Tartu Ülikool
Loodus- ja täppisteaduste valdkond
Matemaatika ja statistika instituut

Siim Viigand

Otsesuunatud tehisnärvivõrgud ja nende treenimine

Matemaatilise statistika eriala

Bakalaureusetöö (9 EAP)

Juhendajad Riho Klement, Taavi Unt

Tartu 2016

Otsesuunatud tehisnärvivõrgud ja nende treenimine

Käesoleva bakalaureusetöö eesmärk on tutvustada tehisnärvivõrgustikke ning anda lugejale praktilisi soovitusi, mis aitaksid kaasa meetodi rakendamisel. Töö on valdavalt teoreetiline ja kirjeldab detailselt närvivõrgustikega seotud mõisteid ja valemeid. Töös on räägitud närvivõrgustike struktuursetest erinevustest, kirjeldatud enamlevinud aktiveerimisfunktsioone, uuritud gradientlaskumise ning tagasilevi meetodit ja käsitletud treenimisega seotud probleeme. Töö annab nii detailse kirjelduse kui ka proovib edastada intuitiivset arusaama.

Märksõnad: *Neurovõrgud, klassifitseerimine, veafunktsioonid, gradientlaskumine, tagasilevi*

P160 Statistika, operatsioonanalüüs, programmeerimine, finants- ja kindlustusmatemaatika

Structure and learning of feedforward artificial neural networks

The purpose of this thesis is to give a brief review about artificial neural networks and give hints and recommendations for practical use. Thesis is mostly theoretical and offers detailed explanations of neural network related terms and equations. Although theoretical part has the biggest role, paper is implemented with sufficient amount of practical advice which should allow any reader to work with artificial neural networks. Thesis covers all the main components: structural differences, most commonly used activation functions, training with gradient descent and backpropagation and some of the threats like overfitting.

Keywords: *Artificial neural networks,classification, error functions, gradient descent, backpropagation*

P160 Statistics, operation research, programming, actuarial mathematics

Sisukord

Sissejuhatus

Läbi ajaloo on proovitud leida täpsemaid, tõhusamaid ja kiiremaid meetodeid. Olgu tegu sõjapidamise, ehitamise või matemaatikaga. Viimasel paarikümnel aastal on väga palju arenenud meetodid, mis toimivad efektiivselt ainult tänu arvutitele. Üheks selliseks meetodite klassiks on tehisnärvivõrgustikud. Tehisnärvivõrgustike puhul on tegu siiani üsna kiirelt areneva meetodiga. Tänapäevased mudelid võivad olla tohutult suured ja hakkama saada väga keeruliste ülesannetega. Antud töös nii keerulisi mudeleid siiski ei vaadata ning pigem antakse pidepunkt, millele uusi teadmisi rajada.

Tehisnärvivõrgustikud on idee saanud bioloogilistest närvivõrgustikest, mis suudavad hakkama saada väga keeruliste ülesannetega. Bioloogiliste närvivõrkude jäljendamisest tulenevalt on ka tehisnärvivõrgustikes kesksel kohal neuron, mis analüüsib informatsiooni. Sidudes kokku mitmeid neuroneid on võimalik lahendada väga keerulisi probleeme. Väga levinud on klassifitseerimis- ja regressioonülesanded. Kuigi töö teoreetiline osa on pigem üldist laadi, siis esitatud näidetes keskendutakse rohkem klassifitseerimisülesannetele.

Töö eesmärk on anda ülevaade närvivõrkude meetodist ning seega on töö pigem referatiivne. Samas on sisse lisatud mitmeid näpunäiteid ja soovitusi, et antud töö lugeja saaks hakkama lihtsamate närvivõrgustike modelleerimisega.

Töö esimeses peatükis antakse detailne ülevaade esimesest ja kõige lihtsamast närvivõrgustikust. Hilisemates peatükkides täiendatakse teadmisi struktuuri, aktiveerimisfunktsioonide, treenimise ja mudeli intuitiivse mõtestamisega. Struktuuris kirjeldatakse mudelite erinevaid suurusi ja edastatakse peamine idee. Aktiveerimisfunktsioonidest tutvustatakse viit populaarsemat funktsiooni ja treenimises seletatakse gradientlaskumise ja tagasilevi ideed. Viimastes peatükkides tuleb juttu ülesobitamisest ja mudeli mõistmisest.

Töös esitatud näidete läbiviimisel on kasutatud tarkvara ***R*** paketti ***neuralnet***. Töö vormistamisel on kasutatud tekstitöötlusprogrammi ***LaTeX*** veebiversiooni ***Overleaf***.

Autor tänab Tambet Matiisenit kasulike näpunäidete ja asjakohaste materjalide soovitamise eest.

1 Pertseptroni mudel

Järgnev peatükk on kirjutatud raamatu „The Nature of Code: Simulating Natural Systems with Processing" (Shiffman, 2012) põhjal.

Tehisnärvivõrkude kasutamise idee sai alguse eelmise sajandi keskpaigas. Sellest ajast alates on meetod palju arenenud. Antud töö eesmärk ei ole kirjeldada ajalooliselt esimesi mudeleid, neist on võimalik täpsemalt lugeda virtuaalsest raamatust „A Brief Introduction to Neural Network" (Kriesel).

Esimene tänapäevast tehisnärvivõrgustikku kirjeldav mudel on pärit aastast 1957, kui Frank Rosenblatt avaldas pertseptroni mudeli. Pertseptroni mudelit võib nimetada kõige lihtsamaks tehisnärvivõrgustikuks, sest see koosneb ainult ühest tehisneuronist. Värskemas kirjanduses kasutatakse pertseptroni mõistet väga erineva tähendusega, aga antud töös mõeldakse pertseptroni mudeli all Rosenblatti pertseptronit, mida kirjeldatakse selles peatükis. Suurematest tehisnärvivõrgustikest, kus kokku on seotud rohkem kui üks tehisneuron, tuleb juttu järgnevates peatükkides. Kuna pertseptroni mudeli kujul on tegu lihtsaima tehisnärvivõrgustikuga, siis antud peatükis kirjeldatakse meetodit väga detailselt, et paremini mõista tehisnärvivõrgustike loogikat.

Frank Rosenblatt pakkus välja mudeli, mis koosneb sisenditest $x_1, x_2, \ldots, x_N$, kus $N \in \mathbb{N}$, sisendeid töötlevast tehisneuronist ja ühest väljundist y. Tehisnärvivõrgustikes on sisenditeks objekti kirjeldavad tunnused ning väljundiks mudeli hinnang uuritavale tunnusele. Mudeli komponendid järgivad otsesuunatud närvivõrgustiku ideed ehk mudelile antakse ette sisendid, mis suunatakse tehisneuronisse. Tehisneuronis sisendeid töödeldakse ning töötluse tulemusena saadakse väljund. Olemas on ka rekurrentsed närvivõrgustikud, kus töötluse tulemusena saadud väljundeid kasutatakse eelnevalt läbitud neuronites uute sisenditena, selliseid mudeleid töös ei kirjeldata, aga neist on võimalik lugeda raamatust „Deep Learning"

(Goodfellow, 2016).

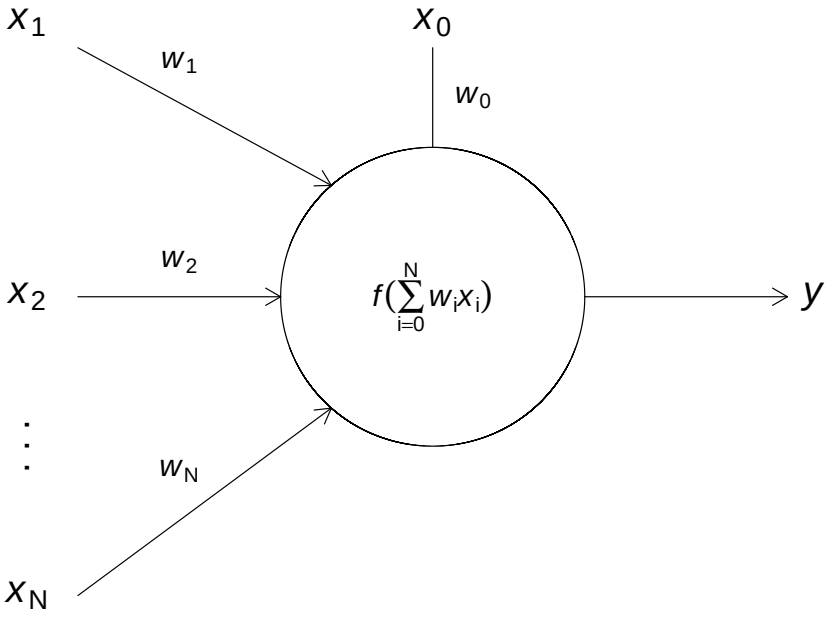
Pertseptronile antud sisendid $x_1, x_2, \ldots, x_N$ kaalutakse enne tehisneuronisse saatmist. Kaalud $w_1, w_2, \ldots, w_N$ näitavad antud sisendi panust väljundi arvutamisesse. Siinkohal tasub mainida, et kaalud on mudelis tundmatuteks parameetriteks ja närvivõrgustiku mudeli hindamine seisnebki sobivate kaalude leidmises. Kaalumine kujutab endast sisendi korrutamist temale vastava kaaluga, seega iga sisend panustab neuronisse suuruse $w_i x_i$, kus $i = 1, \ldots, N$. Kui sisendid on kaalutud, siis korrutised summeeritakse ning suunatakse aktiveerimisfunktsiooni. Aktiveerimisfunktsiooniks f nimetatakse funktsiooni, mille argumendiks on kaalutud sisendite summa ja funktsiooni väärtuseks väljund y. Mudelisse kaasatakse ka konstant x_0 ja temale vastav kaal w_0. Suurus x_0 esindab mudeli vabaliiget, mis tavapäraselt valitakse väärtusega 1.

Kirjeldatud meetodi võib kokku võtta võrrandiga

$$y = f\left(\sum_{i=1}^{N} w_i x_i + w_0 x_0\right) = f\left(\sum_{i=0}^{N} w_i x_i\right). \tag{1}$$

Pertseptroni mudeli tööd iseloomustab joonis 1.

Aktiveerimisfunktsioon f on mudeli koostamisel üks tähtsamaid komponente. Rosenblatti pertseptroni mudelis on kasutatud treppfunktsiooni, mis on näidatud valemiga (6) peatükis 5. Mudeliga prooviti jäljendada bioloogilise neuroni tööd. Bioloogilisest neuronist väljub signaal, kui sisenevate signaalide tugevus ületab lävendit. Sarnaselt toimib ka treppfunktsioon. Samas ei ole suuremate mudelite korral taolise funktsiooni kasutamine eriti levinud. Probleemiks on funktsiooni diferentseeruvus, mis osutub oluliseks mudeli treenimisel. Sellest tulenevalt on hakatud aktiveerimisfunktsioonidena kasutama siledamaid funktsioone, näiteks sigmoidfunktsioone. (Hastie, 2009, lk 394-395) Aktiveerimisfunktsiooni valikust tuleb täpsemalt juttu peatükis 5.



Joonis 1. *Pertseptroni mudel*

Kirjeldatud mudeliga on võimalik ennustada väljundit, aga selleks on vaja teada sobivaid kaale. Sobivate kaalude leidmiseks tuleb mudelit treenida ehk õpetada. Mudeli õpetamiseks kasutatakse treeningandmestikku, mis peaks olema sarnane andmetega, millel mudelit soovitakse kasutada. Treeningandmestiku eripäraks on asjaolu, et selles on mudeliga ennustatav tunnus teada. Treenimine põhineb uuritava tunnuse ennustatud ja tegeliku väärtuse võrdlemisel. Tähistame tegelikku ehk oodatavat väljundit sümboliga y_o. Kui mudeli poolt ennustatud tulemus kattub oodatavaga, siis pole kaale tarvis muuta, sest närvivõrgustik ennustamisel viga ei teinud. Kui tulemused ei kattu, siis tuleb kaale korrigeerida. Treenimise peamiseks ülesandeks on minimiseerida valitud kaofunktsioon, et mudel ennustaks võimalikult täpselt. Kaofunktsioonidest ja tehisnärvivõrgustike treenimisest tuleb täpsemalt juttu peatükis 6.

Pertseptroni mudeli korral on treenimiseks võimalik kasutada järgnevat algoritmi.

Algoritmi idee ja põhjendatus on üldisemal kujul toodud peatükis 6.

1. Valida treeningandmestikust objekt ning ennustada pertseptroniga objekti väärtust y.
2. Arvutada suurus oodatava ja ennustatava väärtuse vahel $e = y_o - y$.
3. Leida kaalude muudud $\Delta_i = \eta e x_i, i = 0, 1, \ldots, N$.
4. Arvutada uued kaalud $w_{i_{uus}} = w_i + \Delta_i, i = 0, 1, \ldots, N$.
5. Korrata järgmise objektiga treeningandmestikust.

Kaalude muutmise valemis on õpikiirust mõjutav konstant η, mis võimaldab reguleerida kaalude muutumise kiirust. Nimetatud konstant valitakse tavaliselt vahemikust $0,01$ kuni $0,9$, aga täpsem väärtus sõltub väga palju lahendatavast ülesandest. Konstandi suur väärtus võimaldab parameetritel kiiresti muutuda ja seega võib sobivast lahendist pidevalt üle hüpata ja seda mitte tabada. Õppimiskonstandi väike väärtus muudab kaalude muutused väikeseks ja võimaldab seega leida täpsemat lahendit, aga samas suurendab tunduvalt treenimiseks kuluvat aega. (Kriesel, lk 92)

Antud algoritmi jooksutatakse üle kõigi treeningandmestikus olevate andmete ja vajadusel korduvalt. Treenimine lõppeb olukorras, kus objektid on lineaarselt eraldatud. Tihti ei ole nimetatud olukorra saavutamine võimalik. Neil juhtudel tuleb treenimine lõpetada, kui saavutatakse piisavalt hea lähend. Pärast treenimist on võimalik mudelit rakendada andmetel, kus ennustatavat väljundit eelnevalt teada ei olnud.

Rosenblatti pertseptron on lihtsaim tehisnärvivõrgustik ja seega võib eeldada, et ka selle rakendatavus on piiratud. Kuna mudel koosneb ühest neuronist, siis suudab see käsitleda vaid lineaarset juhtu, mis klassifitseerimise korral tähendab lineaarset eraldatavust. Pertseptroni mudeli puhul avaldub lineaarne eraldaja kujul

$w_1x_1 + w_2x_2 + \cdots + w_Nx_N + w_0x_0 = 0$, kus w_0x_0 on vabaliige. Lineaarsusest tulenev piirang ongi peamiseks põhjuseks, miks läheb tarvis mitmetest tehisneuronitest koosnevaid mudeleid.

Järgneva näitega proovib autor iseloomustada detailselt pertseptroni tööd.

Näide 1.1. *Olgu kasutada järgnev andmestik, mis esitub all oleva tabelina:*

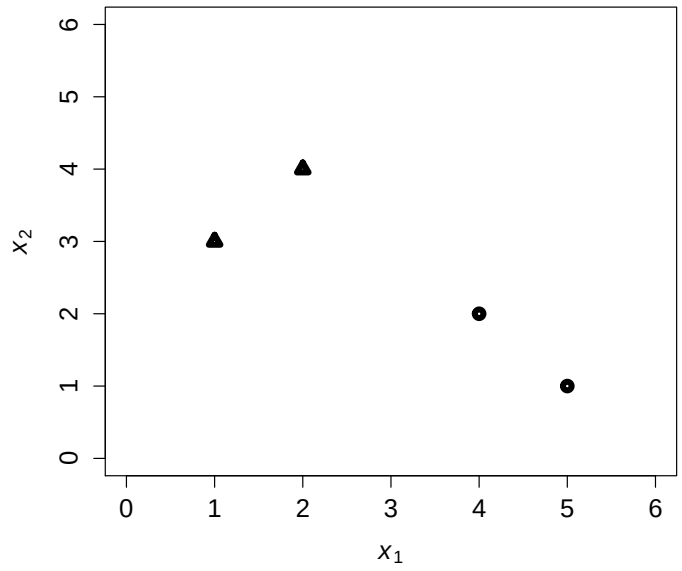
Grupp	*Tunnus* x_1	*Tunnus* x_2
Ring	*4*	*2*
Ring	*5*	*1*
Kolmnurk	*1*	*3*
Kolmnurk	*2*	*4*

Ülesandeks on treenida pertseptroni mudel, mis suudaks eristada kahte erinevat gruppi: ringid ja kolmnurgad. Ringide ja kolmnurkade kohta on andmestikus kaks tunnust, mida soovitakse klassifitseerimiseks kasutada. Joonisel 2 on näha gruppide asetus graafikul. Jooniselt on näha, et valitud gruppe saab sirgega eraldada ning seega otsitakse lahendit, kus mudel ennustab kõiki andmestikus olevaid objekte õigesti.

Lahenduse võimalikkuse huvides tuleb kodeerida ringid ja kolmnurgad. Olgu ringid tähistatud väärtusega 1 ja kolmnurgad väärtusega -1.

Gruppe soovitakse klassifitseerida kahe tunnuse põhjal, seega on mudelis kaks sisendit ja konstant, mis on valitud võrdseks ühega. Veel tuleb valida aktiveerimisfunktsioon. Olgu selleks treppfunktsiooni tuntud modifikatsioon, mida nimetatakse märgifunktsiooniks. Kui märgifunktsiooni korral on kaalutud sisendite summa mittenegatiivne, siis on väärtuseks 1, negatiivse summa korral -1. Sellega on mudeli ülesehitus määratud.

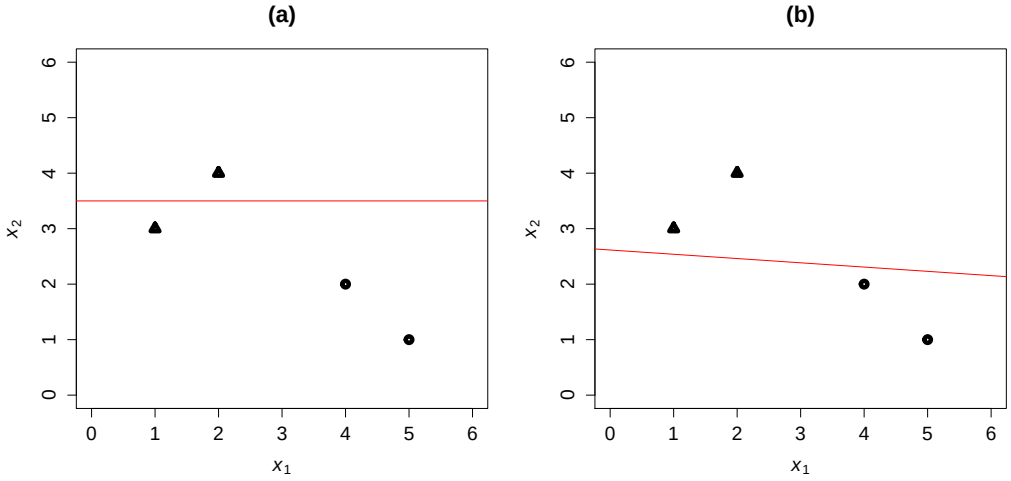
Enne mudeli treenimist on vaja veel määrata esialgsed kaalud. Olgu algkaaludeks $w_1 = 0, w_2 = -1$ *ja* $w_0 = 3,5$*. Selliste kaalude korral on lineaarne eraldaja*



Joonis 2. *Valitud andmepunktide paiknemine graafikul*

kujul $0x_1 - 1x_2 + 3,5x_0 = 0$. *Algne seisund on kujutatud joonisel 3 (a). Jooniselt on näha, et mudel ei suuda hetkel ringe ja kolmnurki korrektselt klassifitseerida. Soovitud lahendi korral peaksid kõik ringid olema ühel pool joont ja kolmnurgad teisel pool.*

Soovitud lahendi leidmiseks tuleb mudelit treenida. Mudeli treenimine käib eelnevalt kirjeldatud algoritmi kohaselt. Andmestikust valitakse objekt, kus on teada klassifitseerimise grupp. Olgu selleks esimene rida. Esimese rea korral on tegu ringiga ning sisenditeks on $x_1 = 4$ *ja* $x_2 = 2$. *Seega on oodatavaks väärtuseks* $y_o = 1$. *Esmalt korrutatakse sisendid läbi algsete kaaludega:* $w_0x_0 = 3,5 \cdot 1 = 3,5, w_1x_1 = 0 \cdot 4 = 0$ *ja* $w_2x_2 = -1 \cdot 2 = -2$. *Seejärel summeeritakse saadud tulemused:* $w_0x_0 + w_1x_1 + w_2x_2 = 3,5 + 0 + (-2) = 1,5$. *Pärast summeerimist rakendatakse aktiveerimisfunktsiooni ja saadakse mudeli ennustus andmerea grupile:* $y = sgn(1,5) = 1$. *Kuna* $y_o = y$, *siis ennustas mudel õiget tulemust ja kaale*



Joonis 3. *(a) Esialgsete kaaludega eraldaja, (b) eraldaja pärast kaalude uuendamist*

korrigeerida pole tarvis.

Protsess jätkub järgmise objektiga. Valime selleks objektiks kolmanda andmerea. Tegu on kolmnurgaga, mille korral tunnusteks on $x_1 = 1$ *ja* $x_2 = 3$*. Oodatavaks väärtuseks on seega* $y_o = -1$*. Joonisel 3 (a) on näha, et valitud kolmnurk ei asu klassifitseerimise jaoks sobival poolel, seega on oodata, et mudel ei saa oodatavat väärtust. Arvutame tulemuse analoogiliselt eelnevale korrale:*
$y = sgn(3,5 \cdot 1 + 0 \cdot 1 + (-1) \cdot 3) = sgn(0,5) = 1$*. Oodatav tulemus on tõepoolest erinev ennustatud tulemusest. Seega tuleb kaale korrigeerida.*

Kaalude korrigeerimiseks kasutatakse algoritmi, mis on samuti kirjeldatud eespool. Leitakse oodatava ja ennustatava väärtuse vahe: $e = y_o - y = -1 - 1 = -2$*. Seejärel leitakse kaalude muudud. Muutude arvutamisel valitakse õppimiskiiruse konstant, mis antud juhul on võetud* $\eta = 0,05$*. Konstant on valitud väike, sest andmestikus saavad tunnused* x_1 *ja* x_2 *väärtusi ühest viieni. Oodatava ja ennustatava väärtuse vahe* e *võib olla -2, 0 või 2. Kuna viga* e *on üsna suur võrreldes tunnuste* x_1 *ja* x_2 *väärtustega, siis väike konstant võimaldab ära hoida suuri kaalude muu-*

tusi, mis hõlbustab sobiva lineaarse eraldaja leidmise protsessi. Pärast suuruse e *leidmist, tuleb arvutada kaalude muudud:* $\Delta_0 = -2 \cdot 1 \cdot 0,05 = -0,1; \Delta_1 = -2 \cdot 1 \cdot 0,05 = -0,1; \Delta_2 = -2 \cdot 3 \cdot 0,05 = -0,3$*. Kaalude muudud liidetakse vanadele kaaludele, et saada uued kaalud:* $w_{0_{uus}} = 3,5 + (-0,1) = 3,4; w_{1_{uus}} = 0 + (-0,1) = -0,1; w_{2_{uus}} = -1 + (-0,3) = -1,3$*. Nüüd on uus lineaarne eraldaja kujul* $-0,1x_1 - 1,3x_2 + 3,4x_0 = 0$*, mis on nähtav joonisel 3 (b).*

Pärast teist sammu on leitud sobivad kaalud, mis võimaldavad andmeid klassifitseerida. Korrektsuse mõttes tuleks korrata esialgset algoritmi uuesti ja veenduda, et nüüd klassifitseeritakse kõik andmed õigesti. Hetkel jääb see tegemata, sest näide oli lihtne ja jooniselt 3 (b) on näha, et leitud lineaarne eraldaja tõepoolest jagab kolmnurgad ja ringid erinevatesse gruppidesse.

2 Ühe varjatud kihiga närvivõrgustik

Ühe varjatud kihiga närvivõrgustiku ülesehitus ei ole väga palju keerulisem pertseptroni mudelist. Sarnaselt Rosenblatti pertseptroniga koosneb ühe varjatud kihiga närvivõrgustik sisenditest, neuronitest ja väljunditest. Erinevuseks on tehisneuronite ja väljundite arv, mis pole enam piiratud. Suuremate närvivõrgustike lihtsamaks mõistmiseks on sisendid, neuronid ja väljundid jaotatud kihtidesse. Ühe varjatud kihiga närvivõrgustik koosneb kolmest kihist: sisendkiht, varjatud kiht ja väljundkiht. Peatükist selgub, et varjatud kiht ja väljundkiht on üsna sarnased, mistõttu nimetatakse ühe varjatud kihiga närvivõrgustikku ka kahekihiliseks närvivõrgustikuks. (Swingler, 2001, lk 10)

2.1 Sisendkiht

Sisendkiht sisaldab sisendeid $x_1, x_2, \ldots, x_N, N \in \mathbb{N}$ ning seega ei erine see Rosenblatti mudelist. Sisendite arvu määramine ei pruugi olla alati lihtne. Võimalik on mudelisse kaasata kõik andmestikus olevad tunnused, mis väiksemate mõõtmetega andmestike puhul on igati õigustatud. Probleem tekib suuremaid andmestikke kasutades. Iga sisend on varjatud kihis asuvate neuronitega ühendatud kaalude abil. Kui varjatud kihis on kaks neuronit, siis igal sisendil on kaks erinevat kaalu. Sellest tulenevalt kasvab hinnatavate kaalude arv väga kiiresti ning närvivõrgustiku treenimine muutub aeglasemaks. Lisaks sellele muutub närvivõrgustik väga keeruliseks ja rohkemate kaalude hindamiseks läheb vaja suuremat treeningandmestikku. Seega tuleks andmestikku alles jätta tunnused, mis kirjeldavad soovitud väljundit kõige rohkem. Üheks võimaluseks on vaadata korrelatsioonimaatriksit. Kui kahe tunnuse vahel on väga suur korrelatsioon, siis võib ühe tunnuse neist eemaldada. Teiseks võimaluseks on algselt kaasata mudelisse kõik

tunnused ning omistada neile väikesed juhuslikud kaalud ning alustada treenimist. Pärast lühiajalist treenimist kontrollida sisendite kaale ning eemaldada sisendid, mille kaalud ei ole muutunud. Korrates protsessi on võimalik jõuda mõistliku arvu sisenditeni. Antud lahenduse miinuseks on suur ajakulu. (Swingler, 2001, lk 24-25, 28)

2.2 Varjatud kiht

Varjatud kihis paiknevad tehisneuronid, millest igaüks on sarnane pertseptroni mudelis olevaga. Tähistame varjatud kihi neuronite arvu tähega $R \in \mathbb{N}$. Iga varjatud kihis olev tehisneuron saab informatsiooni sisenditest, mis kaalutakse kaaludega $w^1_{l1}, w^1_{l2}, \ldots, w^1_{lN}$, kus $l = 1, 2, \ldots, R$. Kaasatakse ka konstant x_0, koos kaaludega $w^1_{l0}, l = 1, 2, \ldots, R$. Kaalutud sisendid summeeritakse analoogselt pertseptroniga ning summale rakendatakse aktiveerimisfunktsiooni, mille väärtusena saadakse neuroni väljund $v_l, l = 1, 2, \ldots, R$. Varjatud kihis olevate neuronite väljundeid otseselt ei uurita, mistõttu nimetataksegi kihti varjatuks. (Swingler, 2001, lk 55-56)

Varjatud kihis olevate neuronite arvu valikuks ei ole väga kindlaid juhendeid. Järgnevalt on välja toodud mõned soovitused, mida tasub neuronite arvu valides silmas pidada:

1. Varjatud kihis ei tohiks olla üle kahe korra rohkem neuroneid kui mudelis sisendeid.
2. Kui on soov andmestikust informatsiooni eraldada, siis on mõistlik kasutada vähem neuroneid kui on sisendeid.
3. Kui on soov andmestikust uusi seoseid leida, siis on mõistlik kasutada rohkem neuroneid kui on sisendeid.

4. Neuronite arv on kompromiss mudeli üldistavuse (vähem neuroneid) ja täpsuse (rohkem neuroneid) vahel.
5. Suuremad mudelid nõuavad pikemat treenimise aega.
6. Parima variandi leidmine nõuab tihtipeale katsetamist.

2.3 Väljundkiht

Ühe varjatud kihiga närvivõrgustiku väljundkiht on väga sarnane varjatud kihile. Ka väljundkihis on tehisneuronid, mille arvu tähistame tähega $M \in \mathbb{N}$. Selles kihis olevate neuronite sisenditeks on varjatud kihi väljundid $v_1, v_2, \ldots, v_R$. Need on seotud kaaludega $w^2_{m1}, w^2_{m2}, \ldots, w^2_{mR}$, kus $m = 1, 2, \ldots, M$. Lisaks eelmisest kihist tulevatele sisenditele lisatakse ka väljundkihti konstant v_0, koos kaaludega $w^2_{m0}, m = 1, 2, \ldots, M$. Pärast sisendite summeerimist ja aktiveerimisfunktsiooni rakendamist saadakse lõplikud väljundid $y_1, y_2, \ldots, y_M$. (Swingler, 2001, lk 11)

Väljundite arv sõltub lahendatavast ülesandest ning seega jääb täpne arv rakendajale endale valida.

Matemaatiliselt on varjatud kihiga närvivõrgustikku võimalik iseloomustada järgneva kahe valemi abil:

$$v_l = f\left(\sum_{i=0}^{N} w^1_{li} x_i\right), \tag{2}$$

kus $l = 1, 2, \ldots, R$ ja

$$y_m = f\left(\sum_{l=0}^{R} w^2_{ml} v_l\right), \tag{3}$$

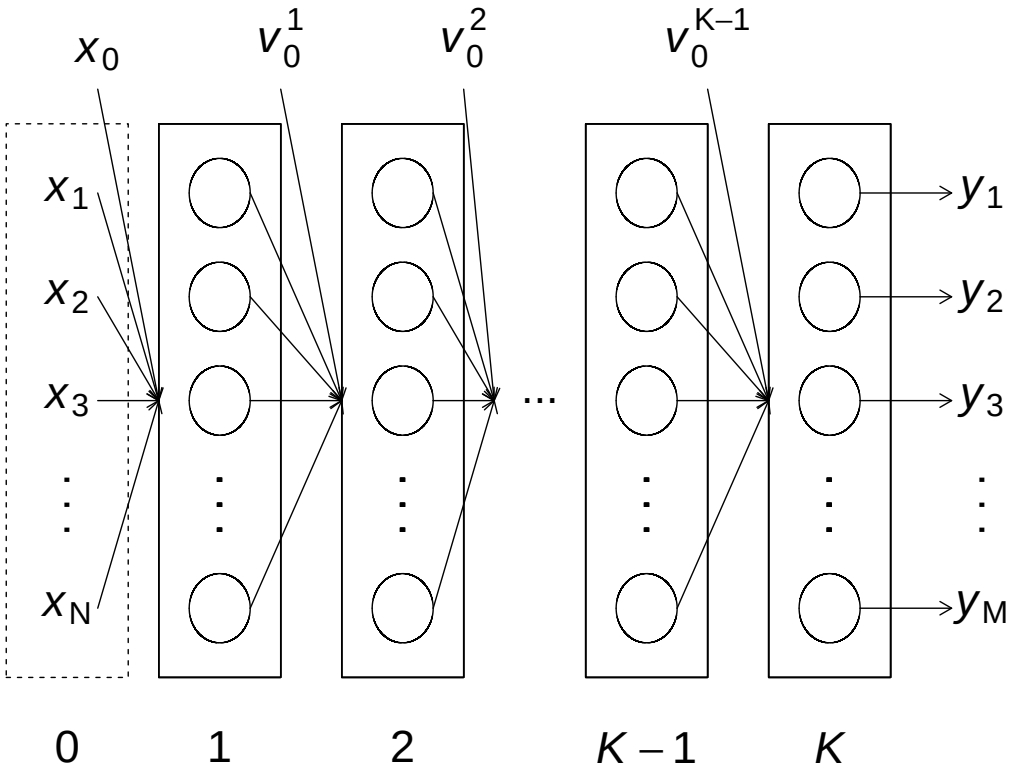
kus $m = 1, 2, \ldots, M$. (Swingler, 2001, lk 19)

3 Mitme varjatud kihiga närvivõrgustik

Mitme varjatud kihiga närvivõrgustik on edasiminek ühe varjatud kihiga mudelist. Nagu nimetusest oletada võib, siis mitme varjatud kihiga mudelis on varjatud kihte rohkem kui üks. Mudel toimib sarnaselt eelnevas peatükis kirjeldatud ühe varjatud kihiga mudelile. See tähendab, et varjatud kihtide vahel toimib sarnane loogika nagu varjatud ja väljundkihi vahel. Uued sisendid saadakse alati eelmisest kihist ning sedasi liigutakse mudeli väljundite poole. Igale kihile lisatakse eraldi juurde konstantne liige ning temale vastavad kaalud. Kihtide arvu tähistame antud töös tähega K ja kihis k olevate neuronite arvu R^k, kus $k = 1, 2, \ldots, K$. Mitme varjatud kihiga närvivõrgustikku iseloomustab joonis 4.

Varjatud kihtide arv sõltub lahendatavast ülesandest. Kõige lihtsamal juhul on ülesannet võimalik lahendada ainult sisend- ja väljundkihiga. Kui väljundkihis on ainult üks neuron, siis on tegu Rosenblatti pertseptroniga. Väljundkihis võib olla ka rohkem neuroneid, aga kui mudelis ei ole ühtegi varjatud kihti, siis suudab närvivõrgustik lahendada ainult lineaarseid juhte. Kui andmestiku kohta eelnev informatsioon puudub, siis tasub mudeli sobitamisel alustada alati taolisest juhust. Mudeli mittesobivuse korral tasub jätkata ühe varjatud kihiga närvivõrgustikuga. Varjatud kihid võimaldavad tunnuseid ümber paigutada, et väljundkihis taanduks ülesanne jällegi lineaarsele juhule. Taolise loogikaga on võimalik jätkata. (Swingler 2001, lk 61-62)

Tavaliselt piisab ülesannete lahendamiseks ühest või kahest varjatud kihist, aga neis kihtides võivad neuronite arvud minna liiga suureks ning sellepärast on mõtet kasutada rohkemaid kihte. See võimaldab vähendada kihis olevate neuronite arvu ja säästa mudeli treenimiseks kuluvat aega. Eelmises peatükis käsitletud valemid



Joonis 4. *Mitme varjatud kihiga närvivõrgustik, kus 0 on sisendkiht, K on väljundkiht ja vahepealsed on varjatud kihid*

(2) ja (3) üldistuvad kujule:

$$v_l^k = f\left(\sum_{i=0}^{R^{k-1}} w_{li}^k v_i^{k-1}\right), \tag{4}$$

kus $l = 1, 2, \ldots, R^k$ ja $k = 1, 2, \ldots, K$. Seejuures iga $R^0 = N$ ja $v_i^0 = x_i, i = 0, \ldots, R^0$ ning $R^K = M$ ja $v_j^K = y_j, j = 1, \ldots, R^K$.

4 Maatrikskuju

Tehisnärvivõrgustike kujutamiseks on vahel mugavam kasutada maatriksitel põhinevaid tehteid. Ülevaate saamiseks on kirjeldatud tehted mitme varjatud kihiga mudeli näitel.

Olgu tähistatud sisendite vektor

$$\mathbf{x} = \begin{pmatrix} x_1 & x_2 & \dots & x_N \end{pmatrix}^T,$$

väljundite vektor

$$\mathbf{y} = \begin{pmatrix} y_1 & y_2 & \dots & y_M \end{pmatrix}^T,$$

kaalude maatriksid

$$\mathbf{W^1} = \begin{pmatrix} w^1_{11} & w^1_{12} & \cdots & w^1_{1N} \\ w^1_{21} & w^1_{22} & \cdots & w^1_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w^1_{R^1 1} & w^1_{R^1 2} & \cdots & w^1_{R^1 N} \end{pmatrix},$$

$$\vdots$$

$$\mathbf{W^K} = \begin{pmatrix} w^K_{11} & w^K_{12} & \cdots & w^K_{1R^{K-1}} \\ w^K_{21} & w^K_{22} & \cdots & w^K_{2R^{K-1}} \\ \vdots & \vdots & \ddots & \vdots \\ w^K_{M1} & w^K_{M2} & \cdots & w^K_{MR^{K-1}} \end{pmatrix}.$$

Konstantidele $x_0, v^1_0, \dots, v^{K-1}_0$ vastavad kaalud on

$$\mathbf{w^1_0} = \begin{pmatrix} w^1_{10} & w^1_{20} & \dots & w^1_{R^1 0} \end{pmatrix}^{\mathbf{T}}$$

$$\mathbf{w^2_0} = \begin{pmatrix} w^2_{10} & w^2_{20} & \dots & w^2_{R^2 0} \end{pmatrix}^{\mathbf{T}}$$

$$\vdots$$

$$\mathbf{w_0^K} = \begin{pmatrix} w_{10}^K & w_{20}^K & \dots & w_{M0}^K \end{pmatrix}^{\mathbf{T}}.$$

Antud maatriksite kaudu on võimalik mitme varjatud kihiga närvivõrku avaldada kujul:

$$\mathbf{y} = \mathbf{f}(\mathbf{W^K f}(\dots \mathbf{f}(\mathbf{W^1 x} + \mathbf{w_0^1 x_0}) \dots) + \mathbf{w_0^K v_0^{K-1}}), \tag{5}$$

kus funktsiooni f rakendamine vektorile tähendab funktsiooni rakendamist elementhaaval. Töö järgnevates peatükkides on kasutatud närvivõrkude maatrikskuju, et tähistused oleks kergemini jälgitavad.

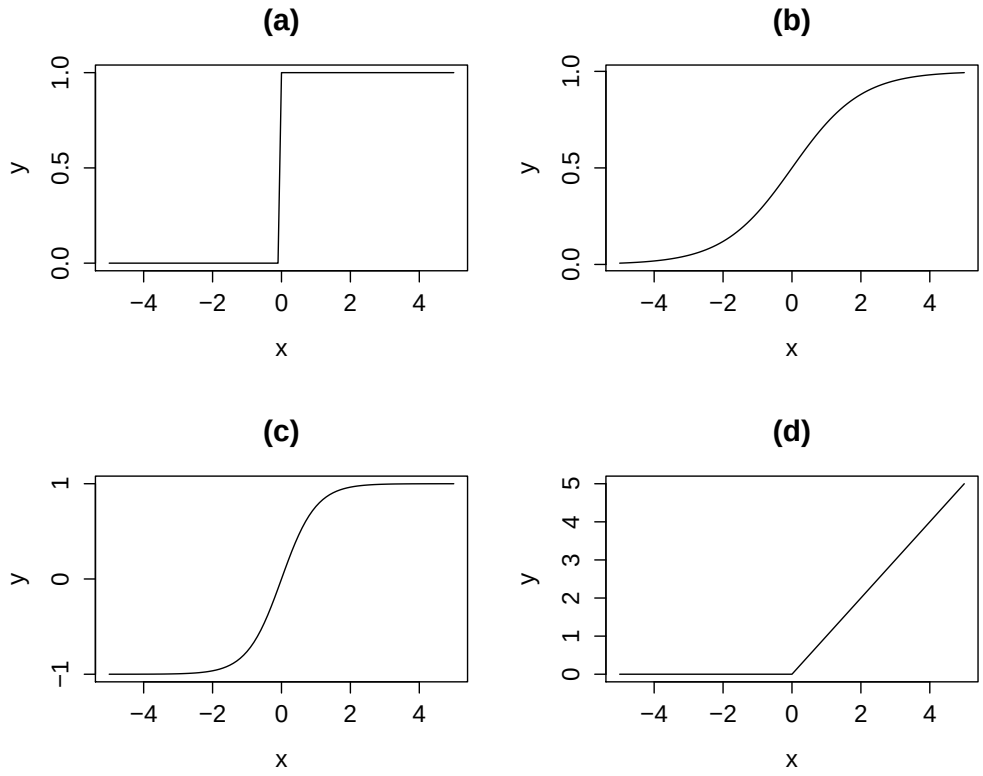
5 Aktiveerimisfunktsioonid

Aktiveerimisfunktsioonid mängivad mudeli koostamisel olulist rolli. Olenevalt ülesandest võivad aktiveerimisfunktsioonid oluliselt kiirendada meetodi koondumist, mistõttu on tähtis teada valikute häid ja halbu külgi. Idee poolest pole funktsioonide valik piiratud. Valida võib rakendajale endale meeldivaid funktsioone, aga vastavalt ülesandele peavad neil olema täidetud teatud omadused. Näiteks gradientlaskumisega meetodi korral on tarvilik funktsiooni diferentseeruvus. Keerukamates närvivõrkudes on võimalik iga neuroni jaoks valida oma aktiveerimisfunktsioon ning saavutada sellega optimaalne tulemus, antud töös selliseid juhte ei kajastata ja kõigis neuronites on võetud ühesugune aktiveerimisfunktsioon. Selles peatükis on kirjeldatud viit enamlevinud aktiveerimisfunktsiooni ning välja on toodud nende head ja halvad küljed.

Treppfunktsioon ehk lävendifunktsioon on üks esimestest kasutusele võetud aktiveerimisfunktsioonidest. Nimetatud funktsioon lävendiga a on kujul:

$$f(x) = \begin{cases} 1, & \text{kui } x \geq a, \\ 0, & \text{kui } x < a. \end{cases} \tag{6}$$

Funktsiooni graafik on näha joonisel 5 (a).Treppfunktsiooni korral ei pruugi väljunditeks olla ainult väärtused üks ja null. Arvud saab valida vastavalt ülesandele. Valemist (6) on näha, et funktsioon väljastab väärtuse 0, kui sisend on alla lävendi ja väärtuse 1, kui sisend ületab lävendit. Sarnane loogika toimib ka bioloogiliste närvivõrkude korral, kus neuron väljastab impulsi, kui sisendsignaal on piisavalt tugev. Just bioloogiliste närvivõrkude loogikast tulenevalt kasutati esimestes tehisnärvivõrkudes aktiveerimisfunktsioonina treppfunktsiooni, sest prooviti täpselt jäljendada bioloogilist närvivõrku. Funktsioon kaotas oma tähtsuse, kui kasutusele tuli gradientlaskumise meetod, milles on väga tähtsal kohal tuletised. Kuna treppfunktsioon ei ole lävendpunktis diferentseeruv ning ülejäänud osas on tuletis 0,



Joonis 5. *(a) Treppfunktsiooni graafik, (b) sigmoidfunktsiooni graafik, (c) hüperboolse tangensfunktsiooni graafik, (d) kärbitud lineaarse funktsiooni graafik*

siis ei olnud treppfunktsiooni kasutamine enam hea lahendus ning asemele tulid siledamad funktsioonid. Samuti ei ole treppfunktsiooniga võimalik väljendada nulli ja ühe vahel asuvaid tõenäosusi. (Kriesel, lk 37)

Sigmoidfunktsioon on väga levinud aktiveerimisfunktsioon. Sigmoidfunktsioon avaldub kujul:

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}. \tag{7}$$

Valemist (7) on näha, et sigmoidfunktsioon on samaväärne logistilises regressioonis kasutatava *logit* funktsiooni pöördfunktsiooniga. Funktsiooni muutumispiirkond on nullist üheni, mis tagab sisendite kokkusurumise. Kokkusurumine on vajalik, et järgmise kihi neuronite jaoks ei oleks sisendid liiga suured. Sigmoidfunkt-

siooni laialdase leviku põhjus on sarnane treppfunktsioonile. Nimelt on võimalik sellega kirjeldada neuroni ergastumist, kus väikese väärtuse korral on väljundiks ligikaudu null ehk neuron ei edasta informatsiooni ja suure väärtuse korral on väljundiks ligikaudu üks ehk neuron on ergastunud olekus ja edastab informatsiooni. Sigmoidfunktsiooni graafik on näha joonisel 5 (b). Sigmoidfunktsioon ei ole täiuslik ja sellel esinevad mõned probleemid. Absoluutväärtuselt suurte sisendite korral läheneb tuletis nullile, mistõttu tagasilevi meetodiga treenimine muutub väga aeglaseks. Seetõttu on oluline valida sobivad algkaalud, et neuronid ei oleks kohe küllastunud olekus, tagades sellega kiirema õppimise. Teiseks sigmoidfunktsiooni puuduseks on alati positiivne väljund, mistõttu mitme kihiga mudelis saavad hilisemad kihid alati mittenegatiivseid sisendeid. Samas ei ole tegu eriti suure probleemiga ja vajadusel võib kasutada järgmisena kirjeldatud funktsiooni, mis selle puuduse eemaldab. (Nielsen, 2015)

Hüperboolne tangensfunktsioon on väga sarnane sigmoidfunktsioonile. Antud funktsiooni graafik on joonisel 5 (c) ning on esitatav kujul:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{8}$$

Hüperboolse tangensfunktsiooni muutumispiirkond on miinus ühest üheni ning tema interpretatsioon on sarnane sigmoidfunktsioonile. Nimetatud funktsiooni puuduseks on samuti halb õppimisvõime absoluutväärtuselt suurte sisendite korral. Hüperboolset tangensfunktsiooni võib kujutada kui sigmoidfunktsiooni teisendust, mis on keskmistatud nulli ümber. Sigmoidfunktsiooni on võimalik tangensfunktsioonist avaldada kujul: $\sigma(x) = \frac{1+\tanh(x/2)}{2}$. (Nielsen, 2015)

Kärbitud lineaarne funktsioon (*ReLU*) on levinud väga mitme kihiga närvivõrkude korral ning on soovituslik aktiveerimisfunktsioon otsesuunatud närvivõrkude kasutamisel. Nimetatud funktsioon eemaldab puuduse, kus sigmoid- ja tangensfunktsiooni tuletised on nullilähedased väga suurte ja väga väikeste väärtuste kor-

ral. Kärbitud lineaarse funktsiooni korral on tuletis iga positiivse sisendi korral konstante ning positiivne, mis väga paljusid kihte sisaldavate närvivõrkude korral võimaldab treenida ka esimesi kihte. Funktsioon esitub kujul:

$$f(x) = \max(0, x). \tag{9}$$

Kärbitud lineaarne funktsioon suurendab paljudel juhtudel treenimise kiirust, sest positiivse sisendi korral on tuletis alati üks. Graafik on nähtav joonisel 5 (d). Lisaks sellele on tegu arvutuslikult kiire protsessiga, sest ei pea leidma eksponenti ja jagatist. Kuigi antud funktsioon ei ole nullpunktis diferentseeruv, siis praktikas leitakse lihtsaid viise, kuidas sellest mööda pääseda (nt. kasutatakse mõnd siledat lähendit või võrdsustatakse tuletis punktis 0 samuti väärtusega 0). Suurimaks puuduseks on funktsiooni „suremine". Kui funktsiooni argument on negatiivne, siis tuletis on null ja neuron lõpetab treenimise. Sellest tulenevalt peab treenimisel olema väga tähelepanelik, et liiga suur osa neuronitest tarbetuks ei muutuks. Puuduste lahendamiseks on kasutatud erinevaid kärbitud lineaarse funktsiooni modifikatsioone. (Goodfellow, 2016, lk 173-174) Võimalikest modifikatsioonidest antud töös lähemalt ei kirjutata, aga parema ülevaate annab artikkel „Empirical Evaluation of Rectified Activations in Convolution Network"(Xu, et al., 2015).

Softmax funktsiooni kasutatakse klassifitseerimisülesannete korral. Tavaliselt kasutatase seda funktsiooni väljundkihis, sest *softmax* funktsiooni väljundeid on võimalik kasutada tõenäosushinnangutena. Sigmoidfunktsioone kasutades see alati võimalik ei ole, sest väljundite summa võib tulla suurem ühest. Arvutamiseks kasutatakse valemit:

$$v_l^k = \frac{e^{z_l^k}}{\sum_{i=0}^{R^k} e^{z_i^k}}, \tag{10}$$

kus $z_l^k = \sum_{i=0}^{R^{k-1}} w_{li}^k v_i^{k-1}, l = 1, 2, \ldots, R^k$ ja $k = 1, 2, \ldots, K$. Valemist on näha, et funktsiooni väljundi arvutamiseks peame teadma kõigi kihis olevate neuronite

sisendeid. Klassifitseerimisülesande korral näitab igast väljundkihi neuronist saadud väljund vastava klassi tõenäosust. (Nielsen, 2015)

6 Närvivõrgustiku treenimine ja tagasilevi meetod

Treenimise ülesandeks on leida sobivad kaalud, et mudel sobiks hästi andmetega. Selliste kaalude leidmiseks on üks levinumaid treenimismeetodeid tagasilevi meetod. Selles peatükis seletatakse täpsemalt, mida kujutab endast gradientlaskumisega tagasilevi meetod ja kuidas seda närvivõrgustike treenimisel kasutada.

Treenimise kesksel kohal on kaofunktsioon, mis mõõdab mudeli sobivust treeningandmetega. Mudeli treenimine kujutab kaofunktsiooni minimiseerimist eesmärgiga viia otsitav mudel võimalikult hästi kooskõlla andmestikuga. Oluline on teada, et minimiseerides on võimalik muuta ainult mudeli kaale, sest sisendid ja oodatavad tulemused on treeningandmestikuga fikseeritud.

Minimiseerimisülesande lahendamiseks tuleb valida kaofunktsioon, millega opereerima hakata. Vastavalt lahendatavale ülesandele on võimalik kasutada erinevaid funktsioone. Regressioonülesande lahendamisel kasutatakse tihti kaofunktsioonina keskmist ruutviga

$$L(W) = \frac{1}{2n} \sum_{i=1}^{n} \|\mathbf{y_{o_i}} - \mathbf{y_i}\|^{\mathbf{2}}, \tag{11}$$

kus n on treeningandmestiku suurus ja W on mudeli kaalud. Konstant $\frac{1}{2}$ lihtsustab keskmise ruutvea tuletise valemit ning seega on ta kaasatud valemisse. Seejuures tähistab $\|\mathbf{v}\|$ vektori $\mathbf{v}$ pikkuse funktsiooni ning $\mathbf{y_i}$ ja $\mathbf{y_{o_i}}$ on vektorid, mis näitavad ennustatud ja oodatavaid väärtuseid. Klassifitseerimisülesande korral on samuti võimalik kasutada keskmist ruutviga, aga paremaid tulemusi annab ristentroopia, mis esitub kujul

$$L(W) = -\frac{1}{n} \sum_{i=1}^{n} (\mathbf{y_{o_i}} \ln \mathbf{y_i} + (\mathbf{1} - \mathbf{y_{o_i}}) \ln(\mathbf{1} - \mathbf{y_i})), \tag{12}$$

kus tähistused on samad, mis keskmise ruutvea valemis (11). (Nielsen, 2015)

6.1 Gradientlaskumine

Gradientlaskumist kasutatakse veafunktsiooni miinimumi leidmiseks. Kahemõõtmelises ruumis võib protsessi kujutada järgnevalt. Funktsiooni $L(w_1, w_2)$ minimiseerimiseks muudetakse tunnuseid w_1 ja w_2. Olgu pisikesed muutused suurustega Δw_1 ja Δw_2. Gradienti kasutades muutub kaofunktsiooni väärtus

$$\Delta L(w_1, w_2) \approx \frac{\partial L(w_1, w_2)}{\partial w_1}\Delta w_1 + \frac{\partial L(w_1, w_2)}{\partial w_2}\Delta w_2 \tag{13}$$

võrra. Muutude Δw_1 ja Δw_2 väärtused valitakse nõnda, et $\Delta L(w_1, w_2)$ väärtus oleks negatiivne. Seejärel muudetakse tunnuste w_1 ja w_2 väärtusi. Korrates analoogset protsessi on võimalik jõuda funktsiooni $L(w_1, w_2)$ miinimumi. (Nielsen, 2015)

Muutude Δw_1 ja Δw_2 leidmiseks võib valemi (13) ümber kirjutada kujul

$$\Delta L(w_1, w_2) \approx \boldsymbol{\nabla L}(w_1, w_2)^T \Delta \boldsymbol{w}, \tag{14}$$

kus $\boldsymbol{\nabla L}(w_1, w_2) = \left(\frac{\partial L(w_1,w_2)}{\partial w_1}, \frac{\partial L(w_1,w_2)}{\partial w_2}\right)^T$ ning $\boldsymbol{\Delta w} = (\Delta w_1, \Delta w_2)^T$. Antud valem võimaldab näha, et kui valida muutuse suuruseks $\boldsymbol{\Delta}\mathbf{w} = -\eta \boldsymbol{\nabla L}(w_1, w_2)$, kus η on väike positiivne konstant, siis kaofunktsiooni muut avalduks kujul

$$\Delta L(w_1, w_2) \approx -\eta \boldsymbol{\nabla L}(w_1, w_2)^T \boldsymbol{\nabla L}(w_1, w_2) = -\eta \|\boldsymbol{\nabla L}(w_1, w_2)\|^2.$$

Kuna $\|\boldsymbol{\nabla L}(w_1, w_2)\|^2 \geq 0$, siis $\Delta L(w_1, w_2) \leq 0$ ning seetõttu saabki kindel olla, et antud algoritmi kasutades väheneb kaofunktsioon. (Nielsen, 2015)

6.2 Tagasilevi meetod

Antud peatükis kirjeldatakse lähemalt keskmise ruutvea kasutamist tagasilevi meetodi korral. Ristentroopia puhul on käsitlus analoogne.

Alapeatükk on kirjutatud raamatu „Neural Networks and Deep Learning" (Nielsen, 2015) põhjal.

Meetodi lihtsamaks mõistmiseks vaatame esialgu keskmise ruutvea $L(W)$ ühte liidetavat $L_j = \frac{1}{2}\|\boldsymbol{y_o} - \boldsymbol{y}\|^2, j = 1, \ldots, n$. See on võimalik, sest $L(W) = \frac{1}{n}\sum_{j=1}^{n} L_j$ puhul on tegu keskmisega. Tihti on treeningandmestik piisavalt suur, et kõigi treeningandmestikus olevate objektide läbimine ja keskmise ruutvea leidmine võtaks palju aega. Selle probleemi lahendamiseks on võimalik treeningandmestik jagada miniplokkideks, mille pealt arvutatud kaofunktsiooni väärtus on piisavalt heaks hinnanguks treeningandmestiku keskmisele. Seega $L(W) \approx \frac{1}{n_i}\sum_{j=1}^{n_i} L_j$, kus n_i on vastava miniploki suurus. Pärast iga miniploki läbimist uuendatakse mudeli kaale ning sedasi minimiseeritakse tegelik kaofunktsioon $L(W)$.

Antud töös vaadatakse miniplokktreeningu erijuhtu, kus $n_i = 1$ iga i korral. Sellist treeningut nimetatakse ka *online* treeninguks, kus pärast iga objekti töötlemist uuendatakse mudeli kaalud. *Online* treening võimaldab lihtsustada kaofunktsiooni arvutamist, sest korraga uuritakse ainult ühte liidetavat L_j. Esituse lihtsuse huvides tähistame $L_j = L$. Keskmist ruutviga on võimalik nüüd ühe andmestiku objekti põhjal arvutada valemist $L = \frac{1}{2}\sum_{m=1}^{M}(y_{o_m} - y_m)^2$. Suuremate miniplokkide kasutamisega on võimalik tutvuda eelpool nimetatud raamatu abil.

Tagasilevi meetod seisneb informatsiooni liikumises vastupidises suunas, seega alustame väljundkihist. Väljundkihi vead olgu vektoris $\boldsymbol{\delta^K}$. Vektorit $\boldsymbol{\delta^K}$ on võimalik arvutada valemist

$$\boldsymbol{\delta^K} = \boldsymbol{\nabla_y L} \otimes f'(\boldsymbol{z^K}), \tag{15}$$

kus $\boldsymbol{\nabla_y L}$ on vektor, mis sisaldab elemente $\frac{\partial L}{\partial y_m}, m = 1, 2, \ldots, M$, $\otimes$ on Hadamardi korrutis ja f on valitud aktiveerimisfunktsioon. Vektor $\boldsymbol{z^K}$ sisaldab väljundkihi neuronite kaalutud sisendite summasid. Antud juhul, kui veafunktsiooniks on kesk-

mine ruutviga, siis $\boldsymbol{\nabla_y L} = (\boldsymbol{y} - \boldsymbol{y_o})$ ning vead on arvutatavad valemist

$$\boldsymbol{\delta^K} = (\boldsymbol{y} - \boldsymbol{y_o}) \otimes f'(\boldsymbol{z^K}). \tag{16}$$

Järgnevate kihtide korral on arvutuskäik sarnane. Ainukeseks erinevuseks on korrutise esimene tegur. Esimene liige leitakse eelmisest kihist tekkinud vigade ja kaalude kaudu. Kui edasilevi korral mõjutab neuroni väljund järgmist kihti kaalutud väljundite võrra, siis tagasilevi korral mõjutatakse neuronit kaalutud vigade võrra. Arvutuslikult on võimalik vigu leida järgnevalt:

$$\boldsymbol{\delta^k} = ((\boldsymbol{W^{k+1}})^T \boldsymbol{\delta^{k+1}}) \otimes f'(\boldsymbol{z^k}), \tag{17}$$

kus $k = K-1, K-2, \ldots, 1$. Konstantide kaalude vigade arvutamine käib sarnaselt.

Iga kaalu kohta on võimalik arvutada osatuletis järgnevalt:

$$\frac{\partial L}{\partial w_{ji}^k} = v_i^{k-1} \delta_j^k. \tag{18}$$

Tuletades meelde eelmises alapeatükis kasutatud gradientlaskumise meetodit, siis on teada kõik vajalik, et uuendada kaale. Täpsemalt kasutatakse uute kaalude leidmiseks valemit:

$$w_{ji_{uus}}^k = w_{ji}^k - \eta v_i^{k-1} \delta_j^k. \tag{19}$$

Konstantide kaalude uuendamiseks kasutatakse valemit, kus liige $v_i^{k-1} = 1$, sest konstant ise on alati üks. Korrates treenimise protsessi on võimalik leida miinimum kaofunktsioonile.

Kirjeldatud algoritmi põhjal on võimalik anda edasi ideed, miks toimib peatükis 1 kirjeldatud algoritm vaatamata sellele, et pertseptronis kasutatav treppfunktsioon ei ole diferentseeruv. Võrdleme pertseptroni mudelis kasutatut tagasilevi algoritmiga. Antud juhul on otstarbekas minimiseerida valesti klassifitseerimiste

arvu. Antud algoritmi puhul on samuti tegemist online treeninguga. Kaale muudetakse vastavalt $e = y_0 - y$ väärtusele. Juhul kui $y_0 = -1$ ja $y = 1$, siis $e = -2$. Kui $y_0 = 1$ ja $y = -1$, siis $e = 2$. Ülesanne seisneb summa muutmises. Kui $e = -2$, siis $\sum w_i x_i > 0$, aga tarvis oleks vastupidist ning seega peame summat vähendama. Kui $e = 2$, siis $\sum w_i x_i < 0$, aga jällegi on vaja vastupidist ning seega peame summat suurendama. Näeme, et summa vajalik muutmise suund on määratud e-ga. Teame valemist (13), et $\Delta F \approx \sum_{i=0}^{N} \frac{\partial F}{\partial w_i} \Delta w_i$, kus Δw_i on väikesed ja $F = \sum_{i=0}^{N} w_i x_i$. Olgu Δw_i meie valemis antud kujul $\Delta w_i = \eta x_i e$. Siis $\Delta F \approx \eta \boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{x} e = \eta \|\boldsymbol{x}\|^2 e$. Kuna $\eta > 0$ ja $\|\boldsymbol{x}\|^2 \geq 0$, siis üle jääb e, mis paneb paika õige suuna. Seega idee on analoogne, aga diferentseeruva funktsiooni korral toimub treenimine üldiselt kiiremini ja on rakendatav ka keerulisematel juhtudel.

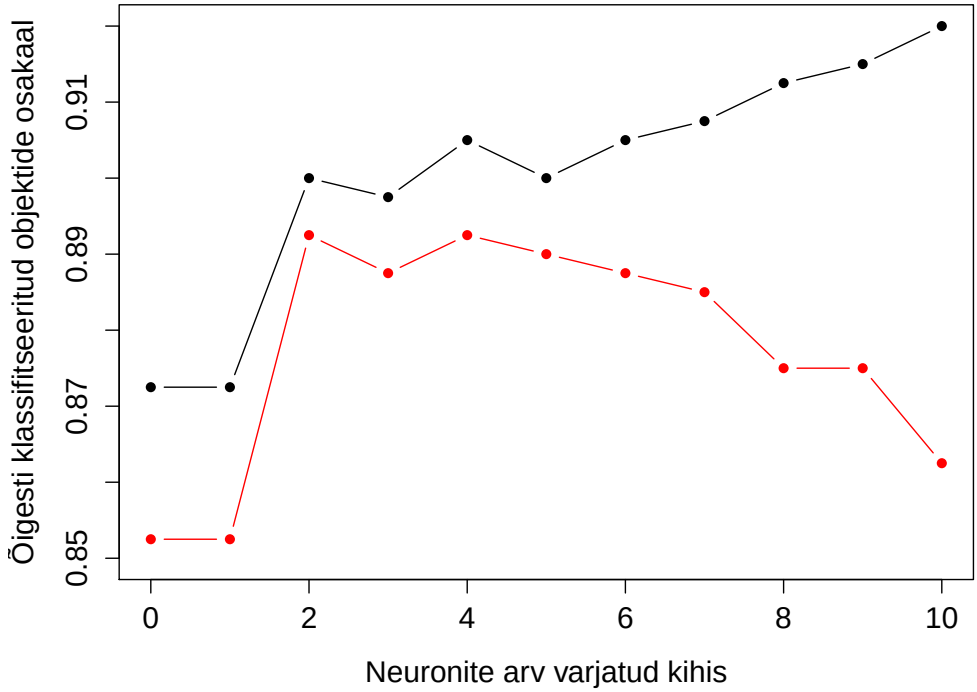
7 Ülesobitamine

Peale tehisnärvivõrgustiku struktuuri, aktiveerimisfunktsioonide ja treenimise peab teadma veel mitmeid asju, et närvivõrkudega korrektselt tegutseda.

Kaalude algväärtuste valik omab suurt mõju lõpptulemusele. Siiani on töös mainitud, et kaalude valik peaks olema juhuslik. Juhuslikkus võimaldab kaaludel koonduda erinevatesse lokaalsetesse miinimumidesse. Seega võib mudelit mitu korda algusest treenima hakata ja jõuda erinevate lahendusteni. Algkaaludeks on soovituslik valida juhuslikud nullilähedased väärtused. Juhul, kui kasutatakse sigmoidfunktsiooni, siis nullpunktis on funktsiooni tõus peaaegu lineaarne. Seega alustatakse lineaarsest mudelist ning mittelineaarsus tekitatakse kohtadesse, kus seda ka tegelikult vaja on. Kui kaalude algväärtused võtta võrdseks nullidega, siis tekib sümmeetrilisus ja tuletised on võrdsed nulliga, mistõttu pole mudelit võimalik treenida. Liiga suured algkaalud võivad viia halva lahenduseni.(Hastie, 2009, lk 397-398)

Tehisnärvivõrgustikega töötades on ülesobitamine sage probleem. Teema paremaks valdamiseks on mõistlik teha selgeks treeningandmestiku ja testandmestiku mõisted. Treeningandmestikuks nimetatakse andmestikku, mis sisaldab mudeli treenimiseks kasutatavaid objekte. Testandmestikuks nimetatakse andmestikku, kus on treenitud mudeli headuse kontrollimiseks kasutatavad andmed. Üldjuhul on mõlema andmestiku korral uuritav ehk ennustatav tunnus teada. Juhul kui uuritav tunnus on teada, siis nimetatakse treenimist juhendajaga õppeks. Siiani on terve töö põhinenud taolistel juhtudel. Võimalik on rakendada ka juhendajata õpet ehk mudel proovib sisendite põhjal ise leida mustreid. Kuna tegu on laia valdkonaga, siis antud töös seda varianti ei kajastata. (Kriesel, lk 52-53,57)

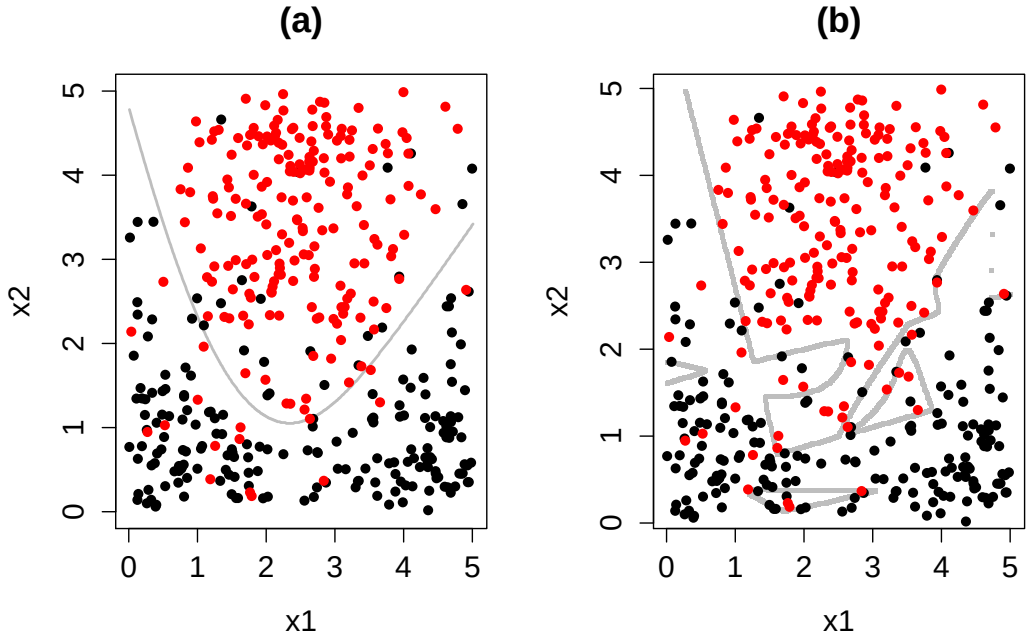
Ülesobitamine on olukord, kus mudel treenitakse liiga täpselt kirjeldama treenin-



Joonis 6. *Õigesti klassifitseeritud objektide osakaal treeningandmestikus (must) ja testandmestikus (punane) varjatud kihis olevate neuronite arvu kaupa*

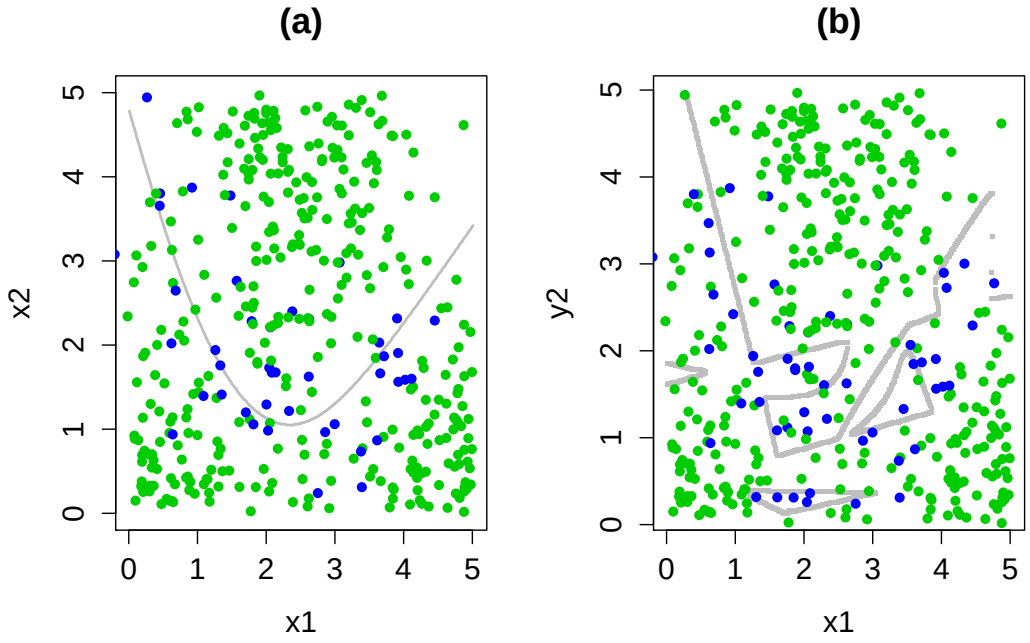
gandmestikku ning seetõttu võib langeda ennustamise täpsus testandmestikus või andmetel üldse. Üsna sagedasti puututakse ülesobitamisega kokku, kui valitakse tehisneuronite arvu. Olukorda kirjeldab hästi joonis 6. Jooniselt on näha, et ülesobitamine ilmneb juba kolme varjatud kihis oleva neuroniga. Siinkohal tasub mainida, et joonis on genereeritud andmetel, kus oli kahte liiki punkte ning punktide grupi ennustamiseks oli võimalik kasutada kahte tunnust. Andmete genereerimiseks ja jooniste tegemiseks kasutatud programmi *R* kood on välja toodud töö lisades. Ülesobitamise tunneb ära iseloomuliku graafiku järgi. Alates mingist neuronite arvust ei muutu testandmestikus õigesti klassifitseeritavate objektide osakaal paremaks, vaid jääb samaks või hakkab hoopis halvenema nagu joonisel 6.

Sellisel juhul tuleks treenimine lõpetada punktis, kus testandmestikul saadud viga on kõige väiksem. Antud juhul oleks sobivaks valikuks kahe neuroniga mudel. Ülesobitamine on kerge esinema, sest ülesanne on treenida võimalikult väikese veaga mudel. Treeningandmetel vea leidmine viib olukorrani, kus iga uus neuron suudab kirjeldada mingi osa, mis eelnevalt kirjeldamata jäi ning seega viga väheneb. Sedasi võime jõuda väga suure mudelini, mis testandmestikul annab suure vea.



Joonis 7. *(a) Kahe varjatud kihis oleva neuroniga mudeli eralduspiirid ning treening andmestiku punktid, (b) kümne neuroniga mudeli klassifitseerimispiir ja treening andmestik.*

Teiseks võimaluseks ülesobitamist ära tunda on vaadata graafikul asuvaid piirkondi. Kahte võimalikku piirkonda on näha joonisel 7. Joonisel (a) on näha eraldajat, mis võiks olla antud ülesande lahendamiseks üsna sobiv. Joonisel (b) on treenimine tekitanud eraldatud piirkondi ning teravaid poolsaari, mis iseloomustab



Joonis 8. *(a) Kahe varjatud kihis oleva neuroniga mudeli eralduspiirid ja valesti klassifitseeritud punktid (sinine), (b) kümne neuroniga mudeli klassifitseerimispiir ja valesti klaasifitseeritud punktid (sinine)*

ülesobitamist. Joonisel 7 on visualiseeritud treeningandmestikku, mille puhul on näha, et kümne neuroniga mudel ennustab täpsemini kui kahe neuroniga mudel. Kui vaadata tulemusi testandmestikul, joonis 8, siis on näha, et olukord on vastupidine. Seega on tegu ülesobitamisega.

Ülesobitamise vastu on võimalik rakendada mitmeid erinevaid meetmeid. Üheks võimalikuks lahenduseks on lõpetada treenimine enne kaofunktsiooni miinimumi jõudmist. Teiseks võimaluseks on rakendada regulariseerimist, mis seisneb kaofunktsioonile karistusliikme lisamises. Karistusliige mõõdab funktsiooni siledust ning seega sunnib leidma optimaalset lahendit kaofunktsiooni ja karistusliikme minimiseerimisel. Regulariseerimisest annab hea ülevaate raamat „Neural Networks for Pattern Recognition”(Bishop, 1995).

8 Funktsiooni lähendamine närvivõrkude abil

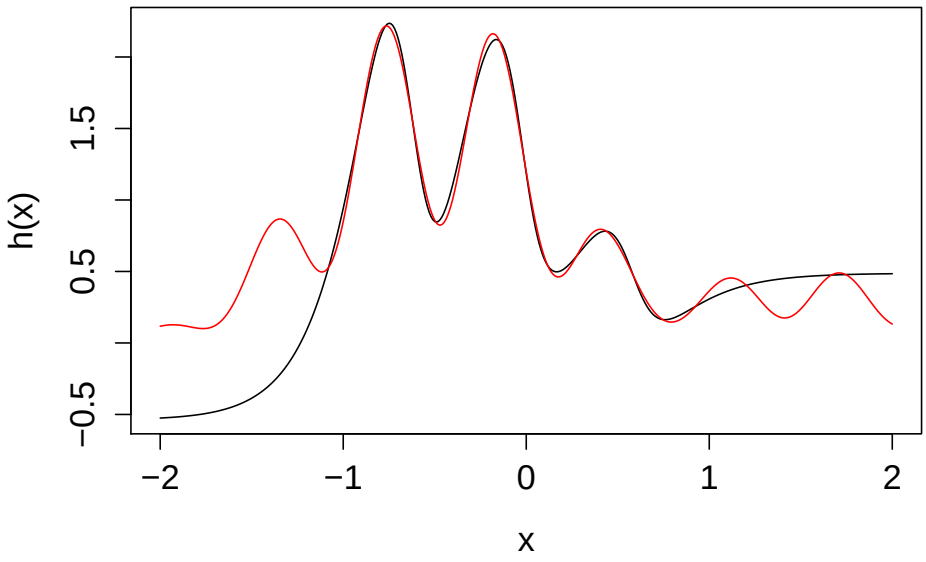
Tehisnärvivõrkudega töötamine ja nende täielik mõistmine nõuab rakendajalt üsna palju matemaatilist tausta. Samas ei ole kõige paremini sobiva mudeli leidmiseks võimalik teha ainult analüütilisi arvutusi, vaid tihti on vaja läheneda eksperimentaalselt ning võrrelda mudelite headust pärast treenimist. Eksperimenteerimise teel leitakse sobiv arv kihte, iga kihi neuronite arv, treenimisel kasutatav õppimiskonstant ja vajadusel ka aktiveerimisfunktsioon, kui see ei ole eelnevalt ülesande püstitusega ette määratud. Antud peatükk sisaldab autori poolt tehtud näiteid närvivõrkude kohta, mis peaksid lihtsustama närvivõrkude toimimise loogika mõstmist.

Peatüki näidete illustreerimiseks kasutatakse funktsiooni

$$h(x) = \frac{(2 - \sin(10x)(1,5 - \sin(2x))e^{-0.5x^2}}{\sqrt{2\pi}}. \tag{20}$$

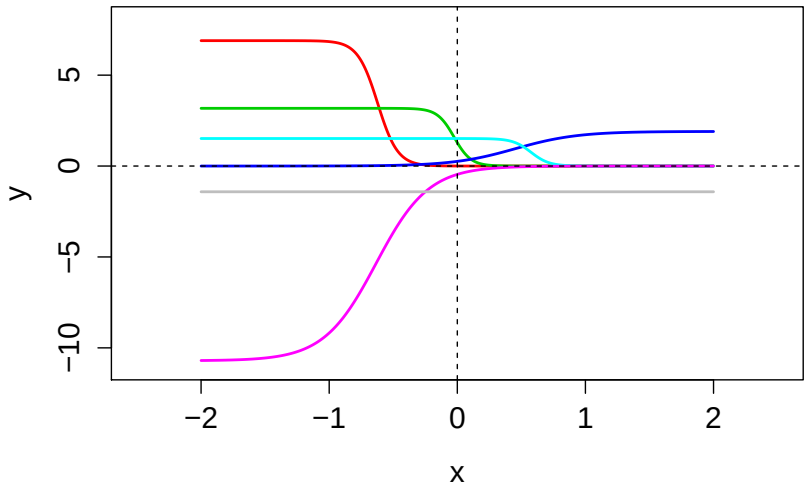
Funktsiooni lähendamiseks on genereeritud treeningandmestik, kus $x \in [-1; 1]$. Kuna funktsiooni analüütiline kuju on teada, siis on seda ka funktsiooni oodatavad väärtused iga x korral. Saadud andmete põhjal on funktsioonile sobitatud ühe varjatud kihiga mudel, mis koosneb viiest neuronist. Aktiveerimisfunktsioonina on kasutatud sigmoidfunktsiooni. Funktsioonist (20) ja sobitatud mudel on esitatud joonisel 9.

Esimene järeldus, mis on väga oluline ka teiste rakendatavate närvivõrgustike puhul, on mudeli ennustamise piirkond. Mudel treeniti andmetega, mis olid genereeritud vahemikus -1 kuni 1 ning seega suudab ka mudel ennustada ainult nimetatud vahemikus. Kirjeldatud olukorda iseloomustab hästi joonis 9, kus on näha, et punase joonega tähistatud analüütiline funktsioon ja musta joonega tähistatud mudeli poolt ennustatav funktsioon kattuvad üsna hästi vahemikus $x \in [-1; 1]$, aga lahknevad piirkonnast väljaspool.



Joonis 9. *Funktsioon (20) on punane ja sobitatud mudel must*

Eelnev järeldus on üsna intuitiivne ning aimatav. Järgnevas lõigus proovitakse lahti mõtestada iga neuroni ülesannet närvivõrgustikus ning autori arvates parandab see närvivõrgustike idee mõistmist tunduvalt. Selleks vaadatakse iga neuronit eraldi, aga jäetakse talle samad kaalud, mis olid terviklikus mudelis. Parema ülevaate saamiseks võib vaadata joonist 10. Jooniselt on näha, et iga neuron varjatud kihis kirjeldab osa lähendatavast funktsioonist (20). Seega võib kujutada, et lõplik hinnatav mudel pannakse kokku nii mitmest sigmoidfunktsioonist, kui on varjatud kihis neuroneid. Summeerides üle kõigi tekkinud sigmoidfunktsioonide on võimalik taastada esialgne mudeliga hinnatud kuju. Siinkohal võib märkida, et kui mudelisse on kaasatud liiga palju neuroneid, siis liigsed neuronid on peaaegu lineaarsed ja seetõttu panustavad lõplikusse mudelisse vähe. Kui neuroneid on võetud liiga vähe, siis võib jääda mõni funktsiooni konarus kirjeldamata ning seega hinnatakse mudel valesti. Seetõttu on funktsioonide lähendamisel mõistlikum



Joonis 10. *Iga varjatud kihi neuroni poolt kirjeldatav osa*

valida suurem mudel, aga näiteks klassifitseerimisülesannet lahendades tasub olla ettevaatlikum, sest suurem mudel kipub kergesti ülesobituma.

Kasutatud kirjandus

[1] Bishop, C. M., (1995), *Neural Networks for Pattern Recognition*, Oxford University Press.

[2] Goodfellow, I., Bengio, Y. ja Courville, A., (2016), *Deep Learning*, http://www.deeplearningbook.org [26.04.2016].

[3] Hastie, T., Tibshirani, R. ja Friedman, J., (2009) *The Elements of Statistical Learning. Data Mining, Inference, and Prediction. Second Edition*, https://web.stanford.edu/∼hastie/local.ftp/Springer/OLD/ESLII_print4.pdf [21.04.2016].

[4] Kriesel, D., *A brief Intoduction to Neural Networks*, http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf [21.04.2016].

[5] Nielsen, M. A. (2015), *Neural Networks and Deep Learning*, Determination Press.

[6] Shiffman, D., (2012), *The Nature of Code: Simulating Natural Systems with Processing*, http://natureofcode.com/book/chapter-10-neural-networks/ [15.04.2014].

[7] Swingler, K., (2001), *Applying Neural Networks. A Practical Guide*, Academic Press.

[8] Xu, B., et al., (2015), *Empirical Evaluation of Rectified Activations in Convolution Network*, http://arxiv.org/pdf/1505.00853.pdf [26.04.2016].

Lisad

```
#Laen alla kasutatavad paketid
library(dplyr)
library(neuralnet)
library(pheatmap)
library(nnet)
#Genereerin andmestiku
par(mfrow=c(1,1))
generate.data<-function(n,seed){
  set.seed(seed)
  n<-n/2
  x1<-c(rchisq(n,3)/3, 5-rexp(n,1))
  y1<-c(rchisq(n,3)/3, rchisq(n,3)/3)

  x2<-rnorm(2*n,2.5)
  y2<-c(5-rchisq(n,3)/3, rnorm(n,2.5))
  return(data.frame(x=c(x1,x2), y=c(y1,y2), type=c(rep("a",n*2),
      rep("b",n*2))))
}

#Genereerin treening andmed
training.data<-generate.data(200,15467)
training.data1<-cbind(training.data[, 1:2], class.ind(training.
    data$type))

#Joonis
plot(training.data$x,training.data$y, xlim=c(0,5), ylim=c(0,5),
    col = training.data$type, pch=16)


#Genereerin test andmed
test.data<-generate.data(200,5642542)
test.data1<-cbind(test.data[, 1:2], class.ind(test.data$type))


#Funktsioon, mis otsustab klassifitseerimise klassi
zeroone.fct <- function(x){
  if (x>=0.5) return(1)
```

```
  else return(0)
}
zeroone.data<-function(matrix){
  return(apply(matrix, MARGIN=c(1,2),FUN=zeroone.fct))
}


#\~Oigesti klassifitseeritud neuronite arv

correct<-function(data, model){
  correct=0
  for(i in 1:dim(data)[1]){
    pred.1<-compute(model, data[i,-3:-4])
    pred.2<-zeroone.data(pred.1$net.result)
    if (pred.2[,1] == data[i,3])
      correct=correct+1
  }
  return(correct/dim(data)[1])
}

#Leian 0-10 neuroniga \~oigete objektide klassifitseerimise arvu
training<-c()
test<-c()
#Iga mudeli juures on m\"a\"aratud see, et k\~oik koonduksid.
    Kiirema koodnumise jaoks on kasutatud tagasilevi meetodi
    modifikatsiooni,
#mis on R-is vaike valikuks
set.seed(1)
model0<-neuralnet(formula=a+b~x+y, data=training.data1, hidden
    =0, linear.output = FALSE, err.fct="ce")
training<-c(training, correct(data=training.data1, model=model0)
    )
test<-c(test, correct(data=test.data1, model=model0))

set.seed(15)
model1<-neuralnet(formula=a+b~x+y, data=training.data1, hidden
    =1, linear.output = FALSE, err.fct="ce")
training<-c(training, correct(data=training.data1, model=model1)
    )
```

```
test<-c(test, correct(data=test.data1, model=model1))

set.seed(545)
model2 <- neuralnet(formula=a+b~x+y, data=training.data1, hidden
    =2, linear.output = FALSE, err.fct="ce")
training<-c(training, correct(data=training.data1, model=model2)
    )
test<-c(test, correct(data=test.data1, model=model2))

set.seed(54541)
model3 <- neuralnet(formula=a+b~x+y, data=training.data1, hidden
    =3, linear.output = FALSE, err.fct="ce")
training<-c(training, correct(data=training.data1, model=model3)
    )
test<-c(test, correct(data=test.data1, model=model3))

set.seed(5450)
model4 <- neuralnet(formula=a+b~x+y, data=training.data1, hidden
    =4, linear.output = FALSE, err.fct="ce")
training<-c(training, correct(data=training.data1, model=model4)
    )
test<-c(test, correct(data=test.data1, model=model4))


set.seed(72060)
model5 <- neuralnet(formula=a+b~x+y, data=training.data1, hidden
    =5, linear.output = FALSE, err.fct="ce")
training<-c(training, correct(data=training.data1, model=model5)
    )
test<-c(test, correct(data=test.data1, model=model5))

set.seed(4405)
model6 <- neuralnet(formula=a+b~x+y, data=training.data1, hidden
    =6, linear.output = FALSE, err.fct="ce")
training<-c(training, correct(data=training.data1, model=model6)
    )
test<-c(test, correct(data=test.data1, model=model6))

set.seed(982)
```

```
model7 <- neuralnet(formula=a+b~x+y, data=training.data1, hidden
    =7, linear.output = FALSE, err.fct="ce")
training<-c(training, correct(data=training.data1, model=model7)
    )
test<-c(test, correct(data=test.data1, model=model7))

set.seed(781)
model8 <- neuralnet(formula=a+b~x+y, data=training.data1, hidden
    =8, linear.output = FALSE, err.fct="ce")
training<-c(training, correct(data=training.data1, model=model8)
    )
test<-c(test, correct(data=test.data1, model=model8))

set.seed(10511)
model9 <- neuralnet(formula=a+b~x+y, data=training.data1, hidden
    =9, linear.output = FALSE, err.fct="ce")
training<-c(training, correct(data=training.data1, model=model9)
    )
test<-c(test, correct(data=test.data1, model=model9))

set.seed(5546)
model10 <- neuralnet(formula=a+b~x+y, data=training.data1,
    hidden=10, linear.output = FALSE, err.fct="ce")
(training<-c(training, correct(data=training.data1, model=
    model10)))
(test<-c(test, correct(data=test.data1, model=model10)))

x<-0:10
#Visualiseerin saadud tulemusi
plot(x,training, pch=16, type="b", ylim=c(0.85,0.92), xlab="
    Neuronite arv varjatud kihis",
      ylab="\~Oigesti klassifitseeritud objektide osakaal")
points(x,test,pch=16, col="red", type="b")


#Tekitan v\~orgustiku
net<-function(){
  netx<-seq(0,5,0.01)
  netx1<-c()
```

```
  nety1<-c()
  for(i in netx){
    temp<-rep(i,length(netx))
    netx1<-c(netx1,temp)
    nety1<-c(nety1,seq(5,0,-0.01))
  }
  return(data.frame(x=netx1, y=nety1))
}
net<-net()



#Kasutan mudel2 ja mudel10

#Pilt
pred2<-compute(model2,net)
output2<-matrix(pred2$net.result[,1], byrow=F,nrow=501)
zo.output2<-zeroone.data(output2)
#pheatmap(zo.output, cluster_rows = F,cluster_cols = F)


#Pilt
pred10<-compute(model10,net)
output10<-matrix(pred10$net.result[,1], byrow=F,nrow=501)
zo.output10<-zeroone.data(output10)
#pheatmap(zo.output10, cluster_rows = F,cluster_cols = F)

#Leian piirjoone
#Leian asukohad, kus v\"a\"artus null muutub \"uheks igas veerus

ext.line<-function(matrix){
  changex<-c()
  changey<-c()
  last<-0
  for(i in 1:dim(matrix)[2]){
    for(j in 1:dim(matrix)[1]){
      if (last != matrix[j,i]){
        changex<-c(changex,i)
        changey<-c(changey,j)
```

```
        last = matrix[j,i]
      }
    }
  }
  return(data.frame(x=changex,y=changey))
}
#Lisan andmestikku tunnuse, kas mudel ennustas v\"a\"artuse \~
    oigesti v\~oi mitte
cor.clas<-function(data, model){
  for(i in 1:dim(data)[1]){
    pred.1<-compute(model, data[i,-3:-5])
    pred.2<-zeroone.data(pred.1$net.result)
    if (pred.2[,1] == data[i,3]){
        data$pred[i] = 0 } else {
        data$pred[i] = 1
      }
  }
  return(data)
}

#Leian treenimise k\"aigus tekkinud piiri
dots2<-ext.line(zo.output2)
dots2<-dots2 %>%
  filter(y != 1) %>%
  mutate(y=(500-y)/100, x=x/100)

training.data2 <- cor.clas(training.data1, model2)

#Visualiseerin tulemust
par(mfrow=c(1,2))
plot(dots2$x,dots2$y, type="l",col="grey", lwd=2, ylim=c(0,5),
    xlab="x1", ylab="x2", main="(a)")
points(training.data2$x,training.data2$y, xlim=c(0,5), ylim=c
    (0,5), col = training.data$type, pch=16)

#Sama suurema andmestiku korral

training.data10<-cor.clas(training.data1,model10)
```

```
dots10<-ext.line(zo.output10)
dots10<-dots10 %>%
  filter(y != 1) %>%
  mutate(y=(500-y)/100, x=x/100)
plot(dots10$x,dots10$y, pch=".", cex=3, col="grey",xlab="x1",
    ylab="x2", main="(b)")
points(training.data10$x,training.data10$y, xlim=c(0,5), ylim=c
    (0,5), col = training.data$type, pch=16)


test.data2 <- cor.clas(test.data1, model2)

#Visualiseerin tulemust
par(mfrow=c(1,2))
plot(dots2$x,dots2$y, type="l",col="grey", lwd=2, ylim=c(0,5),
    xlab="x1", ylab="x2", main="(a)")
points(test.data2$x,test.data2$y, xlim=c(0,5), ylim=c(0,5), col
    = test.data2$pred+3, pch=16)

#Sama suurema andmestiku korral

test.data10<-cor.clas(test.data1,model10)

dots10<-ext.line(zo.output10)
dots10<-dots10 %>%
  filter(y != 1) %>%
  mutate(y=(500-y)/100, x=x/100)
plot(dots10$x,dots10$y, pch=".", cex=3, col="grey",xlab="x1",
    ylab="y2", main="(b)")
points(test.data10$x,test.data10$y, xlim=c(0,5), ylim=c(0,5),
    col = test.data10$pred+3, pch=16)

#Kood peat"ukile praktilised n\"apun\"aited
#Loen sisse kasutatavad paketid
par(mfrow=c(1,1))
library(neuralnet)
#Funktsiooni anal\"u\"utiline kuju
f=function(x)((2-sin(10*x))*exp(-0.5*x**2)*(1.5-sin(2*x))/sqrt(2
    *pi))
#Funktsiooni joonis
```

```
curve(f,-5,5, n=1000)

#Tekitan treenimiseks andmestiku
set.seed(154)
x<-runif(100,-1,1)
y<-f(x) #Uuritav tunnus y=f(x)
training.data<-data.frame(x,y)

#Treenin andmestiku, kus on varjatud kihis 5 neuronit
model<-neuralnet(y~x,training.data,hidden=5)

#Treenitud andmestiku visualiseerimiseks tekitan vektori x
x.seq <- seq(-2,2,0.01)

#Arvutan mudeliga ennustatavad v\"a\"artused
pred.model<-compute(model,x.seq)

#Kujutan joonisel
plot(x.seq,pred.model$net.result, type="l", xlab="x", ylab="h(x)
    ")
curve(f,-2,2,add=T,col=2,n=1000)

#Joonis, kus on kujutatud iga neuroni panust v\"aljundisse

#Sigmoidfunktsiooni valem
sigmoid<-function(x)1/(1+exp(-x))

x<-seq(-2,2,0.1)
#Leian v\"aljundkihti neuroni argumendi.
one.neuron <- function(x,nr)sigmoid(x*model$weights[[1]][[1]][2,
    nr]+model$weights[[1]][[1]][1, nr])*
                                        model$weights[[1]][[2]][nr
                                            +1,1]

argument<-function(x){
  argument = one.neuron(x,1)+one.neuron(x,2)+one.neuron(x,3)+one
      .neuron(x,4)+one.neuron(x,5)+model$weights[[1]][[2]][1,1]
  return(argument)
}
```

```
#Vabaliike kujutamiseks graafikul
bias <-function(x)rep(model$weights[[1]][[2]][1,1], length(x))
#Graafiku joonestamine
plot(0,0,type="n",xlim=c(-2.5,2.5),ylim=c(-11,8), xlab="x", ylab
    ="y")
lines(x.seq,one.neuron(x.seq,1), col=2, lwd=2)
lines(x.seq,one.neuron(x.seq,2), col=3, lwd=2)
lines(x.seq,one.neuron(x.seq,3), col=4, lwd=2)
lines(x.seq,one.neuron(x.seq,4), col=5, lwd=2)
lines(x.seq,one.neuron(x.seq,5), col=6, lwd=2)
lines(x.seq,bias(x.seq), lwd=2, col=8)
abline(0,0,lty=2)
abline(v=0, lty=2)
```

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Siim Viigand,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Otsesuunatud tehisnärvivõrgud ja nende treenimine", mille juhendajad on Riho Klement ja Taavi Unt,

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace'i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, 29.04.2016