizvodi kao signal ne završavaju u centru za vid već u centru za sluh. Centar za sluh je naučio da vidi u punom smislu te reči. Slično je urađeno i sa centrom za dodir. Prespojen je na isti način i naučio je da vidi.

Ovakvi eksperimenti su nazvani neuronskim prespajanjima (*eng. Neuro re-wiring*). Zahvaljujući njima, stičemo utisak da jedan isti deo fizičkog tkiva mozga može da obrađuje čulo vida, sluha ili dodira. Onda je moguće postojanje jednog algoritma učenja koji obrađuje sve moguće čulne "ulaze". Cilj je naći neku aproksimaciju načina na koji mozak obavlja obradu svih tih različitih podataka.



Slika 2.1. Centar za osećaj uči da gleda

Postoji još primera u kojima su naučnici uspešno manipulisali čulima na ovaj način. Gledanje jezikom funkcioniše tako što se na glavu stavi kamera koja uzima niskorezolucione nijanse sive slike ispred osobe, iz koje izlazi žica sa nizom elektroda koja se stavi na jezik.

Svaki piksel se preslikava na određenu lokaciju na jeziku, gde se na primer visoka voltaža preslikava na taman piksel a niska na svetao. Ekolokacija ili ljudski sonar je tehnika koja se

već uveliko koristi kako bi pomogla slepim osobama da imaju neku predstavu o okruženju. Funkcioniše tako što se proizvodi konzistentan zvuk, npr. pucketanje prstima nakon čega se nauči kako da se interpretira šablon zvukova koji se odbijaju o površine i time ustanovi udaljenost od objekata.

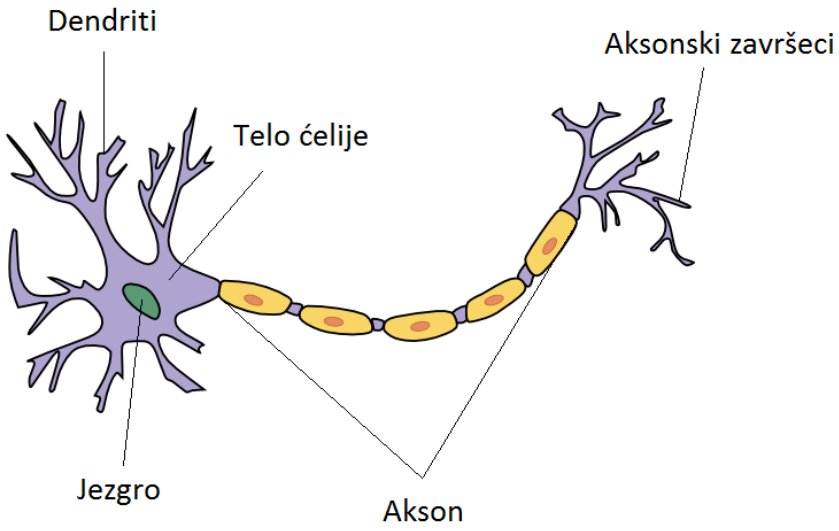
Haptički kaiš je kaiš koji šalje vibracije na određen deo struka u zavisnosti od objekata u blizini, što omogućuje usmeravanje osobe u određenom smeru. Ovakav kaiš mogao bi da nađe razne primene, između ostalog kao pomagalo slepim osobama. Američka vlada finansira projekat za razvoj ovakvog kaiša u vojne svrhe, radi usmeravanja vojnika na bojnom polju. Jedan bizarniji primer je implantiranje trećeg oka žabi, nakon čega je žaba naučila da ga koristi. Ostaje utisak da može da se priključi bilo kakav senzor na mozak, a on će naučiti kako da uči na osnovu novodobijenih podataka i kako da se nosi sa tim podacima.

Slika 2.2. Tehnike neuro prespajanja

Neuronske mreže su razvijene tako da simuliraju neurone, tj. mreže neurona mozga. Neuron se sastoji od dve celine. Telo, koje se sastoji od ulaznih vlakana, koje prima signale sa drugih lokacija. Izlazno vlakno ili akson je u obliku repa i služi za propagaciju informacija nastalih pod uticajem ulaznih signala ka drugim neuronima.

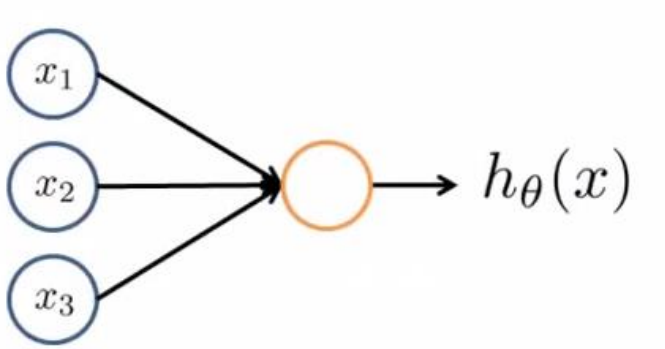
Neuron je po svojoj suštini jedinica izračunavanja, koja dobija određen broj ulaza, obrađuje ih i izlaz prosleđuje dalje kako bi se izračunavanje nastavilo. Informacije se prenose preko pulseva elektriciteta. Pulsevi se šalju kroz akson do sinaptičkih završetaka, gde se preko sinapse prenose na dendrite drugih neurona. Ovo je princip po kome se odvija ceo čovekov misaoni proces. Neuroni koji rade izračunavanja, prosleđuju poruke koje su rezultat ulaza koje su primili.

Na istom principu rade naša čula i mišići. Na primer, signal kojim se mišićima govori da se sakupljaju ili oko koje šalje mozgu informacije koje prima iz spoljašnjeg sveta.



Slika 2.3. Neuron

U mreži veštačkih neurona koji se modeliraju, koristi se jednostavan model onoga što neuron radi. Neuron se modelira kao logička jedinica. Žuti krug (*slika 2.4.*) predstavlja telo neurona a ulazne grane predstavljaju dendrite. Takođe, postoji jedna izlazna grana koja predstavlja akson. Ovakva jedinica se u literaturi naziva *perceptron*.



Slika 2.4. Model neurona u veštačkim neuronskim mrežama (*Perceptron*)

Ulazni podaci modela su dati vektorom *x*, a težinski faktori vektorom *θ (formula 2.1)*. Težinski faktori su od ključnog značaja za prilagođavanje modela podacima i generalno za ceo proces učenja. Obično se nasumično inicijalizuju i menjaju posle svake iteracije algoritma.

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \tag{2.1}$$

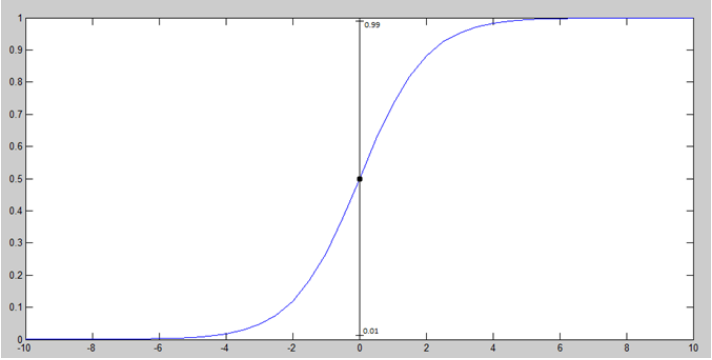
Funkcija aktivacije perceptrona definiše njegov izlaz u zavisnosti od ulaza. Rezultat koji se formira na izlazu obeležava se sa h_θ (*formula 2.2*). Funkcija aktivacije koja se najčešće koristi je sigmoidna funkcija:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} \tag{2.2}$$

gde je x ulaz, θ težinski faktori, a T funkcija transpozicije. Funkcija aktivacije se u literaturi takođe zove i hipoteza ili model, što je opšti naziv koji se odnosi na rešenje problema nekog algoritma istraživanja podataka. Opšti oblik sigmoidne funkcije je:

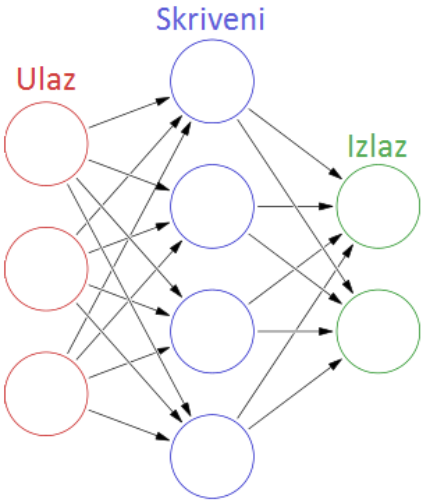
$$g(z) = \frac{1}{1+e^{-z}} \tag{2.3}$$

a njen grafik je prikazan na slici 2.5. Pored sigmoidne funkcije, za ulogu funkcije aktivacije koriste se još i funkcija jediničnog stepenika (*eng. Unit step*), deo po deo linearna (*eng. Piecewise Linear*), Gausova funkcija tj. normalna raspodela, kao i mnoge druge. Više o funkcijama aktivacije može se naći u [4].



Slika 2.5. Sigmoidna funkcija

Neuronska mreža može se definisati kao grupa različitih neurona koji su spojeni, a jedinično uvećanje (x_0) kao neuron koji uvek ima vrednost 1. Jedinično uvećanje se obično dodaje svakom sloju koji nije izlazni i služi kako bi se ustanovio prag odlučivanja izlazne vrednosti. Iako sama jedinica ima vrednost 1, bitan je i težinski faktor koji joj je pridružen. Ova jedinica se spaja samo sa drugim jedinicama koje nisu jedinice uvećanja. U osnovnom modelu neuronske mreže učestvuju tri sloja: ulazni sloj, skriveni sloj i izlazni sloj (*slika 2.6.*). Svaki sloj koji nije ulazni ili izlazni je skriveni sloj.



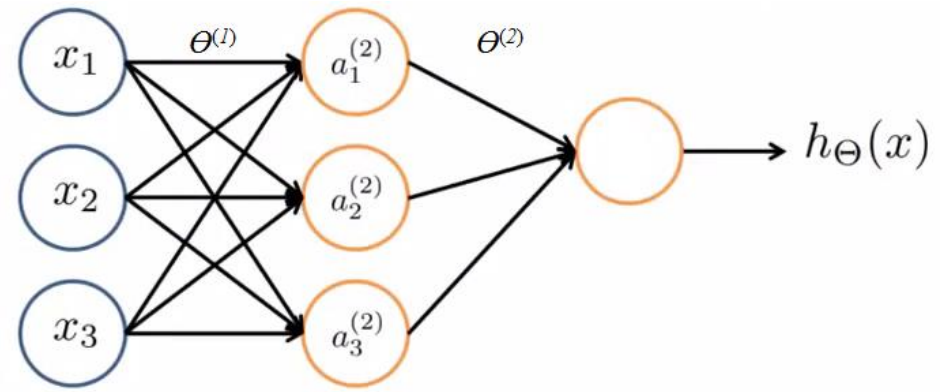
Slika 2.6. Neuronska mreža

U neuronskoj mreži može da postoji više skrivenih slojeva, gde je $a_i^{(j)}$ *aktivacija* neurona *i* u sloju *j*. Aktivacija neurona je vrednost u konkretnom neuronu nakon obrađivanja ulaza, tj. nakon primene funkcije aktivacije na linearnu kombinaciju. $\theta^{(j)}$ je matrica težina koja parametrizuje i preslikava vrednosti iz sloja *j* u sloj *j* + 1. Kada se ulaznim vrednostima jednog sloja pridruže težinski faktori dobija se linearna kombinacija. Kako bi dobili vrednost jedinice u sledećem sloju, potrebno je na određenu linearnu kombinaciju primeniti izabranu funkciju aktivacije (u konkretnom slučaju sigmoidnu funkciju). Sigmoidna funkcija primenjena na linearne kombinacije ulaza data je u sledećoj formuli:

$$
\begin{aligned}
a_i^{(j+1)} &= g\left(\sum_{k=0}^{s_j} \theta_{ik}^{(j)} x_k\right) \\
a_1^{(2)} &= g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3) \\
a_2^{(2)} &= g(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3) \\
a_3^{(2)} &= g(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3) \\
h_\theta(x) &= a_1^{(3)} = g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})
\end{aligned}
\tag{2.4}
$$

U konkretnom primeru (*slika 2.7.*), postoje tri ulazne jedinice i tri skrivene jedinice. $\theta^{(1)}$ je matrica preslikavanja ulaznog sloja na skriveni sloj. Ako mreža ima s_j jedinica u sloju *j*, s_{j+1} jedinica u sloju $j+1$, onda će $\theta^{(j)}$ biti dimenzije $s_{j+1} \times (s_j + 1)$. Npr. $\theta^{(1)} \in \Re^{3x4}$. Za različite parametre θ dobijamo različite funkcije preslikavanja i različite modele. Ovaj iterativni algoritam, gde se rezultati propagiraju od levih slojeva ka desnim prilikom izračunavanja vrednosti u neuronima se zove algoritam propagacije unapred. Radi efikasnog izračunavanja, pravi se vektorizovana implementacija (podaci se tretiraju kao vektori i matrice, kako bi se operacije nad istim odvijale u jednom koraku a ne iterativno) linearne kombinacije iz prethodne formule (*formula 2.4*). Zbog preglednosti, uvodi se oznaka z koja označava ove kombinacije:

$$
\begin{aligned}
z_1^{(2)} &= \theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3 ; \text{ sledi } a_1^{(2)} = g(z_1^{(2)}) \\
z_2^{(2)} &= \theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3 ; \text{ sledi } a_2^{(2)} = g(z_2^{(2)}) \\
z_3^{(2)} &= \theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3 ; \text{ sledi } a_3^{(2)} = g(z_3^{(2)})
\end{aligned}
\tag{2.5}
$$



Slika 2.7. Slojevi neuronske mreže

Ove tri linearne kombinacije odgovaraju množenju matrice i vektora $\theta^{(1)}x$. Ovime se vektorizuje izračunavanje neuronske mreže. Definiše se vektor atributa x (x_0 je jedinično uvećanje) i trodimenzionalni vektor z koji sadrži z vrednosti.

$$
x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}
\tag{2.6}
$$

Nakon toga, vektorizuje se izračunavanje vrednosti $a_1^{(2)}$, $a_2^{(2)}$, $a_3^{(2)}$, tako što se na vektor z primeni funkcija aktivacije g:

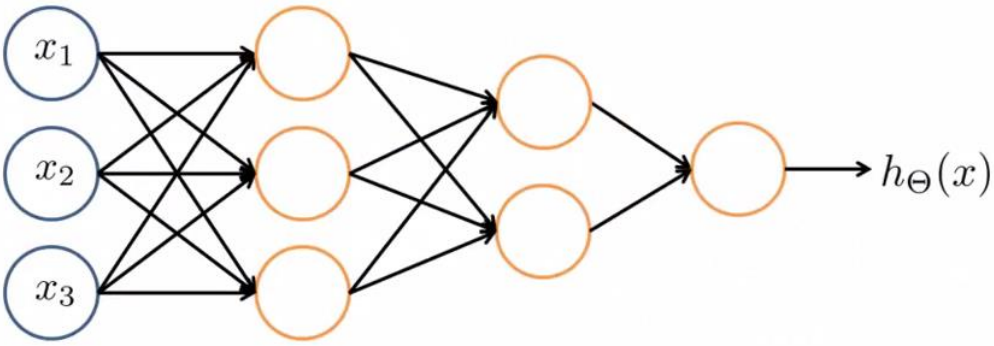
$$
\begin{aligned}
&z^{(2)} = \theta^{(1)} x \\
&a^{(2)} = g(z^{(2)}) \text{, gde su } z^{(2)} \text{ i } a^{(2)} \in \Re^3
\end{aligned}
\tag{2.7}
$$

Sigmoidna funkcija g se primenjuje na svaki element ponaosob. Radi konzistentnosti notacije, umesto da se ulazni sloj obeležava sa x, obeležava se sa $a^{(1)} = x$ i predstavlja aktivaciju prvog sloja. Posledično, $z^{(2)} = \theta^{(1)}x$ postaje $z^{(2)} = \theta^{(1)}a$. Nakon izračunavanja $a^{(2)}$ u formuli 2.4, dodaje se jedinično uvećanje $a_0^{(2)} = 1$. Na ovaj način se vektor a iz $\Re^3$ slika u vektor iz $\Re^4$.

Izraz $\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)}$ je u stvari $z^{(3)}$

$$
\begin{aligned}
&z^{(3)} = \theta^{(2)} a^{(2)} \\
&h_\theta(x) = a^{(3)} = g(z^{(3)})
\end{aligned}
\tag{2.8}
$$

Kada se na vektor *z* primeni funkcija aktivacije g, dobija se vrednost $a^{(3)}$ (*formula 2.8*). Na osnovu ulaza, vrši se predviđanje izlaza $a^{(3)}$ tj. $h_\theta(x)$. Razlika je u tome što, umesto da se koriste samo ulazni atributi x_1, x_2, x_3, koriste se i uvedeni atributi $a_1^{(2)}$, $a_2^{(2)}$, $a_3^{(2)}$. Ovi novi atributi, su naučeni i predstavljaju rezultat funkcije nad ulazom, tj. funkcije koja preslikava sloj 1 u sloj 2, koju određuje drugi skup parametara $\theta^{(1)}$. Od slučaja do slučaja, koriste se različite arhitekture, tj. načini povezivanja neurona u veštačkim neuronskim mrežama. Različite arhitekture (*slika 2.8.*) imaju svoje uloge u različitim domenima i primenama mreža. Neke su više ili manje efikasne u rešavanju konkretnih problema.



Slika 2.8. Primer mreže sa dva unutrašnja sloja

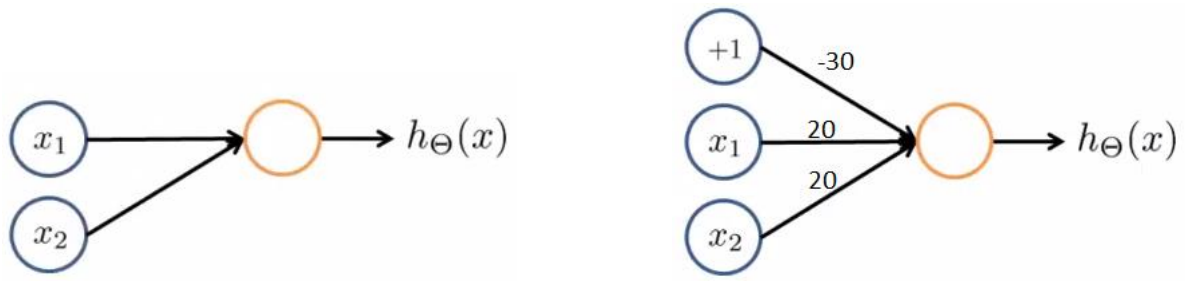
U daljem tekstu dati su primeri nekih elementarnih logičkih funkcija koje su simulirane od strane neuronskih mreža.

Primer 1. Logička *AND* funkcija:

$x_1, x_2 \in \{0,1\}$;

$y = x_1 \; AND \; x_2$;

Neuronska mreža koja simulira *AND* funkciju se sastoji od jednog neurona. Pre bilo kakvog izračunavanja dodaje se jedinično uvećanje. Nakon toga, dodaju se vrednosti težina (-30, 20, 20) tj. parametri θ. Tada je $h_\theta(x) = g(-30 + 20x_1 + 20x_2)$ i $\theta = [-30, 20, 20]$. Vrednosti θ predstavljaju samo jedno od više mogućih rešenja. Ovako postavljena neuronska mreža, za sve moguće ulaze x_1 i x_2 ima vrednosti navedene u tabeli 2.1.



Slika 2.9. *AND* funkcija

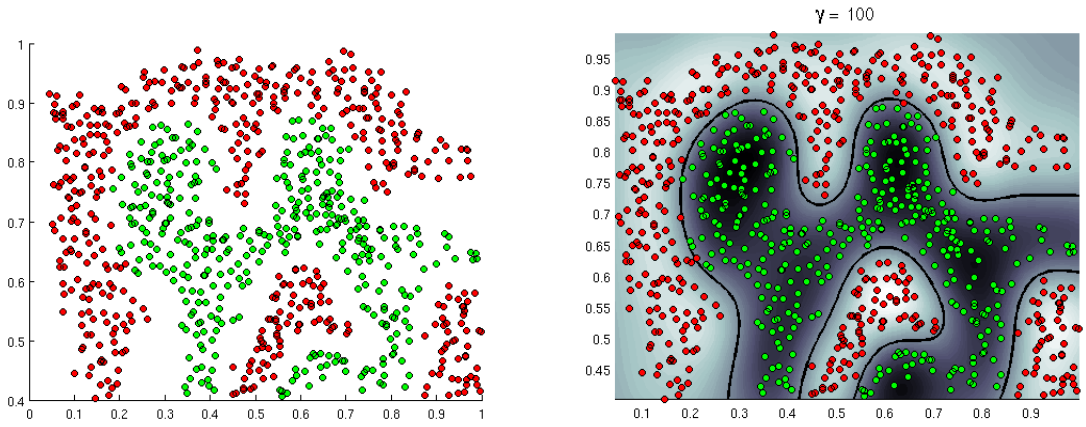
Tabela 2.1. Vrednosti modela *AND* funkcije u zavisnosti od ulaza

x_1	x_2	$h_\theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Ovo je upravo logička *AND* funkcija; $h_\theta(x)$ će biti 1 ukoliko su x_1 i x_2 jednaki 1. Analogno, *OR* funkcija bi se simulirala neuronskom mrežom sa težinskim koeficijentima (-10, 20, 20), a negacija sa (10, -20) (*formula* 2.9).

$$\begin{aligned} \theta &= [-10, 20, 20] \;\; ; \;\; h_\theta(x) = g(-10 + 20x_1 + 20x_2) \\ \theta &= [10, -20] \qquad ; \;\; h_\theta(x) = g(10 - 20x_1) \end{aligned} \tag{2.9}$$

U nekim slučajevima, prilikom primene nekog od algoritama istraživanja podataka, moguće je dobiti linearno neseparabilne podatke. U tom slučaju podaci nisu linearno separabilni (*slika* 2.10. a)). Za ovakve podatke, moguće je koristiti polinomijalnu logističku regresiju dovoljno velikog stepena. Ovakav pristup je dobar kada postoji mali broj (npr. 2) atributa. Realna situacija je da postoji veći broj atributa. U tom slučaju logistička regresija nije zgodna. Na primer, za 100 atributa, kada se koriste samo izrazi drugog reda, pri logističkoj regresiji se dobija oko 5000 atributa, što može dovesti do preprilagođavanja i većih zahteva za obradom. Broj kvadratnih atributa (izrazi drugog reda oblika θx^2, $\theta x^2 y$, $\theta x y^2$ itd...) u zavisnosti od početnog broja atributa n, asimptotski raste $O(n^2)$.



(a) (b)

Slika 2.10.
(a) Linearno neseparabilni podaci (b) Preprilagođavanje

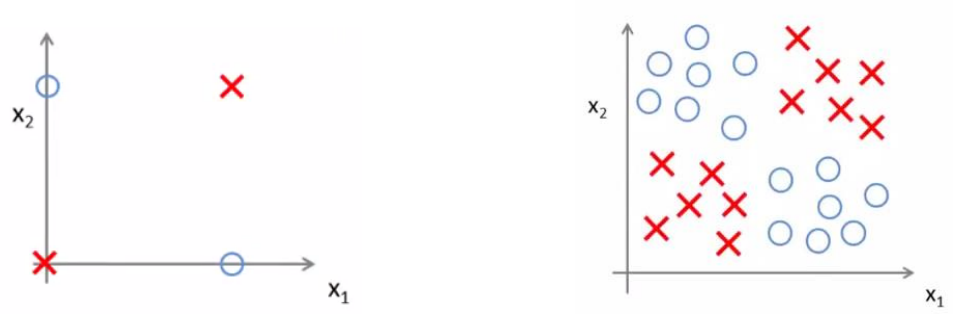
Pri korišćenju modela logističke regresije sa polinomom dovoljno velikog stepena dobijaju se ispravno klasifikovani podaci ali model je preprilagođen podacima, što prouzrokuje veću stopu grešaka prilikom klasifikacije novih neobeleženih podataka. (*slika* 2.10. b)).

Jednostavna logistička regresija, sa kvadratnim ili kubnim atributima nije dobar način za učenje složenih nelinearnih modela kada je *n* veliko. Kako bi se efikasno rešio problem klasifikacije podataka koji nisu linearno separabilni, koriste se neuronske mreže. Podaci koji se obrađuju iz realnog sveta mogu biti dosta složeni i mogu da sadrže dosta atributa koji ih opisuju. Iz tog razloga, neuronske mreže se najčešće koriste nad takvim podacima.

Primer 2. Nelinearna klasifikacija (*XNOR* funkcija):

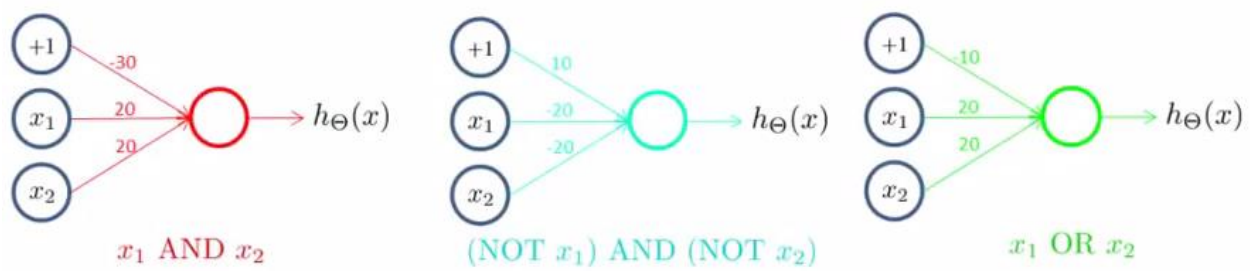
$x_1, x_2 \in \{0,1\}$;

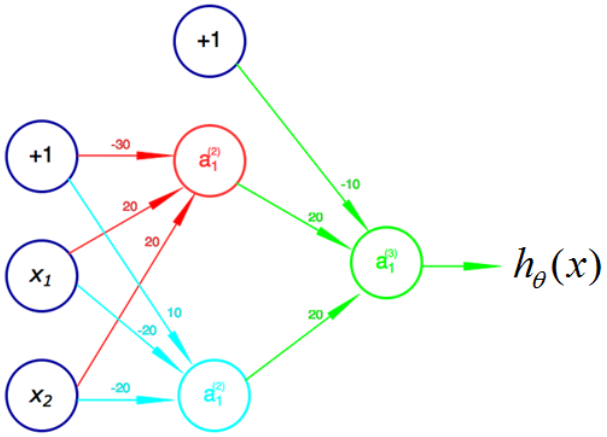
$y = x_1 \ XNOR \ x_2$



Slika 2.11. Linearno neseparabilni podaci

Potrebno je naučiti nelinearnu granicu odlučivanja koja deli pozitivne od negativnih primeraka. Pozitivni slučajevi, u slučaju *XNOR*-a, su obeleženi sa x (y=1) a negativni sa o (y=0).





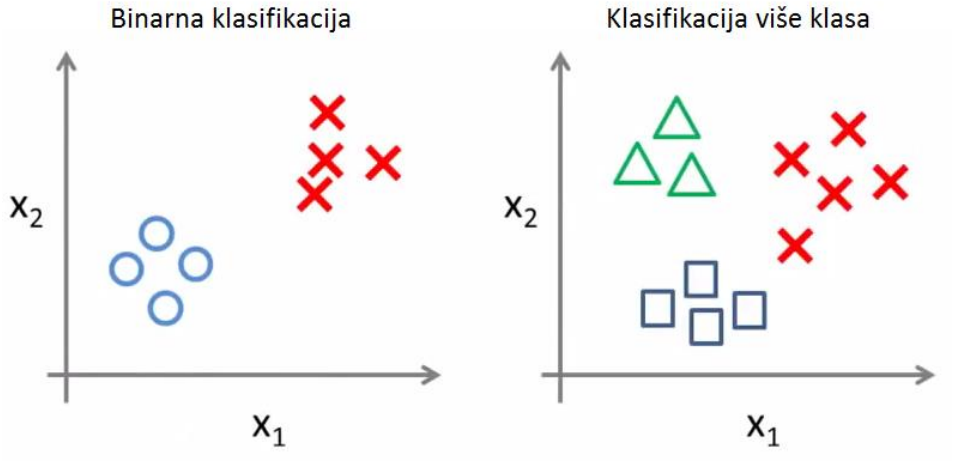
Slika 2.12. *XNOR* funkcija

Neuronska mreža koja simulira složeniju funkciju *XNOR*, sastavljena je od tri jednostavnije mreže koje simuliraju *AND, OR* i mreže koja je kombinacija osnovnih logičkih funkcija *NOT* i *AND*. U ovakvoj kombinaciji mreža, *AND* je potrebna kako bi se dobilo 1 kad su oba ulaza 1. Kombinacija *NOT* i *AND* je potrebna kako bi se dobilo 1 kad su oba ulaza 0. Na kraju, mreža *OR* objedinjuje ova dva rešenja kako bi se dobilo 1 ukoliko je izlaz bilo koje prve dve funkcije 1.

Tabela 2.2. Vrednosti modela *XNOR* funkcije u zavisnosti od ulaza

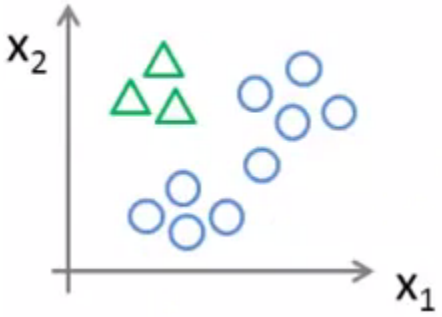
x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\theta(x)$
1)	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	0	0	1

Uz ovu neuronsku mrežu, koja ima jedan ulazni, jedan skriveni i jedan izlazni sloj, dobijamo nelinearnu granicu odluke koja simulira *XNOR* funkciju. Ovaj primer prikazuje kako neuronske mreže mogu da se nose sa složenim funkcijama. Što se više napreduje u propagaciji, to se povećava složenost, tj. mogućnost simuliranja složenih funkcija. Kada postoji više klasa pri klasifikaciji, za razliku od prošlih primera gde u izlaznom sloju postoji samo jedan neuron, potrebno je primeniti tehniku *jedan protiv svih*.



Slika 2.13. Oblici klasifikacije
(a) Binarna klasifikacija (b) Klasifikacija više klasa

Problem klasifikacije više klasa se svodi na binarnu klasifikaciju, tako što se napravi više zasebnih binarnih klasifikatora. U konkretnom primeru (*slika* 2.13. b)) gde postoje tri klase, prave se tri binarna klasifikatora.



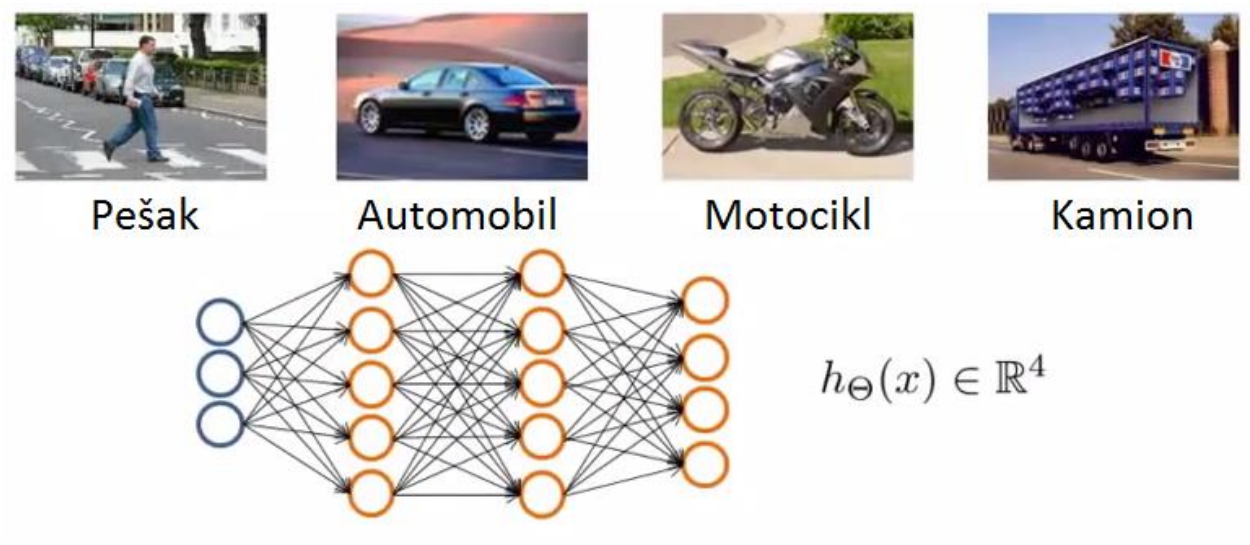
Slika 2.14. Jedna iteracija metode jedan protiv svih

U svakoj od ove tri binarne klasifikacije imamo ciljanu pozitivnu klasu i ostale klase koje su spojene u jednu negativnu klasu. Ostalo je da se izračuna verovatnoća za sve klase *i* tako da $i = y$, gde je *y* labela, tj. klasa kojoj pripadaju konkretni ulazni podaci iz trening skupa a *i* je jedna od pozitivnih klasa iz jedne od binarnih klasifikacija za koju se tvrdi da joj pripadaju ulazni podaci. Sa novim ulazom *x*, bira se klasa *i* takva da maksimizuje vrednost h_θ (*formula* 2.10), tj. klasifikator koji daje najbolje rezultate prilikom klasifikacije.

$$\max_i h_\theta^{(i)}(x) \tag{2.10}$$

Primer:

Na ulazu su slike pešaka, automobila, motocikla i kamiona. Potrebno je klasifikovati ih ispravno. Za razliku od prethodnih primera, model $h_\theta(x)$, tj. izlaz, više nije realan broj već postaje vektor. Neuronska mreža koja implementira ovakvu klasifikaciju, deluje kao da se sastoji od četiri logistička klasifikatora, od kojih svaki pokušava da svrsta ulaz u jednu od više klasa za koju je zadužen.



Slika 2.15. Klasifikacija više klasa

Neka je dat trening skup: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})$ gde su $x^{(i)}$ slike *i*. Potrebno je za svako $x^{(i)}$ da važi $h_\theta(x^{(i)}) \approx y^{(i)}$, gde su $h_\theta(x)$, $y^{(i)} \in \Re^4$.

$$h_\theta(x) \text{ tj. } y^{(i)} \approx \underbrace{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}}_{\text{pešak}} \quad y^{(i)} \approx \underbrace{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}}_{\text{automobil}} \quad y^{(i)} \approx \underbrace{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}_{\text{motocikl}} \quad y^{(i)} \approx \underbrace{\begin{bmatrix}0\\0\\0\\1\end{bmatrix}}_{\text{kamion}} \quad i = \overline{1,m} \tag{2.11}$$

2.1. Funkcija cene

Funkcija cene je funkcija kojom se određuje preciznost modela. Cilj modela je minimizovanje funkcije cene, tj. minimizovanje razlike između rezultata modela $h_\theta(x^{(i)})$ i realnih vrednosti $y^{(i)}$. Za prostu linearnu regresiju od dva parametra θ_1, θ_2, funkcija cene bi glasila:

$$J(\theta_1,\theta_2)=\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2 \tag{2.1.1}$$

Metoda zasnovana na minimizaciji funkcije iz formule 2.1.1 se zove metoda najmanjih kvadrata (*eng. Squared error function*). Vrednost funkcije cene se minimizuje u zavisnosti od parametara θ, uz pomoć neke metode. Ta metoda, za neuronske mreže je obično metoda *gradijentnog spusta***,** mada se koriste i druge metode. Više informacija o gradijentnom spustu se može naći u [5]. Kao primer funkcije cene uzima se funkcija cene u linearnoj regresiji:

$$J(\theta)=\frac{1}{m}\sum_{i=1}^{m}(\frac{1}{2}h_\theta(x^{(i)})-y^{(i)})^2 \tag{2.1.2}$$

Ako se uvede oznaka Cost:

$$Cost(h_\theta(x),y)=\frac{1}{2}(h_\theta(x)-y)^2 \tag{2.1.3}$$

tada funkcija cene postaje:

$$J(\theta)=\frac{1}{m}\sum_{i=1}^{m}Cost(h_\theta(x^{(i)}),y^{(i)}) \tag{2.1.4}$$

Menja se definicija *Cost* funkcije tako da odgovara za neuronske mreže (*formula* 2.1.5).

$$Cost(h_\theta(x),y)=\begin{cases}-\log(h_\theta(x)) & \text{ako y} = 1\\ -\log(1-h_\theta(x)) & \text{ako y} = 0\end{cases} \tag{2.1.5}$$

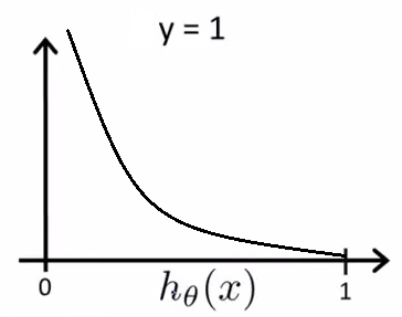
Skraćeni zapis ove funkcije je:

$$Cost(h_\theta(x),y)=-y\log(h_\theta(x))-(1-y)\log(1-h_\theta(x))$$

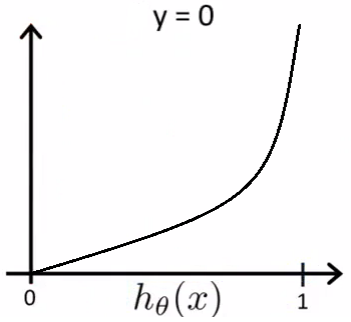
Na taj način se dobija funkcija cene kod izračunavanja neuronskih mreža:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right] \quad (2.1.6)$$

Ovakva funkcija cene je najrasprostranjenija kada su u pitanju neuronske mreže. U nastavku teksta je objašnjeno zašto se pomoćna *Cost* funkcija ovako definiše.



Slika 2.1.1. Funkcija cene za $y = 1$



Slika 2.1.2. Funkcija cene za $y = 0$

Ako je $y = 1$ i $h_\theta(x) = 1$ tada je *Cost* = 0; znači da je algoritam tačno izvršio klasifikaciju. Kako $h_\theta(x)$ teži nuli ($h_\theta(x) \rightarrow 0$), *Cost* teži beskonačno (*Cost* $\rightarrow \infty$); znači da je algoritam pogrešno izvršio klasifikaciju. U tom slučaju algoritam se kažnjava sa velikom cenom, tj. *Cost* $\rightarrow \infty$; analogno, kada je $y = 0$, funkcija cene teži beskonačno kako $h_\theta(x)$ teži jedinici, a nuli kada $h_\theta(x)$ teži nuli.

Ako bi se za primer posmatrala mreža sa slike (*slika 2.15.*), njene karakteristike su:

L = ukupan broj slojeva u mreži
S_l = broj neurona (ne računajući jedinicu uvećanja) u sloju l
$S_1 = 3$; $S_2 = 5$; $S_3 = 5$; $S_4 = 4$
$K = 4$ je broj neurona u sloju izlaza

S obzirom da je četvrti sloj ujedino i sloj izlaza koji ima četiri neurona, u pitanju je klasifikacija više klasa. Potrebno je da se minimizuje $J(\theta)$ gde je:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right] \quad (2.1.6)$$

Na kraju, funkciji cene neuronske mreže se dodaje sumacija K izlaznih jedinica. Detaljnije o funkcijama cene se može naći u [5]. Krajnji izgled funkcije cene:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-h_\theta(x^{(i)}))_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_l+1}(\theta_{ji}^{(l)})^2 \quad (2.1.7)$$

Gde je $h_\theta(x) \in \Re^K$ i $(h_\theta(x))_i = i$ - ti izlaz (i-ti element vektora $h(x)$). Izraz koji se nalazi na kraju formule 2.1.7 se naziva regularizacioni izraz i koristi se kao pouzdana metoda kažnjavanja modela, korišćena za pospešavanje generalizacije i sprečavanje preprilagođavanja modela usled preterane složenosti.

$$\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_l+1}(\theta_{ji}^{(l)})^2 \quad (2.1.8)$$

Više o regularizaciji se može naći u [5].

2.2. Algoritam propagacije unazad

Algoritam propagacije unazad računa optimalne vrednosti θ kako bi se minimizovala funkcija cene $J(\theta)$:

$$\min_{\theta} J(\theta) \tag{2.2.1}$$

U algoritmu propagacije unazad, pored funkcije cene $J(\theta)$ se izračunavaju i parcijalni izvodi. Funkcija cene se računa po formuli *2.1.7*. Parcijalni izvodi se koriste u metodi gradijentnog spusta za iterativno ažuriranje vektora parametra θ *(formula 2.2.2)*. Zahvaljujući parcijalnim izvodima, algoritam je svakom iteracijom bliži globalnom minimumu, ukoliko je funkcija cene ispravno definisana. Stopa ili korak učenja α, kontroliše brzinu odvijanja gradijentnog spusta. Odabir optimalne vrednosti ovog parametra je veoma bitan. Ukoliko je ova vrednost prevelika, moguće je prekoračiti minimum (divergirati, tj. udaljavati od minimuma). Kod premale vrednosti, postojaće više iteracija gradijentnog spusta a samim tim dužina celog procesa treniranja će se povećati.

$$\theta = \theta - \alpha \frac{\partial}{\partial \theta} J(\theta) \tag{2.2.2}$$

Primer:

Za dati trening primerak (x, y), prikazan je niz izračunavanja koji se primenjuje nad njim. Inicijalno se radi propagacija unapred, kako bi se izračunala vrednost modela u zavisnosti od ulaza:

$$\begin{aligned}
a_0^{(2)} &= 1 \\
a_0^{(3)} &= 1 \\
a^{(1)} &= x \\
z^{(2)} &= \theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \\
z^{(3)} &= \theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \\
z^{(4)} &= \theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\theta(x) = g(z^{(4)})
\end{aligned} \tag{2.2.3}$$

U koraku gde se računaju vrednosti funkcija aktivacije za drugi $a^{(2)}$ i treći $a^{(3)}$ sloj, dodaju se jedinice uvećanja $a_0^{(2)}$ i $a_0^{(3)}$. Ovo je vektorizovana implementacija propagacije unapred. Ona omogućava efikasno računanje vrednosti aktivacija kroz slojeve. Vektorizovanost implementacije je odlika koja je veoma bitna prilikom ovakvih vrsta izračunavanja jer omogućava da se izračunavanje više vrednosti izvrši prostim množenjem matrica ili vektora i

matrica, umesto da se iterativnom metodom računa jedna po jedna vrednost. Nakon što je dobijena inicijalna vrednost modela, radi se propagacija unazad. Za svaki čvor, tj. neuron j u sloju l, računa se faktor greške $\delta_j^{(l)}$. Ovaj faktor greške se odnosi na tačnost vrednosti funkcije aktivacije.

U slučaju da neuronska mreža ima npr. četiri sloja (L = 4), za svaki neuron u sloju izlaza računa se faktor greške:

$$\delta_j^{(4)} = a_j^{(4)} - y_j \tag{2.2.4}$$

gde se $a_j^{(4)}$ može zapisati i kao $(h_\theta(x))_j$. Faktor greške je odstupanje predviđene vrednosti a_j od stvarne vrednosti y_j. Vektorizovan oblik je:

$$\delta^{(4)} = a^{(4)} - y \tag{2.2.5}$$

gde su δ, a i y vektori istih dimenzija koji je jednak broju izlaznih jedinica, tj. neurona. Nakon izračunavanja greške u poslednjem sloju (*formula* 2.2.5), računaju se greške u prethodnim slojevima, sve do početnog sloja:

$$\begin{aligned} \delta^{(3)} &= (\theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)}) \\ \delta^{(2)} &= (\theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)}) \end{aligned} \tag{2.2.6}$$

Iz ovog razloga se ceo proces računanja grešaka zove propagacija unazad. Operacija .* nije množenje matrica, već pojedinačno množenje odgovarajućih elemenata matrice. Izvod g' je izvod funkcije aktivacije evaluirane za ulazne vrednosti date $z^{(3)}$:

$$\begin{aligned} g'(z^{(3)}) &= a^{(3)} .* (1 - a^{(3)}) \\ g'(z^{(2)}) &= a^{(2)} .* (1 - a^{(2)}) \end{aligned} \tag{2.2.7}$$

Gde je $a^{(i)}$ vektor aktivacije, a 1 vektor jedinica. Izraz $\delta^{(1)}$ ne postoji, jer u prvom sloju su ulazni podaci, te nema smisla računati grešku za njih. Moguće je dokazati, kroz složen dokaz, uz ignorisanje regularizacije λ (strana 22.), da za λ = 0 važi:

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \delta_i^{(l+1)} \tag{2.2.8}$$

Uz propagaciju unazad, tj. računanje δ izraza, mogu se brzo računati parcijalni izvodi za sve parametre uz pomoć formule 2.2.8.

Pseudokod algoritma propagacija unazad:

Dato je *m* trening primeraka { $(x^{(1)}, y^{(1)}),...,(x^{(i)}, y^{(i)}),...,(x^{(m)}, y^{(m)})$ }. Postavlja se $\Delta_{ij}^{(l)} = 0$ za svako *l, i, j*. Ovi parametri će se koristiti kao akumulatori (kumulativno će se dodavati vrednosti za izračunavanje parcijalnih izvoda).

begin.

for i= 1 to m

{

$a^{(1)} = x^{(i)}$

izračunati $a^{(l)}$ koristeći propagaciju unapred za $l = 2,3,\ldots,L$

koristeći $y^{(i)}$, izračunati $\delta^{(L)} = a^{(L)} - y^{(i)}$

izračunati $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

}

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)} \qquad if \;\; j \neq 0$

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \qquad if \;\; j = 0$

end.

Ako se Δ_{ij} posmatra kao matrica, izraz $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ je moguće vektorisati, te se dobija $\Delta^{(l)} := \Delta^{(l)} + a^{(l)} \delta^{(l+1)}$. Van iteracije su poslednja dva reda. U poslednjem redu, kada je $j = 0$, izostavljen je izraz regularizacije, jer je u pitanju jedinično uvećanje. Uz pomoć još složenih dokaza koji su izostavljeni, dolazi se do formule:

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)} \tag{2.2.9}$$

Ova formula dalje može da se koristi u gradijentnom spustu ili u nekom drugom naprednom algoritmu optimizacije. Kako bi koristili napredne algoritme optimizacije, potrebno je uraditi tzv. razmotavanje parametara. Razmotavanje parametara nije ništa drugo do transformacije podataka koji su u obliku matrice u oblik vektora. Prednost matrične reprezentacije je u tome što takva reprezentacija odgovara algoritmima propagacije unapred i propagacije unazad. Lakše je da se iskoristi vektorizovana implementacija, kada su parametri u obliku matrica. Sa druge strane, prednost vektorske reprezentacije je u tome što mogu da se koriste napredni algoritmi optimizacije, jer ti algoritmi pretpostavljaju da su svi parametri razmotani u jedan veliki dugački vektor.

Primer procesa razmotavanja i vraćanja u matrice pri učenju neuronske mreže:

Funkcija cene *costFunction* kao argumente ima parametre *theta*, a vraća funkciju cene *jVal* i izvode *gradient*. Parametri se prosleđuju naprednom algoritmu optimizacije *fminunc*, koji vraća optimalne vrednosti težinskih parametra. Obe rutine podrazumevaju da su parametri *theta* i *initialTheta* vektori $\Re^n$. Takođe se podrazumeva da će funkcija cene *costFunction*, kao drugu povratnu vrednost vratiti izvode u obliku vektora $\Re^{n+1}$. (primeri koda su u Octave jeziku)

```
function [jVal, gradient] = costFunction(theta);
optTheta = fminunc(@costFunction, initialTheta, options);
```

Kod neuronske mreže od tri sloja $s_1 = 10,\ s_2 = 10,\ s_3 = 1$, gde su dati parametri:

Matrice: $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$ (Theta1, Theta2, Theta3) $\quad \theta^{(1)} \in \Re^{10\times 11}, \theta^{(2)} \in \Re^{10\times 11}, \theta^{(3)} \in \Re^{1x11}$

Matrice: $D^{(1)}, D^{(2)}, D^{(3)}$ (D1, D2, D3) $\quad D^{(1)} \in \Re^{10\times 11}, D^{(2)} \in \Re^{10\times 11}, D^{(3)} \in \Re^{1x11}$

konverzija između matrica i vektora, vrši se na sledeći način:

1) Matrica u vektor (koristi se operator (:) koji od matrice pravi vektor, dok operator ; vrši konkatenaciju):

```
thetaVec = [ Theta1(:); Theta2(:); Theta3(:)];
DVec     = [D1(:); D2(:); D3(:)];
```

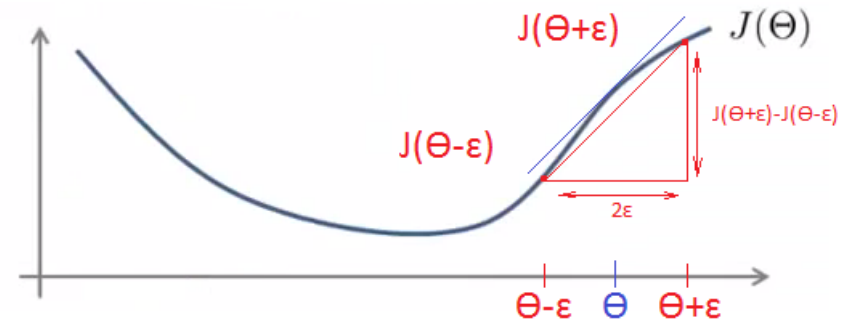
2) Vektor u matricu (koristi se funkcija *reshape* zajedno sa ulaznim vektorom i parametrima koji definišu veličinu matrice):

```
Theta1 = reshape(thetaVec(1:110),10,11);
Theta2 = reshape(thetaVec(111:220),10,11);
Theta3 = reshape(thetaVec(221:231),1,11);
```

2.3. Numerička provera gradijenta

Ponekad je moguća pojava suptilnih grešaka prilikom propagacije unazad. Kada se pokrene gradijentni spust ili neki drugi optimizacioni algoritam, deluje kao da sve radi kako treba i da se funkcija cene smanjuje svakom iteracijom gradijentnog spusta. Ono što je teško uočiti ovim putem, je da neuronska mreža ima veću stopu grešaka, nego što bi imala da implementacija nema neku suptilnu grešku. Iz ovog razloga, uvodi se algoritam provere gradijenta. Uz proveru gradijenta, možemo biti sigurni da su implementacije propagacije unapred i unazad 100% korektne.

Pretpostavimo da postoji neka funkcija $J(\theta)$ i da postoji vrednost θ, tako da $\theta \in \Re$. Potrebno je proceniti vrednost izvoda u tački θ. Izvod je jednak nagibu tangente u toj tački. Procedura za numeričko izračunavanje izvoda u tački θ se računa na sledeći način:



Slika 2.3.1. Izvod sa dvostranom razlikom

Uoče se tačke $\theta+\varepsilon$ i $\theta-\varepsilon$ u okolini tačke θ. Vrednosti funkcije u ove dve tačke se spoje pravom linijom. Nagib ovako dobijene linije je aproksimacija nagiba krive u tački θ. Vrednost nagiba aproksimacije dobija se tako što se podeli visina sa širinom trougla:

$$\frac{\partial}{\partial\theta}J(\theta) \approx \frac{J(\theta+\varepsilon)-J(\theta-\varepsilon)}{2\varepsilon} \tag{2.3.1}$$

Za ε se otprilike uzima vrednost $\varepsilon = 10^{-4}$. Što je ε manje to je aproksimacija bolja i u teoriji kada bi pustili da ε teži nuli, onda bi umesto simbola približno mogao stati simbol jednakosti. Doduše, u praksi, ne sme se staviti previše mala vrednost za ε, jer je moguć nastanak numeričkih problema. Ovakva formula izvoda koristi dvostranu razliku (*slika 2.3.1.*), gde se uzimaju $J(\theta+\varepsilon)$ i $J(\theta-\varepsilon)$. Preciznija je od jednostrane razlike koja se inače češće koristi u definicijama infinitezimalnog računa, gde se uzimaju $J(\theta+\varepsilon)$ i $J(\theta)$.

U Octave jeziku, implementacija provere gradijenta izgleda ovako:

```
gradApprox = (J (theta + EPSILON) - J (theta - EPSILON)) / (2 * EPSILON)
```

Ovo pruža procenu gradijenta u toj tački. U domenu neuronskih mreža, θ nije više realan broj, već matrica $\theta \in \Re^n$. $\theta = [\theta_1, \theta_2, \theta_3, \ldots, \theta_n]$ (θ je razmotana verzija $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}, \ldots, \theta^{(n)}$). Koristeći sličnu ideju, mogu se aproksimirati svi parcijalni izvodi (*formula 2.3.2*).

$$\begin{aligned}
&\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1 + \varepsilon, \theta_2, \theta_3, \ldots, \theta_n) - J(\theta_1 - \varepsilon, \theta_2, \theta_3, \ldots, \theta_n)}{2\varepsilon} \\
&\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2 + \varepsilon, \theta_3, \ldots, \theta_n) - J(\theta_1, \theta_2 - \varepsilon, \theta_3, \ldots, \theta_n)}{2\varepsilon} \\
&\vdots \\
&\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n + \varepsilon) - J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n - \varepsilon)}{2\varepsilon}
\end{aligned} \tag{2.3.2}$$

U tom slučaju, Octave kod bi izgledao ovako:

```
for i = 1:n,
  thetaPlus = theta;
  thetaPlus(i) = thetaPlus(i) + EPSILON;
  thetaMinus = theta;
  thetaMinus(i) = thetaMinus(i) - EPSILON;
  gradApprox(i) = (J(thetaPlus) - J(thetaMinus)) / (2 * EPSILON)
end;
```

Na kraju, dobijamo numerički određene parcijalne izvode funkcije cene u odnosu na sve parametre u mreži. Nakon toga poredimo *gradApprox* sa gradijentom koji se dobija kao rezultat rada propagacije unazad *DVec*. Ove dve vrednosti moraju da budu približno jednake, *gradApprox* ≈ *DVec*. Ukoliko vrednosti jesu dovoljno slične, onda se može zaključiti da implementacija propagacije unazad dobro radi.

2.4. Nasumična inicijalizacija

Kod algoritma gradijentnog spusta ili kod nekih drugih naprednih algoritama za optimizaciju, potrebno je inicijalizovati vrednosti parametara θ, kako bi te vrednosti dalje prosledili funkciji *fminunc*:

```
optTheta = fminunc(@costFunction, initialTheta, options)
```

Prilikom gradijentnog spusta, inicijalizujemo θ na nule, tj. $\theta_{ij}^{(l)} = 0$ za svako i, j, l.

$$initialTheta = zeros(n, 1) \tag{2.4.1}$$

U ovom slučaju, jedinice u istom unutrašnjem, tj. skrivenom sloju računaju iste funkcije nad ulaznim podacima. Za svaki uzorak iz trening skupa, važiće $a_1^{(l)} = a_2^{(l)} = \ldots = a_n^{(l)}$. Isto važi i za vrednosti koje se dobijaju propagacijom unazad $\delta_1^{(l)} = \delta_2^{(l)} = \ldots = \delta_n^{(l)}$, kao i za parcijalne izvode:

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = \frac{\partial}{\partial \theta_{ij+1}^{(l)}} J(\theta) \tag{2.4.2}$$

Posledica ovakve inicijalizacije parametara je da su ove vrednosti, posle jedne iteracije gradijentnog spusta i dalje jednake. Dok je ovo ispravno za logističku regresiju, inicijalizacija svih parametara na nule ne fukncioniše kad se trenira neuronska mreža. Kako bi se izbeglo ovakvo ponašanje mreže, potrebno je nasumično inicijalizovati parametre. Ovaj problem se zove problem simetričnih težina a nasumična inicijalizacija oblik razbijanja simetrije.

Svako $\theta_{ij}^{(l)}$ se inicijalizuje na nasumičnu vrednost iz nekog intervala $\theta_{ij}^{(l)} \in [-\varepsilon, \varepsilon]$, tj. $(-\varepsilon \leq \theta_{ij}^{(l)} \leq \varepsilon)$.

Primer u Octave jeziku:

```
Theta1 = rand(10, 11) * (2 * INIT_EPSILON) - INIT_EPSILON;
Theta1 = rand(1, 11)  * (2 * INIT_EPSILON) - INIT_EPSILON;
```

Funkcija rand generiše matricu veličine određene argumentima, čiji su elementi iz intervala (0,1). Kako bi interval bio [-ε, ε], potrebno je pomnožiti dobijenu vrednost sa dvostrukom vrednošću ε i pritom oduzeti vrednost ε. Obično se za ε uzimaju male vrednosti, blizu nule. Nakon ovog koraka mogu da se primenjuju propagacija unazad, provera gradijenta, gradijentni spust ili neki drugi optimizacioni algoritmi za minimizaciju funkcije cene koji uzimaju ove nasumične vrednosti parametara kao inicijalne.

3. Tipovi neuronskih mreža

Veštačke neuronske mreže mogu biti hardverski (neuroni predstavljeni kroz fizičke komponente) ili softverski bazirane (računarski modeli) i mogu da koriste širok dijapazon topologija i algoritama učenja. Veštačke neuronske mreže se u najopštijem smislu mogu podeliti na mreže prosleđivanja unapred, rekurentne mreže i na njihove hibride.

3.1. Mreža prosleđivanja unapred

Mreža prosleđivanja unapred (*eng. Feedforward neural network*) je bila prva, a ujedino i najjednostavnija neuronska mreža. U ovoj mreži, protok informacija je jednosmeran; od ulaznih jedinica, podaci prolaze kroz skrivene jedinice (ukoliko postoje) sve do izlaznih jedinica. Ne postoje ciklusi u mreži za razliku od rekurentnih neuronskih mreža.

Najjednostavnija vrsta neuronske mreže je jednoslojna mreža perceptrona, koja se sastoji samo od jednog sloja izlaznih jedinica; ulazi se pohranjuju direktno u izlaz preko niza težina. Suma proizvoda težina i ulaza se izračunava u svakoj jedinici i ukoliko je vrednost iznad nekog praga (tipično 0), neuron ispaljuje signal i uzima aktiviranu vrednost (tipično 1); u suprotnom uzima deaktiviranu vrednost (tipično -1). Neuroni sa ovakvom vrstom funkcije aktivacije se još zovu i veštački neuroni ili linearne jedinice praga (*eng. Linear threshold units*). U literaturi termin perceptron često referiše mreže koje sadrže samo jednu ovakvu jedinicu. Sličan neuron je opisan od stane Vorena Mekkuloha i Voltera Pitsa četrdesetih godina prošlog veka.

Perceptron može da se napravi koristeći bilo koje vrednosti za stanja aktivacije i deaktivacije sve dok se vrednost praga nalazi između te dve vrednosti. Većina perceptrona ima izlaze 1 ili -1 sa pragom 0. Postoje dokazi da takve mreže mogu brže da se treniraju za razliku od mreža sa drugim vrednostima. Perceptroni mogu da se treniraju uz jednostavan algoritam učenja, koji se obično naziva delta pravilo. Algoritam računa grešku između sračunatog izlaza i primeraka izlaznih podataka; koristeći tu vrednost, može ispraviti vrednosti težina, tako implementirajući formu gradijentnog spusta.

Jedinični perceptroni (mreža perceptrona od jedne jedinice) su sposobni da uče samo linearno separabilne šablone. Godine 1969. poznata monografija nazvana *Perceptroni*, autora Marvina Minskog (Marvin Minsky) i Sejmora Paperta (Seymour Papert), pokazala je da je nemoguće da jedinični perceptron nauči *XOR* funkciju. Često se (pogrešno) veruje da su pretpostavili da je sličan rezultat tačan i za višeslojne perceptrone. Doduše, to nije bila istina, s obzirom na to, da su i Minski i Papert znali da su višeslojni perceptroni bili u mogućnosti da simuliraju *XOR* funkciju.

Iako je jedinični perceptron ograničen što se njegove računarske moći tiče, pokazano je da mreže paralelnih jediničnih perceptrona mogu da aproksimiraju bilo koju neprekidnu

funkciju, iz nekog sabijenog intervala realnih brojeva, u interval [-1, 1]. Višeslojna mreža može da sračuna neprekidni izlaz umesto funkcije stepenika (*eng. step function*). Obično se za potrebe aproksimacije koristi logistička, tj. sigmoidna funkcija (*formula 2.2*). Prednost ove funkcije je u tome, što je neprekidno diferencijabilna i što se izvodi lako računaju.

Višeslojni perceptroni se sastoje od više slojeva jedinica, obično međusobno povezanih, tako da se informacije propagiraju isključivo unapred. Svaki neuron u konkretnom sloju ima usmerene veze ka neuronima u sledećem sloju. U mnogim slučajevima, jedinice u ovakvim mrežama primenjuju sigmoidnu funkciju kao funkciju aktivacije. Teorema univerzalne aproksimacije za neuronske mreže, tvrdi da svaka kontinualna funkcija, koja preslikava intervale realnih brojeva na neki drugi interval realnih brojeva, može da se aproksimira sa proizvoljnom preciznošću, koristeći višeslojni perceptron sa samo jednim skrivenim slojem. Višeslojne mreže koriste širok dijapazon tehnika učenja, od kojih je najpopularnija propagacija unazad. Izlazne vrednosti se porede sa tačnim vrednostima, kako bi se izračunala vrednost neke predefinisane funkcije greške. Kako se greška propagira unazad kroz mrežu, težinski faktori se ažuriraju kako bi se smanjila greška za neku malu vrednost. Nakon ponavljanja ovog procesa dovoljno puta, mreža obično konvergira ka stanju gde je greška mala. Kako bi se težine ispravno ažurirale, koristi se opšta metoda za nelinearne optimizacije nazvana gradijentni spust. Koriste se izvodi funkcije greške kako bi se težine ispravno ažurirale, samim tim smanjujući grešku. Iz tog razloga, propagacija unazad se koristi samo kod mreža čije su funkcije aktivacije diferencijabilne u svakoj tački.

Kod slučaja gde je broj trening primeraka nizak, potrebno je uvesti dodatne tehnike. Postoji opasnost od preprilagođavanja podacima i pogrešnog razumevanja statističkog procesa koji generiše podatke. Računska teorija učenja (*eng. Computational learning theory*) se bavi treniranjem klasifikatora sa ograničenom količinom podataka. U kontekstu neuronskih mreža, jednostavna heuristika nazvana rano zaustavljanje (eng. *Early stopping*), često osigurava da će mreža uspešno klasifikovati podatke koje nisu iz trening skupa.

Drugi tipični problemi algoritma propagacije unazad, su brzina konvergiranja i moguće zaustavljanje u lokalnim minimumima funkcije greške. Uz sve ove nedostatke, tehnika propagacije unazad je i dalje vodeći alat, kada je zadatak učenja neuronske mreže u pitanju.

Više o mreži prosleđivanja unapred se može naći u [5].

3.2. Radijalno zasnovane funkcije

Radijalno zasnovane funkcije, (*eng. Radial basis function*) su tehnike interpolacije u višedimenzionom prostoru. RBF je funkcija koja je ugrađena u okviru kriterijuma razdaljine u odnosu na centar. RBF funkcije su se primenjivale u oblasti neuronskih mreža, gde su se koristile kao zamena za sigmoidnu funkciju, karakterističnu u višeslojnim perceptronima. RBF mreže se obično treniraju u dva koraka. U prvom koraku, definiše se vektor centralnih jedinica c_i. Ovaj korak može da se izvrši na nekoliko načina; centralne jedinice mogu nasumično da se odaberu iz nekog skupa primeraka ili mogu da se ustanove koristeći neki algoritam klasterovanja. Drugi korak je treniranje, gde se određuju optimalne centralne jedinice i pokrivenosti za svaki neuron. Takođe se određuje trenutak kada se prestaje sa dodavanjem novih neurona u mrežu kako bi se sprečilo preprilagođavanje. Obično je obezbeđena posebna mera greške koja se nadgleda i koja diktira tok algoritma.

Kod problema regresije, izlazni sloj je linearna kombinacija vrednosti skrivenih slojeva koji predstavljaju srednju vrednost predviđenih izlaza. Interpretacija vrednosti izlaza je ista kao i kod modela regresije u statistici. Kod problema klasifikacije, izlazni sloj je obično sigmoidna funkcija nad linearnom kombinacijom vrednosti skrivenih slojeva. RBF mreže imaju prednost što ne pate od lokalnog minimuma na isti način kao višeslojni perceptroni; razlog je što su jedini parametri koji se menjaju u procesu učenja linearna preslikavanja od skrivenog sloja ka izlaznom sloju. Mana im je što je potrebna dobra pokrivenost ulaza od strane radijalnih funkcija. Iako je implementacija veoma drugačija, RBF neuronske mreže su konceptualno slične modelima K najbližih suseda. Osnovna ideja se zasniva na činjenici, da je visoka verovatnoća da je vrednost primeraka koja se predviđa, ista kao kod drugih primeraka koji imaju slične vrednosti atributa, na osnovu kojih se vrši predviđanje.

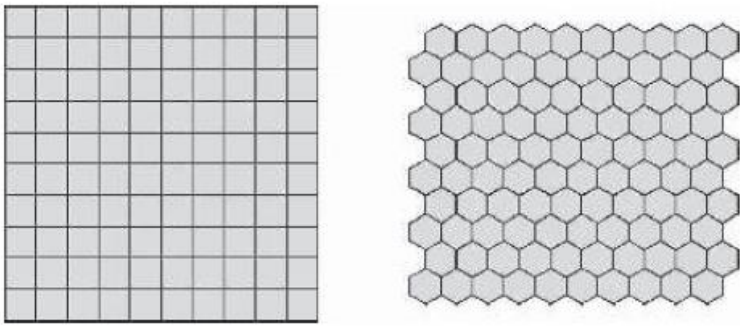
Više o radijalno zasnovanim funkcijama može se naći u [5].

3.3. Kohonenove samoorganizujuće mape

Samoorganizujuća mapa (*eng. Kohonen self-organizing map*) je tip veštačke neuronske mreže koja se trenira koristeći nenadgledano učenje kako bi se proizveo niskodimenzioni (tipično dvodimenzioni) prikaz ulaza, tj. mapa. Samoorganizujuće mape se razlikuju od ostalih veštačkih neuronskih mreža po tome što koriste funkcije susedstva kako bi očuvale topološka svojstva ulaza. Ovo samoorganizujuće mape čini korisnim za vizualizaciju niskodimenzionih pogleda visokodimenzionih podataka. Ova veštačka neuronska mreža razvijena je od strane finskog profesora Teuva Kohonena osamdesetih godina prošlog veka. Kohonenova mreža je računarski zgodna apstrakcija, koja se nadovezuje na rane biološke modele neurona iz sedamdesetih godina prošlog veka, kao i modele morfogeneze koji datiraju još od Alana Turinga iz pedesetih godina prošlog veka.

Kao i mnoge druge neuronske mreže, samoorganizujuće mape rade u dva režima: treniranje i preslikavanje. Treniranje izgrađuje mapu koristeći ulazne primerke (kompetativan proces,

takođe nazvan kvantizacija vektora), dok preslikavanje automatski klasifikuje novi ulazni vektor. Samoorganizujuća mapa se sastoji od jedinica, tj. neurona. Svakoj jedinici je pridružen težinski vektor iste dimenzije kao ulazni vektori, kao i pozicija na mapi. Jedinice su uređene na dvodimenzionoj koordinatnoj mreži, kao šestougli ili kao kvadrati (*slika 3.1.*). Samoorganizujuća mapa opisuje preslikavanje iz višedimenzionog prostora ulaza ka niskodimenzionom prostoru mapa. Procedura smeštanja vektora iz prostora ulaza na mapu podrazumeva pronalaženje jedinice sa najmanjom (metričkom) razlikom između težinskog vektora i vektora ulaznog podatka.



Slika 3.1. Pravougaona i šestougaona topologija

Dok se struktura ovakve mreže tipično smatra kao mreža prosleđivanja unapred gde se jedinice vizualizuju kao povezane, ovakav tip arhitekture je fundamentalno različit. Pokazano je da se samoorganizujuće mape sa malim brojem jedinica ponašaju na način sličan algoritmu K-sredina, dok veće samoorganizujuće mape topološki reorganizuju podatke. Često se pri učenju koristi i U matrica. Vrednost U matrice za konkretnu jedinicu je prosečna razdaljina između težinskog vektora te jedinice i težinskih vektora njenih suseda. Kod mreže gde su jedinice predstavljene kao pravougaonici, mogu se uzeti najbližih četiri ili osam suseda (Fon Nojmanova i Murova susedstva, tim redom) ili šest jedinica na šestougaonoj mreži. Kod velikih mapa koje broje preko hiljade jedinica, moguće je izvršiti operacije klasterovanja na samoj mapi.

Cilj učenja samoorganizujuće mape, je da natera različite delove mreže da se slično odazivaju na konkretne šablone ulaza. Ovo je delimično motivisano načinom kojim se upravljaju vizualni, auditorni i senzorni podaci iz spoljašnjeg sveta u različitim delovima cerebralnog korteksa u ljudskom mozgu. Težine neurona se inicijalizuju na male nasumične vrednosti. Potrebno je da se mreža pohrani sa što većim brojem primera vektora koji predstavljaju, što je bliže moguće, vektore koji se očekuju prilikom preslikavanja.

Trening faza koristi kompetativno učenje. Kada se trening primerak pohrani u mrežu, računa se njegovo Euklidsko rastojanje od svih težinskih vektora. Neuron čiji je težinski vektor najsličniji ulazu se zove jedinica najboljeg poklapanja (*eng. Best matching unit*). Težine jedinice najboljeg poklapanja i neuroni koji su u neposrednoj blizini se podešavaju prema vektoru ulaza. Intenzitet promena se smanjuje vremenom i rastojanjem (na koordinatnoj rešetki) od jedinice najboljeg poklapanja.

Formula ažuriranja za neuron v sa vektorom težine $W_v(s)$ je:

$$W_v(s+1) = W_v(s) + \theta(u, v, s)a(s)(D(t) - W_v(s)) \tag{3.1}$$

Gde je s indeks koraka, t indeks trening primerka, u indeks jedinice najboljeg poklapanja za $D(t)$, $a(s)$ je monotono opadajući koeficijent učenja, $D(t)$ je ulazni vektor i $\theta(u, v, s)$ je funkcija susedstva koja vraća udaljenost između neurona u i neurona v u koraku s. U zavisnosti od implementacije, t može da pretraži skup trening podataka sistematski (t je 0, 1, 2... T-1, gde je T veličina trening skupa), nasumično da se izvuče iz skupa podataka ili da implementira neku drugu metodu uzorkovanja. Funkcija susedstva $\theta(u, v, s)$ zavisi od rastojanja na rešetki između jedinice najboljeg poklapanja (neuron u) i neurona v. U najjednostavnijoj formi vrednost je 1, ukoliko su svi neuroni dovoljno blizu jedinici najboljeg poklapanja i 0 za ostale; takođe može da se koristi i Gausova raspodela. Nezavisno od funkcionalne forme, funkcija susedstva se smanjuje u toku vremena. U početku kad je susedstvo široko, samoorganizovanje se odvija na globalnoj skali. Kad se susedstvo smanji na nekolicinu neurona, težine konvergiraju ka lokalnim procenama. U nekim implementacijama, koeficijent učenja a i funkcija susedstva θ se postepeno smanjuju kako se s povećava, dok se kod drugih (posebno kod onih gde t pretražuje skup trening podataka) smanjuju svaki put posle određenog broja koraka.

Ovaj proces se ponavlja za svaki vektor ulaza za (obično veliki) broj ciklusa λ. Mreža asocira izlazne jedinice sa grupama ili šablonima ulaznih podataka. Ukoliko ovi šabloni mogu da se imenuju, imena mogu da se pridruže odgovarajućim jedinicama u mreži treniranja.

Tokom preslikavanja, izdvojiće se jedan pobednički neuron čiji je težinski vektor najbliži ulaznom vektoru. Pobednički neuron se jednostavno određuje tako što se izračuna Euklidsko rastojanje između ulaznog vektora i težinskog vektora. Dok je za primer ulaza za konstruisanje samoorganizujuće mape uzet vektor, ulaz može biti bilo kakav objekat koji se može predstaviti u digitalnom obliku, uz odgovarajuću meru rastojanja koja mu je pridružena, zajedno sa potrebnim operacijama za treniranje. Ovo podrazumeva matrice, kontinualne funkcije ili čak druge samoorganizujuće mape.

Više o Kohonenovim samoorganizujućim mapama može se naći u [6].

3.4. Rekurentne neuronske mreže

Nasuprot mreža prosleđivanja unapred, rekurentne neuronske mreže (*eng. Recurrent neural networks*) su modeli sa dvosmernim protokom podataka. Dok mreže prosleđivanja unapred propagiraju podatke od ulaza ka izlazu, RRN propagiraju podatke od daljih stadijuma procesiranja ka ranijim. RRN mogu da se koriste za opštu obradu sekvenci.

Potpuno rekurnetne mreže

Potpuno rekurentna mreža (*eng. Fully recurrent network*) je osnovna arhitektura razvijena osamdesetih godina prošlog veka. Sastoji se od jedinica, svaka sa usmerenom vezom ka svakoj drugoj jedinici. Svaka jedinica ima promenljivu vrednost iz skupa realnih brojeva, koja predstavlja njenu aktivaciju tj. izlaz. Svaka veza ima promenljivu težinu, takođe iz skupa realnih brojeva. Razlikuju se od jednostavnih rekurentnih mreža u tome što koriste konkurentna ažuriranja i propagaciju greške. Mreža prosleđivanja unapred ili jednostavna rekurzivna mreža, prenosi informacije od sloja ulaza ka sloju izlaza u jednom koraku: svaka grupa redom ažurira svoje ulaze i nakon toga ažurira svoje izlaze. U potpuno rekurentnoj mreži, sa druge strane, sve grupe istovremeno ažuriraju svoje ulaze i nakon toga svoje izlaze. Na taj način, informacija se propagira kroz samo jedan skup veza po koraku.

Hopfildova mreža

Hopfildova mreža je od istorijskog značaja iako nije klasična RRN, jer nije dizajnirana da obrađuje sekvence šablona. Umesto toga, ona zahteva stacionarne ulaze i ima simetrične veze. Razvijena je od strane Džon Hopfilda (John Hopfield) 1982. godine. Ako se veze istreniraju koristeći Hebovsko učenje, onda Hopfildova mreža moža da bude veoma robustna i da ima otpornost na menjanja veza.

Bolcmanova mašina

Bolcmanova (*eng. Boltzmann*) mašina može da se shvati kao Hopfield mreža sa šumom. Razvijena od strane Džefri Hintona (Geoffrey Hinton) i Teri Sejnovskog (Terry Sejnowki) 1985. godine, Bolcmanova mašina je bitna, jer predstavlja jednu od prvih veštačkih neuronskih mreža koja demonstrira učenje skrivenih jedinica. Učenje Bolcmanove mašine je inicijalno bilo vrlo sporo. Algoritam divergencije Džefri Hintona omogućuje da se treniranje izvršava mnogo brže.

Jednostavne rekurentne mreže

Jednostavnu modifikaciju osnovne mreže prosleđivanja unapred, prvi je implementirao Džef Elman (Jeff Elman) zajedno sa Majkl Džordanom (Michael I. Jordan). Koristi se mreža od tri sloja, uz dodatak skupa kontekstnih jedinica u ulaznom sloju. Postoje veze od skrivenog sloja ili od sloja izlaza ka ovim kontekstnim jedinicama sa fiksiranim težinama čija je vrednost jedan.

Svakim korakom algoritma, ulaz se propagira na standardan način prosleđivanja unapred, a zatim se primenjuje jednostavno pravilo slično propagaciji unazad (ovo pravilo ne primenjuje pravilan gradijentni spust). Fiksne vrednosti veza rezultuju da kontekstne jedinice održavaju kopiju prethodnih vrednosti skrivenih jedinica.

Eho stanje mreže

Eho stanje mreže (*eng. Echo state network*) je rekurentna neuronska mreža sa retko povezanim nasumičnim skrivenim slojem. Težine izlaznih neurona su jedini deo mreže koji

može da se menja i da se istrenira. ESN su dobri kod reprodukovanja određenih vremenskih serija.

Dugo kratkoročna memorija

Dugo kratkoročna memorija (*eng. Long short term memory network*), razvijena od strane Hohritera (Hochreiter) i Šmidhubera (Schmidhuber) 1997. godine, je veštačka neuronska mreža drugačija od tradicionalnih RNN po tome što nema problem nestajućeg gradijenta. Ovaj problem se javlja kad metode koje su zasnovane na gradijentnom spustu predugo traju jer se računske greške akumuliraju kroz nekoliko slojeva neurona. Pored toga što otklanja problem nestajućeg gradijenta ona može da opslužuje signale koji imaju mešavinu komponenti visokih i niskih frekvencija. LSTM RNN su nadmašile ostale RNN i druge metode učenja sekvenci kao što su skriveni Markovljev model (HMM) u brojnim primenama kao što su učenje jezika i prepoznavanje rukopisa.

Dvosmerne RNN

Razvijene su od strane Šustera (Shuster) i Palivala (Paliwal) 1997. godine. Dvosmerne RNN ili BRNN (*eng. Bidirectional RRN*), koriste konačne sekvence kako bi predvidele ili označile svaki element sekvence u zavisnosti od prošlog i od budućeg konteksta elementa. Ovo se radi tako što se dodaju izlazi dve RNN mreže. Jedna obrađuje sekvencu s leva na desno, a druga s desna na levo. Kombinovani izlazi su predviđanja ciljnih signala. Ova tehnika je dokazano korisna kada se kombinuje sa LSTM RNN.

Stohastičke neuronske mreže

Stohastička neuronska mreža (*eng. Stohastic neural networks*) se razlikuje od tipične neuronske mreže jer uvodi nasumične varijacije u mrežu. Sa verovatnosnog pogleda na neuronske mreže, ovakve nasumične varijacije mogu biti posmatrane kao forma statističkog uzorkovanja, kao što je Monte Carlo uzorkovanje.

3.5. Modularne neuronske mreže

Biološke studije su pokazale da ljudski mozak ne funkcioniše kao jedna velika mreža, već kao kolekcija manjih mreža. Ovaj koncept je dao inspiraciju da se naprave modularne neuronske mreže, kod kojih više manjih mreža rade kooperativno kako bi rešili probleme.

Komitet mašina

Komitet mašina (*eng. Committee of machines*) je kolekcija različitih neuronskih mreža koje zajedno "glasaju". Ovakav model daje mnogo bolje rezultate u poređenju sa drugim modelima neuronskih mreža. Kako bi se smanjila opasnost od lokalnog minimuma, sve mreže počinju sa istom arhitekturom i treniranjem, ali koriste različite nasumično inicijalizovane težine, generišući tako veoma različite mreže. CoM teži da stabilizuje rezultate i sličan je uopštenijoj korpa (*eng. Bagging*) metodi, izuzev toga što raznovrsnost

mašina iz komiteta dobija uz treniranje sa nasumičnim inicijalnim vrednostima težina umesto treniranja nad različitim nasumično izabranim podskupovim skupa treniranja.

Asocijativne neuronske mreže

Asocijativna neuronska mreža (*eng. Associative neural network*) je ekstenzija metode komiteta mašina, koja ide dalje od jednostavnog proseka različitih modela. ASNN predstavlja kombinaciju ansambla mreža prosleđivanja unapred i tehnike k-najbližih suseda. Koristi korelaciju između rezultata ansambla kao meru rastojanja kod analiziranih slučajeva za k najbližih suseda.

3.6. Ostali tipovi mreža

Spajking neuronske mreže

Spajking neuronske mreže (*eng. Spiking neural networks*) su modeli koji eksplicitno uzimaju u obzir vreme kod ulaznih podataka. Ulazi i izlazi mreže se uobičajeno predstavljaju kao serija spajkova (delta funkcija ili neki složeniji oblici). Spajking neuronske mreže imaju prednost u tome što mogu da obrađuju podatke u vremenskom domenu (signali koji variraju kroz određeno vreme). Često se se implementiraju kao rekurentne mreže. SNN su takođje jedna forma pulsnog računara. Mreže spajking neurona su korišćene za modeliranje separacije figura/zemlje i za povezivanje regija kod vizuelnog sistema [8].

Juna 2005 godine, IBM je najavio konstrukciju superračunara Blue Gene, koji je korišćen za simulaciju velike rekurentne spajking neuronske mreže.

Dinamičke neuronske mreže

Dinamičke neuronske mreže se bave ne samo nelinearnim višeatributnim problemima, već takođe uključuju učenje vremenski nezavisnog ponašanja kao što su raznoliki prelazni fenomeni i efekti kašnjenja. Tehnike procene procesa sistema kroz posmatrane podatke spadaju pod opštiju kategoriju identifikacije sistema.

Kaskadne neuronske mreže

Kaskadna korelacija je arhitektura i algoritam nadgledanog učenja razvijna od strane Skota Falmana (Scott Fahlman) i Kristijana Lebjera (Christian Lebiere). Umesto da se težine modifikuju u mreži fiksne topologije, kaskadna korelacija počinje sa minimalnom mrežom, a onda u automatizovanom procesu trenira i dodaje nove skrivene jedinice jednu po jednu, tako obrazujući višeslojnu strukturu. Jednom kada se nova skrivena jedinica doda u mrežu, njena težina sa ulazne strane ostaje nepromenjena. Ova jedinica postaje permanentan prepoznavač atributa u mreži, sposobna da proizvodi izlaze ili da pravi druge, složenije detektore atributa. Kaskadno korelaciona arhitektura uči brzo, određuje sopstvenu veličinu i topologiju,

zadržava strukture koje je napravila čak iako se trening skup promeni. Ne zahteva propagaciju unazad ili signale greške kroz veze mreže.

Neuro fazi mreže

Neuro fazi (*eng. Fuzzy*) mreža inkorporira sistem zaključivanja (fazi logiku) u telu veštačke mreže. U zavisnosti od tipa zaključivanja, postoji više slojeva koji simuliraju procese koji su deo fazi zaključivanja kao što su fazifikacija (*eng. Fuzzification*), zaključivanje, agregacija i defazifikacija (*eng. Defuzzification*). Ugrađivanje FIS-a (*eng. Fuzzy inference system*) u opštu strukturu veštačke neuronske mreže, ima kao pogodnost korišćenje raspoloživih metoda treniranja veštačke mreže kako bi se našli parametri fazi sistema.

Kompozicino produkcione neuronske mreže

Kompozicione produkcione neuronske mreže (*eng. Compositional pattern-producing networks*) su varijacija neuronskih mreža koje se razlikuju po skupu funkcija aktivacija i načinom na koji se primenjuju. Dok tipična neuronska mreža sadrži samo sigmoidne funkcije (ponekad i Gausovu funkciju, tj. funkciju normalne raspodele), CPPN mreže mogu da uključuju oba tipa funkcije kao i mnogo drugih. CPPN se primenjuju kroz čitav spektar mogućih ulaza tako da pretstavljaju kompletnu sliku. Kako je u pitanju kompozicija funkcija, CPPN efektivno enkodira slike u željenoj rezoluciji, te mogu da se uzorkuju za određen ekran u optimalnoj rezoluciji.

One-shot asocijativna memorija

Ovaj tip neuronske mreže može da dodaje nove šablone bez potrebe da se ponovno trenira. Ovo se obezbeđuje tako što se pravi posebna memorijska struktura, koja dodeljuje svaki novi šablon ortogonalnoj ravni, koristeći susedno spojene hijerarhijske nizove. Mreža pruža prepoznavanje šablona u realnom vremenu sa visokom skalabilnošću; iziskuje paralelno obrađivanje te je najprikladnija za platforme kao što su bežične senzorne mreže (*eng. Wireless sensor networks*), distribuirane sisteme i GPGUP (*eng. General-purpose computing on graphics processing units*). GPGUP sistem koristi grafičke procesorske jedinice kako bi izvršio proračune koje obično obavlja centralna procesorska jedinica tj. procesor.

Hijerarhijska privremena memorija

Hijerarhijska privremena memorija je model on-line mašinskog učenja (model gde podaci nisu svi odjednom dostupni, već postaju dostupni vremenom, u sekvencijalnom poretku). Razvijen od strane Džefa Hokinsa (Jeff Hawkins) i Dilip Džordža (Dileep George) iz firme Numenta, Inc. Model je inspirisan strukturalnim i algoritamskim svojstvima neokorteksa. HPM je biomimetrički model baziran na teoriji memorijskog predviđanja mozga opisan od strane Džefa Hokinsa u njegovoj knjizi „O inteligenciji“. HPM ne predstavljaju novu ideju ili teoiju, već kombinuje postojeće ideje tako da oponaša neokorteks sa jednostavnim dizajnom koji snabdeva veliki raspon sposobnosti. HPM kombinuje i nadograđuje ideje korišćene kod Bajesovih mreža, prostornih i vremenskih algoritama klasterovanja, koristeći drvoidnu hijerarhiju čvorova koji su uobičajeni kod neuronskih mreža.

Učenje kvantizacije vektora

Učenje kvantizacije vektora (*eng. Learning Vetor Quantization*) je više arhitektura neuronskih mreža nego što je posebna klasa. Prvi put je predložena od strane Teuvo Kohonena. Kod UKV, prototipski predstavnici svoje klase se parametrizuju, zajedno sa prikladnom merom distance. Suštinski je klasifikaciona shema zasnovana na razdaljini.

Fizičke neuralne mreže

Fizička neuronska mreža (*eng. Physical neural network*) podrazumeva električno podesiv materijal kako bi se simulirale sinapse. "Fizička" naglašava zavisnost od hardvera koji simulira neurone, za razliku od softverskih rešenja koja simuliraju neuronsku mrežu.

4. Metode

Postoje tri paradigme učenja gde se koriste neuronske mreže. Svaka odgovara specifičnom apstrakntom zadatku učenja. Ove paradigme su nadgledano učenje, nenadgledano učenje i učenje podrškom.

Nadgledano učenje - klasifikacija

U istraživanju podataka i statistici, klasifikacija je problem identifikacije skupa kome pripada novi primerak, na osnovu trening skupa podataka čija je pripadnost poznata. U terminologiji istraživanja podataka, klasifikacija je primer nadgledanog učenja, tj. učenja gde postoji trening skup čiji su primerci ispravno identifikovani.

Algoritam koji implementira klasifikaciju se naziva klasifikator. Termin "klasifikator" se ponekad takođe odnosi na matematičku funkciju, implementiranu od strane algoritma klasifikacije, koji preslikava ulazne podatke u neku od kategorija. Terminologija može da varira; u statistici, gde se klasifikacija izvršava uz pomoć logističke regresije ili slične procedure, karakteristike primeraka se nazivaju nezavisne promenljive, a kategorije koje se predviđaju se nazivaju ishodima, koje se smatraju kao moguće vrednosti zavisne promenljive. U mašinskom učenju, primerci se često nazivaju instancama, promenljive karakteristikama (grupisane u vektore karakteristika, *eng. Feature vector*) a kategorije koje se predviđaju nazivaju se klasama.

Problemi koji spadaju u okvir paradigme nadgledanog učenja sa neuronskim mrežama su prepoznavanje šablona (tj. klasifikacija) i regresija. Paradigma nadgledanog učenja je takođe primenljiva kod sekvencijalnih podataka (prepoznavanje govora i gestova).

Nenadgledano učenje - klasterovanje

Klaster analiza ili klasterovanje je zadatak grupisanja skupa objekata na takav način da su objekti u istoj grupi (nazvanoj klaster) više međusobno slični (na određen način) nego što su slični objektima iz različitih grupa (klastera). Klasterovanje je jedan od glavnih zadataka istraživanja podataka, kao i česta tehnika za statističku analizu podataka; koristi se u više oblasti, uključujući oblasti mašinskog učenja, prepoznavanja šablona, analize slika, bioinformatike itd. Klaster analiza može se izvršiti uz pomoć raznih algoritama koji se značajno razlikuju u svom poimanju predstavljanja klastera i njihovog efikasnog pronalaženja.

Popularna poimanja klastera podrazumevaju grupe sa malim rastojanjima između članova klastera, gustim oblastima prostora podataka, intervale ili specifične statističke distribucije. Odgovarajući algoritam klasterovanja i parametri podešavanja (uključujući vrednosti kao što su funkcija udaljenosti koja se koristi, prag gustine ili broj očekivanih klastera) zavise od pojedinačnog skupa podataka i namenjene upotrebe rezultata. Klaster analiza kao takva, nije automatizovan proces, već iterativni proces istraživanja podataka koji podrazumeva primenu novih pokušaja i neuspehe. Često je potrebno da se izmene preprocesiranje podataka i parametri modela, sve dok rezultat ne dostigne željena svojstva. Pored termina klasterovanje, postoji još termina sa sličnim značenjem; automatska klasifikacija, numerička taksonomija, botrilogija i tipološka analiza.

Problemi koji spadaju u okvir paradigme nenadgledanog učenja su generalno problemi procene. Primena podrazumeva klasterovanje, procenu statističkih distribucija, kompresiju i filtriranje.

Učenje uslovljavanjem

Učenje uslovljavanjem je oblast mašinskog učenja inspirisana biheviorističkom psihologijom. Bavi se načinom na koji softverski agenti preduzimaju akcije u okruženju, kako bi se maksimizovala kumulativna funkcija tj. nagrada. Podaci uglavnom nisu zadati, već se generišu kroz interakciju agenta sa okolinom. U svakoj tački vremena, agent preduzima akciju i okolina generiše stanje i trenutnu nagradu, prema nekoj (uglavnom nepoznatoj) dinamici. Zbog svoje opštosti se istražuje u mnogim drugim disciplinama, kao što su: teorija igara, teorija upravljanja, teorija informacija, sistemi više agenata, inteligencija roja, statistika i genetski algoritmi.

Okruženje se obično formuliše kao Markovljev proces odlučivanja (*eng. Markov decision process*), gde se u tom kontekstu koriste tehnike dinamičkog programiranja. Problemi koji spadaju u okvir paradigme učenja podrškom su problemi upravljanja (*eng. Optimal control theory*), igre i ostali sekvencijalni problemi odlučivanja.

5. Primene neuronskih mreža

Možda najveća prednost veštačkih neuronskih mreža je u njihovoj mogućnosti da se koriste kao proizvoljan mehanizam aproksimacije funkcija koje uče na osnovu podataka. Međutim, korišćenje nije toliko jednostavno, te je potrebno odlično poznavanje teorije. Odabir modela zavisi od reprezentacije podataka i primene. Preterano složeni modeli imaju tendenciju da stvaraju probleme pri učenju. Skoro svaki algoritam učenja će se ponašati dobro sa ispravnim parametrima za treniranje na konkretnom fiksiranom skupu podataka. Doduše, odabir i štimovanje algoritma za treniranje nad nepoznatim podacima zahteva dosta eksperimentisanja. Ukoliko se model, funkcija cene i algoritam učenja ispravno izaberu, rezultujuća neuronska mreža može biti veoma robustna.

Problemi na koje se primenjuju veštačke mreže uglavnom spadaju u sledeće kategorije:

1) Aproksimacija funkcija ili regresiona analiza, zajedno sa vremenskim serijama.
2) Klasifikacija, uključujući prepoznavanje šablona i sekvenci.
3) Procesiranje podataka, uključujući filtriranje, klasterovanje i kompresiju.
4) Upravljanje, tj. numeričko računarsko upravljanje.

Oblasti primene uključuju sisteme identifikacije i upravljanja (upravljanje vozila, upravljanje trajektorije, upravljanje procesa, upravljanje prirodnim resursima), kvantnu hemiju, igre i donošenje odluka (bekgemon, šah, poker), prepoznavanje šablona (radarski sistemi, prepoznavanje lica, objekata itd.), prepoznavanje sekvenci (gestova, govora, rukopisa), medicinske dijagnoze, finansijske aplikacije (automatski sistemi trgovine), otkrivanje znanja u bazama podataka (*eng. Knowledge discovery in databases*), vizualizacija i filtriranje neželjenih elektronskih poruka (spam), itd.

Neuronske mreže su takođe korišćene za diagnozu nekih vrsta kancerogenih bolesti. Hibridni sistem za prepoznavanje kancera pluća, zasnovan na neuronskim mrežama pospešuje preciznost dijagnoze i brzinu radiologije kancera pluća. Ove mreže su korišćene i za dijagnozu kancera prostate. Dijagnoze mogu da se iskoriste kako bi se napravili specifični modeli, tako da rezultate veće grupe pacijenata porede sa informacijama jednog pacijenta. Modeli ne zavise od pretpostavki korelacija različitih promenljivih. Kolorektalni kancer je takođe diagnostifikovan pomoću neuronskih mreža. Neuronske mreže su mogle da predvide ishod za pacijenta sa kololateralnim rakom sa više preciznosti nego trenutne kliničke metode. Nakon treniranja, mreže su mogle da predvide ishode više pacijenata iz različitih nepovezanih ustanova.

Postoji mnogo različitih primena neuronskih mreža koje podrazumevaju prepoznavanje šablona i donošenje odluka u odnosu na te šablone. U avionima, moguće je koristiti neuronsku mrežu u funkciji jednostavnog autopilota, sa ulaznim jedinicama koje čitaju signale različitih instrumenata kokpita, i izlazim jedinicama koje modifikuju kontrole aviona kako bi se avion bezbedno kretao na kursu. U fabrikama, moguće je koristiti neuronske mreže za kontrolu kvaliteta. Na primer, pri proizvodnji deterdženta, može se meriti finalni proizvod na različite načine (boja, ph vrednost, gustina, itd.), na osnovu kojih će mreža odlučiti da li

treba da se odbaci ili prihvati bačva sa proizvodom. Takođe, postoji mnogo primena neuronskih mreža u domenu sigurnosti. Kod mnogobrojnih transakcija kreditnih kartica, potreban je automatizovan način da se prepoznaju transakcije koje su potencijalno neki oblik prevare. Neuronske mreže perfektno odgovaraju ovom zadatku. Ulazi mogu biti razni faktori kao npr. da li je osoba koja koristi karticu prisutna, da li se koristi ispravan PIN, da li su pet ili više transakcija obavljene u proteklih deset minuta, da li se kartica koristi u državi drugačijoj od one gde je registrovana itd… Sa dovoljnom količinom indicija, neuronska mreža može da obeleži transakcije koje deluju sumnjivo, i tako omogući operateru da ih istraži pažljivije. Na sličan način, banka može da koristi neuronsku mrežu radi procene odobrenja kredita klijentima na osnovu njihove istorije kredita, trenutnih prihoda, istorije zaposlenja, itd...

Većina mnogobrojnih dnevnih aktivnosti čoveka, obuhvata prepoznavanje šablona zarad donošenja odluka, tako da neuronske mreže mogu da nam pomognu na više načina. Mogu da nam pomognu pri prognozi vremena ili kretanja na berzi, da operišu radarskim sistemima ili pak da pomognu lekarima da diagnostifikuju složena oboljenja na bazi njihovih simptoma. Kod mobilnih aplikacija za prepoznavanje rukopisa, koriste se jednostavne neuronske mreže kako bi se odredili karakteri koje korisnik unosi tako što se gledaju istaknute osobine oznaka koji se prave prstima (kao i redosled kojim se unose). Neki oblici softvera za prepoznavanje govora takođe koriste neuronske mreže.

Tabela 4.1. Primene neuronskih mreža

<table>
<tr>
<td>Finansije:

• Predviđanja na berzi
• Dostojnost kredita
• Kreditni rejting
• Predviđanja bankrota
• Procena vlasništva
• Otkrivanje prevara
• Prognoza cena
• Prognoza ekonomskog indikatora</td>
<td>Istraživanje podataka:

• Predviđanje
• Klasifikacija
• Otkrivanje promene i devijacije
• Otkrivanje znanja
• Modeliranje odziva
• Analiza vremenskih serija</td>
</tr>
<tr>
<td>Medicina:

• Medicinske dijagnoze
• Otkrivanje i evaluacija medicinskih fenomena
• Prognoza dužine hospitalizacije pacijenta
• Procena cene lečenja</td>
<td>Prodaja i marketing:

• Prognoza prodaja
• Ciljni marketing
• Prognoza korišćenja servisa</td>
</tr>
<tr>
<td>Industrija:

• Upravljanje procesa
• Kontrola kvaliteta
• Predviđanje temperature i sile</td>
<td>Operaciona analiza:

• Optimizacija zaliha maloprodaje
• Optimizacija zakazivanja
• Menadžersko pravljenje odluka
• Prognoza protoka gotovine</td>
</tr>
<tr>
<td rowspan="2">Nauka:

• Prepoznavanje šablona
• Modeliranje fizičkih sistema
• Evaluacija ekosistema
• Identifikacija polimera
• Prepoznavanje gena
• Botanička klasifikacija
• Procesiranje signala: neuronsko filtriranje
• Analiza bioloških sistema
• Analiza i identifikacija mirisa</td>
<td>Menadžment ljudskih resursa:

• Izbor i zapošljavanje
• Zadržavanje zaposlenih
• Zakazivanje osoblja
• Profilisanje osoblja</td>
</tr>
<tr>
<td>Energija:

• Prognoza električne opterećenosti
• Prognoza energetske potražnje
• Kratkoročna i dugoročna procena opterećenosti
• Predviđanje indeksa cena gasa/uglja
• Sistemi za kontrolu
• Nadzor rada hidroelektrana</td>
</tr>
<tr>
<td>Obrazovanje:

• Predviđanje uspeha studenata
• Istraživanje neuronskih mreža
• Trijaža fakultetskih prijava</td>
<td>Ostalo:

• Sportsko klađenje
• Odabir pobednika trka konja i pasa
• Kvantitativna prognoza vremena
• Razvoj igrica
• Optimizacioni problemi, usmeravanje
• Procena poljoprivredne proizvodnje</td>
</tr>
</table>

6. Paketi

Postoje razna gotova rešenja u obliku softverskih paketa. Ovakvi paketi obično sadrže širok dijapazon raznih alata vezanih za analitiku podataka, takođe sadrže implementacije neuronskih mreža. U nastavku su kratki opisi nekih popularnih softverskih paketa:

Weka

Weka je kolekcija algoritama istraživanja podataka napisanih u Javi. Weka je besplatan softver pod GNU GPL licencom. Sadrži alate za preprocesiranje podataka, klasifikaciju, regresiju, klasterovanje, pravila pridruživanja i vizualizaciju. Dobro je prilagođena za razvijanje novih shema istraživanja podataka. Pored toga, sadrži grafičko korisničko okruženje za pristup ovim funkcijama. Inicijalna verzija se koristila kao alat za analiziranje podataka iz poljoprivrednog domena, dok se danas koristi i u drugim, pogotovu u obrazovnom i istraživačkom domenu.

Link: http://www.cs.waikato.ac.nz/ml/weka/

Knime

Knime je softverska platforma otvorenog koda za analitiku podataka, izveštavanje i integraciju. Takođe napisan u Javi i pod GPL licencom, integriše različite komponente za mašinsko učenje i istraživanje podataka. Grafičko korisničko okruženje omogućava sklapanje različitih čvorova za preprocesiranje (ekstrakciju, transformaciju, učitavanje), za modeliranje, analitiku podataka kao i vizualizaciju. Koristi se kod farmaceutskih istraživanja, poslovne analitike i analize finansijskih podataka.

Link: https://www.knime.org/

RapidMiner

RapidMiner je softverska platforma koja nudi integrisano okruženje za mašinsko učenje, istraživanje podataka, istraživanje teksta, prediktivnu analitiku i poslovnu analitiku. Koristi se u poslovne i industrijske svrhe, kao i za edukaciju, istraživanja, brzo generisanje prototipova, i podržava sve korake procesa istraživanja podataka uključujući vizualizaciju, proveru i optimizaciju.

Link: https://rapidminer.com/us/

SPSS i SPSS Modeler

SPSS Statistics je softverski paket za statističku analizu. Zvanični naziv je IBM SPSS Statistics. SPSS (*eng. Statistical Package for the Social Sciences*) paket, se koristio kao što ime kaže u domenu društvenih nauka. Danas, ovaj softver je popularan i u drugim domenima, uključujući nauke o zdravlju i marketing. SPSS Modeler je softver za istraživanje podataka i analizu teksta. Koristi se za pravljenje prediktivnih modela i sprovođenje ostalih zadataka analitike. Sadrži grafičko korisničko okruženje koje omogućava korišćenje algoritama istraživanja podataka bez programiranja. Neke primene su: otkrivanje prevara, upravljanje rizikom, unapređivanje kvaliteta proizvoda, unapređivanje kvaliteta zdravstvene zaštite, sprovođenje zakona, itd.

Link: http://www-01.ibm.com/software/analytics/spss/
Link: http://www-01.ibm.com/software/analytics/spss/products/modeler/

SAS

SAS (*Statistical Analysis System*) je softver razvijen od strane SAS Instituta, namenjen za naprednu analitiku, analizu više promenljivih, poslovnu analitiku, upravljanje podacima i prediktivnu analitiku. SAS može da istražuje, menja, upravlja i vraća podatke iz različitih izvora i da izvršava statističku analizu na njima. Podržava grafičko korisničko okruženje za netehnički orijentisane korisnike kao i napredne opcije kroz SAS programski jezik. SAS programi imaju korak podataka (*DATA step*), koji obrađuje i vraća podatke, tako praveći SAS bazu podataka i korak procesa (*PROC step*), koji analizira podatke.

Link: https://www.sas.com/en_us/home.html

IBM Intelligent miner

Intelligent miner je paket namenjen za klasifikaciju, pravila pridruživanja, vremenske serije, klasterovanje i predviđanje vrednosti. Dostupnost raznih alata po sebi pruža moćne alternative; nekoliko alata nude veći izbor tehnika. Klasifikacija može da se izvršava koristeći drvo odlučivanja ili neuronske mreže, dok klasterovanje može da se izvrši koristeći demografski algoritam ili neuronske mreže. IBM Intelligent miner koriste integratori i analitičari podataka. Klijentsko aplikacijsko programsko okruženje je obezbeđeno tako da korisničke aplikacije mogu da pristupe algoritmima i funkcijama istraživanja podataka. Analitičari koriste grafičko korisničko okruženje kako bi definisali i izvršili funkcije bez potrebe da se razvija posebna aplikacija. Intelligent miner je integrisan u DB2 Warehouse okruženje.

Link: http://www-01.ibm.com/software/data/db2/linux-unix-windows/warehouse-analytics/

MLC++

MLC++ je biblioteka c++ klasa za nadgledano istraživanje podataka. MLC++ se sastoji od opštih algoritama istraživanja podataka za korišćenje od strane krajnjih korisnika, analitičara i istraživača. Pruža korisnicima širok spektar alata koji pomažu pri istraživanju podataka, povećanju pouzdanosti softvera, pružaju alate za poređenje i vizualizaciju. MLC++ pokušava da izdvoji sličnosti algoritama istraživanja podataka i prikaže ih na jednostavan, povezan i proširiv način.

Link: http://www.sgi.com/tech/mlc/

mlpack

mlpack je još jedna biblioteka istraživanja podataka c++ jezika, koja stavlja akcenat na skalabilnost, brzinu i lakoću korišćenja. Cilj je učiniti da primena algoritama istraživanja podataka bude olakšana za početnike tako što se koristi jednostavan, konzistentan aplikacioni programski interfejs, uz korišćenje odlika c++ jezika kako bi se pružio maksimum performansi i fleksibilnosti.

Link: http://www.mlpack.org/about.html

R

R je programski jezik i okruženje za statističko računarstvo i grafiku. R je GNU projekat koji je sličan jeziku S koji je razvijen u Bell laboratorijama (bivši AT&T, sada Lucent Technologies) od strane Džon Čembersa (John Chambers). R može da se smatra kao drugačija implementacija S-a. Postoje bitne razlike ali većina koda koji je napisan za S, se koristi i u R-u. Jezik S je često prvi izbor kad je u pitanju istraživanje statističkih metodologija, dok R pruža mogućnost (kao jezik otvorenog koda) učestvovanja u toj aktivnosti. R pruža širok spektar statističkih (linearno i nelinarno modelovanje, vremenske serije, klasifikacija, klasterovanje) i grafičkih tehnika i visoko je ekstenzibilan. Jedna od R-ovih jačih strana je lakoća pravljenja dobro dizajniranih i kvalitetnih grafika, uključujući matematičke simbole i formule. R je dostupan pod uslovima korišćenja GPL licence. Kompajlira se i pokreće na različitim UNIX platformama (uključujući FreeBSD i Linux), Windows-u i MacOS-u.

Link: https://www.r-project.org/

7. Primer primene neuronskih mreža

Data je baza podataka [13] koja predstavlja uzorke belog i crnog vina Verde sa severa Portugalije. Zadatak je napraviti model kvaliteta vina na osnovu fizičko-hemijskih svojstava, tj. klasifikovati vina ocenama od 1-10. Klase su uređene ali ne i balansirane (postoji mnogo više vina srednjeg kvaliteta nego odličnih ili lošijih). Korišćeni paketi su Weka, Knime, RapidMiner.

Karakteristike baze podataka:	Baza više promenljivih	Broj instanci:	6497
Karakteristike atributa:	Realni brojevi	Broj atributa:	12
Zadatak:	Klasifikacija	Nedostajuće vrednosti	Nema

Link: http://archive.ics.uci.edu/ml/datasets/Wine+Quality

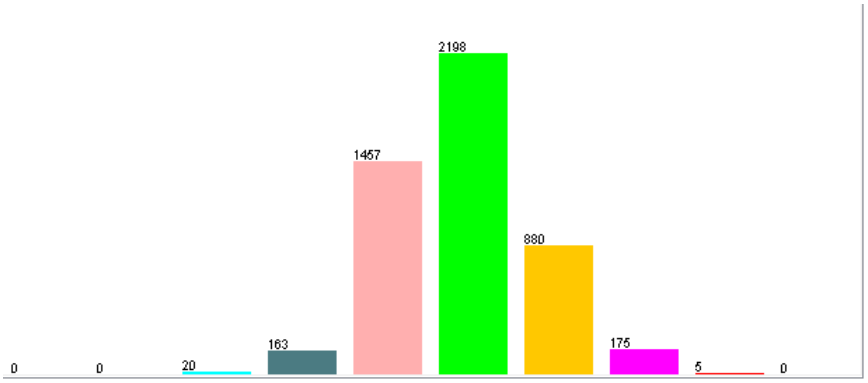
Ulazni atributi:

1. Stalna kiselost
2. Promenljiva kiselost
3. Limunska kiselina
4. Rezidualni šećer
5. Hloridi
6. Slobodni sumpor dioksid
7. Ukupni sumpor dioksid
8. Gustina
9. pH
10. Sulfati
11. Alkohol

Izlazni atribut (labela, tj. atribut po kome se vrši klasifikacija):

12. Kvalitet

Ova baza podataka je dobar primer kako kvalitet modela zavisi od količine podataka i deskriptivnosti same baze podataka. Nebalansiranost klasa se može videti na slici 7.1.



Slika 7.1. Grafik klasa kvaliteta vina (1-10)

S obzirom da ne postoji test skup podataka, koristi se testiranje u trening fazi, tzv. unakrsna provera (*eng. Cross-validation*). Unakrsna provera je tehnika procene generalizacije rezultata statističke analize u nezavisan skup podataka. Uglavnom se koristi kod zadataka predviđanja, gde je potrebno proceniti koliko precizno će model predviđanja raditi u praksi. Kod zadatka predviđanja, baza podataka se deli na dva dela: deo nad kojim se pravi model i deo nad kojim se testira tako napravljen model. U procesu unakrsne provere se polazni skup podataka (iterativno) deli na različite kombinacije podataka za formiranje modela i podataka za testiranje modela, sa ciljem da se izbegne preprilagođavanje modela skupu podataka koji je odabran za pravljenje modela.

Jedna iteracija unakrsne provere podrazumeva particionisanje trening skupa podataka na komplementarne podskupove. Analiza, tj. treniranje se izvršava nad jednim trening podskupom, dok se provera analize izvršava nad drugim testnim podskupom. Kako bi se smanjila varijabilnost, izvršava se više iteracija unakrsne provere, koristeći različite particije, a rezultati provere se uprosečavaju u zavisnosti od broja iteracija.

Struktura koja sadrži tačno pozitivne/negativne mere (*eng. true positive/negative rate*) se naziva matrica konfuzije i služi za poređenje, jer se na osnovu nje može izračunati preciznost klasifikacije.

Weka:

Weka podržava tri algoritma neuronskih mreža.

1) Algoritam jednoslojnog perceptrona (*eng. Singlelayer Perceptron*)

- Rezultat rada jednoslojnog perceptrona:

```
Time taken to build model: 2.1 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        2195               33.7848 %
Incorrectly Classified Instances      3194               49.1612 %
Kappa statistic                          0.1532
Mean absolute error                      0.1243
Root mean squared error                  0.337
Relative absolute error                111.5151 %
Root relative squared error            142.7568 %
UnClassified Instances                1108               17.054  %
Total Number of Instances             6497
```

- Matrica konfuzije jednoslojnog perceptrona:

```
=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j   <-- classified as
    0    0    0    0    0    0    0    0    0    0 |    a = 1
    0    0    0    0    0    0    0    0    0    0 |    b = 2
    0    0    0    3   13    2    6    2    0    0 |    c = 3
    0    0    0    8  101   37   28    8    0    0 |    d = 4
    0    0    0   32 1032  403  301   73    0    0 |    e = 5
    0    0    0    9  649  830  644  177    0    0 |    f = 6
    0    0    0    1   88  387  306   89    0    0 |    g = 7
    0    0    0    0    7   70   60   19    0    0 |    h = 8
    0    0    0    0    0    1    3    0    0    0 |    i = 9
    0    0    0    0    0    0    0    0    0    0 |    j = 10
```

2) Algoritam višeslojnog perceptrona sa argumentom skrivenih slojeva postavljen na 0

- Rezultat rada višeslojnog perceptrona:

```
Time taken to build model: 9.2 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        3352               51.593  %
Incorrectly Classified Instances      3145               48.407  %
Kappa statistic                          0.2217
Mean absolute error                      0.1171
Root mean squared error                  0.2476
Relative absolute error                 87.0953 %
Root relative squared error             95.5528 %
Total Number of Instances             6497
```

- Matrica konfuzije višeslojnog perceptrona:

```
=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j   <-- classified as
    0    0    0    0    0    0    0    0    0    0 |    a = 1
    0    0    0    0    0    0    0    0    0    0 |    b = 2
    0    0    1    0   16   10    3    0    0    0 |    c = 3
    0    0    0    2  129   80    5    0    0    0 |    d = 4
    0    0    0    3 1275  841   19    0    0    0 |    e = 5
    0    0    0    2  787 1870  175    2    0    0 |    f = 6
    0    0    0    0  110  765  204    0    0    0 |    g = 7
    0    0    0    0   10  127   56    0    0    0 |    h = 8
    0    0    0    0    0    1    4    0    0    0 |    i = 9
    0    0    0    0    0    0    0    0    0    0 |    j = 10
```

3) Algoritam višeslojnog perceptrona (*eng. Multilayer Perceptron*)

- Rezultat rada višeslojnog perceptrona:

```
Time taken to build model: 875.09 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        3558               54.7637 %
Incorrectly Classified Instances      2939               45.2363 %
Kappa statistic                          0.2823
Mean absolute error                      0.1129
Root mean squared error                  0.2416
Relative absolute error                 84.0357 %
Root relative squared error             93.2186 %
Total Number of Instances             6497
```

- Matrica konfuzije višeslojnog perceptrona:

```
=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j   <-- classified as
    0    0    0    0    0    0    0    0    0    0 |    a = 1
    0    0    0    0    0    0    0    0    0    0 |    b = 2
    0    0    0    0   18    9    2    1    0    0 |    c = 3
    0    0    0    1  139   73    2    1    0    0 |    d = 4
    0    0    0   13 1354  753   18    0    0    0 |    e = 5
    0    0    0    4  721 1877  234    0    0    0 |    f = 6
    0    0    0    3   43  706  325    2    0    0 |    g = 7
    0    0    0    0    5  112   75    1    0    0 |    h = 8
    0    0    0    0    0    1    4    0    0    0 |    i = 9
    0    0    0    0    0    0    0    0    0    0 |    j = 10
```

4) Algoritam radijalno zasnovanih funkcija (*eng. RBF Network*)

- Rezultat rada *RBF*:

```
Time taken to build model: 12.95 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        3393               52.2241 %
Incorrectly Classified Instances      3104               47.7759 %
Kappa statistic                          0.2307
Mean absolute error                      0.1193
Root mean squared error                  0.2449
Relative absolute error                 88.7715 %
Root relative squared error             94.4859 %
Total Number of Instances             6497
```

- Matrica konfuzije *RBF*:

```
=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j   <-- classified as
    0    0    0    0    0    0    0    0    0    0 |    a = 1
    0    0    0    0    0    0    0    0    0    0 |    b = 2
    0    0    1    1   18   10    0    0    0    0 |    c = 3
    0    0    2    2  121   89    2    0    0    0 |    d = 4
    0    0    2    3 1275  845   12    0    1    0 |    e = 5
    0    0    2    4  769 1899  158    4    0    0 |    f = 6
    0    0    1    0   72  787  215    2    2    0 |    g = 7
    0    0    0    0   12  132   47    1    1    0 |    h = 8
    0    0    0    0    0    2    3    0    0    0 |    i = 9
    0    0    0    0    0    0    0    0    0    0 |    j = 10
```

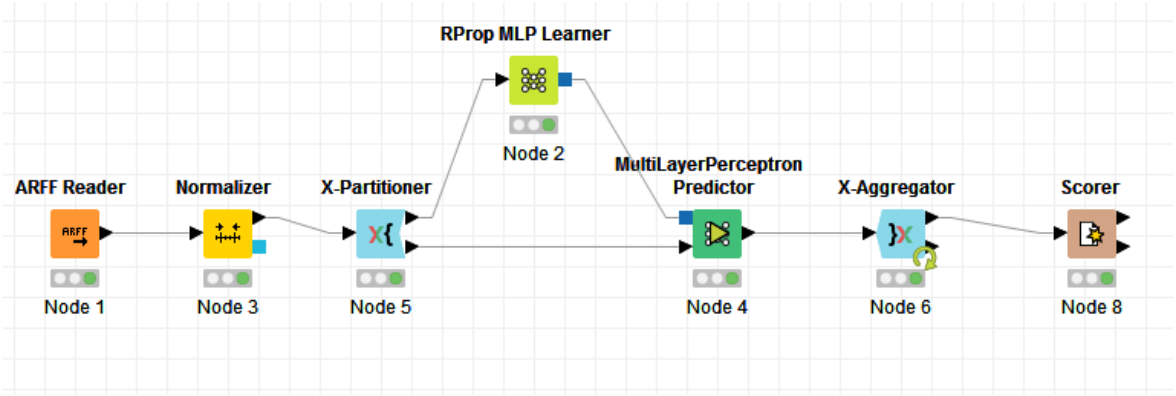
Može se zaključiti da su oba modela pod istim okolnostima proizvela slične rezultate. Zajednička potreba svih modela, je određen nivo kvaliteta trening baze podataka. Ne postoji model koji je otporan na nedostatke baze nad kojom se trenira. Iz tog razloga, bitnija je veličina, kvalitet i deskriptivnost same baze, dok izbor algoritma može doprineti u skromnijem opsegu.

Kako bi se dodatno dokazala ova veoma bitna činjenica u svetu istraživanja podataka, u nastavku su dati rezultati paketa Knime i RapidMiner nad istim podacima.

Knime:

Knime podržava dva algoritma neuronskih mreža.

1) Algoritam višeslojnog perceptrona (*eng. Multilayer Perceptron*)



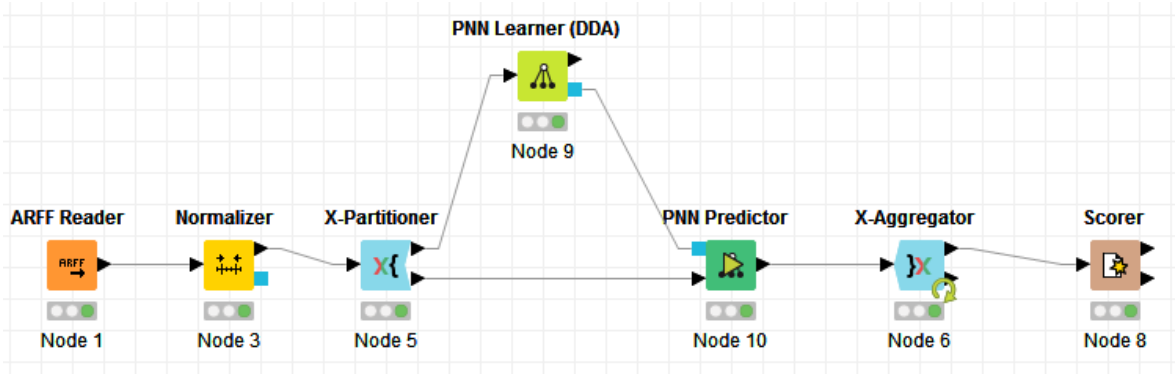
- Rezultat rada višeslojnog perceptrona:

Columns: 11	Column Type	Column Index	Color Handler	Size Handler	Shape Han…	Lower Bound	Upper Bound
TruePositives	Number (integer)	0				0.0%	203000.0%
FalsePositives	Number (integer)	1				0.0%	186400.0%
TrueNegatives	Number (integer)	2				179700.0%	649700.0%
FalseNegatives	Number (integer)	3				0.0%	87500.0%
Recall	Number (double precision)	4				0.0%	71.58%
Precision	Number (double precision)	5				25.0%	59.92%
Sensitivity	Number (double precision)	6				0.0%	71.58%
Specifity	Number (double precision)	7				49.08%	100.0%
F-measure	Number (double precision)	8				0.91%	60.47%
Accuracy	Number (double precision)	9				54.49%	54.49%
Cohen's kappa	Number (double precision)	10				26.27%	26.27%

- Matrica konfuzije višeslojnog perceptrona:

Row ID	4	5	6	7	1	2	3	8	9	10
4	1	132	78	5	0	0	0	0	0	0
5	2	1318	806	12	0	0	0	0	0	0
6	1	670	2014	151	0	0	0	0	0	0
7	0	53	820	206	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	17	12	1	0	0	0	0	0	0
8	0	6	138	49	0	0	0	0	0	0
9	0	0	3	2	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

2) Probabilističku neuronsku mrežu (*eng. Probabilistic Neural Network - PNN*)



- Rezultat rada *PNN*:

Columns: 11	Column Type	Column Index	Color Handler	Size Handler	Shape Han...	Lower Bound	Upper Bound
TruePositives	Number (integer)	0				0.0%	218600.0%
FalsePositives	Number (integer)	1				0.0%	159000.0%
TrueNegatives	Number (integer)	2				207100.0%	649700.0%
FalseNegatives	Number (integer)	3				0.0%	75300.0%
Recall	Number (double precision)	4				0.0%	77.08%
Precision	Number (double precision)	5				0.0%	78.38%
Sensitivity	Number (double precision)	6				0.0%	77.08%
Specifity	Number (double precision)	7				56.57%	100.0%
F-measure	Number (double precision)	8				7.63%	67.48%
Accuracy	Number (double precision)	9				61.89%	61.89%
Cohen's kappa	Number (double precision)	10				38.76%	38.76%

- Matrica konfuzije *PNN*:

Row ID	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	19	10	1	0	0	0
4	0	0	2	9	127	78	0	0	0	0
5	0	0	3	6	1471	656	2	0	0	0
6	0	0	0	5	569	2186	74	2	0	0
7	0	0	0	0	35	712	326	6	0	0
8	0	0	0	0	1	129	34	29	0	0
9	0	0	0	0	0	5	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

RapidMiner:

RapidMiner podržava dva algoritma neuronskih mreža.

1) Algoritam višeslojnog perceptrona (*eng. Multilayer Perceptron*)

    - Rezultat rada višeslojnog perceptrona:

accuracy: 55.15% +/- 2.17% (mikro: 55.15%)

	true 5	true 6	true 7	true 4	true 8	true 3	true 9	class precision
pred. 5	1362	722	50	144	6	17	0	59.19%
pred. 6	756	1924	734	66	126	11	2	53.16%
pred. 7	15	189	295	4	61	1	3	51.94%
pred. 4	3	1	0	2	0	0	0	33.33%
pred. 8	2	0	0	0	0	1	0	0.00%
pred. 3	0	0	0	0	0	0	0	0.00%
pred. 9	0	0	0	0	0	0	0	0.00%
class recall	63.70%	67.84%	27.34%	0.93%	0.00%	0.00%	0.00%	

2) AutoMLP - Algoritam višeslojnog perceptrona sa dinamičkim (u fazi treniranja) podešavanjem sopstvene veličine, kao i stope učenja - α. Ovaj algoritam kombinuje različite ideje iz domena genetskih algoritama i stohastičke optimizacije.

    - Rezultat rada AutoMLP-a:

accuracy: 54.38% +/- 1.38% (mikro: 54.38%)

	true 5	true 6	true 7	true 4	true 8	true 3	true 9	class precision
pred. 5	1216	596	36	124	4	16	0	61.04%
pred. 6	901	2062	794	85	117	12	3	51.89%
pred. 7	14	177	247	1	70	0	2	48.34%
pred. 4	6	0	0	6	0	1	0	46.15%
pred. 8	1	1	2	0	2	1	0	28.57%
pred. 3	0	0	0	0	0	0	0	0.00%
pred. 9	0	0	0	0	0	0	0	0.00%
class recall	56.88%	72.71%	22.89%	2.78%	1.04%	0.00%	0.00%	

Da se primetiti da je RapidMiner automatski izbacio pred. 1, pred. 2 i pred. 10 (vrednost kvaliteta sa ocenom 1, 2 i 10) jer su to vrednosti labele koje nisu dodeljene ni jednom primerku iz baze.

Za zadatak klasifikacije vina, najbolje su se pokazale probabilističke neuronske mreže iz paketa RapidMiner sa 61.89% ispravno klasifikovanih primeraka. Najlošije se pokazao algoritam jednoslojnog perceptrona iz paketa Weka sa 33.7848% ispravno klasifikovanih primeraka. Razlog je što je u pitanju stariji algoritam koji je izbačen iz nove verzije Weka paketa. Umesto njega se koristi višeslojni perceptron sa argumentom skrivenih slojeva postavljen na 0, koji daje znatno bolje rezultate (51.593%).

Višeslojni perceptroni iz ovih paketa imaju slične rezultate (Weka: 54.7637%, Knime: 54.49%, RapidMiner: 55.15% i 54.38%). AutoMLP iz paketa RapidMiner, se pokazao lošijim od običnog algoritma višeslojnog perceptrona istog paketa, sa razlikom od 0.77% ispravno klasifikovanih primeraka.

8. Zaključak

Cilj ovog rada je da pruži osnovne infomacije kada su neuronske mreže u pitanju. Zahvaljujući velikim uspesima u primeni neuronskih mreža, sve se više ulaže u razumevanje i plasiranje neuronskih mreža kao rešenje za mnoge probleme. Čini se da ne postoji problem koji zahteva neki oblik odlučivanja zasnovan na podacima iz realnog sveta, a da neuronske mreže ne mogu da ga reše. Sve dok je problem dobro definisan, a podaci iz realnog sveta pronađu svoju odgovarajuću digitalnu reprezentaciju, neuronske mreže uz određeno vreme prilagođavanja konkretnom problemu mogu predstavljati rešenje za mnoge složene izazove.

Uz moderne termine kao što su “Big Data” i “Internet of Things”, koji predstavljaju sisteme koji generišu ogromne količine podataka, stoje neuronske mreže kao efikasan alat za obradu svih tih podataka.

Moderni trendovi danas iziskuju ljudski potencijal koji se bavi analitikom ove ogromne količine podataka, a pretpostavlja se da će ovaj trend potražnje za stručnjacima koji vladaju tom materijom stagnirati tek 2018. godine. Stoga je odlična prilika biti deo te informacione revolucije u nauci koja je relativno mlada; upustiti se u proučavanje alata za analitiku podataka od kojih je razumevanje i primena neuronskih mreža samo mali postotak.

Razumevanjem teorije neuronskih mreža i njihove prirode, zajedno sa njihovim nastankom i predmetom inspiracije, može se zaključiti da je to definitivno jedna od tehnologija budućnosti, koja će prožimati skoro sve aspekte ljudske delatnosti.

Literatura:

[1] Walter Pitts, Warren McCulloch, A Logical Calculus of Ideas Immanent in Nervous Activity, 1943

[2] Donald Heb, The Organisation of Behaviour, John Wiley & Sons Ltd, 1949

[3] John Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc. NatL Acad. Sci. USA, 1982

[4] Alexander I. Galushkin, Neural Networks Theory, Springer-Verlag Berlin Heidelberg, 2007

[5] Ke-Lin Du, M. N. S. Swamy, Neural Networks and Statistical Learning, Springer-Verlag London, 2014

[6] Pragya Agarwal, Andre Skupin, Self-Organising Maps: Applications in Geographic Information Science, John Wiley & Sons Ltd, 2008

[7]Stanford University, Machine Learning Course, http://online.stanford.edu/course/machine-learning

[8] Reitboeck et al.in Haken and Stadler: Synergistics of the Brain, Berlin, 1989

[9] http://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History

[10]http://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Architecture

[11] https://en.wikipedia.org/wiki/Artificial_neural_network

[12] https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks

[13] P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.