Uvod u neuronske mreže

Židov, Ivan

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike

Ivan Židov

Uvod u neuronske mreže

Završni rad

Osijek, 2018.

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike

Ivan Židov

Uvod u neuronske mreže

Završni rad

Mentor: izv. prof. dr. sc. Domagoj Matijević

Osijek, 2018.

Sažetak

U ovom završnom radu ukratko ćemo se upoznati s neuronskim mrežama. U uvodnom dijelu navedeni su primjeri u kojima su neuronske mreže već nadmašile čovjeka. Glavni dio rada bavi se matematikom skrivenom iza njih koja je prikazana i na primjeru.

Ključne riječi

neuronske mreže, aktivacijske funkcije, funkcije troška, metoda gradijenata, algoritam propagacije pogreške unatrag

Abstract

In this paper we will be introduced to the neural networks. In first part of the paper we will show examples how the neural networks have already outperformed humans. The main part of the paper covers necessary mathematics which is also shown on the example.

Key words

neural networks, activation functions, cost functions, gradient descent, backpropagation

Sadržaj

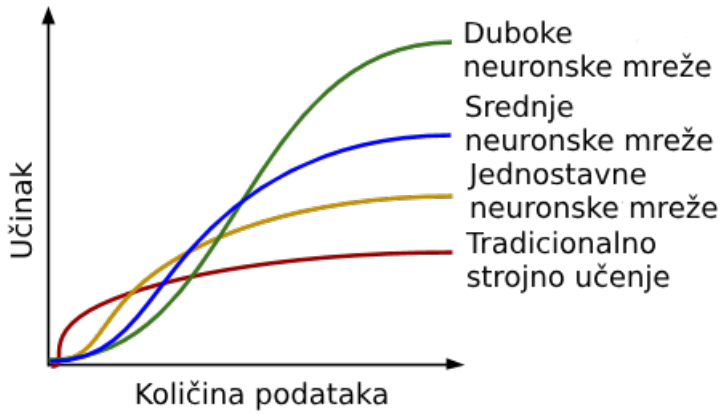
1 Uvod

Od samog početka razvoja računala, ljude je zanimalo hoće li ikad postojati računalo koje će moći razmišljati, odnosno hoće li ikad postojati umjetna inteligencija (eng. artificial intelligence - AI). Danas se pojam neuronskih mreža jako usko veže uz pojam umjetne inteligencije. Razlog tome je što su neuronske mreže već značajno promijenile internetske poslove kao što su web pretraživanja, online reklame i kreiranje preporuka.

To je tek početak jer se neuronske mreže počinju primjenjivati i u medicini, prepoznavanju govora, prevođenju tekstova, prepoznavanju objekata, vožnji autonomnih vozila, procjeni rizika u bankarstvu i druge. Činjenica da je algoritam uspio nadmašiti radiologe s dugogodišnjim iskustvom u prepoznavanju upale pluća na temelju rengenskog snimka, jedan je od pokazatelja u kakvom smjeru kreće napredak i da je to stvarno budućnost. Još jedan pokazatelj je i AlphaGo, koji je pobijedio prvaka u igri Go. Mislilo se da je nemoguće da računalo pobijedi čovjeka, pošto u igri Go postoji 10^{172} mogućih pozicija.

Iako nam se čini da je tehnologija autonomnih vozila još daleko, tvrtke poput Tesle, Googlea i Ubera već su izradile autonomne aute koji dostižu razinu znatno veću od razine čovjeka. Zanimljiv je podatak da je Googleov autonomni auto na prijeđenih više od 3 milijuna kilometara uzrokovao samo jednu prometnu nesreću, što ga čini 10 puta sigurnijim od ljudskih najsigurnijih vozača i 40 puta sigurnijim od ljudskih vozača s malo iskustva. Također je potvrđeno da će od srpnja 2018-te godine, u gradu Frisco u Texasu, autonomna vozila biti dostupna za javno korištenje.

Iako su koncept umjetnih neuronskih mreža Warren McCulloch i Walter Pitts predstavili 1943. godine, postavlja se pitanje zašto su se one popularizirale tek prije nekoliko godina. Razlog tome je što su bile računalno jako skupe. Računala u to vrijeme nisu bila dorasla takvom zadatku. Pojavom velikih obrađenih baza podataka i snažnijih računala, performanse dubokih neuronskih mreža su dobile svoj zamah. Na slici 1 prikazan je odnos učinka rada neuronskih mreža u odnosu na količinu podataka.



Slika 1: Odnos količine podataka i učinka

2 Osnove strojnog učenja

Duboko učenje (eng. Deep Learning) je posebna vrsta stojnog učenja (eng. Machine Learning). Da bi razumjeli duboko učenje, moramo prvo razumjeti osnovne koncepte strojnog učenja. Strojno učenje je algoritam koji može učiti iz podataka. Većina algoritama može se podijeliti u 2 vrste: nadzirano učenje i učenje bez nadzora.

Nadzirano učenje sadrži bazu podataka, koja ima mnogo podataka koji opisuju svaki od primjera i svaki primjer ima neki opis. Taj opis obično vrši čovjek. Učenje bez nadzora prima bazu podataka koja je nestrukturirana. Program mora sam shvatiti kako naučiti svojstva o strukturi podataka te samostalno urediti bazu. Nadzirano učenje je puno lakše i postoje jako dobri algoritmi, no ponekad je skupo ili teško skupljati podatke za svaki primjer. S druge strane, kreiranje baze za učenje bez nadzora je relativno jeftino, ali je programski puno kompleksnije i rjeđe se viđa u praksi.

Puno problema može biti riješeno pomoću strojnog učenja. Najčešći problemi su:

1. **Klasifikacija** - za određen input traži se kojoj od k kategorija pripada, odnosno tražimo funkciju $f : \mathbb{R}^n \to \{1, ..., k\}$. Npr. na temelju slike ručno napisanog broja traži se kojoj kategoriji iz 0,1,...9 pripada.

2. **Regresija** - na temelju inputa traži se neki realan broj, odnosno traži se funkcija $f : \mathbb{R}^n \to \mathbb{R}$ koja će za dani input vratiti realan broj. Npr. na temelju podataka o kvadraturi kuće, broju soba i broju kupaonica, traži se procjena vrijednosti kuće.

3. **Transkripcija** - računalo prima nestrukturiran input i mora od njega napraviti strukturiran tekst. Na primjer, na temelju zvučnog zapisa govora generiraju se titlovi.

4. **Prevođenje** - računalo prima tekst iz jednog jezika i prevodi ga u drugi.

5. **Prepoznavanje anomalija** - na temelju dosadašnjeg uzorka ponašanja, procjenjuje se je li nova akcija neuobičajena. Na primjer, na temelju dosadašnjih platnih navika, procjenjuje se je li nova transakcija uobičajena ili se možda radi o prijevara.

6. **Generiranje podataka** - od algoritma se traži da na temelju nekoliko primjera generiraju nove slične primjere. Koristi se u izradi igara, gdje se na temelju nekoliko slika generira stil umjetnika, jer je crtanje rukom dugotrajno.

7. **Nadopunjavanje podataka koji nedostaju** - za input $\mathbf{x} \in \mathbb{R}$ nedostaje neka vrijednost x_i i nju treba procijeniti. Koristi se u medicini, jer nisu svi podaci dostupni (npr. jer su skupi, podatak zahtjeva invazivan zahvat,...) pa moramo procijeniti kakvi će biti podaci.

8. **Uklanjanje smetnji** - dobijemo neki podatak $\tilde{\mathbf{x}} \in \mathbb{R}^n$ koji ima neku smetnju i od programa se traži da ukloni tu smetnju. Npr. ako imamo sliku auta koji je u magli, tražimo od algoritma da ukloni tu maglu.

2.1 Linearna regresija

Linearna regresija je najjednostavniji oblik regresije. Cilj je konstruirati sustav koji će za vektor $\mathbf{x} \in \mathbb{R}^n$ napraviti procjenu za $y \in \mathbb{R}$. U našem slučaju y će linearno ovisiti o $\mathbf{x}$. Ako je $\hat{y}$ vrijednost koju predviđa naš model tada $\hat{y}$ mora biti što sličniji y. Za procjenu $\hat{y}$ računamo kao

$$\hat{y} = \boldsymbol{\omega}^\top \mathbf{x} = \sum_{i=1}^{n} \omega_i x_i = \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_n x_n \tag{1}$$

gdje je $\boldsymbol{\omega} \in \mathbb{R}^n$ vektor parametara. Parametri su vrijednosti koje kontroliraju ponašanje sustava. U ovom slučaju ω_i je koeficijent koji množimo s karakteristikom x_i prije sumiranja doprinosa svih karakteristika.

Odabirom vektora parametara ω aproksimiramo y s $\hat{y}$, zanima nas koliko je dobra aproksimacija. Jedan od načina za računanje preciznosti je određivanje srednje kvadratne udaljenosti (eng. *mean square error - MSE*). Neka je zadano m test podataka na kojima ćemo mjeriti preciznost. Tada je $\hat{\boldsymbol{y}}^{(test)} \in \mathbb{R}^m$ vektor svih procjena modela na zadanim test podacima, a MSE je dana s:

$$MSE_{test} = \frac{1}{m} \sum_i (\hat{\boldsymbol{y}}_i^{(test)} - \boldsymbol{y}_i^{(test)})^2 \tag{2}$$

Jasno je vidljivo da će (2) biti jednaka 0 ako je $\hat{\boldsymbol{y}}^{(test)} = \boldsymbol{y}^{(test)}$. Također (2) se povećava kako se povećava i Euklidska udaljenost pa je

$$MSE_{test} = \frac{1}{m} \|\hat{\boldsymbol{y}}^{(test)} - \boldsymbol{y}^{(test)}\|_2^2. \tag{3}$$

Sada treba pronaći algoritam koji će minimizirati udaljenost (3) s obzirom na parametre $\boldsymbol{\omega}$. Ideja je da se minimizira MSE_{train} na bazi podataka $(\boldsymbol{X}^{(train)}, \boldsymbol{y}^{(train)})$ te provjeri točnost na $(\boldsymbol{X}^{(test)}, \boldsymbol{y}^{(test)})$. Jedan od načina za minimizaciju je traženje minimuma, odnosno gleda se gdje je gradijent od MSE_{train} jednak 0:

$$\nabla_{\boldsymbol{\omega}} MSE_{train} = 0 \tag{4}$$

$$\Rightarrow \nabla_{\boldsymbol{\omega}} \frac{1}{m} \|\hat{\boldsymbol{y}}^{(train)} - \boldsymbol{y}^{(train)}\|_2^2 = 0 \tag{5}$$

$$\Rightarrow \frac{1}{m} \nabla_{\boldsymbol{\omega}} \|\boldsymbol{X}^{(train)}\boldsymbol{\omega} - \boldsymbol{y}^{(train)}\|_2^2 = 0 \tag{6}$$

$$\Rightarrow \nabla_{\boldsymbol{\omega}} (\boldsymbol{X}^{(train)}\boldsymbol{\omega} - \boldsymbol{y}^{(train)})^\top (\boldsymbol{X}^{(train)}\boldsymbol{\omega} - \boldsymbol{y}^{(train)}) = 0 \tag{7}$$

$$\Rightarrow \nabla_{\boldsymbol{\omega}} (\boldsymbol{\omega}^\top \boldsymbol{X}^{(train)\top} \boldsymbol{X}^{(train)} \boldsymbol{\omega} - 2\boldsymbol{\omega}^\top \boldsymbol{X}^{(train)\top} \boldsymbol{y}^{(train)} + \boldsymbol{y}^{(train)\top} \boldsymbol{y}^{(train)}) = 0 \tag{8}$$

$$\Rightarrow 2\boldsymbol{X}^{(train)\top} \boldsymbol{X}^{(train)} \boldsymbol{\omega} - 2\boldsymbol{X}^{(train)\top} \boldsymbol{y}^{(train)} = 0 \tag{9}$$

$$\Rightarrow \boldsymbol{\omega} = (\boldsymbol{X}^{(train)\top}\boldsymbol{X}^{(train)})^{-1}\boldsymbol{X}^{(train)\top}\boldsymbol{y}^{(train)} \tag{10}$$

Sustav jednadžbi čije je rješenje dano s (10) naziva se normalne jednadžbe. Za više vidi [7]. Obično se termin linearne regresije izvodi s jednim dodatnim parametrom b (eng. bias). Tada je model

$$\hat{y} = \boldsymbol{\omega}^\top \mathbf{x} + b = \sum_{i=1}^{n} \omega_i x_i + b = \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_n x_n + b \tag{11}$$

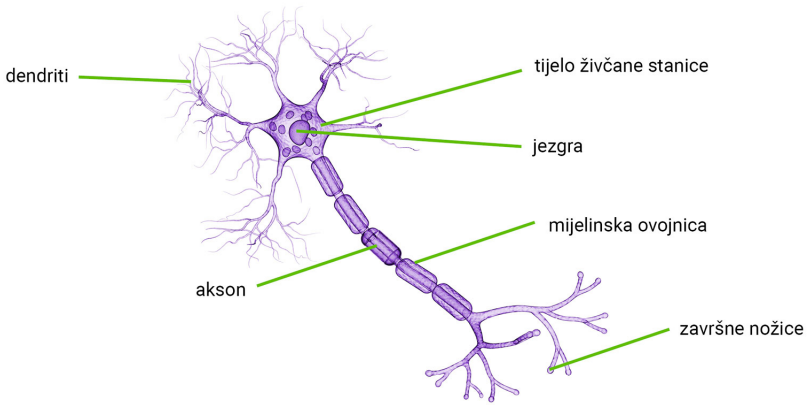
i dalje linearan, ali se može pomicati po y-osi.

Linearna regresija je jako ograničena i potrebno nam je svojstvo nelinearnosti da bi se rješavali kompleksniji problemi.

3 Umjetne neuronske mreže

3.1 Neuron

Neuron ili živčana stanica je osnovni element živčanog sustava. Sastoji se od 3 glavna elementa: tijelo s jezgrom, kratki ogranci (dendriti) i dugi ogranak (akson). Najlakše objašnjeno, neuron prima signal kroz dendrite, obrađuje informacije u jezgri te ih šalje u sljedeći neuron pomoću aksona.



Slika 2: Građa neurona

Na sličan način funkcionira i umjetni neuron. To je funkcija koja prima neke podatke (dendriti) i nakon računanja šalje podatke dalje (akson). Najveći problem je pronaći tu funkciju f. Na primjer, ako želimo napraviti klasifikator koji za neki ulazni podatak $\mathbf{x}$ mora odrediti klasu y, tada tražimo funkciju koja mapira $\mathbf{Y} = f(\mathbf{X};\boldsymbol{W})$ gdje su $\boldsymbol{W}$ takvi parametri pomoću kojih je ta aproksimacija najbolja moguća.

Jedan neuron zapravo radi sljedeće: izračunava sumu ulaznih podataka koji ovise o parametrima $\boldsymbol{W}$ i još se dodaje neki prag (eng. bias). Nakon toga neuron odlučuje da li će poslati signal dalje ili ne.

$$Y = \sum(parametri * ulaznipodaci) + prag$$

Pošto je $Y \in \langle -\infty, +\infty \rangle$ neuron i dalje ne zna treba li poslati signal ili ne, odnosno treba li se aktivirati ili ne. U tu se svrhu uvode aktivacijske funkcije.

3.2 Aktivacijske funkcije

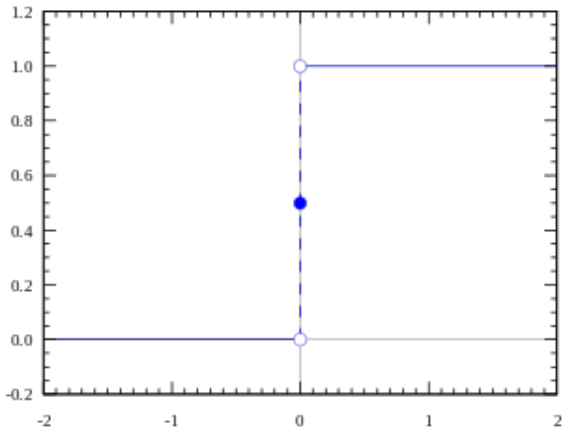
Aktivacijske funkcije važan su dio umjetnih neuronskih mreža.

3.2.1 Step funkcija

Najjednostavnija aktivacijska funkcija je step funkcija. Ako je Y veći od neke težine t, onda neka bude neuron aktiviran i neka pošalje signal. Ako bude manje od t, onda nije aktiviran.

Aktivacijska funkcija A bi za $t = 0$ izgledala ovako:

$$A(Y) = \begin{cases} 1, \text{ ako je Y} \geq 0 \\ 0, \text{ ako je Y} < 0 \end{cases}$$

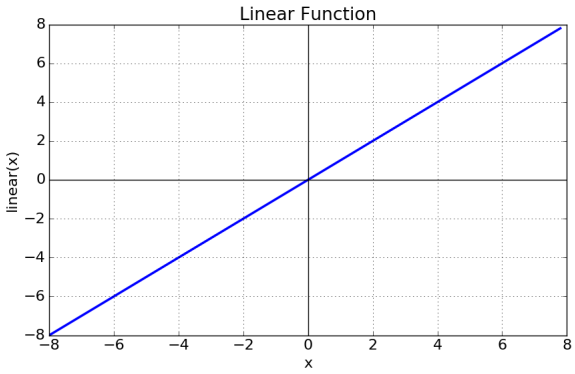


Slika 3: Primjer step funkcije

No takva funkcija se nije pokazala dobra u praksi. Jedan od problema je to što je gotovo nemoguće izgraditi klasifikator koji radi za više klasa.

3.2.2 Linearna funkcija

Linearna funkcija je funkcija oblika $A(x) = cx$, gdje je $c \in \mathbb{R}$. U našem slučaju promatramo $A(Y) = cY$. Na taj način nemamo binarnu aktivaciju i možemo povezivati više neurona.

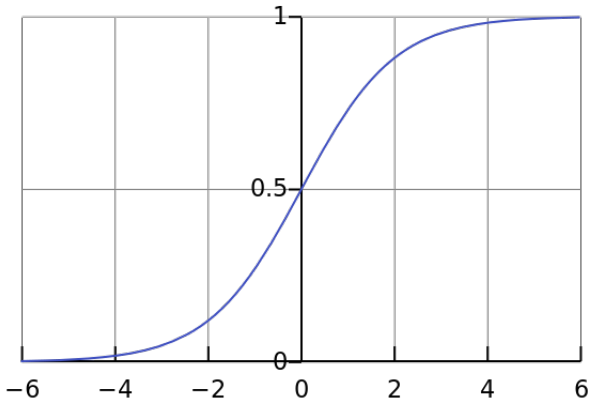


Slika 4: Primjer linearne funkcije

Problemi koji se javljaju s linarnom aktivacijskom funkcijom su ti što ako komponiramo te funkcije u konačnici opet dobivamo novu linearnu funkciju. Kako je nelinearnost svojstvo koje omogućava naprednije zakonitosti, moramo pronaći bolju funkciju.

3.2.3 Sigmoidalna funkcija

Sigmoidalna funkcija je funkcija oblika $A(Y) = \frac{1}{1+e^{-Y}}$. Slična je step funkciji, ali je glatka. To nam omogućuje svojstvo nelinearnosti te su njezine kompozicije također nelinearne. Još jedno važno svojstvo koje ima je to što je njezin izlaz uvijek između 0 i 1. Obično se označava s σ.



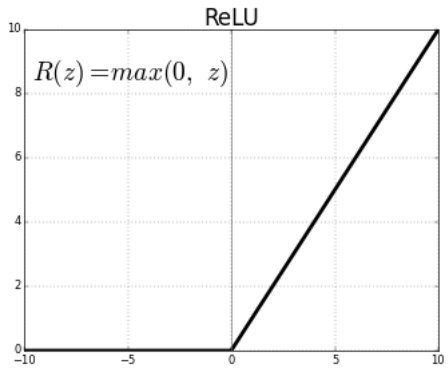
Slika 5: Sigmoidalna funkcija

Slična njoj je i tangens funkcija $f(x) = tg(x) = 2\sigma(2x) - 1$, ali ona daje vrijednosti između -1 i 1.

3.2.4 ReLu funkcija

Rectified linear unit (kraće ReLu) je vrsta aktivacijske funkcije definirana formulom:

$$A(Y) = \begin{cases} 0, \text{ ako je Y} < 0 \\ \text{Y, ako je Y} \geq 0 \end{cases}$$

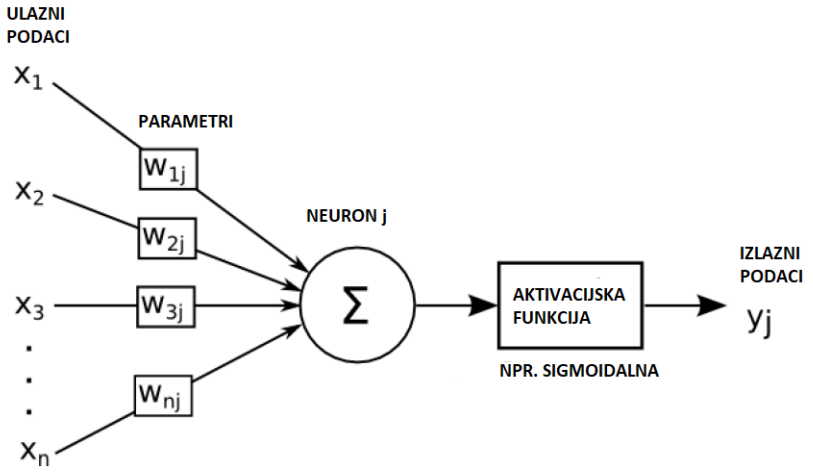


Slika 6: ReLu funkcija

Iako je naizgled vrlo jednostavna, pokazala se jako korisnom u praksi zbog svoje nelinearnosti i gradijenta koji je uvijek 1 ili 0.

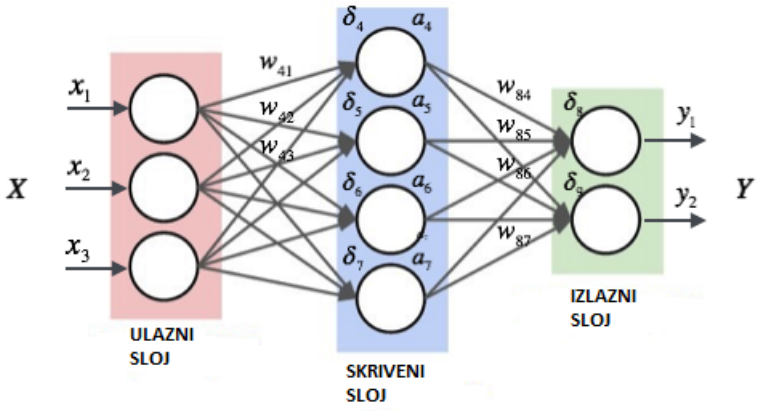
3.3 Struktura neuronskih mreža

Umjetni neuron možemo shematski prikazati na sljedeći način $\sum = \langle w_j, x \rangle + b_j$. Na tu sumu se primjenjuje aktivacijska funkcija te se izračunata vrijednost prosljeđuje do sljedećeg neurona ili je baš ta vrijednost traženo predviđanje.



Slika 7: Umjetni neuron

Sama neuronska mreža sastoji se od više neurona koji su raspoređeni u više slojeva. Postoje 3 osnovne vrste slojeva: ulazni sloj, skriveni sloj i izlazni sloj. Ulazni sloj je prvi sloj mreže i on obrađuje ulazne podatke iz sustava. Izlazni sloj je posljednji sloj mreže u kojemu su izlazi neurona zapravo predviđanja rješenja sustava. Svi ostali slojevi su skriveni.



Slika 8: Primjer strukture neuronske mreže

3.4 Funkcije troška

Najveća razlika između linearnih modela i neuronskih mreža je to što svojstvo nelinearnosti, kod neuronskih mreža, uzrokuje nekonveksnost funkcija troška. Iz toga razloga nije moguće precizno izračunati parametre kao što smo mogli, na primjer, s normalnim jednadžbama kod linearne regresije. Da bi se riješio taj problem, koriste se iterativne metode koje postepeno smanjuju funkcije troška. Metode se temelje na izračunima gradijenata i ne garantiraju konvergenciju prema globalnom minimumu funkcije, odnosno moguće je zapeti u lokalnom minimumu. Važno je da se svi početni parametri $\boldsymbol{W}$ postave na male nasumične vrijednosti, a pragovi mogu biti 0 ili male pozitivne vrijednosti.

Nakon što se postave početni parametri i početni prag treba izabrati funkciju troška i izračunati trošak. Jednu od najkorištenijih funkcija troška smo već spomenuli. To je srednja kvadratna udaljenost (MSE) koja se računa tako što za svaki primjer računamo razliku između predviđanja modela i stvarne vrijednosti, taj broj kvadriramo te od svih tih vrijednosti uzmemo srednju vrijednost.

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_i (\hat{\boldsymbol{y}}_i^{(test)} - \boldsymbol{y}_i^{(test)})^2$$

Često se još koristi i unakrsna entropija (eng. *cross entropy*). Njezina najčešća primjena je kod binarnih klasifikatora.

$$J(\boldsymbol{\theta}) = \frac{-1}{m}\sum_i (\boldsymbol{y}_i \log(\hat{\boldsymbol{y}}_i) + (1-\boldsymbol{y}_i)\log(1-\hat{\boldsymbol{y}}_i))$$

3.5 Izlazi neurona

Izbor funkcije troška usko je vezan sa izborom izlaza neurona. Ti izlazi mogu biti na samom kraju mreže ili u skrivenom sloju. Nabrojati i objasniti ćemo najvažnije.

3.5.1 Linearni izlaz za Gaussovu distribuciju

Za dani vektor svojstava $\boldsymbol{h}$, neuron daje izlaz oblika

$$\hat{\boldsymbol{y}} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}. \tag{12}$$

Linearni izlazi se često koriste za određivanje aritmetičke sredine uvjetovane Gaussove distribucije $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$. Maksimiziranje logaritamske vjerodostojnosti tada se svodi na minimiziranje srednje kvadratne pogreške.

3.5.2 Sigmoidalni izlaz za Bernoullijevu distribuciju

Mnogi zadaci zahtijevaju predviđanje vrijednosti binarne varijable y. Klasifikacija problema s dvije klase je jedan od tih problema. Princip maksimalne vjerodostojnosti nam nalaže da definiramo Bernoullijevu distribuciju za y uz uvjet $\boldsymbol{x}$. Kako je Bernoullijeva distribucija definirana sa samo jednim brojem, neuronska mreža mora predvidjeti samo $P(y=1|\boldsymbol{x})$. To predviđanje ima smisla samo ako leži u intervalu $\langle 0, 1\rangle$.

Da bismo postigli takva predviđanja, koristimo sigmoidalan izlaz oblika

$$\hat{\boldsymbol{y}} = \sigma(\boldsymbol{w}^\top \boldsymbol{h} + \boldsymbol{b}). \tag{13}$$

Možemo na sigmoidalan izlaz gledati kao da ima dvije komponente. Prva koristi linearan izlaz za računanje $z = \boldsymbol{w}^\top \boldsymbol{h} + \boldsymbol{b}$, a druga sigmoidalnu aktivacijsku funkciju za pretvaranje z u vjerojatnost. Dobije se da je vjerojatnost

$$P(y) = \sigma((2y-1)z), \tag{14}$$

što nam daje funkciju troška

$$J(\boldsymbol{\theta}) = -\log \sigma((2y-1)z). \tag{15}$$

3.5.3 Softmax izlaz za multinomnu distribuciju

Kada želimo prikazati distribuciju nad diskretnim varijablama s n mogućih vrijednosti, korisno je koristiti softmax funkciju. To je na neki način generalizacija sigmoidalne funkcije. Softmax se često koristi kao izlazni neuron za klasifikatore, jer daje distribuciju za n različitih klasa, odnosno daje nam vektor $\hat{\boldsymbol{y}}$ gdje su $\hat{y}_i = P(y = i|\boldsymbol{x}), i = 1, \ldots, n$. Bitno nam je da svi $\hat{y}_i$ imaju vrijednost između 0 i 1 te da je suma svih $\hat{y}_i$ jednaka 1. Tako dobivamo dobro definiranu distribuciju.

Prvo izračunamo linearni izlaz $z = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$ koji daje nenormalizirane log vjerojatnosti, gdje su $z_i = \log P(y = i|\boldsymbol{x})$ za svaki i. Softmax funkcija je dana sa:

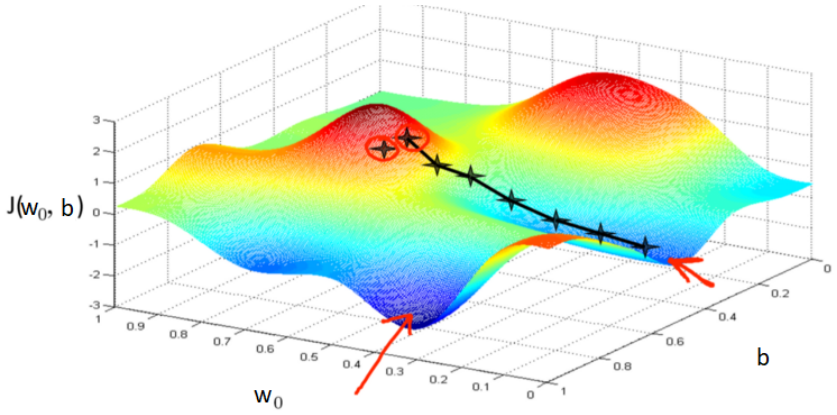
$$softmax(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}. \tag{16}$$

3.6 Učenje neuronskih mreža

U uvodnom dijelu rekli smo da neuronske mreže imaju sposobnost učenja, a u ovom dijelu objasniti ćemo kako se to postiže. Kako bi neuronska mreža mogla rješavati stvarne probleme, potrebno je podesiti njezine parametre. Da bi se to postiglo koriste se dvije važne tehnike: algoritam propagacije pogreške unatrag (eng. *Backpropagation*) i gradijentnu metodu (eng. *Gradient Descent*).

3.6.1 Gradijentna metoda

Gradijentna metoda je optimizacijska metoda koja kod neuronskih mreža minimizira funkciju troška. Da bi bolje shvatili kako funkcionira, zamislimo da se nalazimo pri samom vrhu brežuljka. Naš cilj je spustiti se u najnižu točku. Pretpostavimo da je oko nas gusta magla i vidimo samo 5 metara oko sebe. Moramo odlučiti u kojem se smjeru želimo kretati da se spustimo s brežuljka. To ćemo napraviti tako što pogledamo oko sebe i odredimo u kojem smjeru je najveći nagib, odnosno gdje je najstrmije. U tome smjeru se krećemo tih 5 metara. Nakon što smo se pomakli na novu točku, ponovo gledamo oko sebe i krećemo se u smjeru gdje je najveća nizbrdica. Postupak ponavljamo sve dok ne dođemo do takve točke iz koje se više ne možemo spuštati nizbrdo već se samo penjati uzbrdo.

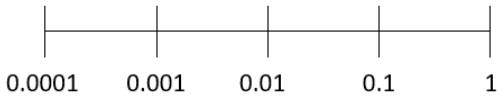


Slika 9: Spuštanje gradijenata

Na sličan način funkcionira gradijentna metoda. Kako se početni parametri sustava postave na neke nasumične vrijednosti, vrijednost funkcije troška u početku je jako velika (vrh brežuljka). U toj točci računamo gradijent funkcije troška koji nam govori u kojem je smjeru minimum (gdje je najstrmije). Tada s unaprijed određenim koeficijentom učenja α (5 metara), ažuriramo parametre tako da napreduju prema minimumu. Taj postupak ponavljamo sve dok se krećemo prema minimumu.

$$\begin{aligned} \boldsymbol{\theta} &\leftarrow Prethodno\, zadano \\ Ponavljati\, dok\, konvergira\, &\{ \\ \boldsymbol{\theta_j} &\leftarrow \boldsymbol{\theta_j} - \alpha \frac{\partial}{\partial \boldsymbol{\theta_j}} J(\boldsymbol{\theta}) \\ &\} \end{aligned} \tag{17}$$

Koeficijent učenja (eng. *learning rate*) α je koeficijent brzine konvergiranja. Ako je α prevelik, nećemo doći do minimuma već ćemo divergirati, a ako je α premalen, konvergirat će jako sporo prema minimumu. Da bi pronašli odgovarajući α možemo se koristiti sljedećom skalom. Krećemo s $\alpha = 0.01$ te ako nam model presporo uči, povećamo ga na 0.1. Analogno ako nam model krene divergirati, smanjimo koeficijent na 0.001.



Slika 10: Skala za koeficijent učenja

Za korištenje gradijentne metode potrebni su nam gradijenti. No to nije trivijalno izračunati za neuronske mreže. Zato se koristi algoritam propagacije pogreške unazad.

3.6.2 Algoritam propagacije pogreške unatrag

Jednostavno rečeno, algoritam propagacije pogreške unatrag je metoda za računanje gradijenta funkcije troška u neuronskim mrežama.

Krećemo od početnog izračuna vrijednosti funkcije troška. Znamo kako ju derivirati u odnosu na prethodni sloj. Ako znamo derivaciju od svakog sloja, možemo se vraćati unatrag i popraviti parametre u svakom sloju. Pogledajmo rad prvo na primjeru.

Neka je naša mreža zadana s 3 neurona gdje su težine w_1, w_2 i w_3 te pragovi b_1, b_2 i b_3. Neka je $a^{(n)}$ aktivacija posljednjeg neurona te $a^{(L-1)}$ aktivacija predposljednjeg neurona. Želimo da nam $a^{(L)}$ bude jednak stvarnoj vrijednosti y. Trošak za jedan primjer je

$$J_0(w_1, w_2, w_3, b_1, b_2, b_3) = (a^{(L)} - y)^2.$$

Jedan od načina na koji možemo definirati $a^{(L)}$ je sa $a^{(L)} = \sigma(z^{(L)})$, gdje je $z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)}$. Želimo izračunti $\frac{\partial J_0(\boldsymbol{\theta})}{\partial w^{(L)}}$, a za to su nam potrebne tehnike složenog deriviranja.

$$\frac{\partial J_0(\boldsymbol{\theta})}{\partial w^{(L)}} = \frac{\partial J_0(\boldsymbol{\theta})}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial w^{(L)}} \tag{18}$$

Pogledajmo pojedine parcijalne derivacije vezane za naš primjer:

$$\begin{aligned}
\frac{\partial J_0(\boldsymbol{\theta})}{\partial a^{(L)}} &= 2(a^{(L)} - y) \\
\frac{\partial a^{(L)}}{\partial z^{(L)}} &= \sigma'(z^{(L)}) \\
\frac{\partial z^{(L)}}{\partial w^{(L)}} &= a^{(L-1)}
\end{aligned}$$

Jednadžba 18 nam daje parcijalnu derivaciju samo za jedan primjer. Mi moramo izračunati za sve primjere i uzeti prosječnu vrijednost.

$$\begin{aligned}
\frac{\partial J(\boldsymbol{\theta})}{\partial w^{(L)}} &= \frac{1}{n}\sum_{i=1}^{n}(2(a_i^{(L)} - y_i) \cdot \sigma'(z_i^{(L)}) \cdot a_i^{(L-1)}) \\
\frac{\partial J(\boldsymbol{\theta})}{\partial b^{(L)}} &= \frac{1}{n}\sum_{i=1}^{n}(2(a_i^{(L)} - y_i) \cdot \sigma'(z_i^{(L)})) \\
\frac{\partial J(\boldsymbol{\theta})}{\partial a^{(L-1)}} &= \frac{1}{n}\sum_{i=1}^{n}(2(a_i^{(L)} - y_i) \cdot \sigma'(z_i^{(L)}) \cdot w_i^{(L)})
\end{aligned}$$

Sada kad smo izračunali derivaciju za zadnji sloj, izračunajmo ju za predzanji tako što koristimo informacije iz zadnjeg sloja. Postupak ponavljamo sve dok ne izračunamo sve derivacije.

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w^{(L-1)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} \tag{19}$$

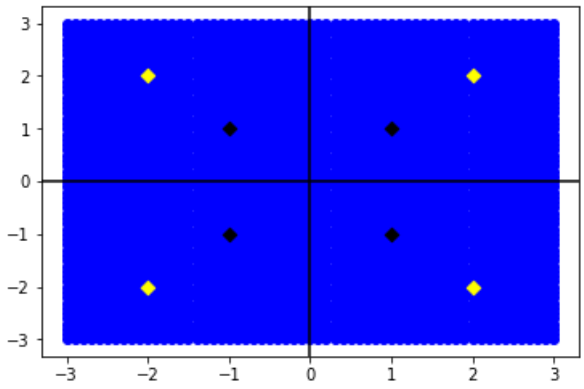
Vidimo da su nam potrebne derivacije aktivacijskih funkcija. Izdvojimo neke od njih:

Aktivacijska funkcija	Formula	Derivacija
Linearna	$f(x) = cx + b$	$f(x)' = c$
Sigmoidalna	$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f(x)' = f(x)(1 - f(x))$
Tangens	$f(x) = tg(x)$	$f(x)' = 1 - f(x)^2$
ReLu	$f(x) = \begin{cases} 0, \text{ ako je x} < 0 \\ \text{x, ako je x} \geq 0 \end{cases}$	$f(x) = \begin{cases} 0, \text{ ako je x} < 0 \\ 1, \text{ ako je x} \geq 0 \end{cases}$

Tablica 1: Aktivacijske funkcije i njihove derivacije

4 Primjer rada neuronske mreže

Pogledajmo sljedeći problem. Imamo 8 točaka, od kojih je 4 iz jedne kategorije i 4 iz druge. One su raspoređene i zadane na sljedeći način.



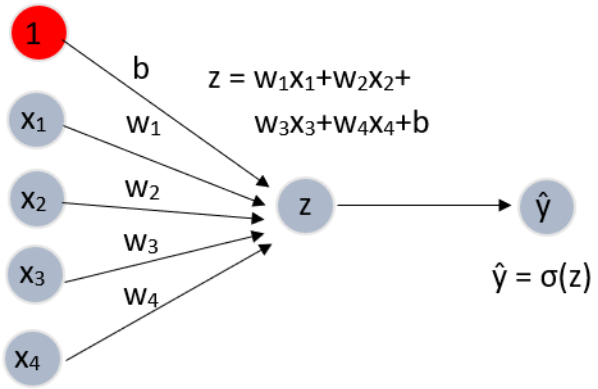
Slika 11: Skala za koeficijent učenja

Crne točke pripadaju jednoj klasi, a žute drugoj. Cilj nam je napraviti klasifikator koji će za novu točku odrediti kojoj od te dvije klase pripada. Vidimo da nije moguće odvojiti te dvije klase linearno pa je potrebno pronaći neku nelinearnu funkciju. Intuitivno vidimo da je jedan od načina na koji se to može napraviti je s kružnicom ili elipsom. Za to nam je uz koordinate točaka potrebno proslijediti i njihove kvadrate. Neka su x_1 i x_2 koordinate točaka, a $x_3 = x_1^2$ i $x_4 = x_2^2$ njihovi kvadrati.

	$\boldsymbol{X}$				$\boldsymbol{Y}$
i	x_{i_1}	x_{i_2}	x_{i_3}	x_{i_4}	y_i
1	-2	-2	4	4	0
2	-2	2	4	4	0
3	-1	-1	1	1	1
4	-1	1	1	1	1
5	1	-1	1	1	1
6	1	1	1	1	1
7	2	-2	4	4	0
8	2	2	4	4	0

Tablica 2: Ulazni podaci i pripadne klase

Da bi riješili ovaj problem, uzmemo jednostavnu neuronsku mrežu. Ona će raditi svoje predviđanje tako što će prvo pomnožiti vrijednosti x_1, x_2, x_3, x_4 s težinama w_1, w_2, w_3, w_4 te im dodati prag b, odnosno $z = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$. Nakon toga će se primijeniti sigmoidalna aktivacijska funkcija koja će nam dati predviđanje $\hat{y} = \sigma(z)$.



Slika 12: Grafički prikaz zadane neuronske mreže

$$\boldsymbol{z} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b} = \begin{bmatrix} -2 & -2 & 4 & 4 \\ -2 & 2 & 4 & 4 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 2 & -2 & 4 & 4 \\ 2 & 2 & 4 & 4 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \\ b \\ b \\ b \\ b \\ b \end{bmatrix} = \begin{bmatrix} -2w_1 - 2w_2 + 4w_3 + 4w_4 + b \\ -2w_1 + 2w_2 + 4w_3 + 4w_4 + b \\ -1w_1 - 1w_2 + 1w_3 + 1w_4 + b \\ -1w_1 + 1w_2 + 1w_3 + 1w_4 + b \\ 1w_1 - 1w_2 + 1w_3 + 1w_4 + b \\ 1w_1 + 1w_2 + 1w_3 + 1w_4 + b \\ 2w_1 - 2w_2 + 4w_3 + 4w_4 + b \\ 2w_1 + 2w_2 + 4w_3 + 4w_4 + b \end{bmatrix}$$

$$\hat{\boldsymbol{y}} = \sigma(\boldsymbol{z})$$

Kako smo prethodno rekli, kod učenja neuronskih mreža prvo se moraju zadati početne vrijednosti parametara i praga. Mi ćemo ih zadati s $w_1 = w_2, = w_3 = w_4 = 0$ i $b = 1$. Pogledajmo kakva predviđanja dobivamo za sve naše primjere:

$$z_1 = -2 \cdot 0 - 2 \cdot 0 + 4 \cdot 0 + 4 \cdot 0 + 1 = 1$$

$$\hat{y}_1 = \sigma(z_1) = \sigma(1) \approx 0.73106$$

Kako su $w_1 = w_2, = w_3 = w_4 = 0$ i $b = 1$, predviđanja za sve slučajeve su jednaka i iznose $\sigma(1) \approx 0.73106$. Ako je vrijednost predviđanja veća od 0.5 tada mu klasifikator pripisuje da je član klase 1, što znači da trenutno predviđa da su sve točke elementi klase 1. Nakon što smo odredili očekivanja slijedi nam računanje vrijednosti funkcije troška u prvom koraku. Pogledajmo kakva su naša predviđanja u odnosu na stvarne vrijednosti.

$\hat{y}_i$	y_i	Kvadratna udaljenost
0.73106	0	$(0.73106-0)^2$
0.73106	0	$(0.73106-0)^2$
0.73106	1	$(0.73106-1)^2$
0.73106	1	$(0.73106-1)^2$
0.73106	1	$(0.73106-1)^2$
0.73106	1	$(0.73106-1)^2$
0.73106	0	$(0.73106-0)^2$
0.73106	0	$(0.73106-0)^2$

Vrijednost funkcije troška računamo tako što sumiramo sve kvadratne udaljenosti i uzmemo srednju vrijednost.

$$J(w_1, w_2, w_3, w_4, b) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

$$J(0,0,0,1) = \frac{1}{8}((0.73106-0)^2 + (0.73106-0)^2 + (0.73106-1)^2 + \dots)$$

$$J(0,0,0,1) = \frac{1}{8}(4(0.73106-0)^2 + 4(0.73106-1)^2)$$

$$J(0,0,0,1) \approx 0.30339$$

Naravno, nismo zadovoljni s takvim predviđanjima i želimo popraviti model. Krećemo u optimizaciju parametara koristeći metode spuštanja gradijenata i propagacije pogreške unatrag. Prvo trebamo parcijalne derivacije funkcije troška u odnosu na sve parametre. Izračunati ćemo za w_3, a ostali se izračunavaju analogno.

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w_1} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)\cdot\sigma(z_i)\cdot(1-\sigma(z_i))\cdot x_{i_1}$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w_2} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_2} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)\cdot\sigma(z_i)\cdot(1-\sigma(z_i))\cdot x_{i_2}$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w_3} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_3} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)\cdot\sigma(z_i)\cdot(1-\sigma(z_i))\cdot x_{i_3}$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w_4} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_4} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)\cdot\sigma(z_i)\cdot(1-\sigma(z_i))\cdot x_{i_4}$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial b} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)\cdot\sigma(z_i)\cdot(1-\sigma(z_i))$$

$$\begin{aligned}\frac{\partial J(\boldsymbol{\theta})}{\partial w_3} = \frac{2}{8}((&0.73106-1)\cdot 0.73106\cdot(1-0.73106)\cdot 4+\\ &(0.73106-1)\cdot 0.73106\cdot(1-0.73106)\cdot 4+\\ &(0.73106-0)\cdot 0.73106\cdot(1-0.73106)\cdot 1+\\ &\dots)\end{aligned} \tag{20}$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w_3} = \frac{1}{4}(4 \cdot (0.73106 - 0) \cdot 0.73106 \cdot (1 - 0.73106) \cdot 4 + 4 \cdot (0.73106 - 1) \cdot 0.73106 \cdot (1 - 0.73106) \cdot 1) \tag{21}$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w_3} \approx 0.52206 \tag{22}$$

Derivacija	Aproksimacija
$\frac{\partial J(\boldsymbol{\theta})}{\partial w_1}$	0
$\frac{\partial J(\boldsymbol{\theta})}{\partial w_2}$	0
$\frac{\partial J(\boldsymbol{\theta})}{\partial w_3}$	0.52206
$\frac{\partial J(\boldsymbol{\theta})}{\partial w_4}$	0.52206
$\frac{\partial J(\boldsymbol{\theta})}{\partial b}$	0.09085

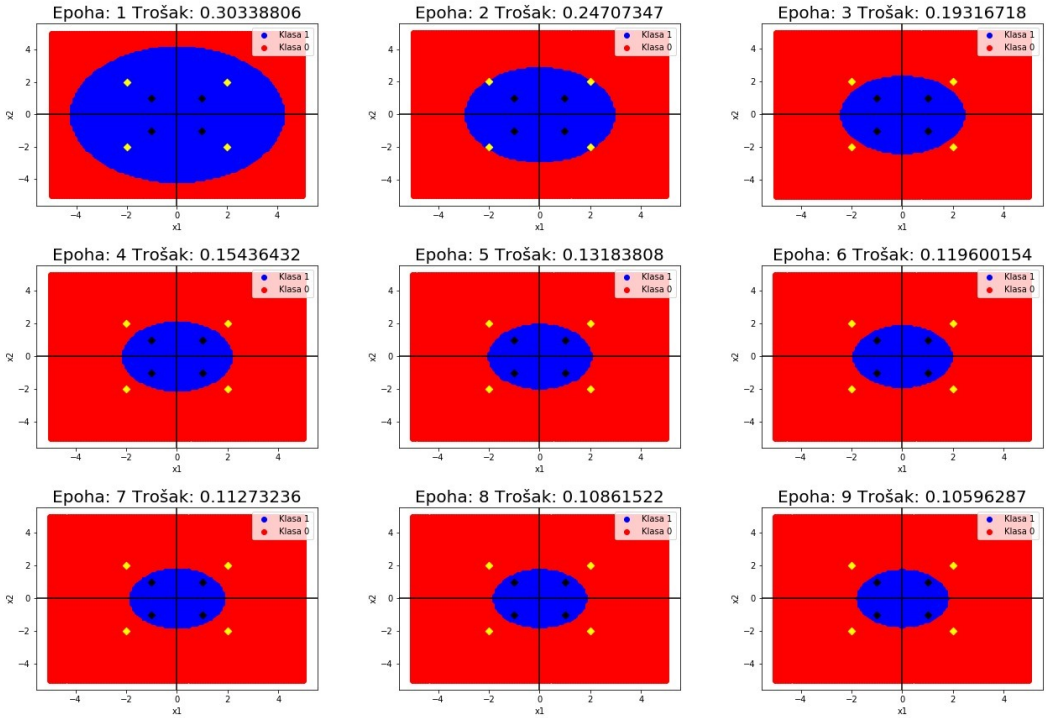
Sada dok smo izračunali parcijalne derivacije, preostaje nam primijeniti metodu spuštanja gradijenata. Prvo zadamo koeficijent učenja $\alpha = 0.1$, a zatim nam preostaje nam podesiti parametre.

$$w_1 \leftarrow w_1 - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial w_1}$$
$$w_2 \leftarrow w_2 - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial w_2}$$
$$w_3 \leftarrow w_3 - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial w_3}$$
$$w_4 \leftarrow w_4 - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial w_4}$$
$$b \leftarrow b - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial b}$$

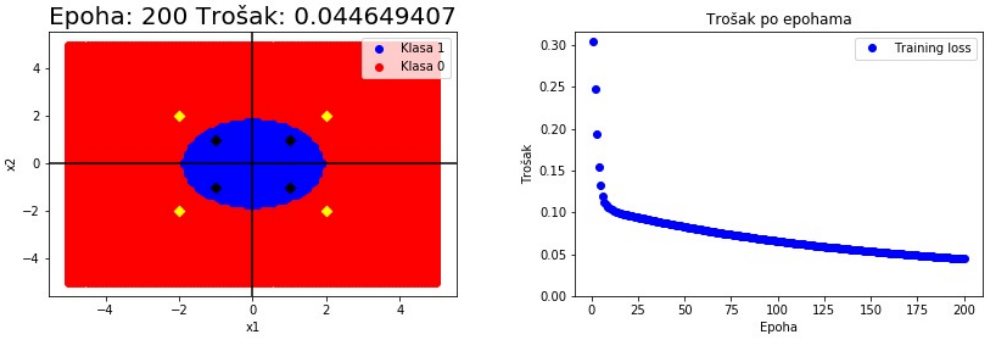
$$w_1 \leftarrow 0 - 0.1 \cdot 0$$
$$w_2 \leftarrow 0 - 0.1 \cdot 0$$
$$w_3 \leftarrow 0 - 0.1 \cdot 0.52206$$
$$w_4 \leftarrow 0 - 0.1 \cdot 0.52206$$
$$b \leftarrow 1 - 0.1 \cdot 0.09085$$

$$\begin{aligned} w_1 &= 0 \\ w_2 &= 0 \\ w_3 &= -0.052206 \\ w_4 &= -0.052206 \\ b &= 0.990915 \end{aligned}$$

Upravo smo završili jednu epohu treniranja neuronske mreže. To je proces sastavljen od računanja troška i podešavanja parametara. Preostaje nam ponavljati taj proces sve dok se funkcija troška smanjuje. Na slici 13 možemo vidjeti kakva je predviđanja dao model i kakav je bio trošak u odnosu na epohe, te na slici 14 možete vidjeti predviđanje nakon 200 epoha i pad troška u tih 200 epoha.



Slika 13: Predviđanja i troškovi po epohama



Slika 14: Predviđanja i trošak nakon 200 epoha

Literatura

[1] https://arxiv.org/abs/1711.05225

[2] https://www.scientificamerican.com/article/how-the-computer-beat-the-go-master

[3] https://www.usatoday.com/story/tech/news/2016/02/29/google-car-hits-bus-first-time-fault/81115258

[4] https://medium.com/@andrewng/self-driving-cars-are-here-aea1752b1ad0

[5] https://edutorij.e-skole.hr/share/proxy/alfresco-noauth/edutorij/api/proxy-guest/4f77c550-2b26-4717-b65e-82b845685f3f/biologija-8/m03/j01/index.html

[6] https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

[7] http://www.mathos.unios.hr/nla/NLA.pdf

[8] I.Goodfellow, Y.Bengio, A.Courville, Deep Learning, MIT Press, 2016