Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Luka Murn

Paralelna implementacija novih pristopov učenja z nevronskimi mrežami in njihovo vrednotenje na biomedicinskih podatkih

diplomsko delo

univerzitetni študij računalništva in informatike

Mentor: prof. dr. Blaž Zupan

Ljubljana, 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01978 / 2014
Datum: 9.1.2014

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **LUKA MURN**

Naslov: **PARALELNA IMPLEMENTACIJA NOVIH PRISTOPOV UČENJA Z NEVRONSKIMI MREŽAMI IN NJIHOVO VREDNOTENJE NA BIOMEDICINSKIH PODATKIH**
PARALLEL IMPLEMENTATION OF IMPROVED NEURAL NETWORK CLASSIFIERS AND THEIR EXPERIMENTAL ASSESSMENT ON BIOMEDICAL DATA SETS

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

V diplomski nalogi implementirajte strojno učenje z umetnimi nevronskimi mrežami, ki so nadgrajene z nedavno predlaganimi izboljšavami. Te naj bi mrežam zvišale napovedno točnost. Izboljšave naj vključujejo metodo izločanja, skladanje avtomatskih enkoderjev z odstranjevanjem šuma in skladanje omejenih Boltzmannovih naprav. Programsko rešitev razvijte za paralelne arhitekture (grafične kartice). Implementirane tehnike preskusite na klasifikaciji podatkov iz molekularne biomedicine. Poročajte o napovednih točnostih, vplivu izboljšav in primerjave z drugimi tehnikami strojnega učenja.

Mentor:

prof. dr. Blaž Zupan

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Luka Murn, z vpisno številko **63080186**, sem avtor diplomskega dela z naslovom:

Paralelna implementacija novih pristopov učenja z nevronskimi mrežami in njihovo vrednotenje na biomedicinskih podatkih

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Blaža Zupana,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 17. januarja 2014 Podpis avtorja:

Zahvaljujem se vsem prijateljem, ki so verjeli vame, me sprejeli v svoj krog in mi s tem dali energijo, samozavest in zagon, ne samo za pričujoče delo, temveč za vse uspehe in pomembne trenutke v mojem življenju. Prav tako se zahvaljujem staršem za vso podporo skozi čas odraščanja in študija. Nenazadnje gre velika zahvala tudi mentorju, saj je bil navkljub veliki geografski in časovni razliki vedno dosegljiv in odziven.

Kazalo

Povzetek

Na področju strojnega učenja je bila v zadnjih letih razvita vrsta pristopov, ki lahko bistveno izboljšajo napovedne točnosti umetnih nevronskih mrež. V diplomskem delu predstavimo tri take tehnike: metodo izločanja, skladanje avtomatskih enkoderjev z odstranjevanjem šuma in skladanje omejenih Boltzmannovih naprav. Te metode so bile do sedaj večinoma uporabljene na velikih podatkovnih naborih, v diplomskem delu pa nas je zanimalo njihovo obnašanje na manjših podatkovnih naborih s področja molekularne biologije. Učenje umetnih nevronskih mrež je računsko zahtevno, zato našo implementacijo, ki teče na grafičnih karticah, razvijemo z uporabo knjižnice Theano. Rezultati vrednotenja pokažejo, da novi pristopi dejansko izboljšajo napovedne točnosti mrež, ki pa na manjših podatkih ne presegajo rezultatov klasičnih tehnik strojnega učenja, kot je logisitična regresija. Pokažemo tudi, da smo s paralelno implementacijo izrazito zmanjšali čas učenja in tako razvili knjižnico, ki je uporabna tudi za večje podatkovne nabore.

Ključne besede: umetne nevronske mreže, strojno učenje, metoda izločanja, diskriminativno učenje, DNA mikromreže, CUDA

Abstract

The field of artificial neural networks has been buzzing with increased activity in the past few years. Many new methods were proposed to improve the classification accuracy of neural networks. We present three such methods in this thesis: dropout, stacking denoising autoencoders (SdAs) and stacking restricted Boltzmann machines (SRBMs). Up to now, those methods have mostly been used on large datasets. In this thesis, we test them on small datasets representing data from the area of molecular biology. As the process of learning artificial neural networks is very time consuming, our implementation utilizes the GPU using Theano Python library. Our results show that while the proposed methods increase the classification accuracy of neural networks, they still fall behind classic machine learning models, such as logistic regression, on small datasets. We also show that parallel implementation greatly reduces time needed to learn the model, and present a library that's usable for larger datasets as well.

Keywords: artificial neural networks, machine learning, dropout, discriminative learning, DNA microarrays, CUDA

Poglavje 1

Uvod

Z izrazom umetne nevronske mreže označujemo vrsto matematičnih modelov, ki v grobem ponazarjajo delovanje človeških možganov. Umetne nevronske mreže imajo pestro zgodovino; od prvih definicij v začetku 20. stoletja pa do danes so bile namreč predmet mnogih raziskav; ko se je področje razvijalo, so številne znanstvene panoge videle uporabno vrednost nevronskih mrež ter jih tako prikrojile za svojo uporabo. Ena izmed teh disciplin je tudi strojno učenje, v okviru katerega smo umetne nevronske mreže proučili v pričujočem delu. V sledečem poglavju želimo bralca predvsem seznaniti, zakaj so se umetne nevronske mreže sploh razvile in kakšno je stanje na področju raziskovanja nevronskih mrež dandanes. Sledi opis ciljev diplomske naloge in obseg opravljenega dela.

1.1 Motivacija

Medicina in molekularna biologija sta v zadnjih 50. letih izjemno napredovali. Od odkritja dvojne vijačnice [29] pa do določitve zaporedij nukleotidov v človeškem genomu [11] se je področje razvijalo in širilo ter iz akademskih okvirjev prešlo v mnoge druge discipline. Poleg človeškega genoma je študij genskega zapisa drugih organizmov na nek način vzpostavil novo disciplino - biotehnologijo, ki ne le, da je že v sedanjem času med najbolj hitro rastočimi

panogami, temveč vse kaže, da bo njena vloga v prihodnosti razvoja človeštva le še bolj pomembna. Ravno hiter napredek in pripadajoč razvoj tehnologije botrujeta dejstvu, da se je v zadnjih desetih letih s pomočjo številnih tehnik, kot so npr. sekvenciranje DNA, DNA mikromreže in verižna reakcija s polimerazo, nakopičila ogromna količina bioloških podatkov. Ti podatki so s pravilno interpretacijo uporabni v medicini, industriji hrane, farmaciji, kmetijstvu in še mnogih drugih panogah. Ravno zaradi velike količine in dimenzije teh podatkov pa je pri njihovi obdelavi potreben informacijski pristop, kar je povzročilo razmah področja bioinformatike.

Na drugi strani akademske sfere se je podoben, nezadržan in izredno hiter razvoj odvijal v računalništvu. Če so bili še ob koncu 80. let računalniki počasni in dragi, je danes domači računalnik marsikatere družine zmožen obdelave velikih podatkovnih naborov, pa naj bo to s področja ekonomije, družboslovja ali molekularne biologije. Predvsem strojno učenje [26] je disciplina, ki je neposredno uporabna za veliko zgoraj omenjenih področij, saj med drugim obravnava problem klasifikacije - iz danih podatkov želi izluščiti neke zakonitosti, na podlagi katerih novim objektom - podatkom določi posamezne oznake. Ravno napredek v zmogljivosti računalnikov je povzročil, da so se nekatere metode, ki teoretično obstajajo že dalj časa, šele zdaj zares pojavile v praksi in začele konkurirati do sedaj uporabljenim, preprostešim metodam, saj prej računalniki enostavno niso premogli dovoljšnje računske moči. Primer take metode so umetne nevronske mreže, ki so se po odkritju učinkovitejšega algoritma za učenje (vzvratno razširjanje napak, [30]) izkazale za zelo močan in prilagodljiv model. Še bolj obetavna pa je razširitev umetnih nevronskih mrež, tako imenovana metoda izločanja [10], ki se je nedavno izkazala za zelo uspešno pri odpravljanju enega bolj perečih problemov umetnih nevronskih mrež - prevelikemu prilagajanju podatkom. Na podatkovnem naboru ročno napisanih števk MNIST, ki je popularen primerjalni test za različne klasifikacijske algoritme, se ta metoda uvršča med najboljše algoritme po deležu napak [16]. Navkljub temu dejstvu pa se, kot sklepamo na podlagi pregleda literature, prav ta metoda zaenkrat še ni uporabila na

področju analiz podatkov s področja molekularne biologije.

1.2 Cilj diplomske naloge

V diplomski nalogi smo implementirali tehniko strojnega učenja z osnovno umetno nevronsko mrežo in vzvratnim razširjanjem, nato pa le-to razširili na dva načina:

- za regularizacijo smo uporabili metodo izločanja;
- za predhodno učenje uteži smo uporabili dva različna tipa diskriminativnega učenja:
  - skladanje avtomatskih enkoderjev z odstranjevanjem šuma,
  - skladanje omejenih Boltzmannovih naprav.

Tako izpopolnjeno nevronsko mrežo smo nato želeli primerjati z obstoječimi modeli strojnega učenja, ki se na bioloških podatkih dobro obnesejo. V primerjavo smo vključili tri splošno znane in uporabljane napovedne modele: logistično regresijo, metodo podpornih vektorjev ter naključne gozdove. Kot tipičen primer podatkov s področja molekularne biologije smo za testiranje uporabili rezultate študij genskih izrazov na področju napovedovanja rakastih obolenj. Te študije so bile izvedene s pomočjo DNA mikromrež [25]. Gre za postopek, kjer nas za posamezne vzorce tkiv zanimajo izraženosti genov oziroma uporabljanost informacije, ki jo gen kodira, v namene izdelave proteinov. Predvsem nas je zanimalo, ali zgoraj omenjene variante nevronskih mrež na danih podatkih dejansko izboljšajo klasifikacijo v primerjavi z osnovnimi nevronskimi mrežami, ter če, ali so pri tem značilno boljše od napovedih drugih popularnih tehnik strojnega učenja.

1.3 Vsebina dela

V okviru diplomske naloge smo najprej programsko implementirali nevronske mreže. Zaradi preprostosti kode, skriptne narave jezika ter dobre podpore

knjižnjic za strojno učenje smo se odločili za programski jezik Python. V njem je napisana celotna koda diplomske naloge. Pri strojnem učenju samo učenje napovednega modela pogosto traja tudi več dni, saj so podatkovni nabori lahko res veliki. Zato smo se odločili, da za pospešitev delovanja uporabimo Pythonovo knjižnico *Theano*. Ta omogoča izvajanje izračunov na grafični kartici, kar zaradi paralelizma močno pohitri določene matrične operacije.

Tako implementirane nevronske mreže smo razširili z metodo izločanja ter diskriminativnim učenjem. Za druge napovedne modele smo uporabili implementacije v knjižnici *Orange*. Za vse te napovedne modele smo nato s prečnim preverjanjem izračunali ploščino AUC na 15 različnih naborih podatkov. Mero AUC smo izbrali zato, ker je neodvisna od velikosti in strukture podatkov. S pomočjo Wilcoxonovega testa smo nazadnje raziskali, kateri modeli so med seboj značilno različni.

Poglavje 2

Nevronske mreže

Termin "nevronske mreže" že nekaj časa v ljudeh vzbuja različne odzive. Strokovnjaki s področja matematike, fizike, računalništva ali drugih znanosti so večinoma seznanjeni z izrazom, prav tako pa vedo, zakaj se nevronske mreže lahko uporabljajo. V netehnični javnosti pa je izraz celo zdaj, po več kot 50. letih obstoja, še vedno sprejet z začudenjem - nekateri nevronske mreže povezujejo z biologijo, drugi vedo, da gre za matematični model, velika večina pa nima povsem jasne predstave, kam jih uvrstiti.

Glavni predmet pričujoče diplomske naloge so torej nevronske mreže. Kot klasifikator so nevronske mreže matematični model. Če želimo učinkovito implementirati katerikoli model, ga je potrebno najprej dobro opredeliti. Poglavje, ki sledi, zatorej oriše vse elemente, ki so potrebni pri gradnji umetnih nevronskih mrež - od zgodovine in biološkega ozadja, do osnovne postavitve modela ter učnega algoritma. Proti koncu poglavja so opisane še različne tehnike in izboljšave nevronskih mrež, katerih namen je izboljšati točnosti napovedi.

2.1 Zgodovina

Delovanje človeških možganov je predmet zanimanja znanstvenikov že od vekomaj. Številni zdravniki, logiki, matematiki, psihologi in inženirji so se

trudili dognati, na kakšen način ljudje dojemamo dražljaje iz okolice in kako se učimo. Pa vendar, šele napredek v elektroniki v 20. stoletju je povzročil, da so se akademiki začeli zanimati ne le o razlaganju, temveč o posnemanju delovanja možganov. Prvo prelomno delo na področju nevronskih mrež sta opravila W. S. McCulloch in W. H. Pitts. Njun članek iz leta 1943 (*A logical calculus of the ideas immanent in nervous activity*) [19] je matematično opisoval način delovanja nevronov. Njuni nevroni so bili binarne enote, ki so na podlagi izbranega praga in podanega vhoda simulirali neko logično resnico. V tistem času so znanstveniki namreč mislili, da se da delovanje možganov dobro opisati z logičnim modelom.

Zaradi razvoja modernega računalnika, kot ga poznamo danes, je raziskovanje modela človeških možganov v 50. letih nekoliko zamrlo. Leta 1958 pa je F. Rosenblatt uvedel model perceptrona [22]. Perceptron je bila prva praktično uporabna umetna nevronska mreža, ki je bila simulirana na tedaj še zelo počasnih računalnikih. Perceptroni so bili v zgodnjih 60. letih predmet mnogih raziskav, ker so omogočali učenje modela na testnih podatkih in posledično klasifikacijo. Leta 1969 sta M. Minsky in S. Papert v svoji knjigi [20] naštela zmogljivosti in številne omejitve perceptronov. Med drugim sta dokazala, da perceptroni lahko rešujejo le linearne probleme. Mnogo znanstvenikov je tedaj menilo, da v knjigi omenjene omejitve veljajo za vse modele nevronskih mrež. Zaradi tega je raziskovanje področja močno upadlo. Navkljub temu dejstvu se perceptroni še danes uporabljajo na podatkovnih naborih z velikim, večmilijonskim številom značilk.

Na podlagi dela v drugih modelih je P. Werbos prenesel algoritem vzvratnega razširjanja napak tudi v nevronske mreže [30]. S tem dejanjem so tudi globoke, večnivojske nevronske mreže dobile učinkovit učni algoritem za nelinearne probleme, ki jim je do tedaj tako manjkal. To dejanje je povzročilo ponoven porast zanimanja za večnivojske nevronske mreže, ki do danes ni usahnil.

Leta 1986 so na prvi konferenci nevronskih mrež za računalništvo D. E. Rumelhart, G. E. Hinton in R. J. Williams širši javnosti predstavili pravilo

vzvratnega razširjanja napak [23]. Razvoj na področju se je tako v 80. in 90. letih še dodatno širil, saj so računalniki postali zmogljivejši, nabori podatkov večji, umetne nevronske mreže pa so se začele uporabljati tudi v komercialne namene.

V 21. stoletju so umetne nevronske mreže med najboljšimi in najbolj pogosto uporabljenimi napovednimi modeli. Navkljub dejstvu, da jih še vedno pestijo nekateri problemi, ki obstajajo že od njihovega nastanka, nedavno delo G. E. Hintona in drugih (metoda izločanja, globoke verjetnostne mreže) priča, da se umetne nevronske mreže še da izboljšati in optimizirati. Po drugi strani je napredek strojne opreme takšen, da lahko učimo globoke, večnivojske mreže na res velikih naborih podatkov.

2.2 Osnovna struktura modela umetne nevronske mreže

Umetna nevronska mreža je, kot pove že ime samo, v mrežasto strukturo povezana skupina nevronov. Gre za izredno idealiziran in poenostavljen model dela človeških možganov. Število osnovnih gradnikov, nevronov, je v umetnih nevronskih mrežah danes zaradi sodobne strojne opreme sicer lahko kar veliko, vendar težko preseže red velikosti 10^4. Po drugi strani se ocenjuje, da je število nevronov v človeških možganih med 10 milijardami in 1 bilijonom [31], kar nedvomno govori kompleksnosti bioloških mrež v prid. Toda kaj sploh je nevron?

2.2.1 Nevron

Osnovna enota vseh nevronskih mrež je nevron. Umetni nevron je idealizirana posplošitev naravnega, človeškega nevrona v možganih. Naravni nevron ima strukturo, ki je prikazana na sliki 2.1. Živčni impulz pride preko dendritov v živčno celico, nakar se akcijski potencial prevaja proti drugim živčnim celicam preko aksona. Stik med dendritom ene celice in aksonom druge celice

se imenuje sinapsa. Impulz se ustvari in potuje preko aksona do drugih nevronov vsakič, ko se preko dendritov nakopiči dovolj električnega naboja, da le-ta preseže določen prag. S količino potenciala, ki se prenaša preko sinaps, se te spreminjajo, kar povzroči spremembo praga.

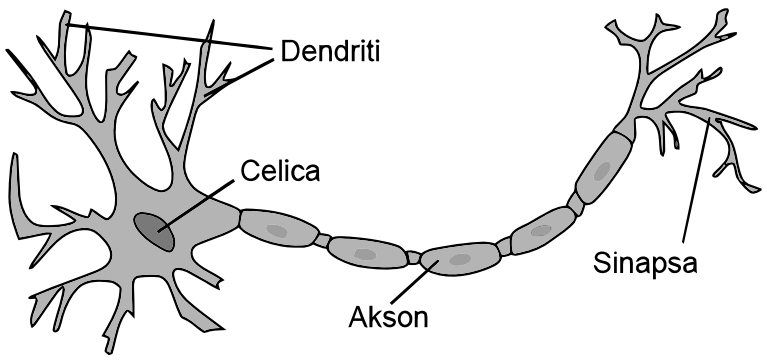
Na osnovi naravnih nevronov so zgrajeni tudi umetni nevroni. Pri tem je treba takoj poudariti, da imajo naravni nevroni diskretne impulze aktivnosti, medtem ko pri idealiziranih umetnih nevronih impulzi lahko predstavljajo tudi realne vrednosti. Izkaže se, da s tem lahko dobro simuliramo tudi diskretne impulze. Na sliki 2.2 je prikazana zgradba umetnega nevrona.

Umetni nevron sprejme vhodni vektor vrednosti $x = [x_1, x_2, ..., x_n]^T$. Da dobi vrednost stanja nevrona z (včasih poimenovana funkcija aktivacije), vhodni vektor pomnoži z vektorjem uteži $w = [w_1, w_2, ..., w_n]$ in jim doda konstanten člen b. Takemu izračunu aktivacijske funkcije pravimo utežena vsota. Izhod umetnega nevrona je funkcija $g(z)$:

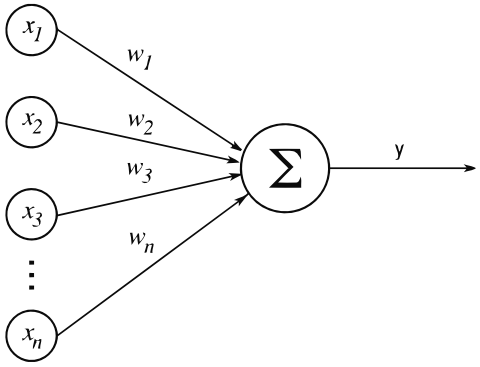
$$y = g(wx + b)$$

Funkciji $g(z)$ pravimo tudi izhodna funkcija nevrona. Kot pri aktivacijski funkciji tudi g lahko predstavlja poljubno funkcijo. V procesu razvoja umetnih nevronskih mrež se je izkazalo, da se najbolje obnesejo mreže, kjer je funkcija g nelinearna - omejuje nek poljuben vhod na omejen izhod.

V praksi se hitro izkaže, da en sam nevron ni dovolj za rešitev kompleks-



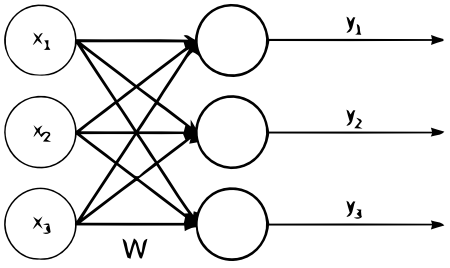
Slika 2.1: Zgradba nevrona.



Slika 2.2: Zgradba umetnega nevrona. Notacije se med različnimi avtorji razlikujejo.

nejših problemov. Tako kot imamo v možganih ogromno količino nevronov, tudi pri umetnih nevronskih mrežah potrebujemo večje število nevronov, povezanih v neko mrežo. Zaradi boljše urejenosti in organizacije predvsem pri usmerjenih nevronskih mrežah posamezne nevrone združujemo v **nivoje** oz. **plasti**. Primer preprostega nivoja nevronov je prikazan na sliki 2.3. V primeru nivoja se enačbe nekoliko spremenijo; uteži postanejo matrika W, konstantni faktor pa vektor b. Izhod nivoja nevronov je tedaj ravno tako vektor:

$$y = g(Wx + b)$$



Slika 2.3: Primer nivoja s 3 vhodi in 3 nevroni.

2.2.2 Vrste umetnih nevronskih mrež

Tako kot se v možganih nevroni povezujejo na različne načine, lahko tudi umetne nevrone med seboj povežemo na več načinov. Najbolj logična delitev umetnih nevronskih mrež je glede na njihovo topologijo.

Usmerjene nevronske mreže. Najbolj pogosto uporabljen model umetnih nevronskih mrež je zgrajen iz več zaporedno povezanih nivojev nevronov. Vsebuje vhodni nivo, izhodni nivo in poljubno število skritih nivojev. Primeren je za regresijske in klasifikacijske probleme, poleg tega pa ima relativno preprost algoritem učenja z vzvratnim razširjanjem napak [23]. Navkljub temu se v primeru večjega števila skritih nivojev pogosto preveč prilagaja podatkom. Splošen algoritem vzvratnega razširjanja napake tudi ne zagotavlja optimalne rešitve, saj se lahko ustavi v lokalnih minimumih. Dokazano je, da lahko mreža z vsaj enim skritim nivojem reši tudi nelinearne probleme. Primer dvonivojske arhitekture je perceptron. Za regresijske probleme mora imeti umetna nevronska mreža na zadnjem nivoju en sam nevron, pri klasifikacijskih problemih pa je na zadnjem nivoju tipično toliko nevronov, kot ima klasifikacijski problem oznak.

Rekurzivne nevronske mreže. Če odstranimo omejitev, da so nevroni med seboj povezani v enosmernih, urejenih nivojih, dobimo rekurzivne nevronske mreže. Primer takih mrež so npr. Hopfieldove mreže [12] ali Boltzmannove naprave [1]. Pri njih so nevroni povezani v nestrukturirano mrežo, lahko vsebujejo tudi cikle. Za take nevronske mreže velja, da imajo sposobnost pomnjenja skozi čas in z njimi lahko realiziramo neke vrste pomnilnik. Njihova slabost pa je, da jih je precej težko naučiti.

V diplomski nalogi smo se ukvarjali s problemom klasifikacije, zato smo uporabili večnivojske usmerjene nevronske mreže, ki se včasih imenujejo tudi **globoke nevronske mreže** [4].

2.3 Model

Naj bo dimenzija vhodnih podatkov (št. atributov) $\mathcal{D} = \mathcal{D}_0$ in L št. razredov klasifikacijskega problema. Število nivojev v umetni nevronski mreži je N. Vhod v nevronsko mrežo je vektor $x = [x_1, x_2, ..., x_{\mathcal{D}}]^T$. Pri usmerjeni nevronski mreži, ki ima vsaj 2 nivoja, vektor $y^{(i)} = [y_1^{(i)}, y_2^{(i)}, ..., y_{\mathcal{D}_i}^{(i)}]^T$ predstavlja izhod vseh nevronov na i-tem nivoju, če je njihovo število enako $\mathcal{D}_i$. Velja enakost $y^{(0)} = x$. Če je $W^{(i)}$ matrika uteži na prehodu iz nivoja $i-1$ v nivo i ter $b^{(i)}$ vektor konstantnih uteži na nivoju i, lahko definiramo aktivacijski vektor vseh nevronov na nivoju i:

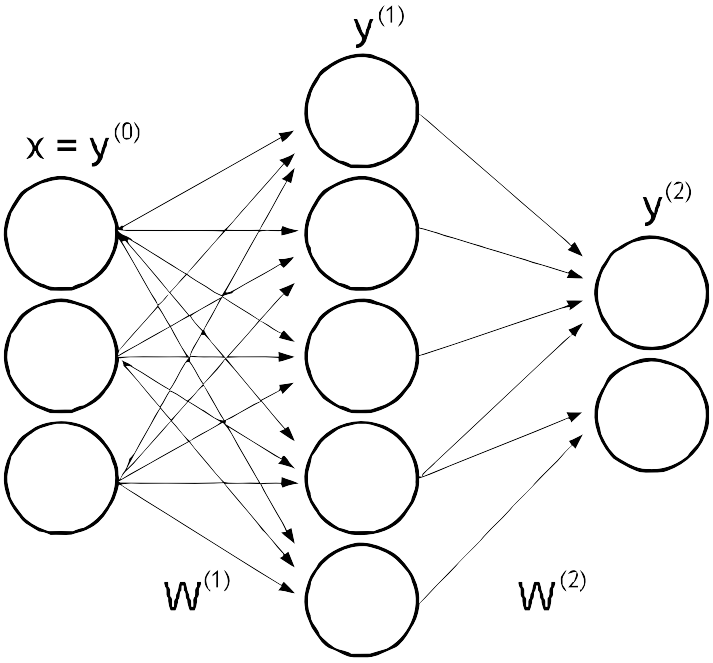
$$z^{(i)} = W^{(i)} y^{(i-1)} + b^{(i)}$$

To je **utežena vsota** in je najbolj pogosto uporabljena aktivacijska funkcija. Obstajajo tudi bolj splošne, nelinearne funkcije. V diplomski nalogi smo uporabili uteženo vsoto kot funkcijo aktivacije nevronov.

Parametri modela, ki se tekom učenja spreminjajo, so za nivo i definirani kot par $\Theta^{(i)} = \{W^{(i)}, b^{(i)}\}$. Vsi parametri umetne nevronske mreže so v tem primeru $\Theta = \{\Theta^{(1)}, \Theta^{(2)}, ..., \Theta^{(N-1)}\}$. Izhod posameznega nevrona je funkcija vrednosti aktivacije nevrona. Ta funkcija je lahko bodisi deterministična ali stohastična. Za celoten nivo nevronov je izhodna funkcija določena kot:

$$y^{(i)} = g(z^{(i)})$$

Primer preproste umetne nevronske mreže s tremi nivoji je prikazan na sliki 2.4. Pri usmerjeni nevronski mreži je prvi nivo vedno vhodni, zadnji nivo izhodni, vmes pa se nahaja poljubno število skritih nivojev. V praksi se sicer najpogosteje uporablja manjše število skritih nivojev, ki redko presega pet nivojev.



Slika 2.4: Preprosta umetna nevronska mreža s tremi vhodnimi in dvema izhodnima nevronoma. Na sliki manjka nekaj povezav na nivoju $W^{(2)}$.

2.3.1 Vrste nevronov

Različne vrste nevronov med seboj razlikujemo po izhodni funkciji $g(z)$ (enako velja tudi za vektor, $g(z^{(i)})$). V diplomski nalogi smo uporabili sledeče štiri vrste nevronov.

Linearen pragovni nevron

Pri njem je izhodna funkcija (glej formulo (2.1)) določena s pragom 0. Formula (2.1) je enaka formuli (2.2), ki je boljša za implementacijo v programskem jeziku. Linearna pragovna funkcija je še najbližje delovanju bioloških nevronov.

$$g(z) = \begin{cases} z & \text{če je } z > 0, \\ 0 & \text{sicer.} \end{cases} \tag{2.1}$$

$$g(z) = max(z, 0) \tag{2.2}$$

Sigmoidni nevron

Sigmoidni nevron uporablja sigmoidno funkcijo, da za poljuben vhod omeji izhodne vrednosti na interval $(0, 1)$. Pogosto se uporablja zaradi lepega odvoda sigmoidne funkcije pri učenju. Izhodno funkcijo podaja enačba:

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

Hiperbolični tangens

Pogosta izhodna funkcija, podobna sigmoidni, a simetrična, ki izhodne vrednosti omeji na interval $(-1, 1)$, je hiperbolični tangens:

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

Popravljeni hiperbolični tangens

V članku [15] je opisan popravek funkcije (2.3), ki za normalizirane

podatke (vhode) teži k temu, da je tudi varianca izhodnih vrednosti blizu 1. Tako popravljeno funkcijo podaja enačba:

$$g(z) = 1.7159\ tanh(\frac{2}{3}z)$$

2.3.2 Metoda "softmax"

Za regresijske probleme, kjer je izhod umetne nevronske mreže realno število, je arhitektura mreže preprosta. Na izhodnem nivoju se nahaja en nevron, ki oddaja izhod umetne nevronske mreže y v obliki realnega števila. Glede na vrsto nevronov v mreži je lahko potrebno y še normalizirati na ustrezen interval vrednosti.

Večji problem se poraja pri klasifikacijskih problemih. Če je število oznak pri klasifikaciji L, ima umetna nevronska mreža na zadnjem nivoju L nevronov. Med nadzorovanim učenjem, kjer poznamo konkretno oznako za posamezen učni primer, določimo izhodne vektorje na spodnji način:

$$y_{oznaka_1} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, y_{oznaka_2} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, ..., y_{oznaka_L} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tag{2.4}$$

Po učenju mreže na izhodu skoraj vedno želimo imeti izhodni vektor y, ki predstavlja verjetnostno porazdelitev oznak za dan vhodni primer. Vsak element izhodnega vektorja y_i mora torej predstavljati zaupanje, s katerim mreža določi vhodnemu primeru i-to oznako. V tem primeru mora veljati vsota, ki velja za vse verjetnostne porazdelitve: $\sum_i y_i = 1$. Toda umetna nevronska mreža se tega dejstva med učenjem ne zaveda. Ne ve, da so v interpretaciji posamezni razredi diskretni in medsebojno izključujoči. Rešitev za dan problem je metoda *softmax*, ki smo jo uporabili v diplomski nalogi. Ta za i-ti nevron v zadnjem (izhodnem) nivoju določa drugačno izhodno

funkcijo:

$$y_i = \frac{e^{z_i}}{\sum\limits_{1 \leq j \leq L} e^{z_j}}$$

Funkcija softmax zagotavlja, da je zgoraj omenjena vsota vedno enaka 1.

2.4 Razširjanje vhodnih vrednosti proti izhodnemu nevronu

Umetne nevronske mreže se v strojnem učenju uporabljajo predvsem za klasifikacijo (tema diplomske naloge) oz. regresijo. V tem primeru mora umetna nevronska mreža za vsak vhodni primer x določiti realno vrednost (regresija) ali pa verjetnostno razporeditev za vse možne oznake (klasifikacija). Slednja porazdelitev je predstavljena z vektorjem realnih vrednosti:

$$h_\Theta(x) = [P(oznaka_1|\Theta, x), P(oznaka_2|\Theta, x), ..., P(oznaka_L|\Theta, x)]^T$$

Elementi te porazdelitve se seštejejo v 1. V zgornji enačbi je izhodni vektor umetne nevronske mreže določen kot funkcija vhoda $h_\Theta(x)$.

Zadnji nivo usmerjene nevronske mreže vselej predstavlja izhod glede na dani vhod (prvi nivo). Če so parametri modela Θ definirani (mreža je naučena), lahko za poljuben vhodni vektor x izračunamo njegov izhodni vektor $h_\Theta(x) = y^{(N-1)}$. To storimo z razširjanjem vhodnih vrednosti proti izhodu. Algoritem razširjanja je podan s psevdokodo 1.

2.5 Algoritem učenja

Učenje usmerjene nevronske mreže je formalno določeno z naslednjim postopkom. Poiskati je potrebno take parametre modela Θ, da bo cenitvena funkcija $J(\Theta)$ na učnih podatkih minimalna. Za učne podatke velja, da poznamo vhodni vektor x in njihovo oznako oz. razred (vektor y). Pri usmerjenih nevronskih mrežah se uporablja več cenitvenih funkcij. V strojnem

Input: $x = [x_0, x_1, ..., x_{\mathcal{D}}]^T$
Output: $h_\Theta(x) = [h_\Theta(x_0), h_\Theta(x_1), ..., h_\Theta(x_L)]^T$
  $y^{(0)} \leftarrow x$
  for $i \leftarrow 1$ **to** $N-1$ **do**
    $y^{(i)} \leftarrow g(W^{(i)}y^{(i-1)} + b^{(i)})$
  end for
  return $h_\Theta(x) = y^{(N-1)}$

Algoritem 1: Razširjanje vhodnih vrednosti proti izhodnemu nevronu

učenju se pogosto uporablja srednja kvadratna napaka, ki pa za klasifikacijske probleme v resnici ni najbolj primerna [21]. Zaradi dobre sinergije z metodo najhitrejšega spusta (opisano v poglavju 2.5.3) smo v diplomski nalogi uporabili cenitveno funkcijo imenovano **navzkrižna entropija**. Naj bo $h_\Theta(x)$ izhodni vektor vrednosti umetne nevronske mreže za vhod x. Če imamo v učni množici m primerov in je $t^{(k)}$ vektor oznake k-tega primera $x^{(k)}$ (predstavljen na način, definiran v formuli (2.4)), $h_\Theta(x^{(k)})$ pa izhod umetne nevronske mreže za dan primer, potem je navzkrižna entropija podana kot:

$$J(\Theta) = \frac{1}{m}\sum_{k=1}^{m}\sum_{j=1}^{L}[t_j^{(k)} log(h_\Theta(x^{(k)})_j) + (1 - t_j^{(k)}) log(1 - h_\Theta(x^{(k)})_j)] \quad (2.5)$$

Drugačno notacijo izhodnega vektorja umetne nevronske mreže ($t^{(k)}$ namesto y) na tem mestu uporabljamo zato, da ne mešamo nivojev nevronske mreže in učnih primerov.

Za minimizacijo cenitvene funkcije lahko načeloma uporabimo poljubno optimizacijsko metodo. V diplomski nalogi smo zaradi hitrosti in preprostosti uporabili **paketno metodo najhitrejšega spusta**. Metoda najhitrejšega spusta je optimizacijska metoda, ki parametre modela popravlja glede na njihove odvode. Zato poleg funkcije $J(\Theta)$ zahteva še odvode vseh parametrov:

$$\frac{\partial J(\Theta)}{\partial W_{ij}^{(l)}} \text{ za } \forall i, j, l$$
$$\frac{\partial J(\Theta)}{\partial b_i^{(l)}} \text{ za } \forall i, l \quad (2.6)$$

Izračun teh odvodov je predmet naslednjega podpoglavja.

2.5.1 Vzvratno razširjanje napake

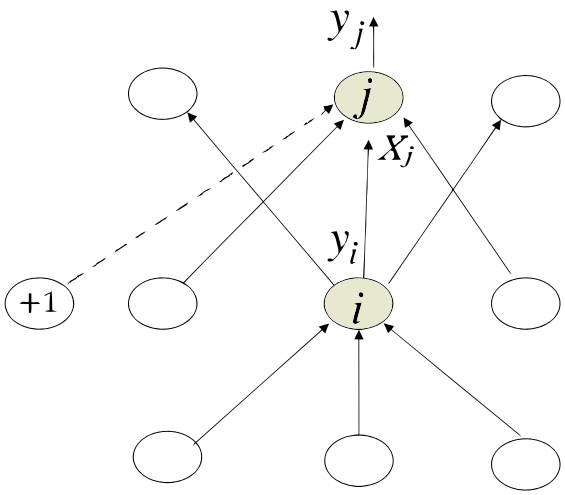
Vzvratno razširjanje napake (včasih poimenovano tudi posplošeno pravilo delta) omogoča učenje globokih usmerjenih umetnih nevronskih mrež. Algoritem je leta 1986 razvil D. Rumelhart [24]. Ideja vzvratnega razširjanja napake je, da za učenje uteži ne uporabljamo izhodnih funkcij nevronov, pač pa odvode napak njihovih izhodov. Intuitivno - mreža ne ve, kaj točno naj bi nevroni počeli, lahko pa izračuna, kako hitro se spreminja napaka, če spreminja izhodne vrednosti. Algoritem začne izvajanje na zadnjem nivoju in se vrača proti prvemu, vhodnemu nivoju. Za vsak nivo na tej poti izračuna odvode napak izhodnih funkcij vseh nevronov na tem nivoju. Na podlagi odvodov napak izhodnih funkcij pa je preprosto izpeljati odvode napak uteži, le-te pa potrebujemo za spreminjanje parametrov modela.

Na sliki 2.5 je predstavljen en korak vzvratnega razširjanja napake. Če predpostavimo, da iz prejšnjega koraka računanja poznamo $\frac{\partial J(\Theta)}{\partial y_j}$ in če x_j predstavlja vse vhodne vrednosti v nevron j, lahko najprej izračunamo odvod cenilne funkcije glede na x_j, kot je predstavljeno v enačbi (2.7). Odvod $g'(x_j)$ je odvisen od izhodne funkcije (tipa) nevrona. Za sigmoidno funkcijo je npr. izredno preprost: $g'(x_j) = y_j(1 - y_j)$. Algoritem za naslednji korak (nivo) razširjanja potrebuje $\frac{\partial J(\Theta)}{\partial y_i}$, ki se izračuna po formuli (2.8). Končno, izračun odvodov parametrov modela (za utež W_{ij}, ki obtežuje povezavo med nevronoma i in j), podajajo enačbe:

$$\delta_j = \frac{\partial J(\Theta)}{\partial x_j} = \frac{dy_j}{dx_j}\frac{\partial J(\Theta)}{\partial y_j} = g'(x_j)\frac{\partial J(\Theta)}{\partial y_j} \tag{2.7}$$

$$\frac{\partial J(\Theta)}{\partial y_i} = \sum_j \frac{dx_j}{dy_i}\frac{\partial J(\Theta)}{\partial x_j} = \sum_j W_{ij}\delta_j \tag{2.8}$$

$$\frac{\partial J(\Theta)}{\partial W_{ij}} = \frac{\partial z_j}{\partial W_{ij}}\frac{\partial J(\Theta)}{\partial z_j} = y_i\delta_j$$



Slika 2.5: Korak vzvratnega razširjanja napake.

Kot se izkaže, si lahko konstantne faktorje uteži b_j predstavljamo kot dodaten nevron na nivoju i ($i = j-1$), ki ima konstanten izhod $+1$, nobenega vhoda ter povezave do vseh nevronov v nivoju j. Ideja je prikazana na sliki 2.5. Tako se na preprost način izračunajo tudi odvodi konstantnih uteži b (glej enačbo (2.6)).

Struktura algoritma za vzvratno razširjanje napake na posameznem učnem primeru je podana v psevdokodi 2. Konstantni parametri b so realizirani na način, prikazan v sliki 2.5.

2.5.2 Ponastavitev uteži

S paketno metodo najhitrejšega spusta računamo odvode parametrov modela. Vendar pa parametrov modela Θ na začetku učenja ne poznamo. Potrebno jih je ponastaviti na neke začetne vrednosti. Pri tem moramo biti nekoliko pazljivi. Če imata dva nevrona enake vhodne, izhodne uteži in konstantni faktor b, potem bodo tudi njuni odvodi vselej enaki. Tudi med učnim procesom se tako ne bosta nikoli razlikovala. V praksi to pomeni, da se ne bosta nikoli naučila predstavljati dveh različnih značilk. To simetrijo

Input: $x = [x_0, x_1, ..., x_{\mathcal{D}}]^T, y = [y_0, y_1, ..., y_L]^T$
Output: $wder_{ij}^{(l)}\ \forall i, j, l$
  $y^{(0)} \leftarrow x$ ▷ Start of forward_prop
  for $i \leftarrow 1$ **to** $N - 1$ **do**
    $y^{(i)} \leftarrow g(W^{(i)}y^{(i-1)} + b^{(i)})$
  end for ▷ End of forward_prop
  $\delta^{(N-1)} \leftarrow y^{(N-1)} - y$ ▷ Start of back_prop
  for $i \leftarrow N - 2$ **down to** 1 **do**
    $\delta^i \leftarrow g'(x^{(i)})W^{(i+1)}\delta^{(i+1)}$
  end for ▷ End of back_prop
  for all i, j, l **do**
    $wder_{ij}^{(l)} \leftarrow y_i^{(l)}\delta_j^{(l+1)}$
  end for
  return $wder$

Algoritem 2: Vzvratno razširjanje napake

razbijemo tako, da ob pričetku učenja uteži ponastavimo na neke majhne, naključne vrednosti.

Za inicializacijo uteži je pomembno tudi število povezav v in iz posameznega nevrona (angl. *fan-in* in *fan-out*) [18]. Če ima nevron veliko vhodnih povezav, morajo biti uteži manjše, saj sicer lahko njihova vsota doseže prevelike vrednosti. Priporočen interval, na katerem se naključno ponastavi uteži $W_{ij}^{(l)}$, je:

$$uniform[-\frac{\sqrt{6}}{\sqrt{fan_{in} + fan_{out}}}, \frac{\sqrt{6}}{\sqrt{fan_{in} + fan_{out}}}] \tag{2.9}$$

Za sigmoidne nevrone je ponastavitev nekoliko drugačna:

$$uniform[-4\frac{\sqrt{6}}{\sqrt{fan_{in} + fan_{out}}}, 4\frac{\sqrt{6}}{\sqrt{fan_{in} + fan_{out}}}] \tag{2.10}$$

V diplomski nalogi smo uporabili **zvezno enakomerno porazdelitev** za naključno ponastavljanje uteži. Vse konstantne uteži $b_i^{(l)}$ smo ob začetku učenja ponastavili na vrednost 0.

2.5.3 Paketna metoda najhitrejšega spusta

Paketna metoda najhitrejšega spusta je različica metode najhitrejšega spusta, kjer odvode izračunamo na majhnem izboru učnih podatkov ("paketu") ter nato popravimo parametre v skladu z odvodi. Naj bo celotna učna množica primerov označena z D. Metoda celotno učno množico primerov razdeli na tri podmnožice:

- učno množico D_{train},
- validacijsko množico D_{valid} in
- testno množico D_{test}.

Učna množica se uporablja za dejansko učenje parametrov modela. V diplomski nalogi smo uporabili varianto metode najhitrejšega spusta s **predhodnim ustavljanjem** (glej poglavje 2.6). Algoritem v tem primeru na validacijski množici izračuna napako. Za izračun le-te uporabi cenitveno funkcijo. Testna množica je lahko tudi prazna; uporablja se za oceno napake modela na neodvisnih podatkih (ravno tako algoritem na njej izračuna cenitveno funkcijo).

V diplomski nalogi smo celotno učno množico podatkov razdelili na D_{train}, D_{valid} in D_{test}. Pri tem smo uporabili **stratifikacijo** - izbor je bil tak, da je bila porazdelitev oznak v vsaki podmnožici čim bolj podobna porazdelitvi oznak v celotni učni množici D. Celotni postopek učenja je predstavljen z algoritmom 3.

Seveda je velikost paketov dodaten parameter učnega algoritma. Velikost paketov je odvisna od velikosti in domene nabora podatkov. V diplomski nalogi smo uporabili pakete z 20-imi primeri.

2.6 Regularizacija

Velik problem umetnih nevronskih mrež je preveliko prilagajanje učnim podatkom (angl. *over-fitting*). V praksi to pomeni, da se algoritem izredno

Input: D
Output: Θ
  $D_{train}, D_{valid}, D_{test} \leftarrow$ construct_datasets(D)
  $\Theta \leftarrow$ initialize_weights()
  $patience \leftarrow 0$
  $cost_{best} \leftarrow \infty$
  while $patience <$ MAX_PATIENCE **do**
    for $minibatch$ **in** D_{train} **do**
      $\Theta' \leftarrow$ train_model($minibatch, \Theta$) ▷ Calc. gradients with back_prop
      $\Theta \leftarrow$ update_model(Θ, Θ', LEARNING_RATE)
      $cost \leftarrow 0, n \leftarrow 0$
      for $minibatch_2$ **in** D_{valid} **do**
        $cost \leftarrow cost +$ validate_model($minibatch_2, \Theta$), $n \leftarrow n + 1$
      end for
      $cost_{avg} \leftarrow \frac{cost}{n}$
      if $cost_{avg} < cost_{best}$ **then**
        $cost_{best} \leftarrow cost_{avg}, patience \leftarrow 0$
      else
        $patience \leftarrow patience + 1$
        if $patience \geq MAX\_PATIENCE$ **then**
          exit
        end if
      end if
    end for
  end while

Algoritem 3: Paketna metoda najhitrejšega spusta

dobro obnese na učnih podatkih, pri tem pa se nauči razpoznavati tudi šum oz. napake, ki so se pojavile pri zajemu učnih podatkov. Tega pri realnih podatkih ni. Sploh pri **globokih nevronskih mrežah**, kjer imamo več kot en skriti nivo, je to zelo pereč problem. Globoke nevronske mreže so bile do nedavnega možne in učinkovite le v primeru, da je bila učna množica res velika, na majhnih podatkovnih naborih pa so se obnesle slabše kot večino preprostejših klasifikatorjev.

Regularizacija je postopek, pri katerem se želimo izogniti prevelikemu prileganju podatkov. V diplomski nalogi smo za regularizacijo uporabili kombinacijo tehnik L_1/L_2, predhodno ustavljanje in metodo izločanja. Predvsem metoda izločanja se je v zadnjem letu, odkar je bila razvita, uveljavila kot odličen regularizator, tudi ko učimo globoke nevronske mreže na majhnem naboru podatkov [10].

2.6.1 Metode

Kar nekaj metod za regularizacijo umetnih nevronskih mrež obstaja že dolgo, nekatere pa so novejšega nastanka. Med splošne metode spadajo L_1/L_2 regularizacija, predhodno ustavljanje, metoda momenta, metoda *rmsprop*, deljenje uteži in druge. Spodaj predstavljamo samo pristope, ki smo jih dejansko implementirali v pričujoči nalogi.

L_1/L_2 regularizacija uteži

Pri tej metodi želimo preprečiti, da bi uteži postale (pre)velike. To storimo tako, da v cenitveno funkcijo dodamo dva člena, ki vsebujeta vsoto vseh uteži in vsoto vseh kvadratov uteži. Prav tako uvedemo dva parametra, L_1 in L_2, s katerima določamo, v kakšni meri želimo, da metoda vpliva na cenitveno funkcijo. Enačbo (2.5) tako razširimo v:

$$J(\Theta) = \frac{1}{m}\sum_{k=1}^{m}\sum_{j=1}^{L}[t_j^{(k)} log(h_\Theta(x^{(k)})_j) + (1 - t_j^{(k)}) log(1 - h_\Theta(x^{(k)})_j)] + L_1 \sum_i \sum_j \sum_l |W_{ij}^{(l)}| + L_2 \sum_i \sum_j \sum_l (W_{ij}^{(l)})^2$$

Ker pri metodi najhitrejšega spusta minimiziramo cenitveno funkcijo, bomo z dodatnimi členi, ki vsebujejo uteži, želeli le-te zmanjšati. Seveda se posledično spremenijo tudi odvodi cenitvene funkcije po posameznih parametrih (utežeh) modela. Tej metodi se reče tudi zmanjševanje uteži (angl. *weight-decay*).

Predhodno ustavljanje

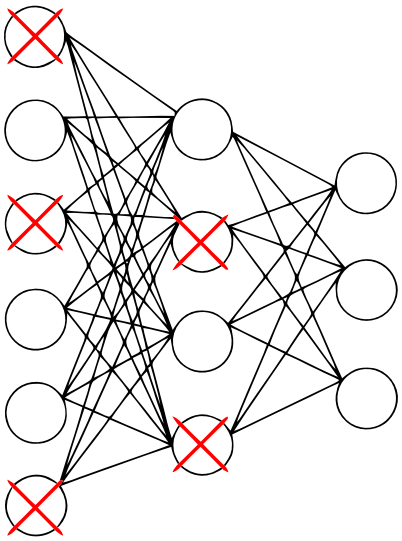
Predhodno ustavljanje temelji na ideji, da učni algoritem ustavimo že na nekem vmesnem koraku učenja. Drugače povedano, ne želimo, da cenitvena funkcija postane premajhna. Predhodno ustavljanje se navadno implementira tako, da se algoritem učenja ustavi, če se v določenem številu zaporednih korakov vrednost cenitvene funkcije ne spremeni za več kot neko vrednost. Na ta način smo predhodno ustavljanje implementirali tudi v diplomski nalogi.

2.6.2 Metoda izločanja

Metodo izločanja (angl. *dropout*) je leta 2012 razvil G. E. Hinton s sodelavci [10]. V osnovi je metoda izredno preprosta. V fazi učenja ob vsakem klicu razširjanja in vzvratnega razširjanja napake izpustimo naključnih 50% nevronov v vseh skritih nivojih, kot je prikazano na sliki 2.6. To je najlažje implementirati tako, da na vsakem nivoju naključnim 50% nevronom postavimo izhod y na vrednost 0. Ko je model naučen, v fazi klasifikacije vse uteži modela Θ razpolovimo.

Ta na videz izredno preprosta sprememba ima v ozadju globok pomen. S tem, ko ob vsaki iteraciji razširjanja naključno izpustimo 50% nevronov, vsakič uporabljamo drugačno arhitekturo nevronske mreže, ki pa si z ostalimi deli enake uteži. Tako gre za kombinacijo metode deljenja uteži in metode povprečenja modelov. Hinton je dokazal, da dejansko povprečimo 2^H modelov, kjer je H število vseh skritih nevronov.

Metoda izločanja se lahko uporablja tudi na vhodnih podatkih (vhodnem nivoju usmerjene nevronske mreže). Ker pa tu ne želimo zavreči kar 50% vseh



Slika 2.6: Skica metode izločanja.

učnih podatkov, je odstotek zavrženih nevronov ponavadi nekoliko manjši. V diplomski nalogi smo uporabili vrednost 30%. Princip je sicer popolnoma enak; v fazi učenja ob vsaki iteraciji naključno izločimo 30% vhodnih nevronov. Ob klasifikaciji pa vse izhodne vrednosti vhodnih nevronov y pomnožimo z vrednostjo 0.7.

2.7 Predhodno učenje uteži

Naključna ponastavitev uteži pred začetkom učenja ima določene probleme. Zaradi naključnosti namreč nimamo nobenega zagotovila, da so ponastavljene uteži kjerkoli blizu optimalni rešitvi. Napačna ponastavitev uteži lahko povzroči, da se cenilna funkcija ustali v lokalnem minimumu, iz katerega metoda najhitrejšega spusta ne najde več izhoda. Če pa so začetne uteži že kolikor toliko blizu dejanskemu minimumu cenitvene funkcije, pa ne le zagotovimo boljšo rešitev (boljši lokalni minimum ali pa celo globalni minimum), temveč tudi pohitrimo čas učenja. Metoda najhitrejšega spusta potrebuje namreč manj korakov, da doseže minimum cenilne funkcije.

Toda kako ponastaviti uteži na pravilne vrednosti? Tehnika, ki smo jo uporabili v diplomski nalogi, izkorišča t.i. nenadzorovano učenje. Cilj nenadzorovanega učenja je ravno tako učenje modela, le da pri tovrstnem učenju za učne primere nimamo na voljo oznak. Model želimo naučiti le, da bi znal razločiti, ali je dan primer podoben tistim v učni množici ali ne. To dejansko pomeni, da se tak model nauči razpoznavati neke zakonitosti in povezave (značilke), ki veljajo le za primere v učni množici.

Ravno tako kot pri učenju je seveda pomembno, da je algoritem za ponastavitev začetnih vrednosti hiter; če je neučinkovit v primerjavi z metodo najhitrejšega spusta, nam ne koristi preveč, saj celoten postopek učenja še dodatno podaljša. V diplomski nalogi smo uporabili dva različna modela, s katerima smo ponastavili začetne vrednosti uteži modela W in b.

2.7.1 Skladanje avtomatskih enkoderjev z odstranjevanjem šuma

Avtomatski enkoder je enota, ki vzame vhodni vektor $x = [x_1, x_2, ..., x_n]^T$ in ga preslika v skrito predstavitev oz. kod $y = [y_1, y_2, ..., y_m]^T$. Pri tem Preslikavo (*kodiranje*) opisuje spodnja enačba, kjer g predstavlja neko nelinearno funkcijo:

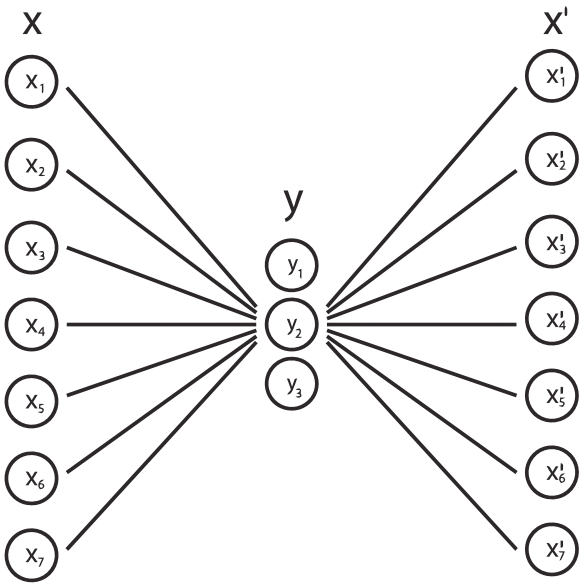
$$y = g(Wx + b)$$

Kod y je nato preslikan nazaj v rekonstrukcijo vektorja x, x' (*dekodiranje*):

$$x' = g(W'y + b')$$

Tudi v tej enačbi g predstavlja nelinearno funkcijo. V primeru deljenih uteži velja enakost $W' = W^T$, kar smo uporabili tudi v diplomski nalogi. Sicer pa so lahko uteži povsem različne. Parametri modela so v tem primeru W, W', b in b'. Primer avtomatskega enkoderja je prikazan na sliki 2.7.

Pri avtomatskih enkoderjih želimo, da je rekonstrukcija vektorja x, označena z x', čim bolj točna. Zato želimo minimizirati rekonstrukcijsko napako. V



Slika 2.7: Primer avtomatskega enkoderja. Zaradi preglednosti so narisane le povezave za y_2.

primeru umetnih nevronskih mrež je najbolj v uporabi navzkrižna entropija:

$$J(\Theta) = \sum_{k=1}^{n} x_k log(x'_k) + (1 - x_k) log(1 - x'_k)$$

Če izračunamo še odvode parametrov modela Θ, lahko uporabimo zelo podobno metodo najhitrejšega spusta, kot je opisana v poglavju 2.5.3 za izračun vrednosti parametrov modela W, b in b' (če uporabljamo deljene uteži). Pri tem uporabimo začetno ponastavitev uteži, opisano v enačbah (2.9) in (2.10). Želja pri učenju avtomatskih enkoderjev je ta, da kod y predstavlja strnjeno interpretacijo vhodnega vektorja x. Ker pri tem gre za kompresijo z izgubami, učenje stremi k temu, da bo rekonstrukcijska napaka za učne primere majhna, za signifikantno različne primere pa velika.

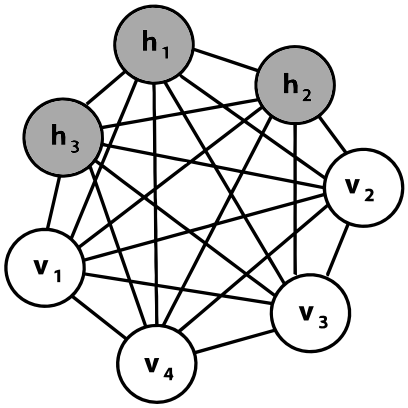
Problem, ki nastane, če je kodni vektor daljši od vhodnega vektorja ($m > n$) je, da se tekom učenja avtomatski enkoder lahko nauči funkcijo identitete I, ki x preslika v samega sebe ($y = x$). V tem primeru se seveda model ni

naučil kakršnekoli predstavitve podatkov. Rešitev za to je preprosta - vhodni vektor popačimo s šumom. Če naključno število vrednosti vhodnega vektorja ob vsakem učnem primeru nadomestimo z 0, dobimo **avtomatski enkoder z odstranjevanjem šuma**. Tak enkoder se namreč trudi rekonstruirati s šumom izgubljene podatke iz neizgubljenih podatkov. Količina šuma je parameter učnega algoritma in odvisna od problema do problema.

V članku [28] so avtomatske enkoderje prvič uporabili tudi v kontekstu umetnih nevronskih mrež. Avtomatske enkoderje skladamo tako, da za vsak skriti nivo v usmerjeni nevronski mreži zgradimo avtomatski enkoder z deljenimi utežmi, ki ima enake dimenzije vhodnega in kodnega vektorja kot trenutni nivo nevronov in en nivo pred njim v nevronski mreži. Zaželjeno je, da avtomatsko enkoder uporablja enako nelinearno funkcijo kot umetna nevronska mreža. Ravno tako si na posameznem nivoju avtomatski enkoder in umetna nevronska mreža delita uteži W in b (uteži b' nam za nevronsko mrežo ne pridejo prav). Uteži sprva ponastavimo na vrednosti, opisane v enačbah (2.9) in (2.10). Učenje avtomatskih enkoderjev začnemo z nivojem 0; ko naučimo prvi avtomatski enkoder, z naučenimi parametri W in b še enkrat izračunamo kod y; ta kod predstavlja vhodni vektor podatkov x za naslednji avtomatski enkoder. Poleg tega naučeni parametri W in b predstavljajo začetno ponastavitev vrednosti na tem nivoju v usmerjeni nevronski mreži. Na ta način se algoritem sekvenčno sprehodi po vseh nivojih usmerjene nevronske mreže od prvega do zadnjega in izračuna začetne vrednosti vseh parametrov W in b v nevronski mreži.

2.7.2 Skladanje omejenih Boltzmannovih naprav

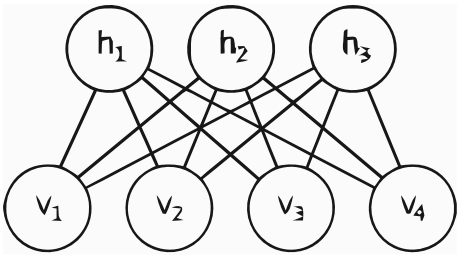
Boltzmannove naprave so stohastične, rekurzivne umetne nevronske mreže. So stohastična različica vzročnih rekurzivnih nevronskih mrež, ki jih imenujemo tudi **Hopfieldove mreže**. Za oba modela je značilno, da imata simetrične povezave, ter da modelirata binarne podatke. Drugače povedano, Boltzmannove naprave in Hopfieldove mreže vsebujejo **binarne pragovne nevrone**; vsak tak nevron ima izhodno vrednost 0 ali 1. Velja torej



Slika 2.8: Primer Boltzmannove naprave s tremi skritimi nevroni ($\{h_1, h_2, h_3\}$) in štirimi vidnimi nevroni ($\{v_1, v_2, v_3, v_4\}$).

$g(z) \in \{0, 1\}$. Rekurzivne nevronske mreže imajo več načinov uporabe. V diplomski nalogi smo jih uporabili za potrebe klasifikacije in nenadzorovanega učenja. Tako smo s tem v mislih tudi ustrezno določili model. Pri rekurzivnih nevronskih mrežah je nemogoče definirati nivoje tako kot pri usmerjenih nevronskih mrežah. Zato za namene klasifikacije vse nevrone razdelimo na vidne oz. vhodne nevrone (njihovo število je enako številu atributov v podatkih) ter skrite nevrone. Za razliko od usmerjenih nevronskih mrež je pri rekurzivnih mrežah zelo pomembno stanje nevronov; rekurzivne mreže se namreč spreminjajo skozi čas (zato so Hopfieldove mreže med drugim primerne za realizacijo pomnilnika). Primer Boltzmannove naprave je prikazan na sliki 2.8.

Problem Boltzmannovih naprav je, da jih je zaradi rekurzivne arhitekture in učnega algoritma (glej spodaj) težko naučiti. Model mnogo lažje naučimo, če omejimo povezave med nevroni. **Omejena Boltzmannova naprava** je vrsta Boltzmannove naprave, kjer so edine obstoječe povezave med vidnimi in skritimi nevroni. Vse povezave so simetrične. Ne obstaja pa nobena povezava, ki bi povezovala vidni nevron z drugim vidnim nevronom ali skritega s skritim. Primer omejene Boltzmannove naprave je prikazan na sliki 2.9.



Slika 2.9: Primer omejene Boltzmannove naprave s 3 skritimi in 4 vidnimi nevroni.

Boltzmannove naprave so za razliko od običajnih nevronskih mrež *energijski modeli*. Vsaka konfiguracija nevronov (vsak nevron je bodisi v stanju 0 ali 1) ima neko skupno energijo. Izkaže se, da lahko z uporabo lokalnega posodabljanja vrednosti nevronov energijsko funkcijo minimiziramo. Cilj učenja omejenih Boltzmannovih naprav je poiskati take parametre modele W in b, da bo verjetnost vhodnih podatkov $p(v)$ čim večja. To pomeni, da iščemo minimum funkcije $-p(v)$. Izkaže se, da je zaradi računanja bolje uporabiti logaritemsko funkcijo. Tako iskano cenitveno funkcijo podaja spodnja enačba:

$$J(\Theta) = -log(p(v))$$

Tudi pri omejenih Boltzmannovih napravah je cilj učenja minimizacija cenitvene funkcije $J(\Theta)$.

Naj bodo stanja vhodnih (vidnih) nevronov podana v vektorju $v = [v_1, v_2, ..., v_n]$ in stanja skritih nevronov podana v vektorju $h = [h_1, h_2, ..., h_m]$. Ker obsega učni algoritem več korakov (najbolj intuitivno je, če si jih predstavljamo kot zaporedni časovni koraki), naj bosta $v^{(t)}$ in $h^{(t)}$ vrednosti omenjenih vektorjev v koraku t. Naj bo W matrika uteži za posamezne povezave (vse povezave so dvosmerne) ter b vektor konstantnih uteži za posamezen nevron. Če na vidne nevrone "položimo" nek vhodni vektor (ki seveda vsebuje samo vrednosti 0 in 1), lahko na preprost način izračunamo pričakovano verjetnost, da bo posamezen skriti nevron zasedel stanje 1, kot je prikazano v spodnji

enačbi:

$$P(h_j = 1|v) = \frac{1}{1 + e^{-(b_j + \sum_i v_i W_{ij})}} \tag{2.11}$$

To je mogoče ravno zato, ker so vsi skriti in vidni nevroni med seboj neodvisni. V diplomski nalogi smo, tako kot drugje, tudi za omejene Boltzmannove naprave uporabili sigmoidno funkcijo za nelinearno funkcijo.

Zelo podobna enačba velja, če določimo skrite nevrone in nas zanima verjetnost, da bo posamezen vidni nevron zasedel stanje 1:

$$P(v_i = 1|h) = \frac{1}{1 + e^{-(b_i + \sum_j h_j W_{ji})}} \tag{2.12}$$

Sledi opis algoritma za učenje omejenih Boltzmannovih naprav. Najprej na vidne nevrone $v^{(0)}$ "položimo" primer iz učne množice. Če atributi nimajo binarnih vrednosti, jih normaliziramo na interval $[0, 1]$ ter s tako verjetnostjo dodelimo vhodnim nevronom vrednost 1. Nato z uporabo enačbe (2.11) izračunamo skriti vektor $h^{(0)}$. Na podlagi skritega vektorja lahko nato z enačbo (2.12) rekonstruiramo vhodni vektor, $v^{(1)}$. Ko gre $t \to \infty$ in ponavljamo opisana koraka, se izkaže, da $(v^{(t)}, h^{(t)})$ konvergirata proti distribuciji $p(h, v)$, ki je določena s parametri W in b. Postopku se reče tudi **Gibbsovo vzorčenje**. Za učenje z metodo najhitrejšega spusta tedaj potrebujemo le še izračun odvodov $\frac{\partial J(\Theta)}{\partial W_{ij}}$ in $\frac{\partial J(\Theta)}{\partial b_i}$, ki pa je precej kompleksen [14]. V diplomski nalogi se z analitičnim računanjem odvodov zaradi knjižnjice Theano nismo ukvarjali (glej poglavje 3).

Hinton je dokazal, da je za učenje parametrov modela dovolj, če izračunamo le en korak Gibbsovega vzorčenja (do $t = 1$). Postopek se imenuje **kontrastna divergenca** [9]. V tem primeru dejansko ne izračunamo odvodov $\frac{\partial J(\Theta)}{\partial W_{ij}}$ in $\frac{\partial J(\Theta)}{\partial b_i}$, ampak zgolj nek približek, s katerim posodabljamo uteži W in b. Tehnika v praksi ravno tako deluje (čeprav se računanje ne vrši striktno po metodi največjega verjetja).

Učni algoritem je tedaj ponovno podoben tistemu, opisanemu v poglavju 2.5.3. Algoritem iterativno na vhodne vektorje $v^{(0)}$ "položi" primere iz učne množice, izvede korak Gibbsovega vzorčenja, ter na podlagi odvodov (oz. aproksimacij) popravi parametre modela W in b.

Sistem, kjer za predhodno učenje uteži globoke nevronske mreže uporabimo omejene Boltzmannove naprave, je Hinton poimenoval **globoke verjetnostne mreže**. Ideja je zelo podobna kot pri avtomatskih enkoderjih, opisanih v podpoglavu 2.7.1. Uteži W in b ponastavimo na vrednosti, kot je opisano v enačbah (2.9) in (2.10). Nato se sprehodimo čez vse nivoje usmerjene nevronske mreže in za vsak posamezen nivo uporabimo učenje omejene Boltzmannove naprave. Ta si deli uteži W in b z usmerjeno nevronsko mrežo. Število skritih nevronov je enako kot število nevronov v nevronski mreži na trenutnem nivoju, število vidnih nevronov pa je enako številu nevronov na predhodnem nivoju mreže. Omejeno Boltzmannovo napravo nato naučimo, s čimer že ponastavimo uteži v nevronski mreži. Izračunan skriti vektor h nato predstavlja vhodni vektor v za naslednji nivo omejene Boltzmannove naprave.

Poglavje 3

Implementacija

Kot smo že omenili v poglavju 2, so nevronske mreže matematični model. Za izvajanje jih je potrebno implementirati na računalniku. Medtem ko so bile prvotne umetne nevronske mreže (perceptroni) implementirane na nivoju take ali drugačne strojne kode, se danes za implementacijo uporabljajo številni programski jeziki. V diplomski nalogi smo se odločili, da za implementacijo uporabimo visokonivojski programski jezik *Python*. Razlogov je kar nekaj:

- *Python* je skriptni programski jezik. Ta lastnost je odlična za hiter razvoj programske kode, saj omogoča sprotno preverjanje funkcij in spremenljivk.
- Sintaksa jezika je zelo sorodna psevdokodi, kar omogoča intuitivno implementacijo matematičnih algoritmov.
- Na voljo je velik nabor knjižnic, ki močno olajšajo delo z matematičnimi izračuni in vsebujejo vse potrebno za implementacijo algoritmov s področja strojnega učenja (npr. NumPy, SciPy, Theano in Orange).

Umetne nevronske mreže je seveda možno implementirati v samem jeziku *Python* brez dodatnih knjižnic. Vendar bi bilo to zelo zamudno opravilo. V diplomski nalogi smo pri implementaciji želeli ugoditi predvsem sledečim dvem zahtevam:

1. Čim hitrejše učenje in klasifikacija umetnih nevronskih mrež, po možnosti s procesiranjem na grafični procesni enoti (GPE).

2. Elegantna programska implementacija algoritmov, predvsem vzvratnega razširjanja napak, za vse različne arhitekture umetnih nevronskih mrež, predstavljenih v poglavju 2.3.

Obema zahtevama zadosti *Pythonovska* knjižnica Theano. Zato je le-ta opravila najpomembnejšo vlogo pri sami implementaciji.

V diplomski nalogi smo uporabili tudi programski paket Orange[1]. Orange je orodje za vizualizacijo in modeliranje podatkov in vsebuje velik nabor komponent in algoritmov s področja podatkovnega rudarjenja in strojnega učenja. Hkrati nudi vmesnik API za programski jezik Python, s katerim lahko do njegovih algoritmov in struktur dostopamo tudi programsko. V diplomski nalogi smo ga uporabili za učenje drugih napovednih modelov - metode podpornih vektorjev, logistične regresije in naključnih gozdov - ter za predobdelavo podatkov. Seveda se v primerjavi s knjižnico Theano koda iz paketa Orange izvaja na CPE.

3.1 CUDA

Transparentna uporaba grafične procesne enote je izrednega pomena. Učenje globokih umetnih nevronskih mrež je računsko zelo potratno opravilo. Čeprav je za vse različne implementacije težko določiti časovno zahtevnost, učenje mrež tudi na najsodobnejših osebnih računalnikih lahko traja več ur ali celo dni. Kakršnakoli pospešitev je zato izredno dobrodošla. Grafične procesne enote na grafičnih karticah v osebnih računalnikih so v zadnjih letih napredovale do te mere, da jih lahko s pomočjo ustreznih vmesnikov API uporabimo tudi za potrebe časovno kritičnega računanja (angl. *GPGPU* oz. *general-purpose computing on graphics processing units*). Daleč najbolj dominantni

[1] http://orange.biolab.si

ogrodji za to sta odprtokodni OpenCL[2] in CUDA[3] (angl. *Compute Unified Device Architecture*), ki je v lasti podjetja NVIDIA. Slednjo smo uporabili v okviru diplomske naloge.

Arhitektura CUDA razdeli računalnik v grobem na:

- Centralno procesno enoto (CPE), poimenovano **gostitelj** (angl. *host*).
- Grafično procesno enoto (GPE), poimenovano **naprava** (angl. *device*).

Gre za dve povsem ločeni entiteti, saj ima vsaka svoj pomnilnik ter svoj naslovni prostor. Glavna ideja arhitekture CUDA je, da se v okviru programa zaporedna koda izvaja na gostitelju, paralelna koda pa na napravi.

Arhitektura CUDA podpira mnogo programskih jezikov, med njimi tudi Python, s katerimi lahko programsko dostopamo in uporabljamo grafično procesno enoto. Osnova za vse pa je t.i. *CUDA C*, vmesnik za dostop do grafične kartice v programskem jeziku C. Ta v grobem deli funkcije programa na dva dela:

- **Funkcije** predstavljajo programsko kodo, ki se izvaja na gostitelju.
- **Jedra** (angl. *kernels*) predstavljajo kodo, ki se izvaja paralelno (v številnih nitih) na GPE. Natančneje - arhitektura CUDA združuje skupino niti v blok (angl. *block*), skupino blokov pa v mrežo (angl. *grid*). Jedro se izvaja v okviru mreže.

Blok je najmanjša neodvisna, samostojna enota izvajanja, saj si niti v posameznem bloku delijo pomnilniški prostor, prav tako se lahko usklajujejo med seboj. Trinivojska arhitektura (niti, bloki in mreže) je zelo močno orodje, saj omogoča tako paralelno (in usklajeno) izvajanje znotraj posameznih osnovnih enot (blokov) kot tudi paralelno izvajanje mnogih neodvisnih blokov na enem nivoju višje.

[2]http://www.khronos.org/opencl
[3]http://www.nvidia.com/object/cuda_home_new.html

Na sliki 3.1 predstavljamo arhitekturo CUDA. Skupine jeder grafične procesne enote si delijo pomnilniški prostor, vse pa imajo dostop tudi do globalnega pomnilnika naprave (angl. *GDDRAM*). S centralno procesno enoto je GPE povezana preko vodila (za grafične kartice je trenutno v uporabi vodilo *PCI Express*). Navkljub velikim hitrostim, ki jih to vodilo dandanes omogoča (zadnja verzija, v4, teoretično omogoča hitrosti do 31.51 GB/s v obe smeri), je prenos podatkov iz CPE v GPE ter obratno še vedno najbolj ozko grlo pri arhitekturi CUDA. Dober program v ogrodju CUDA torej mora čim bolj minimizirati količino prenosov med CPE in GPE. Ustaljena praksa je, da se podatki pred začetkom paralelnega računanja v celoti skopirajo na pomnilnik naprave.

Zaradi omenjenih lastnosti izvajanje posameznega jedra poteka v štirih fazah [3]:

1. Prenos podatkov iz pomnilnika gostitelja v pomnilnik naprave (GDDRAM).

2. Klic jedra z gostitelja.

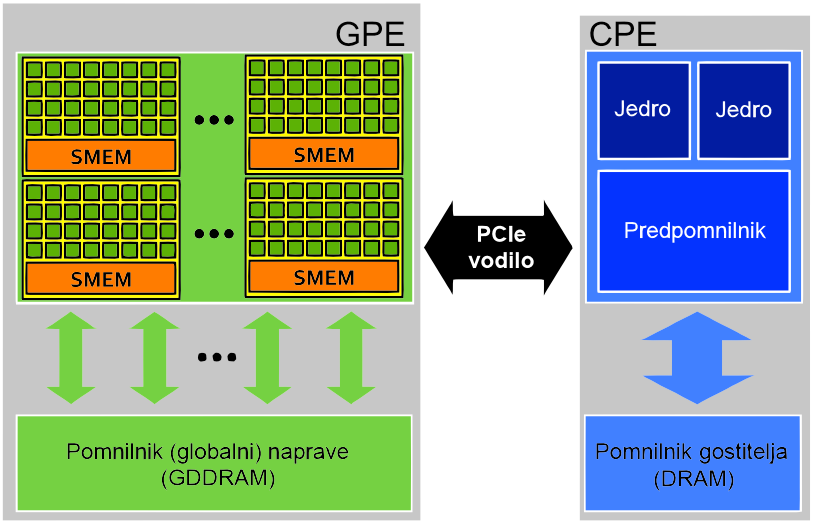
3. Izvajanje jedra na napravi (bloki oz. mreže paralelno izvajajo ukaze, dostopajo do globalnega pomnilnika ter se usklajujejo).

4. Ko je računanje na napravi končano, se podatki prenesejo nazaj v pomnilnik gostitelja.

3.2 Knjižnica Theano

Theano je računska knjižnica za programski jezik *Python*. Razvita je bila z namenom učinkovite implementacije večdimenzionalnih matričnih operacij. Vse lastnosti knjižnice so naštete na uradni strani[4]. Za implementacijo umetnih nevronskih mrež sta najbolj pomembni lastnosti knjižnice transparentna uporaba grafične procesne enote (GPE) za izvajanje časovno kritičnih

[4]http://deeplearning.net/software/theano



Slika 3.1: Arhitektura CUDA. GPE je povezana s CPE preko vodila PCIe.

računskih operacij in simbolična predstavitev matematičnih izrazov in formul [5].

Knjižnica Theano je bila razvita z namenom, da se računanje izvaja na grafični procesni enoti, vendar se lahko v celoti izvaja tudi na centralni procesni enoti. S tem lahko program veliko izgubi na hitrosti (odvisno od programa), še vedno pa knjižnica razvijalcu nudi svojo drugo prednost - simbolične grafe. Knjižnica Theano od uporabnika namreč zahteva, da vse željene izračune poda v simboličnem zapisu. Simbolični zapis nekega izraza je v knjižnjici predstavljen s tremi vrstami objektov:

1. Spremenljivka (angl. *variable*) ima določen tip - lahko gre za skalarje, vektorje ali matrike različnih podatkovnih tipov.

2. Operator (angl. *op*) predstavlja različne tipe matematičnih operacij.

3. Priredba (angl. *apply*) dodeljuje operator eni ali večim spremenljivkam.

Iz zgoraj naštetih objektov knjižnica Theano zgradi **graf operacij**. Za preprost izračun seštevanja dveh števil bi v knjižnici Theano uporabili sledečo Pythonovsko kodo:

```
1 x = T.dmatrix("x")
2 y = T.dmatrix("y")
3 z = x + y
```

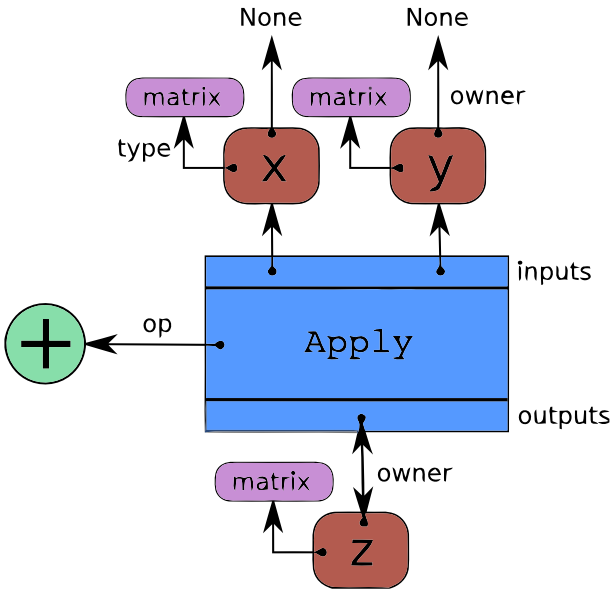
Pri tem se je potrebno zavedati, da elementoma **x** in **y** nismo dodelili še nobene vrednosti; zapis v tretji vrstici predstavlja zgolj navodilo, kako spremenljivko **z** izračunati. Ustrezen graf operacij, ki ga knjižnica zgradi v ozadju, je prikazan na sliki 3.2. Knjižnica nudi pester nabor različnih operatorjev in funkcij, ki jih lahko uporabimo pri simbolični gradnji izrazov. Ni si težko predstavljati, da velikost grafa pri bolj kompleksnih problemih hitro naraste. V diplomski nalogi smo tako zgradili graf, ki je opisoval izračun vrednosti izhodnega nivoja nevronske mreže, vrednosti cenilne funkcije ter vrednosti odvodov parametrov modela; vse to na podlagi dveh spremenljivk, parametrov modela Θ in vhodnega vektorja (začetni nivo nevronske mreže). Uporaba grafa operacij prinaša še dve veliki prednosti:

1. **Izračun odvodov.** Knjižnica Theano zaradi grafa operacij podpira avtomatsko računanje odvodov poljubnega izraza. To stori tako, da se sprehodi po grafu operacij od izhodnih spremenljivk proti vhodnim in sproti upošteva vozlišča s priredbami. V diplomski nalogi smo to lastnost izkoristili za izračun odvodov parametrov modela (Θ).

2. **Optimizacija.** Z analizo grafa operatorjev lahko knjižnica Theano pohitri čas računanja tako, da odstrani odvečne veje v grafu, združi posamezne poti in nadomesti dele grafa z drugimi operatorji, ki so hitrejši ali bolj stabilni.

3.3 Razvita koda

V okviru diplomske naloge smo razvili programsko knjižnico, ki je poleg uporabe na podatkih, predstavljenih v podpoglavju 4.1, uporabna tudi na drugih, poljubnih podatkih iz področja strojnega učenja. Tako se lahko v diplomskem delu opisane metode preverijo tudi na drugih podatkovnih naborih iz



Slika 3.2: Preprost primer grafa operacij. Rdeča vozlišča predstavljajo spremenljivke, zeleno vozlišče je operator, modro vozlišče pa priredba.

drugačnih domen. Knjižnica je napisana v programskem jeziku Python ter izkorišča zgoraj predstavljeni paket Theano za izvajanje kalkulacij na GPE. Del knjižnice, ki je uporaben za poljubne podatke, se nahaja na javno dostopnem repozitoriju[5]. Koda je izdana brez licence, kar pomeni, da ima kdorkoli pravico kodo spreminjati in jo uporabljati brez omejitev.

Knjižnico smo razvili kot Pythonovski projekt v razvojnem okolju *Eclipse*[6]. Razdeljena je na 3 glavne pakete (angl. *package*), ki vsebujejo posamezne module:

.impl vsebuje implementacijo vseh modelov nevronskih mrež, vključno z izboljšavami, predstavljenih v poglavju 2. Ta del je dostopen na javnem repozitoriju in ponuja tudi dokumentacijo.

.data obsega strukture, ki poenostavljajo delo s podatkovnimi nabori.

[5]http://github.com/Ducz0r/NeuralNets
[6]http://www.eclipse.org/

.core združuje kodo, ki predobdela podatkovne nabore, na njih nauči modele nevronskih mrež ter nazadnje primerja in ovrednoti dobljene rezultate klasifikacije.

V jeziku Python ima uporabnik na voljo dva načina razvoja - kodo lahko piše v datoteke (t.i. skripte), lahko pa programira direktno v Pythonovskem tolmaču (angl. *interpreter*). Zaradi ponovljivosti smo kodo iz paketa **.core** spisali kot Pythonovske skripte, medtem ko paketa **.impl** in **.data** vsebujeta objektne strukture, ki so pripravljene za uporabo tudi iz tolmača.

Pythonovska koda je praviloma strnjena in razmeroma kratka, saj izkorišča veliko izrazno moč jezika. Zaradi večje uporabnosti in fleksibilnosti pa je v nasprotju s tem razvita knjižnica precej objektno usmerjena. Zaradi tega razloga in obsežne dokumentacije implementiranih modelov obsega koda 5586 vrstic.

Uporabo knjižnice ponazarjamo s sledečim primerom, ki predstavlja gradnjo in učenje modela nevronske mreže ter klasifikacijo primera po tem, ko je model naučen:

```
1  # Read the train, validation and test data
2  file = open("DATA.pickle", "r")
3  train_set, valid_set, test_set = pickle.load(file)
4
5  # Generate & learn the neural net
6  nn = NeuralNetwork(hidden_levels = [500, 500],
7      in_out = (784, 10),
8      activation = T.tanh, batch_size = 20,
9      dropout_thresholds = (0.8, 0.5), seed = 50)
10
11 nn.learn(train_set, valid_set, test_set)
12
13 # Save the model parameters
14 nn.save_model_to_file("model.pickle")
15
16 # Classify a data example
17 result = nn.classify(example)
```

Zgoraj zgrajena nevronska mreža ima dva skrita nivoja, od katerih vsak obsega 500 nevronov; podatki vsebujejo 784 značilk ter 10 razredov. Vsi možni parametri implementiranih nevronskih mrež in njihovih funkcij so strukturirani in dokumentirani v formatu *PythonDoc*.

3.3.1 Čas izvajanja

Zgoraj predstavljeno kodo smo izvajali na domačem računalniku, ki je imel sledečo strojno in programsko opremo:

- štirijedrni procesor **Intel Core i5 3470**,
- velikost pomnilnika 16GB,
- grafično kartico **GeForce GTX 460**,
- operacijski sistem **Windows 8 64-bit**.

Za dokaz, kako nujno potrebna je implementacija nevronskih mrež z ogrodjem CUDA, smo zgradili preprosto nevronsko mrežo z dvema skritima nivojema (od katerih je vsak vseboval 500 nevronov) ter naučili model na enem izmed podatkovnih naborov, predstavljenih v podpoglavju 4.1. Kot je razvidno v tabeli 3.1, je očitno, da je že sama uporaba knjižnice Theano nujna za učinkovito učenje nevronskih mrež. Pri bolj kompleksnih nevronskih mrežah (ki vsebujejo več skritih nivojev z večjim številom nevronov) ter večjih naborih podatkov, se je izkazalo, da tudi razlika med izvajanjem na GPE in CPE postaja vedno večja. Ker smo v okviru diplomske naloge naučili izredno veliko število nevronskih mrež, smo posledično učenje vseh izvajali na GPE.

Način učenja	**Lokacija**	**Čas izvajanja (v s)**
Brez uporabe knjižnice Theano	CPE	6773.8
Uporaba knjižnice Theano	CPE	65.9
Uporaba knjižnice Theano	GPE	37.1

Tabela 3.1: Primerjava časov izvajanja za različne načine učenja nevronske mreže.

Poglavje 4

Primerjava napovednih modelov

V predhodnih poglavjih so bili predstavljeni podatki, definicija klasifikacijskega modela ter nazadnje še sama implementacija umetnih nevronskih mrež na računalniku. S tem smo pripravili vse, kar je potrebno za samo klasifikacijo oz. učenje napovednih modelov. Zgrajene umetne nevronske mreže smo zatorej naučili na danih podatkovnih naborih. Ker pa je cilj diplomske naloge oceniti uspešnost zgrajenih umetnih nevronskih mrež na izbranih podatkih s področja molekularne biologije, v pričujočem poglavju opišemo, na kakšnih podatkih smo naučili zgrajene nevronske mreže, kako smo izmerili uspešnost posameznih napovednih modelov in kako smo jih primerjali med seboj. Na koncu poglavja sledijo predstavljeni rezultati primerjave modelov.

Cilj primerjave napovednih modelov je bil raziskati, kako se metoda izločanja obnese v primerjavi z navadnimi umetnimi nevronskimi mrežami, ter kako se umetne nevronske mreže (z dodatki, omenjenimi v poglavju 2) obnesejo v primerjavi z uveljavljenimi klasifikatorji. Obe primerjavi smo preverili na naboru bioloških podatkov, opisanem v podpoglavju 4.1. Želeli smo torej *ovreči* sledeči dve ničelni hipotezi:

1. Na danem naboru bioloških podatkov so nevronske mreže brez metode izločanja enako uspešne pri klasifikaciji kot nevronske mreže z metodo

izločanja.

2. Na danem naboru bioloških podatkov so nevronske mreže enako uspešne pri klasifikaciji kot logistična regresija/metoda podpornih vektorjev/naključni gozdovi.

4.1 Seznam podatkovnih množic in njihove osnovne statistike

Za preverjanje uspešnosti zgrajenih nevronskih mrež smo izbrali podatke s področja molekularne biologije. Podatke smo pridobili iz primerjalne študije [27], ki se je ukvarjala z razvrščanjem podatkovnih naborov o genski ekspresiji rakastih obolenj v gruče (angl. *clustering*). Študija je zajela 35 podatkovnih naborov, ki vsebujejo podatke o genski ekspresiji posameznih vzorcev rakastega tkiva in ustreznih oznak. Podatkovni nabori, uporabljeni v študiji, so javno dostopni na Internetu[1]. Na 15 podatkovnih naborih iz te študije smo naučili zgrajene umetne nevronske mreže.

Omenjeni nabori podatkov so bili objavljeni v 25 neodvisnih študijah. Ker so bili biološki vzorci v nekaterih študijah razvrščeni na več načinov v različne razrede rakastih obolenj, je podatkovnih naborov 35. V vsako študijo je bilo vključenih od 22 do 248 različnih bioloških vzorcev, po en vzorec na bolnika. Vsi vzorci v isti študiji so povezani z isto vrsto raka (npr. levkemijo) in izvirajo iz istega tipa tkiva (npr. kri, pljuča, možgani), iz katerega zo izolirali RNA in jo uporabili za profiliranje izražanja genov. Avtorji so vzorce v vsaki študiji razdelili v več razredov, ki predstavljajo različne podtipe raka in ki naj bi se med seboj razlikovali po izražanju genov. Tako so npr. Armstrong s sod. [2] biološke vzorce bolnikov z levkemijo razvrstili v dva tipa, ALL (akutna limfoblastična levkemija) in MLL (akutna mieloična levkemija).

Avtorji so v vseh študijah za zajem podatkov uporabili DNA mikro-

[1] http://bioinformatics.rutgers.edu/Static/Supplements/CompCancer/datasets.htm; nazadnje dostopano 28.11.2013.

Karakteristika	x_{min}	$\bar{x}$	x_{max}	σ
Št. primerov	42	112.73	248	63.05
Št. značilk	85	1441.47	2526	672.97
Št. oznak (razredov)	2	3.33	10	2.05
Št. primerov s posamezno oznako	4	33.82	205	38.8
Delež primerov s posamezno oznako	0.03	0.3	0.83	0.22
Redkost podatkov	0.36	0.7	0.98	0.19

Tabela 4.1: Splošne karakteristike izbranih podatkovnih naborov.

mreže [25]. Vsaka DNA mikromreža nosi nekaj tisoč kratkih DNA sond (angl. *probes*). Za vsako sondo je specifičen del zapisa DNA posameznega gena v človeškem genomu. S tem ko vsak biološki vzorec analiziramo na eni DNA mikromreži, pridobimo informacije o nivoju izražanja za večino genov v tem vzorcu. Izražanje genov je lahko v različnih bolezenskih stanjih različno in se lahko razlikuje tudi med različnimi podtipi iste bolezni. Genski izrazi so lahko osnova za razporejanje bioloških vzorcev v različne bolezenske podtipe (v primeru študij v različne podtipe določene vrste raka).

Nekaj splošnih karakteristik uporabljenih podatkovnih naborov je podanih v tabeli 4.1. Z izrazom redkost podatkov smo poimenovali karakteristiko, ki smo jo dobili na sledeč način. Najprej smo podatke diskretizirali (glej podpoglavje 4.1.1) Za vsak nabor podatkov smo nato poiskali najbolj pogosto vrednost diskretiziranih značilk, redkost podatkov pa predstavlja delež vseh značilk, ki imajo vrednost drugačno od te, najbolj pogoste vrednosti.

4.1.1 Predobdelava

Preden smo zgrajene umetne nevronske mreže uporabili na vseh 15 podatkovnih naborih, smo nekaj zgrajenih nevronskih mrež naučili na enem izmed podatkovnih naborov. Izračunali smo mero AUC, pri čemer smo uporabili različne metode predobdelave podatkov, in nazadnje prišli do kombinacije, ki za zgrajene nevronske mreže izboljša rezultat klasifikacije.

15 danih podatkovnih naborov s področja molekularne biologije smo, preden smo na njih naučili zgrajene nevronske mreže, predobdelali na sledeča dva načina:

Normalizacija je postopek, kjer vse vrednosti podatkovnega nabora transformiramo na željen interval. Za umetne nevronske mreže je normalizacija podatkov bistvenega pomena; brez te tehnike so bili rezultati klasifikacije za zgrajene nevronske mreže izredno slabi. Podatke smo normalizirali na interval $[0, 1]$ z metodo **min-max normalizacije**. Ta tehnika transformira vsako vrednost x v novo vrednost x' na interval $[x_{low}, x_{high}]$, kjer x_{min} predstavlja najmanjšo vrednost podatkov, x_{max} pa največjo vrednost podatkov:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}(x_{high} - x_{low}) + x_{low}$$

Diskretizacija je postopek, kjer realno številske vrednosti značilk transformiramo v diskretne vrednosti. To pomeni, da značilka ne zaseda več poljubne (oz. na interval omejene) realne vrednosti, temveč spada v eno izmed končno mnogo skupin. Podatke smo diskretizirali, ker je za nekatere napovedne modele postopek izboljšal klasifikacijo (glej tudi [17]). Na podatkih smo uporabili privzeto implementacijo **diskretizacije z entropijo** [8] v programskem paketu Orange. Metoda razbija interval vrednosti značilk na podskupine s ciljem minimizacije entropije razreda glede na učne vzorce.

Pri predobdelavi podatkov smo poskusili tudi z različnimi tehnikami izbora značilk (angl. *FSS* oz. *feature subset selection*), implementiranimi v programskem paketu Orange. Ugotovili smo, da na zgrajenih nevronskih mrežah z izborom podmnožice značilk ne izboljšamo klasifikacije, zato smo na vseh podatkih uporabili vse obstoječe značilke.

4.2 Mere uspešnosti

V diplomski nalogi smo za ocenjevanje uspešnosti, podobno kot v mnogih sorodnih študijah na podatkih o genskih izrazih, uporabili **mero AUC** oz. ploščino pod krivuljo ROC. Krivulja ROC se že dolgo časa uporablja kot grafičen prikaz uspešnosti posameznega napovednega modela. Naj problem obsega dve oznaki razreda, $\{P, N\}$. Tedaj krivulja ROC prikazuje razmerje med senzitivnostjo (relativno število pravilno klasificiranih primerov razreda P) in specifičnostjo (relativno število napačno klasificiranih primerov razreda N), ki sta najbolj pomembni meri napovednega modela pri dvorazredni klasifikaciji.

Senzitivnost in specifičnost podajata spodnji enačbi:

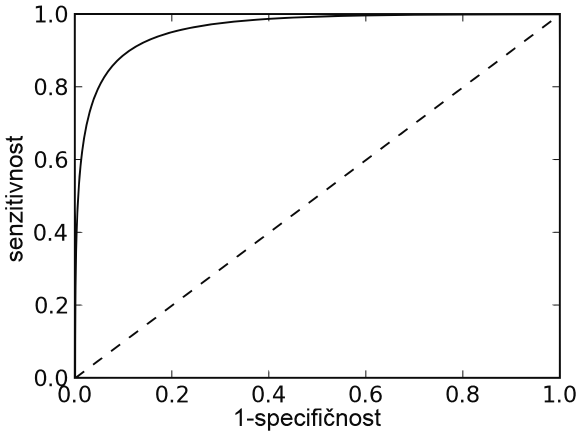
$$Senz = \frac{TP}{POS}$$

$$Spec = \frac{TN}{NEG}$$

Pri tem TP predstavlja število pravilno klasificiranih primerov razreda P, TN število pravilno klasificiranih primerov razreda N, POS število vseh primerov razreda P in NEG število vseh primerov razreda N. Klasifikatorji, ki smo jih uporabili v obsegu diplomske naloge, za vsak učni primer vrnejo par $(p(P), p(N))$, ki predstavlja verjetnost, da je oznaka za dan primer P oz. N prisotna oz. ne. V tem primeru za vsak klasifikator obstaja krivulja ROC. Primer le-te je prikazan na sliki 4.1; na vodoravni osi je prikazana specifičnost, na navpični pa senzitivnost.

Če izračunamo ploščino pod krivuljo ROC, dobimo mero AUC. Ta tudi pove, s kakšno verjetnostjo bo klasifikator primeru razreda P pripisal večjo verjetnost pripadnosti temu razredu kot primeru razreda N. Boljši klasifikator ima torej višjo AUC. Za sam izračun ploščine smo v diplomski nalogi uporabili algoritem, opisan v članku [7].

Pri klasifikaciji, kjer imamo podan več kot en razred, mere AUC neposredno ne moremo uporabiti. V diplomski nalogi smo to rešili tako, da smo



Slika 4.1: Primer krivulje ROC. Črtkana črta predstavlja naključni klasifikator. Bolj se krivulja približa točki $(0, 1)$, boljši je klasifikator.

primerjali posamezne razrede z vsemi ostalimi, jih utežili glede na porazdelitev razreda v učni množici in rezultate povprečili. Naj bo N število vseh primerov v učni množici in N_{oznaka_i} število primerov v učni množici, ki imajo oznako i. AUC_{oznaka_i} je izračun AUC, kjer za pozitivne primere upoštevamo vse primere, ki imajo oznako i, vsi ostali primeri v učni množici pa so negativni. Končna vrednost naše cenilke je potem:

$$AUC = \frac{1}{N} \sum_i \frac{1}{N_{oznaka_i}} AUC_{oznaka_i}$$

4.3 Način testiranja

Za ocenjevanje mere AUC za dani podatkovnem naboru smo uporabili **K-kratno sorazmerno prečno preverjanje** [13], ki je orisano v algoritmu 4. Ta učno množico razdeli na K približno enako velikih podmnožic. Pazimo, da ima vsaka podmnožica približno enako porazdelitev razreda kot celoten nabor. Tej tehniki se reče **stratifikacija**. Nato K-krat, za vsako podmnožico, ki predstavlja testni nabor podatkov, klasifikator naučimo na preostalih $K-1$

podmnožicah in izračunamo mero AUC na K-ti podmnožici. Te rezultate nato seštejemo in povprečimo (delimo s K), da dobimo končen rezultat.

Input: D, K
Output: AUC
$\{D_1, ..., D_K\} \leftarrow$ construct_K_stratified_datasets(D, K)
$AUC \leftarrow 0$
for $i \leftarrow 1$ **to** K **do**
  $\Theta \leftarrow$ train_model$(D \setminus D_i)$
  $AUC \leftarrow AUC +$ calc_auc(Θ, D_i)
end for
$AUC \leftarrow \frac{1}{K} AUC$

Algoritem 4: K-kratno prečno preverjanje

Če imamo na voljo več podatkovnih naborov iz iste domene, je bolje, da napovedne modele primerjamo na večjem številu podatkovnih naborov. Ker nas je zanimalo predvsem obnašanje v poglavju 2.3 opisanih metod v primerjavi z obstoječimi napovednimi modeli, smo za primerjavo napovednih modelov na večih podatkovnih naborih uporabili **Wilcoxonov test predznačenih rangov** [6]. Gre za statistični test, kjer primerjamo uspešnost dveh klasifikatorjev med seboj. Testiramo torej ničelno hipotezo, ki pravi, da sta oba klasifikatorja enako uspešna. Naš cilj je bil ničelno hipotezo zavreči, saj le-to pomeni, da sta klasifikatorja signifikantno različna. Drugače povedano, opažene razlike med primerjanima klasifikatorjema so za določen interval zaupanja statistično signifikantne (niso posledice nekega naključnega procesa). Pri Wilcoxonovem testu najprej izračunamo razliko med mero AUC obeh primerjanih klasifikatorjev (označenih z $AUC^{(1)}$ in $AUC^{(2)}$) za vsak (i-ti) podatkovni nabor:

$$d_i = AUC_i^{(2)} - AUC_i^{(1)}$$

Dobljene razlike d_i nato uredimo po absolutni vrednosti ter jim dodelimo

rang ($rank(d_i)$). Nato izračunamo vrednosti R^+ in R^-:

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i)$$

$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i)$$

Naj bo $T = min(R^+, R^-)$. Tedaj sta klasifikatorja značilno različna, če je $T \leq N_\alpha$, kjer je N_α kritična vrednost, odvisna od zaupanja ter števila podatkovnih naborov.

V diplomski nalogi smo klasifikatorje primerjali na 15 naborih podatkov; za interval zaupanja smo uporabili natančnost $\alpha = 0.05$.

4.4 Rezultati

Sledeče podpoglavje vsebuje rezultate primerjave v predhodnih poglavjih opisanih klasifikatorjev. Pri tem smo s statističnimi metodami želeli ovreči dve ničelni hipotezi, predstavljeni v uvodu poglavja.

4.4.1 Gradnja modelov

V iskanju najboljše umetne nevronske mreže za klasifikacijo danih bioloških podatkov smo v okviru diplomske naloge zgradili:

- 54 navadnih umetnih nevronskih mrež (kratica: **NN**), kjer smo spreminjali sledeče štiri parametre: arhitekturo mreže, vrsto nevronov, hitrost učenja (angl. *learning rate*) in uporabo metode izločanja.

- 36 umetnih nevronskih mrež, kjer smo za predhodno učenje uteži uporabili skladanje avtomatskih enkoderjev z odstranjevanjem šuma (kratica: **NN + SDA**). Spreminjali smo sledeče štiri parametre: arhitekturo mreže, delež šuma na vhodu (angl. *corruption level*), hitrost učenja ter uporabo metode izločanja.

- 36 umetnih nevronskih mrež, kjer smo za predhodno učenje uteži uporabili skladanje omejenih Boltzmannovih naprav (kratica: **NN + SRBM**). Spreminjali smo sledeče štiri parametre: arhitekturo mreže, hitrost učenja Boltzmannovih naprav, hitrost učenja in uporabo metode izločanja.

V sledečem poglavju so v tabelah zaradi lažjega prikaza uporabljene zgoraj podane oznake klasifikatorjev. V celoti razložena predstavitev parametrov testiranja je prikazana v tabeli 4.2.

4.4.2 Metoda izločanja

Izmed vseh 126 različnih modelov umetnih nevronskih mrež, naštetih zgoraj, smo na 120 izmed njih preverjali uspešnost metode izločanja. Teh 120 mrež je razdeljenih na 60 parov mrež, ki so si med seboj identične v vseh parametrih, pri čemer ena izmed para ne uporablja metode izločanja, druga pa jo uporablja (z deležem izločanja 50% na nevronih in 70% na vhodu). Nevronske mreže smo kategorizirali glede na vrsto ter za vsak tip izračunali povprečno razliko (izboljšavo ali poslabšanje) mere AUC ($d_{avg} = \frac{1}{N}\sum_{i=1}^{N} d_i$).

Ta statistika nam žal ne pove vsega o dejanski vrednosti metode izločanja, saj lahko zelo dober (ali slab) rezultat na posameznem podatkovnem naboru (t.i. ubežniku) odloča o primerjavi. To ponavadi ni zaželjeno, saj nas zanima splošna oz. povprečna uspešnost klasifikatorja. Zato smo z Wilcoxonovim testom preverili, ali sta mreži posameznega para signifikantno različni (torej, ali je ničelna hipoteza ovržena). Delež parov, kjer sta mreži signifikantno različni (metoda izločanja signifikantno izboljša klasifikacijo) je označen kot $\%_{\alpha^+}$). Rezultati tega testa so predstavljeni v tabeli 4.3.

4.4.3 Primerjava modelov z obstoječimi klasifikatorji

V diplomski nalogi smo kot predstavnike obstoječih klasifikatorjev izbrali naslednje modele: logistično regresijo (kratica: **LOGREG**), metodo podpornih vektorjev (kratica: **SVM**) in naključne gozdove (kratica: **RF**). Za množico

Klasifikator	**Parameter**	**Vrednosti**
(Skupni parametri)	Arhitektura	$[1000, 200]$, $[200, 200]$, $[400, 400]$, $[100, 100, 100]$, $[1000, 1000]$, $[500, 500, 500]$
	hitrost učenja	0.1, 0.5
	parameter L_1	0
	parameter L_2	0
	delež izločanja	0%, 50%
	delež izločanja na vhodu	0%, 30%
	velikost paketov	20
	max. št. korakov učenja	1000
NN	Vrsta nevronov	hiperbolični tangens, popravljeni hiperbolični tangens, linearen pragovni nevron
NN + SDA	Delež šuma na vhodu	10%, 30%, 50%
NN + SRBM	Hitrost učenja Boltzmannovih naprav	0.01, 0.1, 0.5

Tabela 4.2: Parametri modelov. Vsako število v seznamu pri parametru arhitektura predstavlja število nevronov na posameznem skritem nivoju.

Klasifikator	**Št. parov mrež**	d_{avg}	$\%_{\alpha^+}$
NN	24	+0.1105	70.83%
NN + SDA	18	+0.0039	27.77%
NN + SRBM	18	+0.0089	44.44%
Skupaj	60	+0.0411	47.69%

Tabela 4.3: Povprečna izboljšava mere AUC, če umetna nevronska mreža uporablja metodo izločanja, ter delež nevronskih mrež, kjer uporaba metode izločanja prinaša značilno boljšo klasifikacijo ($\alpha = 0.05$).

Klasifikator	AUC_{avg}	**Parametri**
SVM	0.8730	$C = 1$, $Nu = 0.5$, $p = 0.1$, $\gamma = 1/\#features$, degree = 3, $coef_0 = 0$, shrinking = YES, probability = YES, $\epsilon = 0.001$, normalization = YES
LOGREG	0.9983	$C = 0.1$, $\epsilon = 0.001$, normalization = YES, $bias = -1$
RF	0.9965	nr. of trees = 100

Tabela 4.4: Parametri obstoječih klasifikatorjev.

15 podatkovnih naborov smo po testiranju našli parametre, ki dajejo najboljšo povprečno AUC vrednost na 15 naborih podatkov. Ti parametri so (vključno s povprečno mero AUC, v tabeli označeno kot AUC_{avg}) podani v tabeli 4.4.

Vseh 126 modelov umetnih nevronskih mrež, zgrajenih v okviru diplomske naloge, smo razvrstili glede na vrsto, ter za vsak tip primerjali posamezno nevronsko mrežo z vsemi tremi kontrolnimi klasifikatorji. Tako kot v podpoglavju 4.4.2, sta nas tudi pri tem zanimali meri d_{avg} in delež signifikantno boljših klasifikatorjev pri Wilcoxonovem testu ($\%_{\alpha^+}$). Vsi rezultati so zbrani v tabeli 4.5.

Po primerjanju vseh 126 modelov umetnih nevronskih mrež z zgoraj ome-

Kontr. klasifikator	**Klasifikator**	**Št. mrež**	d_{avg}	$\%_{\alpha^+}$
SVM	NN	54	+0.0698	72.22%
SVM	NN + SDA	36	+0.1172	83.33%
SVM	NN + SRBM	36	+0.1140	72.22%
LOGREG	NN	54	−0.0555	0%
LOGREG	NN + SDA	36	−0.0081	0%
LOGREG	NN + SRBM	36	−0.0113	0%
RF	NN	54	−0.0537	0%
RF	NN + SDA	36	−0.0062	0%
RF	NN + SRBM	36	−0.0094	0%

Tabela 4.5: Primerjava umetnih nevronskih mrež z ostalimi kontrolnimi klasifikatorji.

Kontr. klasifikator	$\#_{AUC^+}$	$\%_{AUC^+}$	$\%_{\alpha^-}$
SVM	115	91.27%	0%
LOGREG	0	0%	73.15%
RF	3	2.38%	22.84%

Tabela 4.6: Primerjava umetnih nevronskih mrež z ostalimi kontrolnimi klasifikatorji.

njenimi klasifikatorji smo ugotovili, da se velik delež umetnih nevronskih mrež obnese bolje kot metoda podpornih vektorjev, medtem ko niti ena umetna nevronska mreža ne preseže klasifikacije logistične regresije. Pri naključnem gozdu je le nekoliko bolje, zgolj tri nevronske mreže imajo boljšo povprečno mero AUC (ni pa nobena signifikantno različna) kot naključni gozd. Število in delež nevronskih mrež, ki imajo boljšo povprečno mero AUC smo označili z $\#_{AUC^+}$ in $\%_{AUC^+}$. Pri tem velja omeniti, da sta v veliki večini primerov logistična regresija in naključni gozd signifikantno boljša kot nevronske mreže (to je v tabeli označeno kot delež $\%_{\alpha^-}$). To je ravno tako ugotovljeno s pomočjo Wilcoxonovega testa. Te ugotovitve so podane v tabeli 4.6.

Poglavje 5

Sklepne ugotovitve

Diplomska naloga je raziskala uporabo nekaterih najnovejših pristopov z učenjem nevronskih mrež. Poročila o uporabi teh tipično poročajo o eksperimentih z zelo velikimi nabori podatkov, takih, kjer je primerov več deset ali sto tisoč. V diplomski nalogi pa nas je zanimala uporaba teh modelov na manjših naborih podatkov, kot so ti s področja molekularne biomedicine.

Pridobljeni eksperimentalni rezultati so vsekakor vsaj delno ustrezali pričakovanjem, zastavljenim v uvodu diplomske naloge. Metoda izločanja se je obnesla točno v skladu s pričakovanji, saj je signifikantno izboljšala klasifikacijo na vseh naborih podatkov. Za predhodno učenje uteži tega ne bi mogli reči, vendar se je vseeno potrebno zavedati, da sta obe uporabljeni metodi še precej neraziskani.

Ker se diplomska naloga ukvarja z relativno novimi modeli, je nadaljnega dela še veliko - področje se bo v prihodnosti nedvomno raziskovalo naprej in naše mnenje je, da je pred umetnimi nevronskimi mrežami tudi zaradi metod, opisanih v pričujočem delu, nekaj pestrih, inovativnih let. Prostora za izboljšave je namreč še veliko.

5.1 Metoda izločanja

Iz rezultatov je jasno razvidno, da se metoda izločanja obnese odlično. Ker v nalogi uporabljeni podatki vsebujejo majhno število primerov, je ta izboljšava verjetno še bolj izrazita. Metoda izločanja se je najbolje obnesla na nevronskih mrežah, ki niso uporabljale predhodnega učenja uteži, nekoliko manjšo izboljšavo pa je doprinesla v mrežah, ki so uporabljale diskriminativno učenje uteži. Glede na rezultate, ki smo jih dosegli na uporabljenih bioloških podatkih, menimo, da se metodo izločanja splača uporabiti na vseh nevronskih mrežah. Pri vseh različnih arhitekturah nevronskih mrež in na vseh naborih podatkov, uporabljenih v diplomski nalogi, ni metoda izločanja niti enkrat poslabšala klasifikacije. Poleg očitne izboljšave rezultatov je metoda izločanja izredno preprosta za implementacijo ter, še važnejše, ne vpliva na časovno zahtevnost učenja modela. V algoritem je namreč potrebno dodati le nekaj linearnih izrazov.

Kaj pa delež izločenih nevronov? Logično bi se zdelo, da sta delež izločenih nevronov znotraj mreže in delež izločenih nevronov na vhodnih podatkih dodatna dva parametra modela. Tekom gradnje umetnih nevronskih mrež smo ugotovili, da temu pravzaprav ni tako. Manjše spreminjanje omenjenih vrednosti ne vpliva veliko na samo klasifikacijo in daje zelo podobne rezultate kot klasifikacija s privzetimi vrednostmi (50% znotraj mreže, 30% na vhodnih podatkih). To se sklada z ugotovitvami, omenjenimi v prvotnem članku o metodi izločanja [10].

V zgrajenih nevronskih mrežah smo uporabili tri različne vrste nevronov; na uporabljenih bioloških podatkih so se najbolje obnesli nevroni, ki so za aktivacijsko funkcijo uporabljali hiperbolični tangens. To je pričakovan rezultat - hiperbolični tangens se pojavlja v večjem delu preučenih člankov kot najboljša aktivacijska funkcija. Po drugi strani pa je potrebno za popravljen hiperbolični tangens vhodne podatke ustrezno transformirati [15], česar v okviru diplomske naloge nismo storili. Zato so se najverjetneje nevroni s popravljenim hiperboličnim tangensom obnesli nekoliko slabše.

5.2 Predhodno učenje uteži

V primerjavi z izboljšavo klasifikacije, ki jo doprinese "preprosta" metoda izločanja, se predhodno učenje uteži na uporabljenih bioloških podatkih ni obneslo najbolje. Obe omenjeni metodi sicer nista poslabšali same klasifikacije, nista pa je signifikantno izboljšali. Kar se tiče dodatnih parametrov modela, smo ugotovili, da je delež šuma pri avtomatskih enkoderjih z odstranjevanjem šuma podoben parameter kot delež izločanja pri metodi izločanja. To pomeni, da različne, sorodne vrednosti (blizu 50%) ne vplivajo veliko na samo rezultat klasifikacije. Podobno velja za ločeno hitrost učenja omejenih Boltzmannovih naprav. Izkazalo se je, da manjše hitrosti učenja sicer izboljšajo klasifikacijo, vendar izredno minimalno. To se sklada s teorijo o Boltzmannovih napravah.

Zakaj so izboljšave v klasifikaciji nevronskih mrež z uporabo predhodnega učenja tako zanemarljive? Če upoštevamo dejstvo, da je logistična regresija od vseh uporabljenih klasifikatorjev dosegla najboljši rezultat, lahko sklepamo, da odvisnosti med podatki morda niso tako kompleksne. Ker imajo vse uporabljene umetne nevronske mreže vsaj dva skrita nivoja je mogoče, da smo zgradili preveč kompleksne arhitekture nevronskih mrež. Prav tako vsebujejo uporabljeni podatki, kot je bilo že večkrat omenjeno, majhno število primerov, kar je morebiti ravno tako razlog za neuspešnost predhodnega učenja.

Tudi avtorji, ki so zasnovali skladanje avtomatskih enkoderjev in skladanje omejenih Boltzmannovih naprav, so izpostavili, da predhodno učenje uteži izboljša klasifikacijo za največ 1-2% ([18], [28]), pri čemer je v obeh primerih za ta rezultat bila potrebna natančna kalibracija parametrov (angl. *fine-tuning*). To je seveda izrednega pomena pri klasifikaciji primerjalnih naborov podatkov (kot je npr. nabor ročno napisanih števk MNIST). Ne prinašata pa ti dve metodi tako očitnih izboljšav kot metoda izločanja. Ravno tako sta metodi novi in relativno neraziskani.

5.3 Primerjava z obstoječimi tehnikami klasifikacije

V diplomski nalogi zgrajene umetne nevronske mreže so se obnesle sicer bolje od metode podpornih vektorjev, vendar so po klasifikaciji malenkostno zaostajale za logistično regresijo in naključnimi gozdovi. Menimo, da je razlog za tak rezultat dvoslojen. Po eni strani so podatki relativno preprosti (1000 značilk, 10-200 primerov v posameznem podatkovnem naboru), kar daje prednost preprostejšim modelom; nedvomno sta tako logistična regresija kot naključni gozdovi preprostejša napovedna modela od uporabljenih nevronskih mrež. Po drugi strani pa razlog tiči v problemu nevronskih mrež, ki ga zelo radi izpostavljajo njihovi kritiki (in je ravno posledica kompleksnosti modela). Ustrezna kalibracija vseh različnih parametrov nevronske mreže traja veliko časa in je zelo različna od problema do problema; zelo verjetno je, da zgrajene nevronske mreže niso optimalne za dan nabor podatkov oz. da bi lahko našli nabor parametrov, ki bi klasifikacijo še malenkostno izboljšal. V primerjavi z logistično regresijo in naključnimi gozdovi, ki sta na uporabljenih podatkih dosegla doober rezultat praktično "out-of-the-box" (z minimalno konfiguracijo parametrov) pri umetnih nevronskih mrežah žal ni tako.

5.4 Nadaljne delo

Zdi se, da umetne nevronske mreže doživljajo svoj preporod. Vse od odkritja učinkovitega učnega algoritma (vzvratno razširjanje napake) pa do nekaj let nazaj se o nevronskih mrežah ni veliko govorilo, saj na področju ni bilo novih algoritmov, poleg tega pa so drugi klasifikacijski modeli napredovali do te mere, da so zasenčili nevronske mreže. A peščica znanstvenikov se je ves ta čas ukvarjala z nevronskimi mrežami in njihov trud se je nazadnje poplačal. Danes je na področju kar nekaj novih člankov, ki opisujejo nove algoritme, modele in izboljšave umetnih nevronskih mrež. Vse te nove metode

je potrebno še natančno preučiti, predvsem pa jih je potrebno uporabiti za klasifikacijo čimbolj raznolikih naborov podatkov. Obnašanje številnih metod namreč še ni povsem definirano in razloženo, saj so nove in enostavno še niso bile uporabljene na dovolj velikemu številu podatkovnih naborov.

Avtomatski enkoderji so primer take metode - avtorji sami navajajo, da je potrebno raziskati, kaj se zgodi, če popačimo ne-le vhodne podatke, temveč tudi skrito reprezentacijo podatkov ter druge sklope modela [28]. Ravno tako je z uvedbo globokih verjetnostnih mrež G. E. Hinton s sodelavci odprl novo področje, ki kombinira usmerjene nevronske mreže z omejenimi Boltzmannovimi napravami. Rekurzivne nevronske mreže (čigar primer so Boltzmannove naprave) so še izredno neraziskano področje. Vsaj del razloga tiči v tem, da jih znanstveniki niso uspeli uspešno uporabiti v strojnem učenju. Z uvedbo globokih verjetnostnih mrež se je seveda vse to spremenilo, in trenutne smernice nakazujejo na to, da bo kombiniranje usmerjenih in rekurzivnih nevronskih mrež izrednega pomena pri gradnji še boljših klasifikatorjev.

Umetne nevronske mreže, zgrajene v diplomski nalogi, bi se prav tako dalo še izboljšati. Ker metoda izločanja odlično deluje kot regularizator globokih nevronskih mrež tudi pri majhnem naboru podatkov, bi lahko zgradili še bolj globoke (večnivojske) nevronske mreže. Prav tako bi lahko poskusili izboljšali klasifikacijo z uporabo predhodnega učenja uteži, kjer bi morali bolj precizno ponastaviti parametre modela. Potrebno bi bilo predvsem zmanjšati hitrost učenja ter odstraniti L1/L2 regularizacijo uteži [10]. Če bi želeli za uporabljen nabor bioloških podatkov res dobiti najboljši klasifikator, bi lahko uporabili povprečenje različnih klasifikacijskih modelov - logistične regresije, naključnih gozdov in nevronskih mrež - kar bi skoraj zagotovo še nekoliko izboljšalo rezultat. Vsekakor pa bi bilo smiselno zgrajene nevronske mreže uporabiti še na drugih naborih podatkov, po možnosti iz druge domene, da bi dobili bolj splošno predstavo o njihovi uspešnosti.

Literatura

[1] D. H. Ackley, G. E. Hinton, T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines," *Cognitive Science*, št. 9, zv. 1, str. 147-169, 1985.

[2] S. A. Armstrong, J. E. Staunton, L. B. Silverman, R. Pieters, M. L. den Boer, M. D. Minden, S. E. Sallan, E. S. Lander, T. R. Golub, S. J. Korsmeyer, "MLL translocations specify a distinct gene expression profile that distinguishes a unique leukemia," *Nature Genetics*, št. 30, zv. 1, str. 41-47, 2002.

[3] J. Balfour. (2011). *Introduction to CUDA* [Online]. Dostopno na: http://mc.stanford.edu/cgi-bin/images/f/f7/Darve_cme343_cuda_1.pdf

[4] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends in Machine Learning*, št. 2, zv. 1, str. 1-127, 2009.

[5] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, Y. Bengio, "Theano: A CPU and GPU Math Expression Compiler," v zborniku *Proceedings of the 9th Python in Science Conference (SciPy 2010)*, Austin, United States of America, 2010, str. 3-10.

[6] J. Demšar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *Journal of Machine Learning Research*, št. 7, zv. 1, str. 1-30, 2006.

[7] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, št. 27, zv. 8, str. 861-874, 2006.

[8] U. M. Fayyad, K. B. Irani, "Multi-interval discretization of continuous valued attributes for classification learning," v zborniku *Proceedings of the 13th International Joint Conference on Uncertainty in AI*, Chambery, Francija, 1993, str. 1022-1029.

[9] G. E. Hinton, S. Osindero, Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, št. 18, zv. 7, str. 1527-1554, 2006.

[10] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. R. Salakhutdinov. (2012). *Improving neural networks by preventing co-adaptation of feature detectors* [Online]. Dostopno na: http://arxiv.org/pdf/1207.0580.pdf

[11] L. Hood, D. Galas, "The digital code of DNA," *Nature*, št. 421, zv. 1, str. 444–448, 2003.

[12] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the United States of America*, št. 79, zv. 8, str. 2554-2558, 1982.

[13] I. Kononenko, *Strojno učenje*. Ljubljana, Založba FE in FRI, 2005.

[14] P. Lamblin. (2007). *Contrastive divergence multi-layer RBMs* [Online]. Dostopno na: http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DBNEquations

[15] Y. LeCun, L. Bottou, G. B. Orr, K. Müller, "Efficient BackProp," v *Neural Networks: Tricks of the trade*, 1. izd., zv. 1. G. Montavon, G. B. Orr, K. Müller, ur. Berlin Heidelberg: Springer, 1999, str. 9-50.

[16] Y. LeCun, C. Cortes, C. J. C. Burges. *The MNIST database of handwritten digits* [Online]. Dostopno na: http://yann.lecun.com/exdb/mnist/

[17] J. L. Lustgarten, V. Gopalakrishnan, H. Grover, S. Visweswaran, "Improving Classification Performance with Discretization on Biomedical Datasets," v zborniku *AMIA 2008 Annual Symposium Proceedings*, Washington, United States of America, 2008, str. 445-449.

[18] Machine Learning Laboratory, University of Montreal. (2013). *Deep Learning tutorial - Multilayer Perceptron* [Online]. Dostopno na: http://www.deeplearning.net/tutorial/mlp.html

[19] W. S. McCulloch, W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, št. 5, zv. 4, str. 115-133, 1943.

[20] M. Minsky, S. Papert, *Perceptrons: an introduction to computational geometry*. Cambridge MA: The MIT Press, 1969.

[21] G. E. Nasr, E. A. Badr, C. Joun, "Cross Entropy Error Function in Neural Networks: Forecasting Gasoline Demand," v zborniku *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference*, Pensacola Beach, United States of America, 2002, str. 381-283.

[22] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, št. 65, zv. 6, str. 386-408, 1958.

[23] D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning representations by back-propagating errors," *Nature*, št. 323, zv. 1, str. 533-536, 1986.

[24] D. E. Rumelhart, J. L. McClelland, PDP Research Group, *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*. Cambridge: MIT Press, 1986.

[25] M. Schena, D. Shalon, R. W. Davis, P. O. Brown, "Quantitative Monitoring of Gene Expression Patterns with a Complementary DNA Microarray," *Science*, št. 270, zv. 5235, str. 467-470, 1995.

[26] R. Solomonoff, "An Inductive Inference Machine," v zborniku *IRE Convention Record*, New York, United States of America, str. 56-62, 1957.

[27] M. C. P. de Souto, I. G. Costa, D. S. A. de Araujo, T. B. Ludermir, A. Schliep. (nov. 2008). Clustering cancer gene expression data: a comparative study. *BMC Bioinformatics* [Online]. 9(497), str. 1-14. Dostopno na:
http://www.biomedcentral.com/1471-2105/9/497

[28] P. Vincent, H. Larochelle, Y. Bengio, P. A. Manzagol, "Extracting and Composing Robust Features with Denoising Autoencoders," v zborniku *Proceedings of the Twenty-fifth International Conference on Machine Learning*, Helsinki, Finland, 2008, str. 1096-1103.

[29] J. D. Watson, F. H. C. Crick, "Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid," *Nature*, št. 171, zv. 1, str. 737–738, 1953.

[30] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences : doktorska disertacija*, Harvard, 1974.

[31] R. W. Williams, K. Herrup, "The Control of Neuron Number," *The Annual Review of Neuroscience*, št. 11, zv. 1, str. 423–453, 1988.