Korištenje neuronskih mreža u iOS aplikacijama

Baić, Hrvoje

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

KORIŠTENJE NEURONSKIH MREŽA U iOS APLIKACIJAMA

Hrvoje Baić

Zagreb, prosinac 2017.

Student vlastoručno potpisuje Završni rad na prvoj stranici ispred Predgovora s datumom i oznakom mjesta završetka rada te naznakom:

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, datum.

Hrvoje Baić

Predgovor

Zahvaljujem svim profesorima koji s entuzijazmom prihvaćaju svoj poziv.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

U ovom radu rješava se problem prepoznavanja rukom pisanih slova na mobilnom uređaju korištenjem neuronske mreže. Takav problem bilo bi gotovo nemoguće riješiti proceduralnim programiranjem. Srž rada je implementacija mreže koja se može koristiti i u drugim aplikacijama za rješavanje sličnih problema. Prvi dio bavi se teorijom, vrstama neurona i načinima treniranja. Drugi dio detaljno opisuje značajne dijelove koda i implementaciju.

Ključne riječi: iOS, neuronske mreže, umjetna inteligencija.

Summary

This thesis solves the problem of handwriting recognition on mobile devices using neural networks. Such problem would be near to impossible to solve using procedural programming. The core is the implementation of the network which could be used in other applications to solve similar problems. First part describes types of neurons and training methods in theory. Second part describes important parts of the code and implementation.

Keywords: iOS, neural networks, artificial intelligence

Sadržaj

.

1. Uvod

Kompleksnost ljudskog mozga je ono što nas razlikuje od ostalih životinja. Intelektualne sposobnosti omogućile su nam svijest o sebi i svijetu oko nas. Prirodno je bilo zapitati se što nam to daje takvu prednost nad drugim bićima i nastojati razumjeti po čemu smo drugačiji. Čovjek, primat koji je iznimno dobar u stvaranju alata, sada nastoji stvoriti alat koji je stvorio njega - mozak. Još uvijek ne znamo hoćemo li uspjeti stvoriti umjetnu inteligenciju sličnu našoj, inteligenciju koja posjeduje svijest, jer još uvijek ne razumijemo svijest u potpunosti. No, uspjeli smo simulirati biološke procese mozga, princip na kojem neuroni kreiraju našu realnost te sada imamo neuronske mreže koje mogu "zamišljati" slike, prepoznavati bolesti, razumjeti jezik, igrati i čak pobijediti čovjeka u strateškim igrama poput šaha. Svoje intelektualne sposobnosti uglavnom uzimamo zdravo za gotovo, ne imajući na umu složenost procesa koji se odvijaju u mozgu kada čitamo. Milijarde neurona sa milijardama sinapsi čine iznimno moćno nelinearno, paralelno i energetski vrlo efikasno računalo. Ovaj rad prikazat će kako pomoću računala možemo kreirati umjetnu mrežu neurona koja uči i rješava problem klasifikacije, ono što mozak radi svaki puta kada prepoznaje stvari iz naše okoline.

2. Hipoteza

Ovaj rad bavi se prepoznavanjem rukom pisanih znakova. U prvom dijelu čitatelja se uvodi u teoriju neuronskih mreža, strukturu i vrste učenja, a u drugom opisuje implementaciju jedne neuronske mreže. Kao algoritam učenja izabrao sam učenje unazadnom propagacijom (engl. *backpropagation*). Korisnik rukom crta znak na ekranu mobilnog uređaja. Aplikacija reže znak iz suvišnih bjelina na ekranu i skalira ga na predefiniranu vrijednost. Skalirana slika pretvara se u polje piksela koje se šalje u neuronsku mrežu na obradu.

Primjenom navedenih metoda na mobilnom uređaju moguće je kreirati neuronsku mrežu i pružiti gotovo rješenje kojim se kroz korištenje jednostavnog aplikacijskog programskog sučelja (engl. *application programming interface*) omogućuje korištenje funkcionalnosti neuronske mreže bez prethodnog poznavanja detalja tematike.

Podaci se prikupljaju od korisnika aplikacije koji može iterativno trenirati mrežu ili testirati klasifikaciju na dosadašnjem setu podataka. Za mjerenje efikasnosti klasifikacije koriste se ROC (engl. *Receiver Operating Characteristic*, skraćeno ROC) krivulje.

3. Neuronske mreže

Zbog relativno nedavnog uspjeha neuronskih mreža u rješavanju složenih problema za koje se donedavno mislilo da su za računala nerješivi, moglo bi se pomisliti kako se radi o novoj tehnologiji, no taj pojam se pojavljuje još 1943. godine zahvaljujući radu neurofiziologa Warrena McCulloha i matematičara Waltera Pittsa koji su svojim radom opisali neuron kao osnovnu logičku jedinicu u mozgu.

Donald Hebb proširio je razumijevanje neurona objasnivši kako se veze među njima pojačavaju učestalijom aktivacijom. Najstarija postojeća mreža je Mark I Perceptron, djelo neurobiologa Franka Rosenblatta. Perceptron (jedna od vrsta implementacija neurona, opisan u kasnijim poglavljima) je nastao iz proučavanja oka muhe, te kako u oku funkcionira donošenje odluka. Služio je kao prvi primjer računala koje uči i donosi odluke kao čovjek.

U knjizi *Perceptrons* Marvin Mlinsky i Seymour Papert istaknuli su nedostatke perceptrona i ukazali na neke dotad nepoznate zapreke u učenju kao što je rješavanje XOR sklopa [2], što je kasnije opovrgnuto implementacijom troslojnih mreža. Njihov rad značajan je i po tome što je uvelike usporio razvoj i financiranje istraživanja neuronskih mreža.

Povećanjem procesorske moći računala, gotovo četrdeset godina kasnije od prvog matematičkog modela, neuronske mreže opet dobivaju zasluženu pažnju i razvijaju se nova područja znanosti. Služe za rješavanje skupa određenih problema korištenjem metoda kao što su klasifikacija (grupacija) i regresija (predviđanje). Svrha mreže je da pronađe funkciju koja aproksimira izlazne vrijednosti na temelju ulaznih i uz dobru sposobnost generalizacije rješava probleme slične onima na kojima je trenirana.

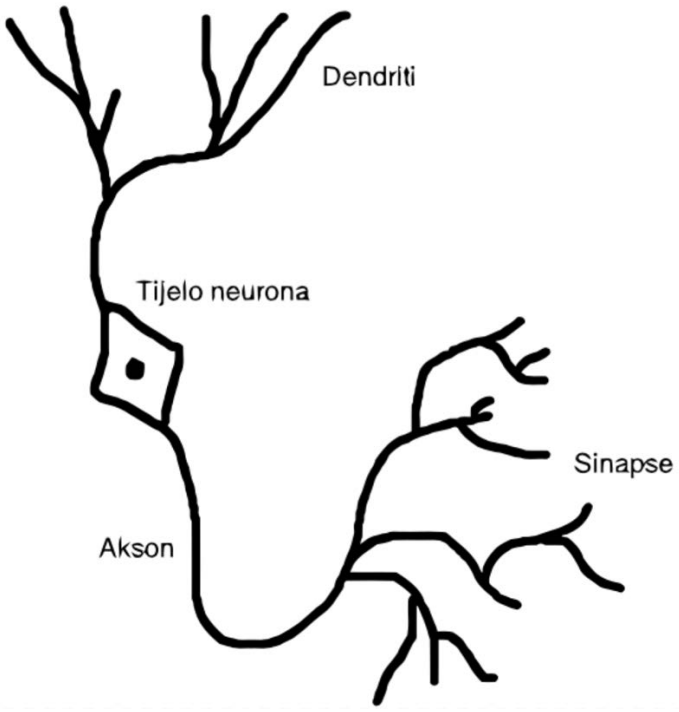
3.1. Neuron

Neuron je osnovna logička jedinica neuronske mreže. Umjetne neuronske mreže primitvna su imitacija bioloških. Milijarde neurona i njihovih spojeva čine iznimno kompleksno paralelno računalo.

3.1.1. Biološki neuron

Sastoji se od dendrita, aksona, sinapsi i tijela neurona [1]. Dendriti primaju ulazne signale od drugih neurona. Akson prenosi impulse prema sinapsama koji se spajaju na dendrite

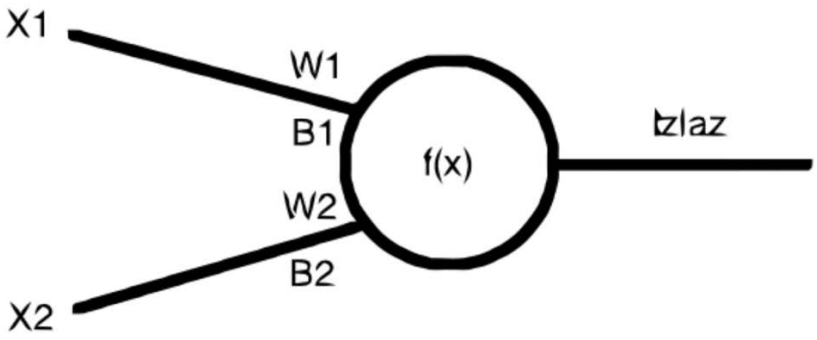
drugih neurona. Sam neuron predstavlja elektropodražajno tijelo koje se aktivira iznad određenog praga.



Slika 1. Biološki neuron

3.1.2. Umjetni neuron

Po uzoru na biološki, umjetni neuron prenosi signal kada je podražen iznad određenog praga. Takav neuron naziva se **perceptron** [3]**.** Rosenblatt je uveo koncept težinskih vrijednosti koje označavaju utjecaj pojedinih ulaza na konačni izlaz.



Slika 2. Umjetni neuron i ulazi sa pripadajućim težinskim vrijednostima i pomacima

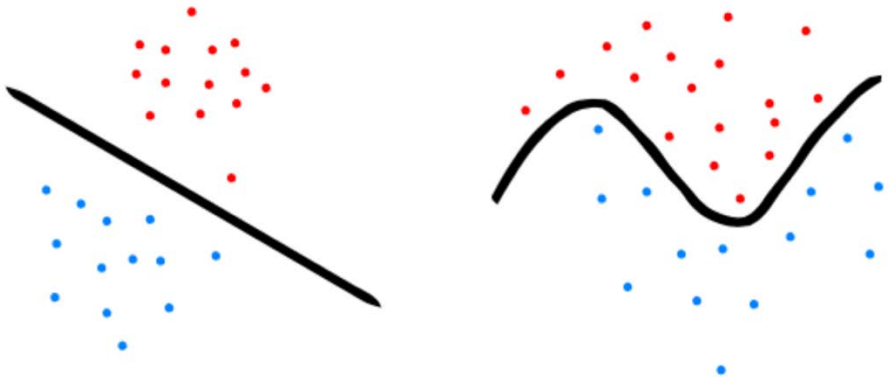
Na gornjoj slici:

- *X1* i *X2* su ulazi
- *W1* i *W2* su težinske vrijednosti
- *B1* i *B2* su pomaci (engl. *bias*) po formuli $f(x) = w * x + b$

Ideja perceptrona je prilično jednostavna. Suma ulaza množenih sa težinskim vrijednostima $\sum_i w_i x_i$ mora biti veća od zadanog praga. Donja formula opisuje slučajeve kada neuron okida.

$$f(x) = \begin{cases} 1 & ako\ je\ w * x + b > 0 \\ 0 & u\ svim\ ostalim\ slučajevima \end{cases}$$

Problem perceptrona jest to što je on linearni klasifikator. To znači da može klasificirati samo one skupove koji su linearno separabilni. U slučaju da razredi nisu linearno separabilni, neuronska mreža neće moći naći rješenje, te moramo primijeniti drugu aktivacijsku funkciju.



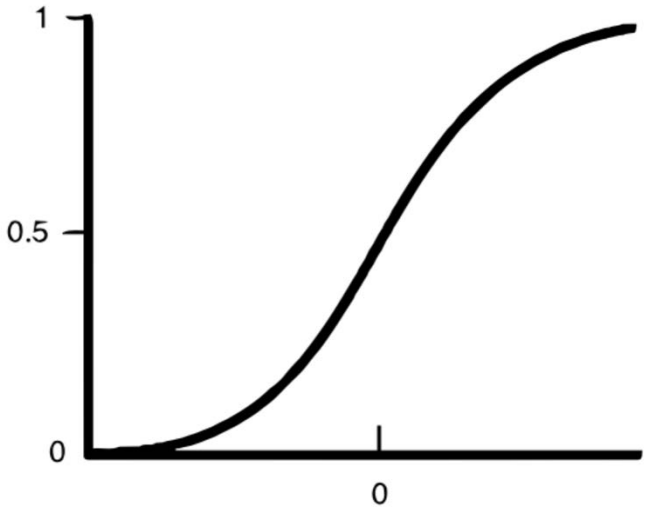
Slika 3. Primjer linearne i nelinearne separabilnosti

Zato se danas, umjesto linearne, koristi **sigmoidna** funkcija:

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

gdje je $z = w * x + b$.

Perceptron ima binarne ulaze i daje 0 ili 1 na izlazu, a sigmoid prima i daje broj u rasponu [0, 1] što znači da mala promjena na ulazu rezultira malom promjenom na izlazu.

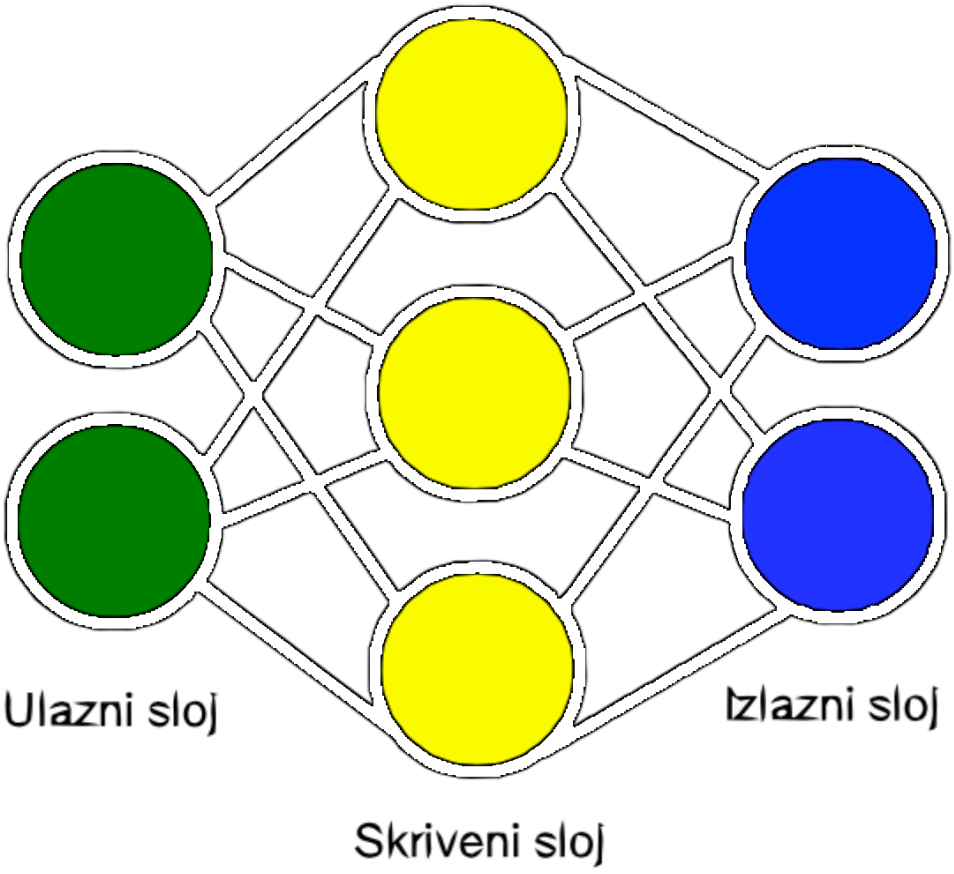


Slika 4. Sigmoidna funkcija

Za vrlo velike i vrlo male vrijednosti sigmoid je sličan perceptronu, ali za sve vrijednosti u sredini perceptronova step funkcija je izglađena, što je glavna značajka sigmoida u odnosu na perceptron.

3.2. Topologija

Svaka neuronska mreža ima barem jedan sloj (izlazni). Ulazni sloj sastoji se od numeričkih (ulaznih) vrijednosti i ne mora biti implementiran kao skupina neurona, jer se u njemu ne vrši procesiranje. Izlazni sloj također se sastoji od numeričkih vrijednosti koje su u slučaju nadziranog učenja, koje je implementirano u ovom radu, predefinarane za svaki razred koji mreža raspoznaje. U sredini može imati skrivene slojeve. Svaki sloj sastoji se od neurona koji su međusobno povezani sa neuronima u drugim slojevima. Srednji ili skriveni slojevi nalaze se između ulaznih i izlaznih slojeva. Broj skrivenih slojeva i broj neurona u njima nije definiran i varijacije se očituju u performansama neuronske mreže.



Slika 5. Primjer strukture neuronske mreže

U odnosu na broj slojeva razlikujemo jednoslojne i višeslojne mreže. Primjer jednoslojne mreže je linearna asocijativna memorija.

3.3. Učenje

Kao i ljudski mozak, umjetna neuronska mreža ima sposobnost učenja i spremanja znanja. Jačanjem veza među neuronima zapravo se stvara pamćenje koje se aktivira za određene podražaje iz okoline, odnosno ulazne vrijednosti. U slučaju klasifikacije, koja se koristi u ovom radu, učenje se svodi na pronalaženje dovoljno dobre funkcije krivulje koja dijeli dva ili više razreda, a da pritom može generalizirati, odnosno da nije previše prilagođena (engl. *fitting*) određenom skupu testnih podataka. U slučaju da mreža ne zna generalizirati, male promjene u ulaznim podacima značit će potpuno krive izlazne vrijednosti. Zanimljivo je da u ambiciji da reproduciramo procese u ljudskom mozgu pretpostavljamo da se ti procesi mogu opisati matematičkim funkcijama. Kao dvije najpoznatije metode učenja poznajemo nenadzirano i nadzirano učenje.

3.3.1. Nenadzirano

Nenadzirano učenje koristimo kada želimo stvoriti razrede iz nekog seta podataka. Želimo da mreža shvati po čemu se ti podaci razlikuju i da nađe način da ih grupira. Primjeri problema koje rješavamo nenadziranim učenjem mogu biti klasteriranje, filtriranje itd.

3.3.2. Nadzirano

Nadziranim učenjem za svaki set ulaznih podataka mreži govorimo koji je točan izlazni podatak. Možemo reći da mapiramo ulaze na izlaze. Uspješnost tog mapiranja u algoritmu mjerimo funkcijom troška (engl. *cost function*)

$$C(w,b) \equiv \frac{1}{2n}\sum_{x} || \, y(x) \, - \, a \, ||^2$$

koja je srednja kvadratna pogreška između ulaznih i izlaznih vrijednosti.

U gornjoj formuli:

- *w* stoji za sve težinske vrijednosti
- *b* stoji za sve vrijednosti pomaka
- *n* je broj ulaza
- *x* je ulaz
- *a* stoji za izlazne vrijednosti
- *suma* je za svaku ulaznu vrijednost
- *y(x)* je aktivacija neurona na izlazu

Algoritam možemo smatrati uspješnim ako je $C(w,b) \approx 0$, odnosno ako je $y(x) \approx a$. Riječima, želimo smanjiti funkciju troška tako da treniranjem mreže predviđeni rezultat što više približimo očekivanom.

3.4. Princip rada temeljen na propagaciji unazad

Princip rada neuronske mreže možemo podijeli u dva koraka:

- treniranje
- testiranje

Za testiranje potrebna je samo **propagacija unaprijed**, a za treniranje još i **propagacija unazad** koja se koristi kao algoritam učenja. Propagacijom unaprijed možemo vidjeti koje vrijednosti dobivamo na izlazu u odnosu na trenutne težinske vrijednosti i pomak. Propagacijom unazad mijenjamo težinske vrijednosti i pomak u odnosu na grešku koja nastaje između predviđene i očekivane vrijednosti.

3.4.1. Propagacija unaprijed

Novoinicijalizirana mreža generira nasumične težinske vrijednosti i vrijednosti pomaka. Puštanjem ulaznih vrijednosti kroz mrežu događa se slijedeće:

- računamo ulaz na svakom neuronu sa $\sum_i w_i * x_i + b_i$
- provlačimo sumu ulaza svakog neurona kroz sigmoidnu funkciju kako bi normalizirali vrijednost između 0 i 1
- ponavljamo isti proces za sve neurone u svakom idućem sloju

Na izlazu iz mreže dobivamo predviđene vrijednosti. Ako je mreža trenirana, dobit ćemo zadovoljavajuće vrijednosti, a ako nije propagacijom unazad možemo korigirati težinske vrijednosti tako da smanjimo grešku na izlazu

3.4.2. Propagacija unazad

Propagacija unazad je algoritam koji služi za treniranje neuronske mreže. Njome se iterira kroz neurone i korigira težinske vrijednosti kako bi se greška na izlazu smanjila. Podsjetimo se, greška na izlazu je razlika između predviđene i očekivane vrijednosti. Kao što smo spomenuli, koristeći sigmoidnu funkciju male promjene na ulazima rezultiraju malim promjenama na izlazu. Krećući se unazad kroz mrežu izračunat ćemo koliko svaka od težinskih vrijednosti izlaznog sloja utječe na grešku i korigirati je.

Algoritamski:

- ukupna greška je zbroj grešaka na svim izlazima
- želimo znati kako se greška na izlazu mijenja sa promjenom težinske vrijednosti $\frac{\partial\, error_{ukupni}}{\partial\, w_{ij}}$ gdje je i indeks neurona, a j index sloja. Ovaj izraz možemo čitati kao

  promjena ukupne greške u odnosu na promjenu težinske vrijednosti w_{ij} ili ***gradijent*** *u odnosu na težinsku vrijednost* w_{ij}

- korištenjem kompozicije funkcija do rezultata dolazimo množenjem svih promjena od izlaza do težinske vrijednosti, suprotno od onoga što radimo kod propagacije unaprijed, formulom

$$\frac{\partial\, error_{ukupni}}{\partial\, w_{ij}} = \frac{\partial\, error_{ukupni}}{\partial\, izlaz_{ij}} * \frac{\partial\, izlaz_{ij}}{\partial\, ulaz_{ij}} * \frac{\partial\, ulaz_{ij}}{\partial\, w_{ij}}$$

- derivacija sigmoidne funkcije je

$$izlaz_{ij} * (1 - izlaz_{ij})$$

- u nekim izvorima izraz $\frac{\partial\, error_{ukupni}}{\partial\, izlaz_{ij}} * \frac{\partial\, izlaz_{ij}}{\partial\, ulaz_{ij}}$ se izražava kao **delta** [4], odnosno δ
- korekcija težinske vrijednosti radi se razlikom trenutne težinske vrijednosti sa umnoškom **stope učenja** $\boldsymbol{\eta}$ i $\frac{\partial\, error_{ukupni}}{\partial\, w_{ij}}$ što daje:

$$w_{ij} = w_{ij} - \eta * \frac{\partial\, error_{ukupni}}{\partial\, w_{ij}}$$

- proces ponavljamo za sve težinske vrijednosti neurona izlaznog sloja

Na sličan način korigiramo težinske vrijednosti u skrivenim slojevima, s jedinom razlikom što moramo uzeti u obzir da promjena na svakom neuronu u tom sloju utječe na promjenu na neuronima u idućim slojevima. U kontekstu skrivenog sloja, algoritamski:

$$\frac{\partial\, error_{ukupni}}{\partial\, w_{ij}} = \frac{\partial\, error_{ukupni}}{\partial\, izlaz_{ij}} * \frac{\partial\, izlaz_{ij}}{\partial\, ulaz_{ij}} * \frac{\partial\, ulaz_{ij}}{\partial\, w_{ij}}$$

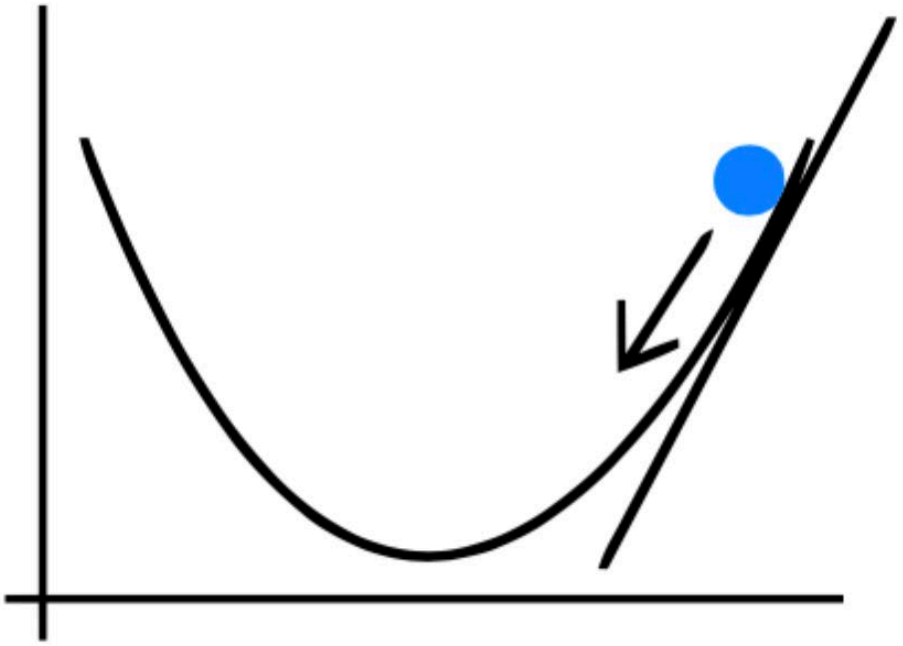
s tim da je

$$\frac{\partial\, error_{ukupni}}{\partial\, izlaz_{ij}} = \frac{\partial\, error_{jk}}{\partial\, izlaz_{ij}} + \frac{\partial\, error_{jl}}{\partial\, izlaz_{ij}} + \frac{\partial\, error_{jm}}{\partial\, izlaz_{ij}} + \ \ldots$$

za svaki neuron u izlaznom sloju. Bitno je voditi računa da u implementaciji **ne ažuriramo** težinske vrijednosti prije nego što se korigiraju sve težinske vrijednosti neuronske mreže u toj iteraciji.

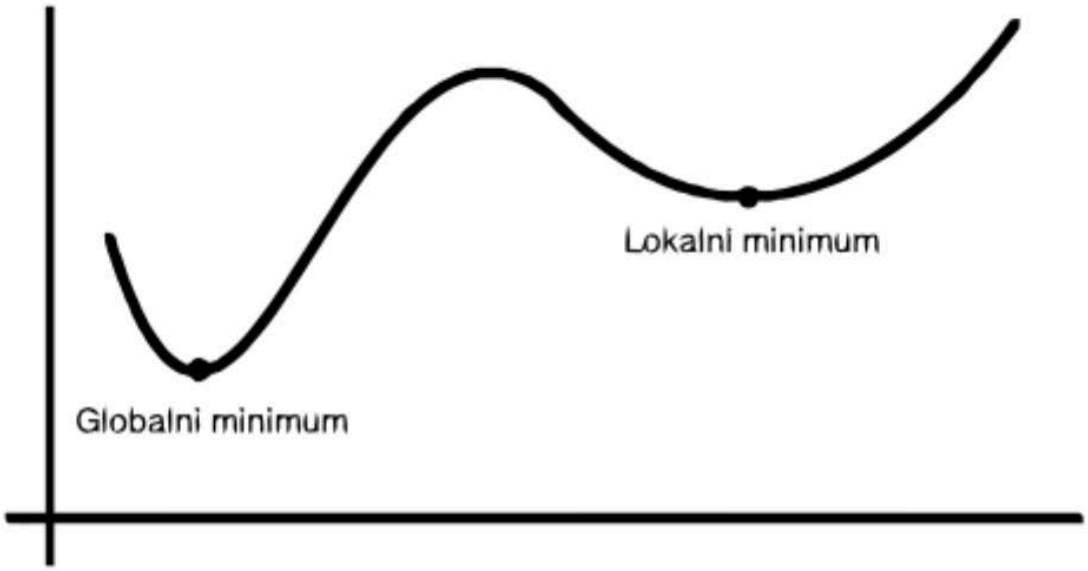
Stopa učenja je proizvoljan koeficijent koji određuje brzinu učenja, odnosno pomak po krivulji u smjeru prema globalnom minimumu. Kao metodu pronalaska globalnog minimuma koristimo **stohastički gradijentni spust** [5]. Tom metodom minimiziramo

funkciju troška na način da uz pomoć tangente na krivulji za mali uzorak podataka (engl. *batch*) tražimo smjer u kojem se trebamo kretati, kao loptica na brežuljku.



Slika 6. Ilustracija stohastičkog gradijentnog spusta

Problem koji se javlja kod treniranja mreže na taj način jest da učenje zapne u lokalnom minimumu.



Slika 7. Primjer funkcije sa globalnim i lokalnim minimumom

Unatoč tome, stohastični gradijentni spust funkcionira za rješavanje vrlo velikog broja problema.

4. Implementacija

Ovo poglavlje opisat će detalje implementacije neuronske mreže i izdvojiti bitne dijelove koda, obrazložiti motivaciju iza ovog rada, te navesti mogućnosti primjene u praksi.

4.1. Motivacija

U vrijeme prijave ovog rada, postojalo je svega nekoliko parcijalno implementiranih neuronskih mreža u Swiftu, relativno novom Appleovom programskom jeziku. U međuvremenu taj broj se povećao, a Apple je izdao CoreML, biblioteku namijenjenu radu sa strojnim učenjem. No to ne umanjuje vrijednost ovog rada koji pruža podršku programerima koji ne žele raditi sa velikim bibliotekama i imati standardizirane modele, već brzo prototipizirati ili u svoju aplikaciju uključiti strojno učenje sa jednostavnim aplikacijskim programskim sučeljem u svega tri klase:

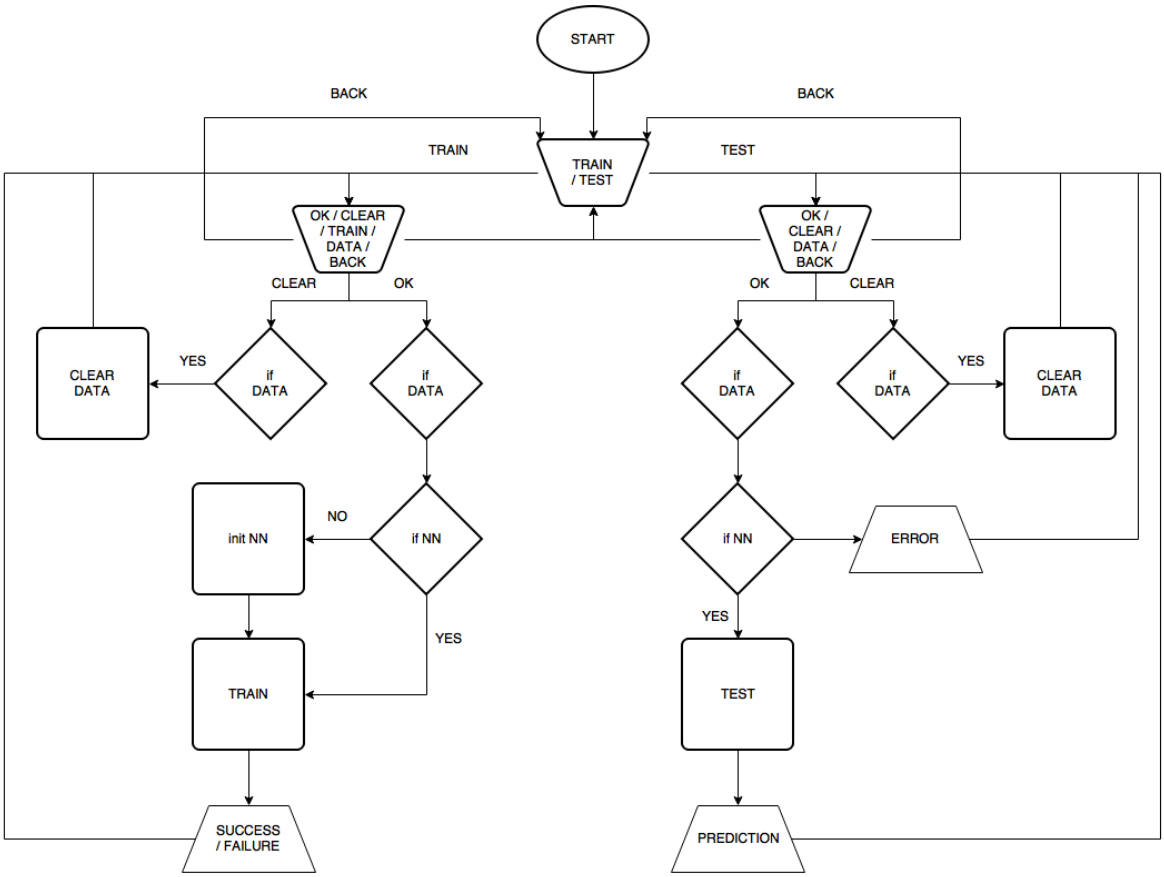
- Neuron
- Sloj
- Neuronska mreža

4.2. Opis aplikacije

Srž rada jest neuronska mreža, ali za demonstraciju je napisana aplikaciju koja ju koristi za prepoznavanje rukom pisanih znakova. Aplikacija se sastoji od ekrana sa izbornikom u kojem su opcije:

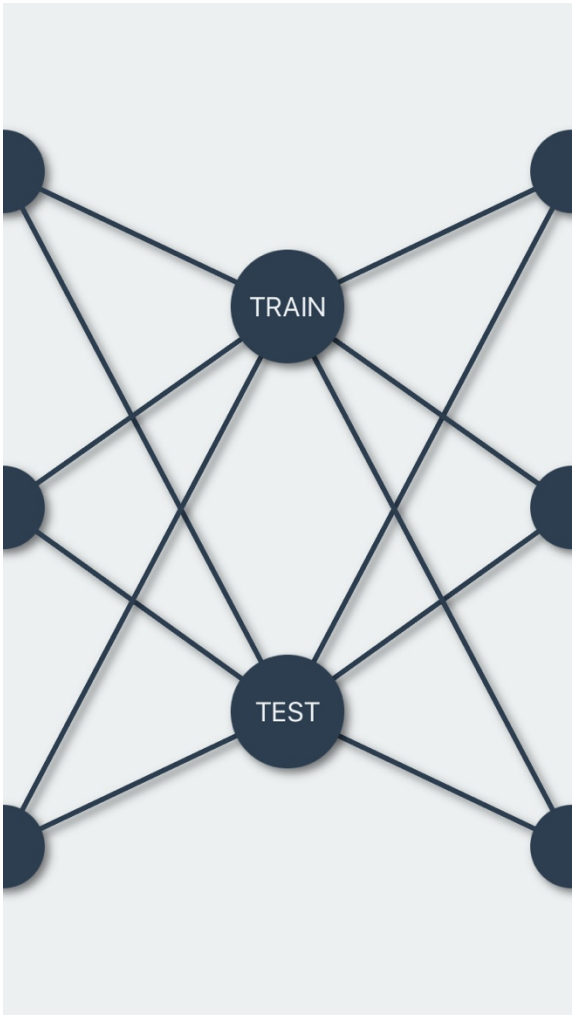
- Treniranje (engl. *train*)
- Testiranje (engl. *test*)

i ekrana namijenjenih za treniranje i testiranje.



Slika 8. Dijagram toka

Dijagram toka opisuje tijek izvršavanja i opcije koje korisnik ima na raspolaganju u točkama grananja. Aplikacija se grana na dvije strane, u granu za treniranje i u granu za testiranje. Korisničko sučelje je slično u oba slučaja što olakšava korištenje aplikacije. Razlika je u pozadinskoj logici gdje se u slušaju testiranja podaci provlače samo kroz algoritam propagacije unaprijed, a kod treniranja još i kroz algoritam propagacije unazad. U oba slučaja vrši se obrada slike, transformacija piksela u polje nula i jedinica te rezanje.



Slika 9. Glavni izbornik

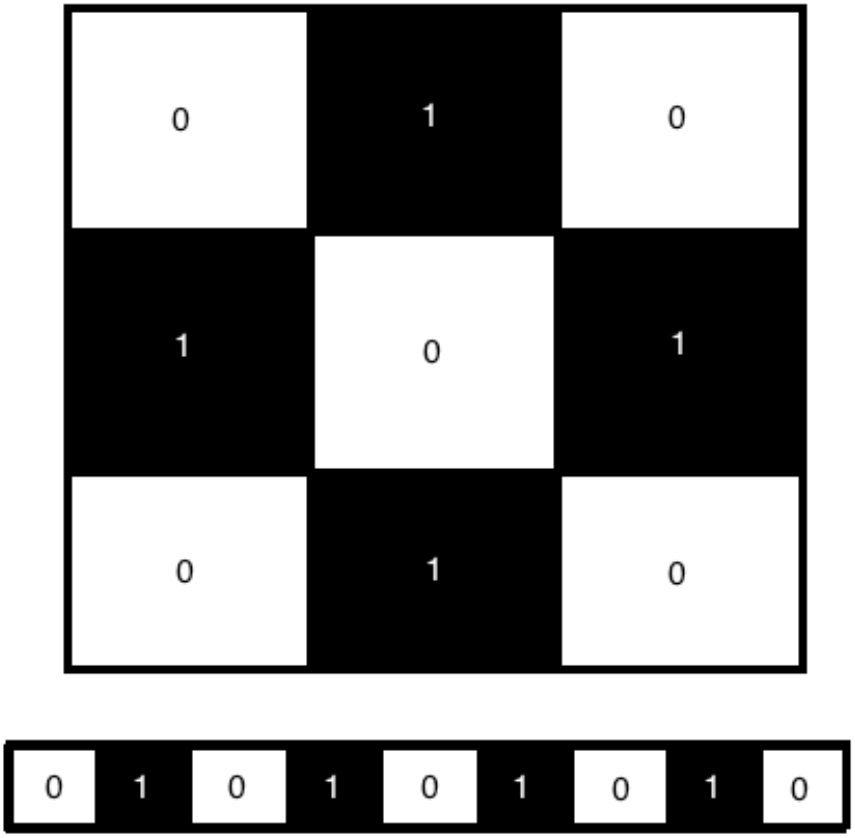
Pritiskom na tipku *TRAIN* aplikacija će otvoriti novi ekran sa opcijama za treniranje svakog slova engleske abecede. Pritiskom na tipku *TEST* otvara se ekran za testiranje na kojem korisnik prstom crta bilo koji znak. Mreža odgovara sa svojom naboljom klasifikacijom pri vrhu ekrana.

Strukturno, mreža se sastoji od klasa *neuron*, *sloj* (engl. *layer*) i *neuronska mreža* (engl. *neural network*). Klasa *neuronska mreža* sadrži slojeve i metode za treniranje i testiranje. Metoda `trainWith(inputs: [Float], targetOutput: [Float], learningRate: Float = 0.3`) prosljeđuje vrijednosti sloju koji radi propagaciju unaprijed i unazad, a metoda `predictFor(inputs: [Float]) -> [Float]` samo propagaciju unaprijed i vraća predviđenu vrijednost sa izlaza. Slojevi sa sastoje od neurona i u njima se nalaze algoritmi za propagaciju koje ćemo obraditi detaljno u kasnijem poglavlju. Neuron čuva sve svoje težinske vrijednosti, pomak i vrijednost sa izlaza nakon

aktivacije koja se dobiva pozivanjem metode `activate(inputs: [Float]) -> Float`

4.2.1. Treniranje

Korisnik prstom na ekranu unosi znak, a u izborniku ispod izabire o kojem se znaku radi. Pritiskom na tipku *OK* uzorak se sprema za treniranje. Slovo sa ekrana u plavom okviru prvo se izreže i skalira na predefeniranu vrijednost. Rezanje i skaliranje rješava problem kada korisnik upisuje veće ili manje slovo, različitim rukopisom, tako da su sva slova jednako skalirana u svojim pravokutnicima. Nakon toga, iteriramo kroz sve piksele skalirane slike po visini i širini. Cilj je od matrice piksela koja predstavlja sliku stvoriti jednodimenzionalno polje koje ćemo predati neuronskoj mreži kao ulazne parametre. Nule predstavljaju bijele piksele, a jedinice crne.

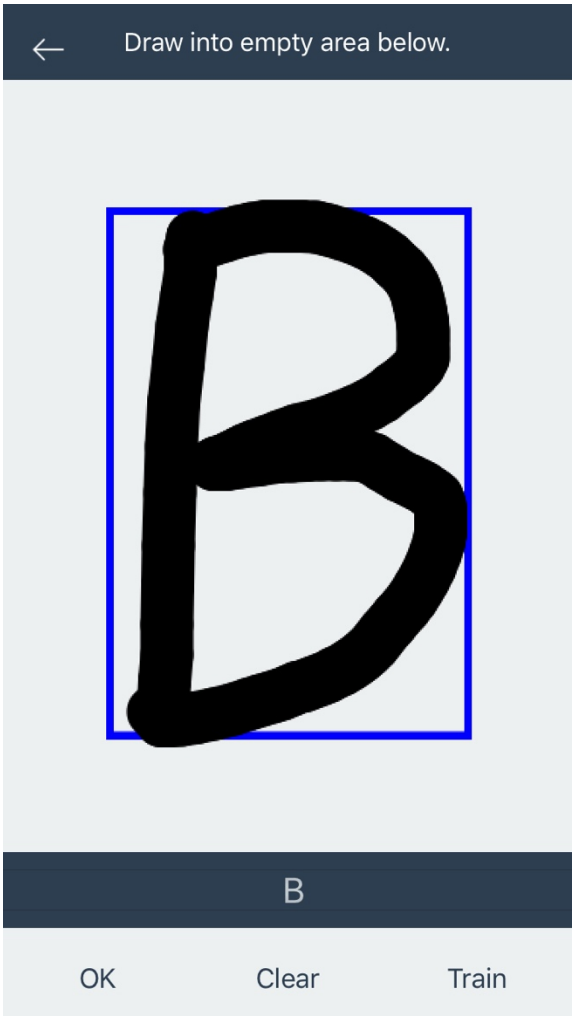


Slika 10. Primjer slike slova "o" u 3x3 matrici, konvertiranog u jednodimenzionalno polje

Polje u koje spremamo uzorke znakova za testiranje je trodimenzionalno i sadrži polje piksela pojedinog slova u najdubljoj razini. Viša razina sadrži polja piksela za svako slovo,

jer ih može biti više za svako slovo. Najviša razina sadrži sve piksele od svih uzoraka slova na odgovarajućem indeksu.

Ako korisnik želi izbrisati slovo, to postiže pritiskom na tipku *Clear*. Pritiskom na strelicu u gornjem lijevom kutu, vraća se u glavni izbornik. Za početak treniranja koristi se tipka *Train*.

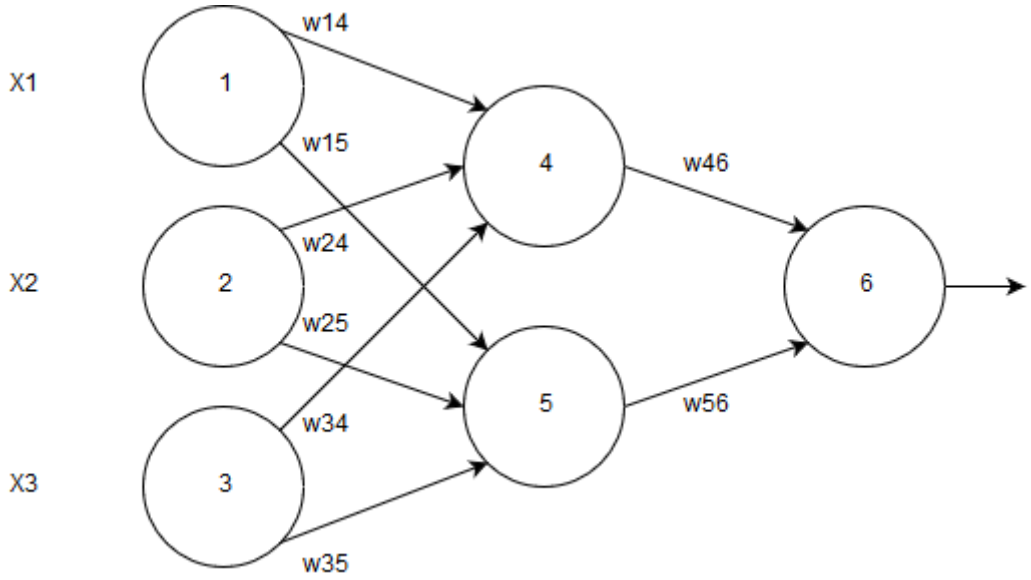


Slika 11. Primjer unosa slova za treniranje mreže

Pritiskom tipke za treniranje, pripremit će se izlazni podaci na temelju izbornika slova. Izlazni podaci bit će u formatu dvodimenzionalnog polja duljine *n*, za svako slovo iz izbornika. Unutarnje polje sadržava odgovarajuću izlaznu vrijednost. Na jednostavnom primjeru treniranja mreže za tri slova *a*, *b* i *c*, izlazne vrijednosti bit će `[[1, 0, 0], [0, 1, 0], [0, 0, 1]]`, redom. Ako u memoriji postoji od prije trenirana mreža, učitat će se sa postojećim težinskim vrijednostima i pomacima, te nastaviti treniranje. Ukoliko ne postoji, inicijalizirat će se nova mreža sa nasumično generiranim vrijednostima. Sa

spremnom neuronskom mrežom, prikazuje se indikator sa postotkom napretka. U pozadini odvija se ranije objašnjeni algoritam propagacije unaprijed i unazad.

Primjer algoritma u praksi možemo predočiti presjekom jednog dijela mreže.



Slika 12. Propagacija unaprijed u višeslojnoj neuronskoj mreži

Tablica 1. Početne ulazne vrijednosti

X1	X2	X3
1	0	1

Tablica 2. Početne težinske vrijednosti

w14	w15	w24	w25	w34	w35	w46	w56
0.2	0.5	0.15	0.9	0.1	0.2	0.4	0.3

Tablica 3. Početne vrijednosti pomaka

Θ4	Θ5	Θ6
0.5	0.5	0.5

Tablica 4. Ulazne i izlazne vrijednosti neurona

Neuron j	**Ukupni ulaz, I_j**	**Izlaz, O_j**
4	1 * 0.2 + 0 * 0.15 + 1 * 0.1 + 0.5 = 0.8	$1 / 1 + e^{-0.8} = 0.68997$
5	1 * 0.5 + 0 * 0.9 + 1 * 0.2 + 0.5 = 1.2	$1 / 1 + e^{-1.2} = 0.76852$
6	0.68997 * 0.4 + 0.76852 * 0.3 + 0.5 = 1.006544	$1 / 1 + e^{-1.006544} = 0.73234$

Tablica 5. Izračun grešaka u neuronima pomoću *delta* pravila

Neuron j	**Greška j**
4	-(1 – 0.73234) * 0.73234 * (1 – 0.73234) * 0.68997 = -0.0362
5	-(1 – 0.73234) * 0.73234 * (1 – 0.73234) * 0.76852 = -0.04032
6	$1 / 2 * (1 – 0.73234)^2 = 0.03582$

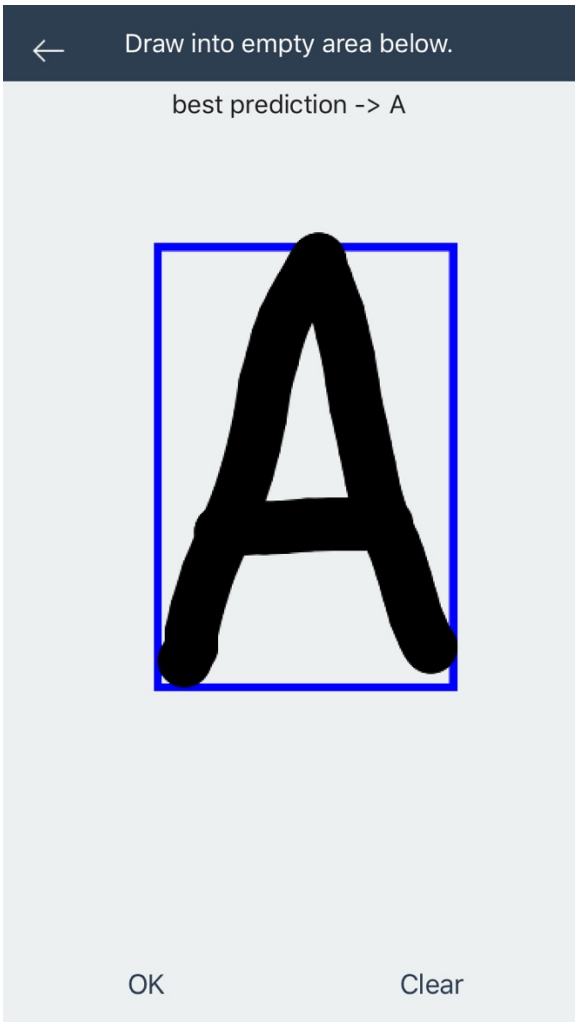
Tablica 6. Promjene u težinskim vrijednostima i vrijednostima pomaka

Težinska ili vrijednost pomaka	**Nova vrijednost**
w46	0.4 + 0.5 * -0.04032 = 0.37984
w56	0.3 + 0.5 * -0.0362 = 0.2819
w14	0.2 + 0.5 * -0.0362 * 1 = 0.1819
w15	0.5 + 0.5 * -0.04032 * 1 = 0.47984
w24	0.15 + 0.5 * -0.0362 * 0 = 0.15
w25	0.9 + 0.5 * -0.04032 * 0 = 0.9
w34	0.1 + 0.5 * -0.0362 * 1 = 0.0819
w35	0.2 + 0.5 * -0.04032 * 1 = 0.17984
Θ4	0.5 + 0.5 * -0.0362 = 0.4819

Θ5	0.5 + 0.5 * -0.04032 = 0.47984
Θ6	0.5 + 0.5 * 0.03582 = 0.51791

4.2.2. Testiranje

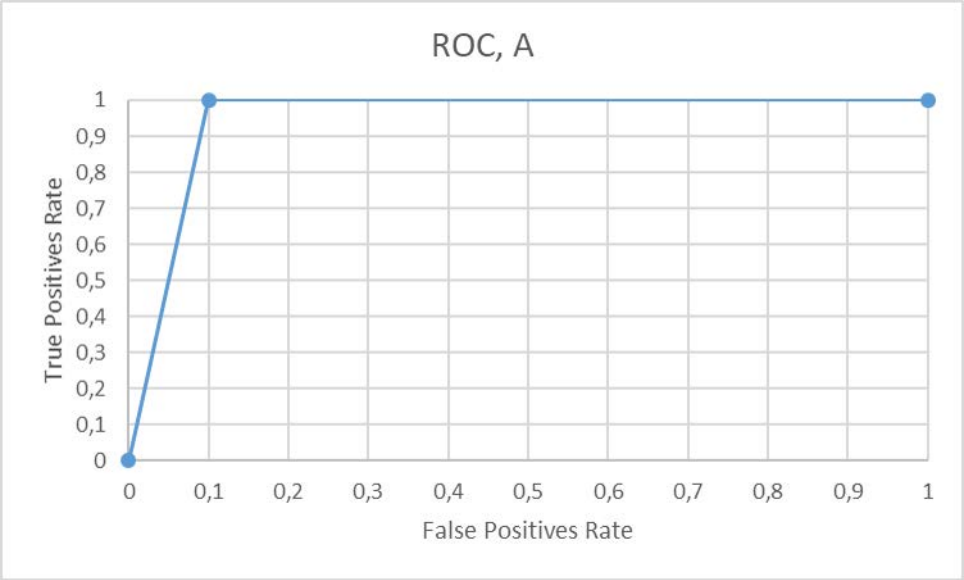
Testiranjem se ponavlja proces rezanja i skaliranja unesenog znaka kako bi se slika pripremila za format ulaznih vrijednosti neuronske mreže. Pritiskom na *OK* učitava se neuronska mreža i ulazne vrijednosti se propagiraju unaprijed. Iz dobivenih izlaza se uzima najviša vrijednost kao klasificirani rezultat i prikazuje u vrhu ekrana. U ovom slučaju izlazne vrijednosti (neuroni) bit će u formatu jednodimenzionalnog polja duljine n, a vjerojatnost na svakom indeksu odgovara indeksu znaka koji se testira. Idealne izlazne vrijednosti za predikciju slova *A* bile bi `[1, 0, 0]`.



Slika 13. Primjer uspješne klasifikacije slova

Učinkovitost klasifikacije može se mjeriti ROC krivoljum (engl. *Receiver Operating Characteristic,* skraćeno ROC). ROC krivulja prikazuje učinkovitost binarne klasifikacije. U višeklasnim sustavima efikasnost mjerimo uspoređivanjem jedne klase u odnosu na sve ostale. Za primjer možemo uzeti klasifikaciju tri slova: *A*, *B* i *C*. Za mjerenje efikasnosti klasifikacije slova *A*, grupirati ćemo *B* i *C* kako bismo dobili binarnu usporedbu *A* i (*B*, *C*). ROC krivulja prikazuje omjer TPR (engl. *True Positive Rate*, skraćeno TPR) i FPR (engl. *False Positive Rate*, skraćeno FPR). TPR računamo kao omjer pogodaka TP (engl. *True Positive,* skraćeno TP) u odnosu na cijeli skup uzoraka klase koju testiramo. Svaki pogodak je jedan TP. FPR računamo kao omjer promašaja FP (engl. *False Positive*, skraćeno FP) u odnosu na skup oprečne klase ili klasa, odnosno koliko puta je klasifikator promašio. Točke na grafu crtamo u odnosu na prag (engl. *threshold*) u rasponu od 0 do 1. Svako od tri slova trenirano je na skupu podataka od 25 uzoraka i testirano na skupu podataka od 20 uzoraka. Testni uzorci sastoje se od 10 uzoraka točne vrijednosti i 10 uzoraka krive vrijednosti.

Uzorak sa krivim vrijednostima sastoji se od dva skupa po 5 uzoraka za svako slovo koje nije u testiranom skupu.



Slika 14. ROC krivulja klasifikacije slova A

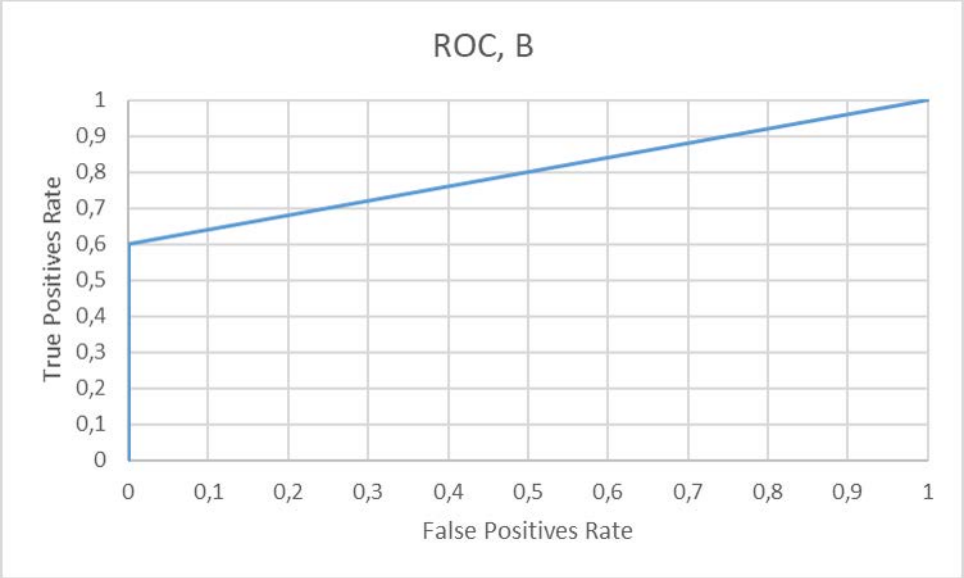
Gornja slika prikazuje ROC krivulju za slovo *A*. Slijedi tablica vjerojatnosti:

Tablica 7. TP i FP vjerojatnosti za slovo A

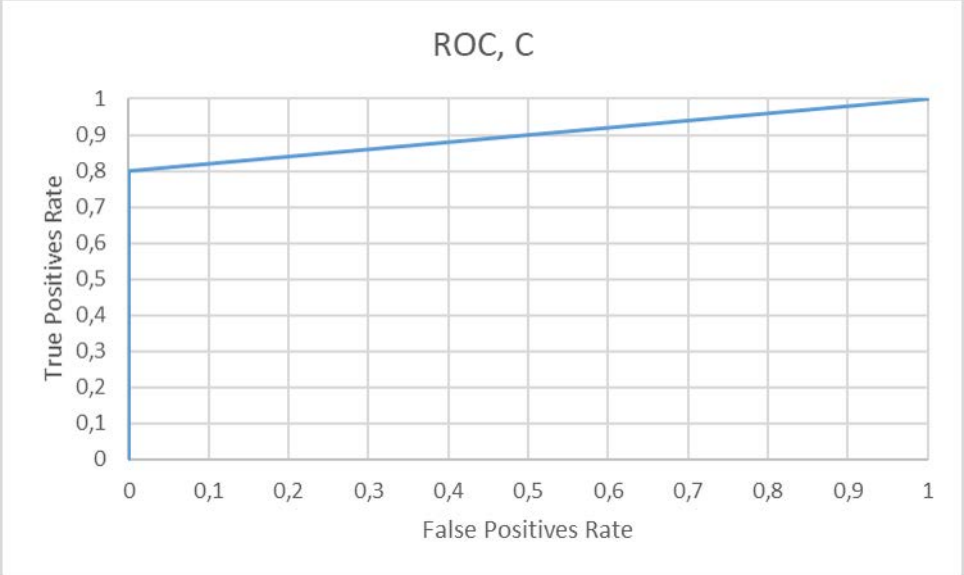
A	B ili C
0.991487622	0.367593884
0.988276184	0.0600785054
0.990563631	$2.15571458 * 10^{-6}$
0.993929505	$7.29445926 * 10^{-9}$
0.78258276	$1.3323897 * 10^{-7}$
0.991627455	$1.86871584 * 10^{-5}$
0.985008836	$1.08059603 * 10^{-5}$
0.96518214	$1.03168163 * 10^{-9}$

0.965712547	$9.641339963 * 10^{-6}$
0.98937875	0.000536001695

Nakon prikupljanja podataka, crtamo krivulju za sve pragove od 0.1 do 0.9, odnosno od 10% do 90%. Isto možemo napraviti i za preostala dva slova.



Slika 15. ROC krivulja kasifikacije slova B



Slika 16. ROC krivulja klasifikacije slova C

4.3. Značajni dijelovi koda

Ovo poglavlje opisuje najznačajnije dijelove koda, grupirane po klasama kojima pripadaju.

4.3.1. Klasa DrawingViewController

U klasi DrawingView controller nalazi se većina koda zaduženog za interakciju s korisnikom putem korisničkog sučelja, ali i metode za pristupanje logici. Jedna takva metoda je

```
func trainButtonPressed() {
// check if network exists
// if yes load and append new values
// train and save network

var outputData: [[Float]] = []
for (index, _) in pickerViewData.enumerated() {
      var outputDataForLetter = Array<Float>(repeating: 0,
count: pickerViewData.count)
      outputDataForLetter[index] = 1
      outputData.append(outputDataForLetter)
}

if let loadedNetwork =
NSKeyedUnarchiver.unarchiveObject(withFile:
NeuralNetwork.ArchiveURL.path) as? NeuralNetwork {
      neuralNetwork = loadedNetwork
}

// Dispatch training on another thread
indicatorView.alpha = 1
indicator.startAnimating()

let epochs = 50000

// Training is done on the main thread because it shouldn't
be interrupted
for iterations in 0..<epochs {
for (character, output) in zip(characterPixelsArray,
outputData) {
```

```
            for i in 0..<character.count {
                  neuralNetwork.trainWith(inputs: character[i],
targetOutput: output)
            }

let percent: CGFloat = CGFloat(iterations) / CGFloat(epochs)
* 100.0
indicatorLabel.text = ("Training: \(percent)%")
                  }
            }

indicator.stopAnimating()
indicatorView.alpha = 0

NSKeyedArchiver.archiveRootObject(neuralNetwork, toFile:
NeuralNetwork.ArchiveURL.path)
}
```

Kôd 1. Metoda koja za izvršava pritiskom na gumb "Train"

Kao što samo ime kaže, metoda se poziva pri početku treniranja neuronske mreže. Prvo se stvore izlazni podaci na temelju odabranog slova iz izbornika, zatim se provjerava postoji li spremljena neuronska mreža u memoriji. Ako postoji, učitava se i treniranje će početi od prethodnih težinskih vrijednosti što omogućuje iterativno treniranje. Prikazuje se indikator napretka i treniranje mreže započinje. Korisnika se postotkom obavještava o napretku. Po završetku treniranja metoda gasi indikator i sprema neuronsku mrežu u memoriju.

Zatim metoda koja služi za prikaz klasifikacije napisanog znaka:

```
fileprivate func bestEstimateFor(prediction: [Float]) {

var maxValue: Float = 0
var predictedLetter = ""

for (prediction, letter) in zip(prediction, pickerViewData) {
      if prediction > maxValue {
            maxValue = prediction
            predictedLetter = letter
      }
}
```

```
let predictionLabel = UILabel()
predictionLabel.text = "best prediction ->
\(predictedLetter)"
predictionLabel.alpha = 0
predictionLabel.textAlignment = .center
view.addSubview(predictionLabel)
view.addConstraintsWithFormat(format: "H:|[v0]|", views:
predictionLabel)
view.addConstraintsWithFormat(format: "V:|-15-[v0(100)]",
views: predictionLabel)

UIView.animate(withDuration: 0.2, animations: {
      predictionLabel.alpha = 1
}) { finished in
      UIView.animate(withDuration: 0.6, delay: 0.2, options:
[], animations: {
            predictionLabel.alpha = 0
            }, completion: { finished in
                  predictionLabel.removeFromSuperview()
            })
      }
}
```

Kôd 2. Metoda koja za izvršava nakon klasifikacije upisanog znaka

Ova metoda na izlazu iz mreže iz polja uzima najveću vrijednost i prikazuje ju kao najbolju estimaciju na vrhu ekrana.

Vrlo važna metoda za pripremu podataka je:

```
fileprivate func imagePixels() -> [Int] {
guard
      let croppedImage =
      drawingImageView.image?.cropImageWith(rect:
      characterBoxFrame),
      let scaledImage = croppedImage.scaleImageTo(size:
      scaledImageSize)
else {
      return []
      }
```

```
        return presenter.pixelize(image: scaledImage)
}
```

Kôd 3. Metoda za pripremu podataka kojom se slika pretvara u polje piksela

Slika se prvo reže na predefiniranu veličinu kako bi se riješili suvišnih bjelina. Za to koristimo kvadrat koji crtamo oko znaka za njegove najudaljenije točke. Također skaliramo sliku kako bismo smanjili broj ulaza na mreži. Sa skaliranjem treba biti oprezan, jer kvaliteta slika postepeno pada te se mogu izgubiti neke vrijednosti. Na kraju se skalirana slika pretvara u polje piksela u metodi koja će biti obrađena u idućem poglavlju.

4.3.2. Klasa DrawingPresenter

Presenter je klasa koja za *ViewController* obrađuje podatke. Jedna od metoda čiju logiku *DrawingViewController* delegira svom *Presenteru* je

```
func pixelize(image: UIImage) -> [Int] {
let pixelData = image.cgImage?.dataProvider?.data
let dataPointer: UnsafePointer<UInt8> =
CFDataGetBytePtr(pixelData)
var pixelsArray = [Int]()
let bytesPerRow = image.cgImage?.bytesPerRow
let bytesPerPixel = ((image.cgImage?.bitsPerPixel)! / 8)
var position = 0
for _ in 0..<Int(image.size.height) {
      for _ in 0..<Int(image.size.width) {
            let alpha = Float(dataPointer[position + 3]) //
[r,g,b,a]
            pixelsArray.append(Int(alpha / 255))
            position += Int(bytesPerPixel)
      }
      if position % Int(bytesPerRow!) != 0 {
            position += (Int(bytesPerRow!) - (position %
            Int(bytesPerRow!)))
      }
}

return pixelsArray
}
```

Kôd 4. Metoda za pretvaranje slike u numeričko polje na temelju piksela

jedna od glavnih metoda za pripremu podataka. Pokazivačem se iterira kroz bajtove po visini i širini. Za svaki piksel provjerava se prozirnost. U slučaju da je u trenutnom pikselu dio iscrtanog znaka, prozirnos ima vrijednost 255. Prije spremanja u polje, vrijednost dijelimo sa 255 kako bi za obojane piksele dobili jedinicu, a za neobojane nulu.

4.3.3. Klasa UIImageExtensions

Koristeći ekstenzije, postojećim klasama čijoj implementaciji nemamo pristup, možemo dodati nove funkcionalnosti. Klasu *UIImage*, čiji objekt će sadržavati sliku upisanog znaka proširujemo sa metodama `cropImageWith(rect: CGRect) -> UIImage?` za rezanje i `scaleImageTo(size: CGSize) -> UIImage?` za skaliranje. Prvo ćemo proučiti metodu pomoću koje se slika reže.

```
func cropImageWith(rect: CGRect) -> UIImage? {
guard let imageRef = self.cgImage?.cropping(to: rect) else {
      return nil }
return UIImage(cgImage: imageRef)
}
```

Kôd 5. Metoda za rezanje slike

Pomoću ove metode rješava se problem skaliranja velikih i malih slova. Metoda prima kvadrat koji opisuje slovo i odstranjuje nepotrebne bjeline.

Slijedi metoda za skaliranje:

```
func scaleImageTo(size: CGSize) -> UIImage? {
UIGraphicsBeginImageContextWithOptions(size, false, 1.0)
guard let context = UIGraphicsGetCurrentContext() else {
return nil
}

// kCGInterpolationNone - There is no smoothing of pixels
after scaling (the image is not blurred)
// which is important for pixelizing the image for neural
network input

context.interpolationQuality = .none
self.draw(in: CGRect(origin: CGPoint.zero, size: size))
let imageRef = (context.makeImage())! as CoreImage.CGImage
```

```
let newImage = UIImage(cgImage: imageRef, scale: 1.0,
orientation: UIImageOrientation.up)
UIGraphicsEndImageContext()

return newImage
}
```

Kôd 6. Metoda za skaliranje

Ova metoda prima objekt *CGSize* koji sadržava visinu i širinu po kojoj se slika treba skalirati. U ovom odlomku najbitnije je postaviti

```
context.interpolationQuality = .none
```

što će onemogućiti interpolaciju između piksela. To znači da će slika imati oštre rubove, a upravo to nam treba kako bismo što bolje istrenirali mrežu za oblik slova.

4.3.4. Klasa NeuralNetwork

Klasa *NeuralNetwork* je klasa koja pruža sučelje za treniranje i korištenje neuronske mreže. Korisnik koji želi napraviti aplikaciju koja koristi neuronsku mrežu ovoj klasi će predati podatke za treniranje, a isto tako će i testirati mrežu preko metoda sučelja.

Korištenje je vrlo jednostavno, a izložene su dvije metode:

```
func trainWith(inputs: [Float], targetOutput: [Float], learningRate: Float
= 0.3)
```

koja služi za treniranje, a prima polje ulaznih vrijednosti, polje izlaznih vrijednosti po kojima se korigira greška i *learning rate.* Learning rate je stopa učenja kojom se određuje korak kojim mreža uči. Nema jednog idealnog broja, već se bira proizvoljno iz iskustva po rezultatima testiranja. Obično se uzima mali broj kako krivulja ne bi izašla iz minimuma. Druga metoda je `func predictFor(inputs: [Float]) -> [Float]`, a prima samo polje ulaznih vrijednosti i vraća polje izlaznih. Najveća vrijednost u polju izlaznih vrijednosti je slovo s najvećom vjerojatnosti.

Detaljnije se može vidjeti iz implementacija navedenih metoda.

```
func trainWith(inputs: [Float], targetOutput: [Float],
learningRate: Float = 0.3) {

if !inputs.isEmpty {
// forward
let networkOutput = forwardPropagate(inputs: inputs)
```

```
// backward
let outputError = zip(targetOutput, networkOutput).map { $0 -
$1 }
backPropagate(error: outputError, inputs: inputs,
learningRate: learningRate)
    }
}
```

Kôd 7. Metoda za treniranje, sučelja neuronske mreže

Kao što se vidi iz dijagrama toka, postoji grananje aplikacijske logike u treniranje i testiranje, odnosno propagaciju unazad i propagaciju unaprijed. Treniranje obuhvaća oba algoritma, redom.

```
private func forwardPropagate(inputs: [Float]) -> [Float] {
var mutableInput = inputs
for layer in layers {
      mutableInput = layer.forwardPropagate(inputs:
mutableInput)
      }
      return mutableInput
}
```

Kôd 8. Metoda neuronske mreže za propagaciju unaprijed

Na razini mreže propagacija unaprijed delegira se u slojeve mreže, kaskadno iz sloja u sloj i na kraju vraća kao izlazna vrijednost.

```
private func backPropagate(error: [Float], inputs: [Float],
learningRate: Float) {
var mutableError = error

for layer in layers.reversed() {
mutableError = layer.backPropagate(error: mutableError,
inputs: inputs, learningRate: learningRate)
      }
}
```

Kôd 9. Metoda neuronske mreže za propagaciju unazad

Slično propagaciji unaprijed, propagacija unazad delegirana je u slojeve mreže, kaskadno po slojevima s greškom kao ulaznim parametrom. Druga metoda sučelja neuronske mreže,

metoda za propagaciju unaprijed poziva spomenutu metodu `private func forwardPropagate(inputs: [Float]) -> [Float]`.

4.3.5. Klasa Layer

Klasa *Layer* predstavlja jedan sloj neuronske mreže. Njezin zadatak je većinska implementacija algoritma učenja (dio je implementiran u klasi *Neuron,* koja će biti objašnjena u kasnijem poglavlju), ali i instanciranje neurona i nasumičnih težinskih vrijednosti. Cijela priprema se vrši u inicijalizacijskoj metodi

```
init(inputSize: Int, numberOfNeurons: Int) {

neurons = []
self.inputSize = inputSize

let sign = Float(arc4random_uniform(3) > 1 ? 1 : -1)
let bias = Float(arc4random_uniform(2))

for _ in 0..<numberOfNeurons {

      var weights: [Float] = []
      for _ in 0..<inputSize {
            weights.append(Float(arc4random_uniform(2)) *
sign)
      }

      neurons.append(Neuron(weights: weights, bias: bias))
      }
}
```

Kôd 10. Inicijalizacijska metoda sloja neuronske mreže

Može se vidjeti da se inicijaliziraju početne vrijednosti, kao i neuroni koji u svojim incijalizacijskim metodama primaju te vrijednosti kao ulazne parametre. Bitno je napomenuti da se nove vrijednosti inicijaliziraju samo u slučaju da već ne postoji neuronska mreža. U tom slučaju vrijednosti će se učitati iz memorije. Na taj način možemo spremiti djelomično treniranu mrežu i naknadno nastaviti treniranje sa novim podacima. Propagacija unaprijed je i u sloju delegirana prema nižim komponentama, odnosno u neuron, kao što možemo vidjeti iz implementacije:

```
func forwardPropagate(inputs: [Float]) -> [Float] {
```

```
var outputs: [Float] = []
for neuron in neurons {
      outputs.append(neuron.activate(inputs: inputs))
      }
      return outputs
}
```

Kôd 11. Metoda sloja za propagaciju unaprijed

Slijedi implementacija metode za propagaciju unazad. Ulazni parametri su greške iz prošlog sloja ili sa izlaza, ulazne vrijednosti u sloju i stopa učenja spomenuta u opisu implementacije metode za propagaciju unazad u sklopu neuronske mreže.

```
func backPropagate(error: [Float], inputs: [Float],
learningRate: Float) -> [Float] {

var errorsToPropagate = [Float](repeating: 0, count:
inputSize)

for (indexN, neuron) in neurons.enumerated() {

let delta = neuron.deltaFor(error: error[indexN])

for (indexW, weight) in neuron.weights.enumerated() {
errorsToPropagate[indexW] = errorsToPropagate[indexW] +
weight * delta
let deltaWeight = inputs[indexW] * delta * learningRate
neuron.weights[indexW] = neuron.weights[indexW] +
neuron.storedWeights[indexW] * neuron.bias + deltaWeight
neuron.storedWeights[indexW] = deltaWeight
            }
      }

      return errorsToPropagate
}
```

Kôd 12. Metoda sloja za propagaciju unazad

4.3.6. Klasa Neuron

Svaki neuron sastoji se od svojih težinskih vrijednosti koje odgovaraju broju ulaza iz prethodnog sloja. Također čuva vrijednost pomaka, izlaznu vrijednost i posebnu kolekciju

težinskih vrijednosti koji se koriste prilikom korekcije originalnih težinskih vrijednosti u algoritmu propagacije unazad. U nastavku bit će opisane metode klase *Neuron* počevši sa metodom `func activate(inputs: [Float]) -> Float` koja za kolekciju ulaznih vrijednosti vraća izlaznu ili aktivacijsku vrijednost. Implementacija je opisana u nastavku.

```
func activate(inputs: [Float]) -> Float {

for (input, weight) in zip(inputs, weights) {

output = output + (input * weight)

}

output = output + bias

output = sigmoid(x: output)

return output

}
```

Kôd 13. Metoda za aktivaciju

Prvo se uparuju težinske i ulazne vrijednosti koje se redom množe i prikupljaju u varijabli *output*. Potom se rezultatu pridružuje vrijednost pomaka, a zatim se prosljeđuje u sigmoidnu funkciju.

```
private func sigmoid(x: Float) -> Float {

return 1 / (1 + exp(-x))

}
```

Kôd 14. Sigmoidna funkcija

Sigmoidna funkcija izlaznu vrijednost skalira na raspon od 0 do 1. Kao što se vidi iz implemetacije algoritma propagacije unazad, neuron također sadrži metode `deltaFor(error: Float) -> Float` i `private func sigmoidDerivative(x: Float) -> Float`. Derivacije sigmoidne funkcije može se svesti na $x * (1 - x)$ kao što se vidi u implementaciji ispod:

```
private func sigmoidDerivative(x: Float) -> Float {

return x * (1 - x)

}
```

Kôd 15. Implementacija derivacije sigmoidne funkcije

Zaključak

U radu je obrađena tema korištenja neuronskih mreža u iOS aplikacijama. Mreža i demo aplikacija implementirani su bez korištenja vanjskih biblioteka. Dokazano je da je na mobilnom uređaju moguće implementirati, trenirati i koristiti neuronsku mrežu za prepoznavanje uzoraka što se može koristiti u različite svrhe kao što su pisanje prstom ili analiza načina korištenja aplikacije što otvara mogućnosti za nova kreativna rješenja u poboljšanju korisničkog iskustva ili prilagodbi aplikacija osobama s posebnim potrebama.

Najteži dio cijelog procesa je priprema podataka, jer za to ne postoji algoritam. U aplikaciji je količina ulaza smanjena skaliranjem slike slova, zbog ograničenih resursa. Za primjere u tekstu, mreža je trenirana sa uzorcima od tri slova sa dvadesetak uzoraka za svako slovo. Za cijelu abecedu potreban je mnogo veći set podataka za treniranje po slovu da bi se postigla što bolja generalizacija. Treniranje takvog seta može potrajati i nekoliko dana, no mrežu je potrebno trenirati samo jednom, a za testiranje mogu se koristiti manji uzorci. Također, kôd je u budućnosti moguće dodatno optimizirati za bolje performanse.

Popis kratica

ROC	*Receiver Operating Characteristic*	krivulja osjetljivosti
TPR	*True Positive Rate*	stopa pogodaka
FPR	*False Positive Rate*	stopa promašaja
TP	*True Positive*	pogodak
FP	*True Positive Rate*	promašaj

Popis slika

Popis tablica

.

Popis kôdova

Literatura

[1] REINGOLD, E., Artificial Neural Networks Technology, University Of Toronto, http://www.psych.utoronto.ca/users/reingold/courses/ai/cache/neural2.html.

[2] MINSKY, M.; PAPERT, S., *Perceptrons: an introduction to computational geometry*, The MIT Press, Cambridge MA, 1972.

[3] LONČARIĆ, S., *Neuronske mreže: Uvod*, Fakultet elektrotehnike i računarstva.

[4] MAZUR, M., *A Step by Step Backpropagation Example, https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/* 62.

[5] NIELSEN, M.A., *Neural networks and deep learning*, Determination Press, 2015.