UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 1. stopnja

Laura Guzelj Blatnik

Nevronske mreže z vzvratnim razširjanjem napak v funkcijskem programskem jeziku

Delo diplomskega seminarja

Mentor: prof. dr. Ljupčo Todorovski
Somentor: asist. dr. Aljaž Osojnik

Ljubljana, 2020

Kazalo

Nevronske mreže z vzvratnim razširjanjem napak v funkcijskem programskem jeziku

Povzetek

Angleški prevod slovenskega naslova dela

Abstract

Math. Subj. Class. (2010):
Ključne besede:
Keywords:

1. Uvod

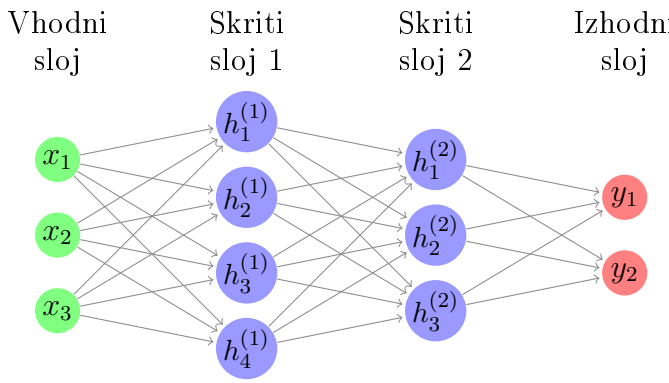
Človeški možgani so kompleksen organ predvsem zaradi neštetih funkcij, ki jih opravljajo. In zgolj vprašanje časa je bilo, kdaj bodo znanstveniki skoraj neskončne zmožnosti možganov prenesli v računalništvo. Ko so poskušali idejo uresničiti, so si predvsem želeli strukture, ki se bo - podobno kot možgani - sposobna učiti, odzivati na sprememe in prepoznati neznane situacije. Tako je 1949 Donal Hebb v svojem delu prvič predstavil idejo umetnih nevronskih mrežah, kjer se je zgledoval predvsem po možganih. Dvoslojni perceptorn se je rodil nekoliko kasneje, leta 1962 je ta pojem prvi omenil Rosenblatt. Z razvojem storjne opreme se je zanimanje za nevronske mreže zopet povečalo v osemdestih lezih prejšnjega stoletja. Leta 1986 je več razskovalcev neodvisno drug od drugega vpelje teorijo o večslojnih nevronskih mrežah in vzvratnem razširjanju napake. Kljub širokemu spektru uporabe, ki jih nudijo nevronske mreže je interes zanje kasneje nekoliko upadel. V zadnjih letih pa so nevronske mreže zopet vroča tema, predvsem zaradi njihove zmožnosti prepoznavanja slik. Do danes so umetne nevronske mreže močno napredovale, namesto običajnih mrež se vedno bolj uporabljajo globoke nevronske mreže, ki v nekaterih lastnostih celo prekašajo možgane. Kljub vsemu pa so možgani sposobni masrcičesa, česar računalniki ne bodo nikoli.

Diplomska naloga se poglobi v umetne nevronske mreže z vzvratnim razširjanjem napake. Primer take mreže sem tudi implementirala v funkcijskem programskem jeziku OCaml.

2. Umetne nevronske mreže

2.1. **Biološko ozadje.** Kot sugerira že samo ime, se umetne nevronske mreže v marsičem zgledujejo po človeških možganih. Osnovni gradniki možganov so nevroni, ki so med seboj povezani s sinapsami. Prav te goste povezave so ključne za delovanje živčevja. Skozi naše življenje se sinapse spreminjajo po jakosti in številu, ko se učimo se ustvarjajo nove povezave med nevroni, že obstoječe pa postajajo močnejše. Nevron se aktivira le, ko po sinapsi do njega pride signal s točno določeno frekvenco, ta impulz nato nevron posreduje sosednjim nevronom. Človeške možgane sestavlja 10^{10} nevronov, vsak ima približno 10^4 sinaps. Umetne nevronske mreže načeloma sestavlja veliko manj nevronov, posledično so zato pri izvajanju operacij veliko hitrejše od možganov. Kot bomo videli v nadaljevanju, je osnovna ideja umetnih nevronskih mrež zelo podobna živčevju v možganih. Lahko bi rekli, da so umetne nevronske mreže zgolj abstraktna poenosatvitev delovanja možganov.

2.2. **Zgradba nevronskih mrež.** Nevronske mreže so sestavljene iz nevronov, ki matematično gledano niso nič drugega kot funkcije. Nevroni so med seboj povezani, vsaka povezava ima svojo težo oziroma utež. Nevrone razdelimo v sloje, pomembno je poudariti, da nevroni v istem sloju med seboj niso povezani, vsak



SLIKA 1. Perceptron z dvema skritima slojema nevronov

nevron je povezan le z vsemi nevroni v sosednjem (oziroma prejšnjem sloju). Nevronska mreža lahko vsebuje le vhodni in izhodni sloj - takrat govorimo o dvoslojni nevronski mreži. Če se med tema slojema skriva še kakšen skriti sloj, potem mreža postane večslojna. Število nevronov v posameznem sloju tako kot tudi število skritih slojev nevronske mreže je odvisno od problema, ki ga nevronska mreža rešuje. Pri mrežah, ki jih bom obravnavala v diplomski nalogi, so vse povezave usmerjene, čeprav je pomembno omeniti, da obstajajo tudi mreže z dvosmernimi povezavami. Usmerjeni nevronski mreži rečemo perceptron, glede na število slojev pa ločimo dvoslojni in večslojni perceptron.

2.3. **Delovanje perceptrona.** Vzemimo dvoslojni perceptron z m nevroni v vhodnem sloju, njihova stanja označimo z $X_1, X_2, \ldots, X_m$ in n nevroni v izhodnem sloju, označimo jih $Y_1, Y_2, \ldots, Y_n$. Običajno sloj nevronov zapišemo v obliki vektorja:

$$X = (X_1, X_2, \ldots, X_m)^\intercal \quad \text{in} \quad Y = (Y_1, Y_2, \ldots, Y_n)^\intercal.$$

Uteži med nevroni predstavimo z matriko:

$$M = \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1m} \\ W_{21} & W_{22} & \cdots & W_{2m} \\ \cdots & & \cdots & \\ W_{n1} & W_{n2} & \cdots & W_{nm} \end{bmatrix}$$

kjer element matrike W_{ij} predstavlja utež med i-tim nevronom v vhodnem sloju in j-tim nevronom v izhodnem sloju.

2.3.1. *Funkcija kombinacije:* Stanje j-tega nevrona po uporabljeni funkciji aktivacije je sledeče:

$$A_j = \sum_i W_{ij} X_i + C_j,$$

kjer je W_{ij} utež med i-tim in j-tim nevronom, C_j pa konstanta aktivacije za j-ti nevron (njen pomen je pojasnjen v poglavju 2.3.3). Za cel sloj izhodnih nevronov

pa lahko funkcijo aktivacije zapišemo z linearno preslikavo:

$$A = WX,$$

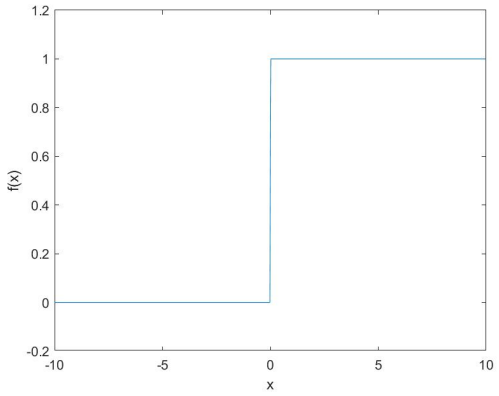
kjer je W matrika uteži, X pa vhodni vektor. Če enačbo zapišemo v vektorski obliki je potrebno upoštevati še konstante aktivacije, zato matriko uteži W in vektor X malo preoblikujemo. Vektorju na konec dodamo še eno komponento z vrednostjo 1 in dobimo $X = (X_1, X_2, \ldots, X_m, 1)^\intercal$ matriki uteži pa dodamo še stolpec s konstantami aktivacije za vsak nevron, torej:

$$M = \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1m} & C_1 \\ W_{21} & W_{22} & \cdots & W_{2m} & C_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ W_{n1} & W_{n2} & \cdots & W_{nm} & C_n \end{bmatrix}$$

2.3.2. *Funkcija aktivacije:* Cilj funkcije aktivacije je normalizirati vrednost, ki jo dobimo s funkcijo kombinacije. Glede na namen uporabe nevronske meže lahko uporabimo različne funkcije aktivacije. Najpogosteje se uporabljaljata stopničasta pragovna funkcija ali pa sigmoidna funkcija. *Stopničasta pragovna funkcija* nevron aktivira le, če je vrodnost dobljena z funkcijo aktivacije nad oziroma pod določeno mejo. Uporaba te funkcije je dokaj omejena. Ker vhodnim podatkom pripiše le dve različni vrednosti, jo lahko uporabimo le pri klasifikaciji v dve kategoriji oziroma za reševanje linearnih problemov. Prav tako jo lahko uporabimo samo pri dvoslojnih perceptronih. Kot bomo videli v nadaljevanju, za večslojne perceptrone ni primerna, saj je njen odvod enak 0, kar pa pomeni, da za učenje ne moremo uporabiti posplošenega pravila delta.

$$f(x) = \begin{cases} 0, & \text{če } x \leq 0 \\ 1, & \text{sicer} \end{cases}$$

Sigmoidne funkcije so nelinearne, zato se uporabljajo pri večslojnih perceptornih



Slika 2. Stopničasta pragovna funkcija aktivacije

in reševanju nelinearnih problemov. So zvezne in neskončnokrat zvezno odvedljive, tako jih lahko uporabimo pri posplošenem pravilu delta. Odvisno na kateri interval želimo normalizirati izhodne vrednosti, uporabimo različne sigmoidne funkcije. Primera sigmoidne funkcije sta:

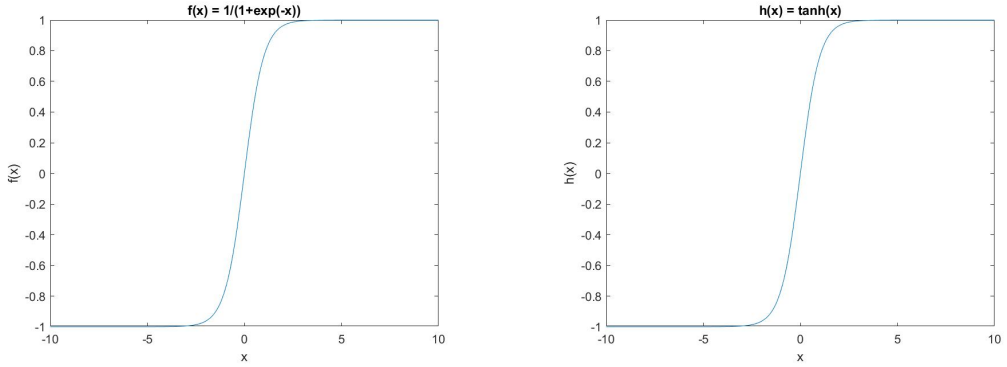
$$f(x) = \frac{1}{1+e^{-x}} \quad in \quad h(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{1}$$

Kot lahko vidimo z grafov (slika 3) je na intervalu $(-2, 2)$ gradient obeh funkcij zelo velik, zato da na omenjenem intervalu mreža pri napovedovanju jasne rezultate in učenje je hitro. Po drugi strani pa je zunaj tega intervala odvod majhen, tako je učenje mreže in posledično napovedovanje zelo počasno in slabo.

Za učenje mreže s posplošenim pravilom delta potrebujemo tudi odvod funkcije aktivacije, ki ga pri zgornjih dveh funkcijah lahko izrazimo z $f(x)$ oziroma $h(x)$ sledeče:

$$f'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = f(x)(1 - f(x)) \tag{2}$$

$$h'(x) = \left(\frac{\sinh(x)}{\cosh(x)}\right)' = \frac{\cosh^2(x) - \sinh^2(x)}{\cosh^2(x)} = 1 - \tanh^2(x) = 1 - h(x)^2 \tag{3}$$

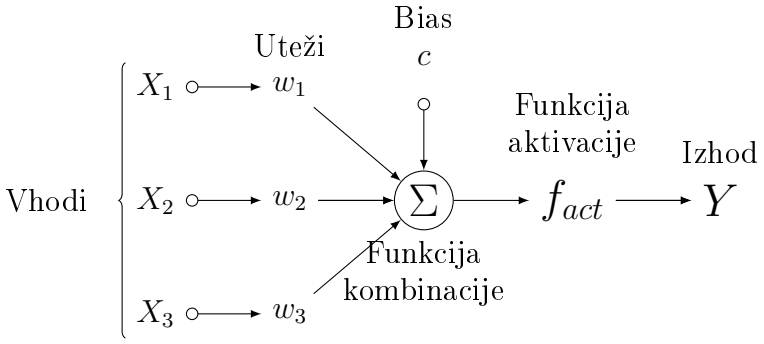


Slika 3. Sigmoidni funkciji aktivacije

Katero funkcijo aktivacije bomo izbarli je odvisno od problema, ki ga želimo z mrežo obravnavati.

2.3.3. *Konstanta aktivacije.* Sloju nevronov pogosto dodamo še en nevron - konstanto aktivacije. To je nevron z vrednostjo 1, njegova naloga pa je, da funkcijo aktivacije primerno premakne levo oziroma desno po grafu. Brez dodatnega nevrona bi mreža pri vhodnem podatku $X = 0$ vedno vračala vrednost 0, posledično bi bila aproksimacija dane funkcije slaba in rezultati netočni.

Pomen konstante aktivacije lahko predstavimo tudi s pomočjo linearne funkcije. Vse premice oblike $y = kx$ prečkajo koordinatno izhodišče in tako izgubimo kar



SLIKA 4. Funkcija nevrona

nekaj premic. Če pa fuknkciji dodamo še začetno vrednost, ki ustreza pomenu konstante aktivacije, potem lahko z enačno $y = kx+n$ zapišemo poljubno premico. Posledično je mreža bolj fleksibilna, problem bolje aproksimira in vrača bolj točne rezultate.

2.3.4. *Funkcija nevrona.* Izhodni sloj nevronov Y dobimo tako, da vhodni sloj pomnožimo z matriko uteži - temu pravimo funkcija kombinacije, nato pa na tem rezultatu uporabimo še funkcijo aktivacije.

Če združimo funkcijo kombinacije in funkcijo aktivacije, lahko napišemo pravilo za funkcijo nevrona, tj. izračun izhodnih vrednosti nevrona. Za posamezen nevron velja:

$$Y_j = f(\sum_i W_{ij}X_i + C_j)$$

oziroma v vektorski obliki za cel sloj izhodnih nevronov:

$$Y = f(WX),$$

kjer velja, da funkcijo aktivacije uporabimo na vsaki komponenti vektorja WX.

2.4. **Lastnosti nevronskih mrež.** Nevronske mreže odlikuje kar nekaj pozitivnih lastnosti. Najpomembnejše med njimi so sledeče:

Paralelizabilnost: Posamičen nevron deluje relativno neodvisno od ostalih nevronov. Paralelizabilnost posledično pomeni zelo hitre operacije in omogoča uporabo nevronskih mrež pri reševanju kompleksnih problemov.

Stabilnost: Izgubljen ali nedelujoč nevron le malo pokvari natančnost delovanja nevronske mreže (več kot je takih nevronov, manj natančna je nevronska mreža). Nevronske mreže so tako stabilne v smislu uničenih nevronov in njihovih povezav. Poleg tega pa so stabilne tudi v smislu nepopolnih podatkov. To pomeni, da mrežo lahko naučimo pravilnega delovanja, tudi če v učnih priemrih del podatkov manjka. Velja pa, da bolj nepopolni kot so podatki, slabša je aproksimacija nevronske mreže.

Matematično ozadje: Cilj nevronske mreže je aproksimirati poljubno funkcijo in kasneje na podlagi te aproksimacije napovedati vrednosti pri poljubnih vhodnih podatkih. Pri računanju se nevronske mreže močno opirajo na linearno algebro, linearnime preslikave, lastne vrednosti in vektorje ter negibne točke.

Glavna pomanjkljivost nevronskih mrež je določanje topologije mreže. Medtem ko je število nevronov v vhodnem in izhodnem sloju določeno s problemom, je število nevronov v posameznem skritem sloju in število skritih slojev odvisno od posameznega primera. Idealno topologijo nevronske mreže je tako težko doseči, do nje lahko pridemo le s empirično s poskušanjem. Še ena slaba lastnost pa je nezmožnost razlaganja odločitev (ang. balckbox). Zaradi kompleksne strukture je vsak rezultat mreže produkt večih neodvisnih operacij nevronov, ki jih ne moremo razložiti.

2.5. **Uporaba nevronskih mrež.** Z razvojem strojne opreme v zadnjih letih so močno napredovale tudi nevronske mreže. Razvite tudi hitrjše implementacije algoritmov, posledično je zaradi njihove efektivnosti uporaba nevronskih mrež danes zelo pogosta v industriji. Ko nevronsko mrežo z učnimi primeri naučimo pravilnega delovanja, se mreža zna odzvati na nepoznane situacije. Tako lahko, zaradi njihove splošnosti in sposobnosti sprejetja velikega števila vhodnih podatkov, nevronske mreže uporabimo za reševanje najrazličnejših probleme. Zato se dandanes uporabljajo na skoraj vseh mogočih področjih. Dva najpomembenjša namena uporabe sta napovedovanje in klasifikacija. Primeri uporabe v industriji so:

- prepoznavanje in klasifikacija slik: konkretno podjetje Google uporablja nevronske mreže za prepoznavanje slik, ki jih nato opremi s ključnimi besedami,
- prepoznavanje ročno napisanih besedil,
- prepoznavanje govora, tu bi veljalo izpostaviti podjete Miscrosoft, ki je razvilo nevronsko mrežo, ki govorjeno angleščino prevaja v kitajščino,
- kontrola kvalitete v produkciji,
- v zdravstvu se uporabljajo za določanje pacientove diagnoze,
- napovedovanje različnih vrednosti glede na trenutne trende.

3. Učenje

Najprej je smiselno definirati, kaj pojem učenje sploh pomeni. Po [1, str. 37] za učenje potrebujemo sistem, ki strmi k izpolnitvi določene naloge oziroma cilja. Pred učenjem sistem ni sposoben zadostno opraviti naloge. S ponavljanjem določenih opravil poskušamo sistem pripraviti do tega, da bo deloval bolje, kjer je bolje lahko hitreje, ceneje, bolj pravilno...Učenje lahko torej definiramo kot zaporedje ponovitev, kjer pri vsaki ponovitvi skušamo zmanjšati napako tako, da bi se zastavljenemu cilju čimbolj približali.

Obstaja kar nekaj različnih pravil, kako umetno nevronsko mrežo naučiti pravilnega delovanja. V svoji diplomski nalogi sem se osredotočila na posplošeno

pravilo delta oziroma vzvratno razširjanje napake, ki je nekako standaren algoritem za učenje pri večslojnih perceptronih.

3.1. **Nadzorovano učenje.** Vzvratno razširjanje napake je primer nadzorovanega učenja. Za tako učenje potrebujemo učne primere - torej vhodne podatke in vredosti, ki želimo, da jih mreža pri danih vhodnih podatkih vrača. Cilj učenja je najti funkcijo, ki bo najbolje aproksimirala relacijo med vhodnimi in izhodnimi podatki. Pri vzvratnem razširjanju napake do take funkcije pridemo tako, da spreminjamo vrednosti na utežeh, pri čemer poskušamo minimalizirati napako med želenim in dobljenim izhodom mreže.

Potrebno je omeniti še, da poleg nadzorovanega učenja poznamo tudi nenazdrovano učenje. Tu izhodnih podatkov nimamo. Mreža vhodne podatke obdela po svojih kriterijih in poskuša najti strukturo, ki povezuje vhodne podatke med seboj. Nenazdorovano učenje se praviloma ne uporablja pri nevronskih mrežah.

3.2. **Pravilo delta.** Pravilo delta je eno izmed pravil, ki ga lahko uporabimo pri učenju nevronske mreže, ponavadi se uporablja na dvoslojnih perceptronih.Če je mreža večslojna, potem govorimo o vzvratnem razširjanju napake oziroma posplošenem pravilu delta. Za pravilom delta stoji povsem preprosta ideja. Najprej uteži med nevroni poljubno nastavimo(pomembno je le, da niso vse uteži nastavljene na 0), nato na izbranem testnem primeru izračunamo za koliko se je naša mreža zmotila glede na pričakovan izhod. Tako dobimo napako s pomočjo katere lahko vrednosti na utežeh popravimo tako, da minimaliziramo dobljeno napako. Postopek ponavljamo na ostalih učnih primerih, dokler se uteži ne ustalijo (včasih lahko isti učni primer uporabimo tudi večkrat). Takrat se učenje konča.

Za začetek vzemimo preprost dvoslojni perceptron z enim izhodnim nevronom Y, število vhodnih nevronov je poljubno. Pravilo delta pri spreminjanju vrednosti uteži upošteva razliko med želenim in dobljenim izhodom mreže. Tako definiramo napako mreže kot kvadrat razlike omenjenih izhodov:

$$\begin{aligned} E(l) &= (d(l) - Y(l))^2 \\ &= (d(l) - WX(l))^2 \end{aligned} \tag{4}$$

kjer je $d(l)$ želen izhod in $Y(l) = WX(l)$ z mrežo izračunana vrednost pri l-tem učnem primeru, vhodnem vektorju $X(l)$ in matriki uteži W.

Pravilo delta temelji na gradientnem iskanju, vektor uteži popravljamo v negativni smeri odvoda napake po utežeh. Izračunamo še odvod napake po utežeh:

$$\frac{dE(l)}{dW} = -2(d(l) - WX(l))X(l)$$

Sedaj lahko zapišemo enačbo za pravilo delta in nove vrednosti uteži W^N:

$$\begin{aligned} W^N &= W - \eta \frac{dE(l)}{dW} \\ &= W + \eta(d(l) - WX(l))X(l) \end{aligned} \tag{5}$$

V zgornji enačbi η predstavlja hitrost učenja in določa kako hitro bodo uteži konvergirale k optimalnim.

3.2.1. *Gradientno iskanje.* Za pravilom delta stoji gradientno iskanje. To je v bistvu iskanje lokalnega minimuma. Začnemo na naključni točki na grafu (zato utežem na začetku nastavimo naključne vrednosti) in se nato v vsakem koraku premaknemo po krivulji v smeri negativnega odvoda, to je smer, kjer funkcija najhitreje pada. Dolžino vsakega premika pri učenju nevronskih mrež določa stopnja učenja η, zato je pomembno kakšno vrednost zanjo izberemo.

3.3. **Vzvratno razširjanje napake.** Vzvratno razširjanje napake temelji na pravilu delta. Problem se pojavi, ker uteži med skritimi sloji niso direktno odvisne od napake med izhodnim stanjem in želeno vrednostjo, zato moramo pravilo malo prilagoditi. Ko popravljamo uteži, to počnemo od izhodnega sloja nevronov proti vhodnemu (od tod tudi ime - vzvratno razširjanje napake oziroma *backpropragation* v anglrščini).

3.3.1. *Notacija.* Definirajmo notacijo, ki jo bomo uporabili pri izpeljavi vzvratnega razširjavnja napake.

- X_i označuje i-ti nevron v vhodnem sloju.
- H_{ki} označuje i-ti nevron v k-ti skriti plasti.
- Y_i označuje i-ti nevron v izhodni plasti
- $W_{ij}^{(k)}$ označuje utež med i-tim nevronom v $k-1$ plasti in j-tim nevronom v k-ti plasti
- A_{ki} označuje stanje i-tega nevrona v k-ti skriti plasti po uporabljeni funkciji kombinacije: $A_{kj} = \sum_{j=1}^{N_{k-1}} W_{ij}^{(k)} X_j$, poseben primer je stanje i-tega nevrona v izhodni plasti, ki ga označimo z A_i

3.3.2. *Izpeljava vzvratnega razširjanja napake.* Vzemimo splošen večsojen perceptron z N_X nevroni v vhodnem sloju in N_Y nevroni v izhodnem sloju. Nevronska mreža naj sestoji iz m skritih slojev, $m > 0$, vsak skriti sloj pa naj vsabuje N_k nevronov, kjer velja $1 \leq k \leq m$.

Preden se lotimo popravljanja uteži, moramo za dani l-ti učni primer izračunamo izhodne vrednosti. To storimo tako, da na vsakem nevronu uporabimo funkcijo aktivacije in kobinacije. Za i-ti nevron v prvem skritem sloju lahko zapišemo:

$$H_{1i}(l) = f\left(\sum_{j=1}^{N_X} W_{ij}^{(1)} X_j(l)\right)$$

in

$$A1i(l) = \sum_{j=1}^{N_X} W_{ij}^{(1)} X_j(l)$$

Za poljubni k-ti sktiti sloj, kjer je $1 \leq k \leq m$,tako zapišemo:

$$H_{ki}(l) = f(A_{ki}(l)),$$

kjer je

$$A_{ki}(l) = \sum_{j=1}^{N_{k-1}} W_{ij}^{(k)} H_{k-1,j}(l).$$

V zgornji enačbi $H_{k-1,j}$ označuje že izračunane vrednosti nevronov v predhodnem sloju.

V izhodnem sloju dobimo vrednosti, ki nas najbolj zanimajo. Velja:

$$Y_i(l) = f(A_i(l)) \quad \text{in} \quad A_i(l) = \sum_{j=1}^{N_m} W_{ij}^{(m+1)} H_{m,j}(l).$$

Tako smo dobili izhodne vrednosti, ki jih je izračunala naša mreža. Vrednosti primerjamo s tistimi, ki bi si jih želeli dobiti za dani l-ti učni primer. Za posamezen nevron ozančimo sledeče:

$$e_i(l) = d_i(l) - Y_i(l), \tag{6}$$

lahko pa enačbo zapišemo tudi v vektoski obliki:

$$e(l) = d(l) - Y(l).$$

Podobno kot pri pravilu delta, napaki celotne nevronske mreže za l-ti učni primer definiramo:

$$E(l) = \frac{1}{2} \sum_{i=1}^{N_Y} e_i^2(l) \tag{7}$$

Napako želimo minimalizirati in podobno kor prej s pomočjo gradientnega iskanja uteži popraviti v smeri negativnega odvoda napake E. Nove uteži izračunamo s pomočjo enake formule kot prej (enačba (5)):

$$W^N = W - \eta \frac{dE}{dW},$$

kjer η označuje stopnjo učenja.

Pri izračunu zgornjega odvoda si bomo pomagali s verižnim pravilom za odvajanje. Za izhodni sloj je napako enostavo izračunati. Problem se pojavi pri skritih slojih, saj uteži niso direktno odvisne od napake. Kot bomo videli v nadaljevanju, si bomo tudi v tem primeru pomgali z verižnim pravilom.

Najprej razpišimo formulo (7), kjer upoštevamo formuli za e_i in Y_i:

$$\begin{aligned} E(l) &= \frac{1}{2}\sum_{i=1}^{N_Y} e_i^2(l) \\ &= \frac{1}{2}\sum_{i=1}^{N_Y} (d_i(l) - Y_i(l))^2(l) \\ &= \frac{1}{2}\sum_{i=1}^{N_Y} (d_i(l) - f(\sum_{j=1}^{N_m} W_{ij}^{(m+1)} H_{m,j}(l)))^2 (l) \end{aligned}$$

Sedaj za izračun odvoda uporabimo verižno pravilo, formulo razpišemo na posamezno utež:

$$\frac{\partial E(l)}{\partial W_{ij}^{(m+1)}} = \frac{\partial E(l)}{\partial e_i(l)} \frac{\partial e_i(l)}{\partial Y_i(l)} \frac{\partial Y_i(l)}{\partial A_i(l)} \frac{\partial A_i(l)}{\partial W_{ij}^{(m+1)}} \tag{8}$$

Zgornje parcialne odvode pa lahko ob pomoči definicij posameznih izrazov zapišemo sledeče:

$$\frac{\partial E(l)}{\partial e_i(l)} = \frac{\partial(\frac{1}{2}\sum_{i=1}^{N_Y} e_i^2(l))}{\partial e_i(l)} = e_i(l) \tag{9}$$

$$\frac{\partial e_i(l)}{\partial Y_i(l)} = \frac{\partial(d_i(l) - Y_i(l))}{\partial Y_i(l)} = -1 \tag{10}$$

$$\frac{\partial Y_i(l)}{\partial A_i(l)} = \frac{\partial(f(A_i(l)))}{\partial A_i(l)} = f'(A_i(l)) \tag{11}$$

$$\frac{\partial A_i(l)}{\partial W_{ij}^{(m+1)}} = \frac{\partial \sum_{j=1}^{N_m} W_{ij}^{(m+1)} H_{m,j}(l)}{\partial W_{ij}^{(m+1)}} = H_{mj}(l) \tag{12}$$

Pri tem velja, da je f' odvod funkcije aktivacije in je odvisen od tega, katero funkcijo aktivacije uporabimo.

Zaradi boljše preglednosti parcialni odvod zapišemo še malo drugače:

$$\frac{\partial E(l)}{\partial W_{ij}^{(m+1)}} = \frac{\partial E(l)}{\partial A_i(l)} \frac{\partial A_i(l)}{\partial W_{ij}^{(m+1)}}$$

Če upoštevamo zgoraj izračunani odvod dobimo:

$$\frac{\partial E(l)}{\partial W_{ij}^{(m+1)}} = \frac{\partial E(l)}{\partial A_i(l)} H_{mj}(l)$$

Označimo:

$$\frac{\partial E(l)}{\partial A_i(l)} = \Delta_i^{(m+1)}(l) \tag{13}$$

Združimo vse zgornje rezultate in zapišemo pravilo za izračun novih uteži pri izhodnem sloju:

$$
\begin{aligned}
W_{ij}^{N(m+1)} &= W_{ij}^{(m+1)} - \eta \frac{dE(l)}{dW_{ij}^{(m+1)}} \\
&= W_{ij}^{(m+1)} - \eta \Delta_i^{(m+1)}(l) H_{mj}(l) \\
&= W_{ij}^{(m+1)} - \eta e_i(l) f'(A_i(l)) H_{mj}(l)
\end{aligned} \tag{14}
$$

Tako smo izpeljali pravilo za spremembno uteži, ki povezujejo izhodni sloj s predhodnikom.

Vzvratno razširjanje napake pride do izraza šele pri skritih slohjih. Ker ne moremo direktno izračunati napake e_i, si pri računanju pomagamo z že spremenjenimi utežmi v izhodnem sloju in pa $\Delta^{(m+1)}$, ki smo ga spotoma izračunali. Za uteži, ki povezujejo predzadnji in zadnji skriti sloj lahko ob upoštevanju verižnega pravila zapišemo:

$$\frac{\partial E(l)}{\partial W_{ij}^{(m)}} = \frac{\partial E(l)}{\partial A_{mi}(l)} \frac{\partial A_{mi}(l)}{\partial W_{ij}^{(m)}}$$

Drugi člen odvoda enostavno izračunamo

$$\frac{\partial A_{mi}(l)}{\partial W_{ij}^{(m)}} = H_{m-1,j}(l)$$

in podobno kot prej označimo

$$\frac{\partial E(l)}{\partial A_{mi}(l)} = \Delta_i^{(m)}(l)$$

Zopet uporabimo verižno pravilo za odvajanje in zapišemo:

$$\Delta_i^{(m)}(l) = \frac{\partial E(l)}{\partial A_{mi}(l)} = \frac{\partial E(l)}{\partial H_{mi}(l)} \frac{\partial H_{mi}(l)}{\partial A_{mi}(l)}$$

Odvod ${}^{\partial H_{mi}(l)}/_{\partial A_{mi}(l)}$ je odvod funkcije aktivacije:

$$\frac{\partial H_{mi}(l)}{\partial A_{mi}(l)} = f'(A_{mi}(l)),$$

$\partial E(l)/\partial H_{mi}(l)$ pa zopet razpišemo s pomočjo verižnega pravila

$$\begin{aligned}\frac{\partial E(l)}{\partial H_{mi}(l)} &= \sum_{j=1}^{N_Y} \frac{\partial E(l)}{\partial A_j(l)} \frac{\partial A_j(l)}{\partial H_{mi}(l)} \\ &= \sum_{j=1}^{N_Y} \frac{\partial E(l)}{\partial A_j(l)} \frac{\partial (\sum_{p=1}^{N_m} W_{pj}^{N(m+1)} H_{mp}(l))}{\partial H_{mi}(l)} \\ &= \sum_{j=1}^{N_Y} \frac{\partial E(l)}{\partial A_j(l)} W_{ji}^{N(m+1)}\end{aligned}$$

Pri tem velja, da so elementi matrike uteži $W^{N(m+1)}$ vrednosti, ki smo jih že popravili v prejšnjem koraku. Uporabimo enačbo (13) in izrazimo vrednost $\Delta_i^{(m)}(l)$ sledeče:

$$\begin{aligned}\Delta_i^{(m)}(l) &= f'(A_{mi}(l)) \sum_{j=1}^{N_Y} \frac{\partial E(l)}{\partial A_j(l)} W_{ji}^{N(m+1)} \\ &= f'(A_{mi}(l)) \sum_{j=1}^{N_Y} \Delta_j^{(m+1)}(l) W_{ji}^{N(m+1)}\end{aligned}$$

Vse združimo in vstavimo v enačbo za izračun spremenjenih uteži $W^{N(m)}$ ter dobimo:

$$\begin{aligned}W_{ij}^{N(m)} &= W_{ij}^{(m)} - \eta \frac{dE(l)}{dW_{ij}^{(m)}} \\ &= W_{ij}^{(m)} - \eta \Delta_i^{(m)}(l) H_{m-1,j}(l) \\ &= W_{ij}^{(m)} - \eta f'(A_{mi}(l)) \left(\sum_{p=1}^{N_Y} \Delta_p^{(m+1)}(l) W_{pi}^{N(m+1)} \right) H_{m-1,j}(l)\end{aligned} \tag{15}$$

Za uteži $W^{(k)}$, ki povezujejo poljuben $k-1$ in k-ti skriti sloj, kjer je $1 < k < m$, podobno zapišemo:

$$\frac{\partial A_{ki}(l)}{\partial W_{ij}^{(k)}} = H_{k-1,j}(l)$$

in

$$\Delta_i^{(k)}(l) = f'(A_{ki}(l)) \sum_{j=1}^{N_{k+1}} \Delta_j^{(k+1)}(l) W_{ji}^{N(k+1)}$$

Spremenjene uteži nato izračunamo sledeče:

$$
\begin{aligned}
W_{ij}^{N(k)} &= W_{ij}^{(k)} - \eta \Delta_i^{(k)}(l) H_{k-1,j}(l) \\
&= W_{ij}^{(k)} - \eta f'(A_{ki}(l)) \left(\sum_{p=1}^{N_{k+1}} \Delta_p^{(k+1)}(l) W_{pi}^{N(k+1)} \right) H_{k-1,j}(l)
\end{aligned} \tag{16}
$$

Ostane nam le še matrika uteži $W^{(1)}$, ki povezuje vhodni sloj s prvim skritim slojem. Zapišemo:

$$\frac{\partial A_{1i}(l)}{\partial W_{ij}^{(1)}} = X_j(l)$$

in

$$\Delta_i^{(1)}(l) = f'(A_{1i}(l)) \sum_{j=1}^{N_2} \Delta_j^{(2)}(l) W_{ji}^{N(2)}$$

Uteži pa popravimo po sledečo formuli:

$$
\begin{aligned}
W_{ij}^{N(1)} &= W_{ij}^{(1)} - \eta \Delta_i^{(1)}(l) X_j(l) \\
&= W_{ij}^{(1)} - \eta f'(A_{1i}(l)) \left(\sum_{p=1}^{N_2} \Delta_p^{(2)}(l) W_{pi}^{N(2)} \right) X_j(l)
\end{aligned} \tag{17}
$$

Tako smo izpeljali pravilo za računanje spremembe uteži pri vzvratnem razširjanju napake.

3.4. **Lastnosti vzvratnega razširjanja napake.** Kljub temu, da je vzvratno razširjanje napake zelo priljubljeno pravilo pri učenju nevronskih mrež, ima tudi nekaj pomanjkljivosti.

Gradientno iskanje temelji na iskanju lokalnega minimuma, posledično uteži ne konvergirajo vedno k globalno najboljši izbiri. Lahko se zgodi, da bo matrika uteži konvergirala k lokalnemu optimumu, taka mreža ne bo dajala zadovoljivih rezultatov. Kam bodo uteži konvergirale je odvisno predvsem od njihove začetne izbire. Če med učenjem pride do omenjenega problema moramo izbrati novo naključno matriko uteži in učenje ponoviti v upanju, da bodo uteži tokrat konvergirale k optimalnim.

Naslednji problem, ki se pojavi kot posledica gradientnega iskanja je določanje parametra η. Stopnja učenja določa kako hitro in kam bo mreža konvergirala. Žal pa ni pravila, ki bi določal kakšno vrednost je za η najbolje izbrati. Do take vrednosti lahko pridemo le s poskušanjem.

Slabost vzvratnega razširjanja napake je tudi to, da je za učenje nevronske mreže potrebno zelo veliko število učnih primerov. Ponavadi se to število giblje med nekaj 10000 in 100000. Posledično je učenje lahko zelo zamudno.

Potrebno je omeniti še časovno zahtevnost, zaradi svoje kompleksne zgradbe so nevronske mreže zelo časovno potratne in njihovo učenje praviloma ni hitro.

Z večjo mrežo le to zahtevnost še povečamo. Kljub temu, pa lahko z različnimi implementacijami malo vplivamo na hitrost mreže.

3.5. **Napaka in varianca.** Po končanem učenju lahko preverimo napako in varianco nevronske mreže. Ta podatka povesta, kako uspešni smo bili pri aproksimaciji funkcije. Ker gre za aproksimacijo, bo napaka vedno prisotna, želimo pa, da bi bila le ta čim manjša. Omenjeni količini izračunamo na poljubnem testnem primeru.

S t označimo točno vrednost, ki bi jo želeli dobiti, s $\hat{t}$ pa vrednost, ki jo mreža vrne. Z $E[\hat{t}]$ označimo pričakovano napoved, to je povprečje vrednosti, ki jih dobimo, če mrežo poženemo na več testnih primerih. Napako mreže pri t-tem učnem primeru označimo z $N(t)$ in izračunamo sledeče:

$$N(t) = E[\hat{t}] - t \tag{18}$$

Ptoblem se pojavi, če točne vrednosti t ne poznamo. Včasih lahko iz ostalih podatkov sklepamo kakšna bi ta vrednost morala biti ali pa jo poskusimo oceniti na kakšen drugačen način. Nadalje definiramo še varianco $Var(t)$, ki meri občutljivost mreže na dan učni nabor podatkov:

$$Var(\hat{t}) = E[E[\hat{t}] - \hat{t}]^2 \tag{19}$$

$$\begin{aligned} E[(\hat{t} - t)^2] &= E[(\hat{t} - E[\hat{t}] + E[\hat{t}] - t)^2] \\ &= (E[\hat{t}] - t)^2 + E[E[\hat{t}] - \hat{t}]^2 \\ &= N(t)^2 + Var(\hat{t}) \end{aligned} \tag{20}$$

Včasih se lahko zgodi, da napaka med učenjem začne rasti. Pomembno je, da učenje takrat prekinemo, saj lahko sicer pride do preprileganja (ang. overfitting). Preprileganje pomeni, da mreža zelo dobro deluje na učnih primerih (napaka je tu majhna), aproksimacija za poljuben primer pa ni natančna.

V nasprotju z preprileganjem p

3.6. **Normalizacija podatkov.** Vhodne in izhodne podatke je pred učenjem potrebno normalizirati. Obstaja več pravil za normalizacijo, najbolj pogosta sta min-max normalizacija in z-score normalizacija.

3.6.1. *Min-max normalizacija.*

3.7. **Z-score normalizacija.**

4. Funkcijski programski jezik OCaml

OCaml spada med funkcijske programse jezike. Poleg funkcijskih poznamo še objektne in proceduralne programske jezike. Vsi načini programiranja imajo svoje prednosti in slabosti. Ponavadi se za programski jezik odločimo glede problem, ki ga želimo rešiti.

Sama sem se za OCaml odločila, ker so implementacije nevronskih mrež v funkcijskih programskih jezikih zelo redke in mi je to predstavljalo izziv.

4.0.1. *Lastnosti funkcijskih programskih jezikov.* V funkcijskem programiranju so osnovni objekti funkcije. To pomeni, da lahko funkcije sprejmejo ozrioma vračajo druge funkcije, poleg tega lahko funkcije shranimo v podatkovne strukture. Funkcijam, ki kot argument sprejmejo funkcije pravimo *funkcije višjega reda.* Ena izmed osnovnih funkcij višjega reda je na primer funkcija map. Le ta sprejme podatkovno strukturo, naprimer seznam ali tabelo, in na vsakem njenem elementu uporabi funkcijo, ki smo jo podali kot argument.

4.1. **Lastnosti OCamla.** - je hiter - kodo lahko pišemo tudi proceduralno ali objektno, ni strikten funkcijskinjezik -tipi - varen (ni napak) - nespremenljivi tipi, to ni vedno ok -> rešitev so reference

4.2. **Tipi in moduli.**

5. Implementacija nevronske mreže

Del diplomske naloge je bila tudi implementacija nevronske mreže v jeziku OCaml. Cilj je bil implementirati generičen večslojni perceptron, kot učno pravilo sem uporabila vzvratno razširjanje napake. Zaradi generičnosti, lahko tako nevronsko mrežo prilagodimo in uporabimo na poljubnem primeru. Lahko tudi testiramo, kakšna mora biti topologija mreže in pa stopnja učenja, da bo mreža na danem primeru delovala optimalno. Celotna koda se nahaja na koncu naloge, v dodatku.

5.1. **Primer nevronske mreže.** Da ni ostalo le pri teoriji, sem mrežo testirala tudi na dejnaskem primeru. Za le to sem uporabila podatke o izposoji koles, pridobljwnw iz spletne strani

5.2. **Ugotovitve.**

6. Priloga

Slovar strokovnih izrazov

Literatura

[1] I. Kononenko in M. Kukar, *Machine Learning and Data Mining: Introduction to Principles and Algorithms*, Horwood Publishing Limited, UK, 2007

[2]