Učenje i interaktivna analiza neuronske mreže

Fabek, Matija

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Matija Fabek

UČENJE I INTERAKTIVNA ANALIZA NEURONSKE MREŽE

Diplomski rad

Voditelj rada:
doc. dr. sc. Matej Mihelčić

Zagreb, travanj, 2023

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Sadržaj **iii**

Uvod **2**

1 Uvod u tehnologije **3**
1.1 Angular i TypeScript 3
1.2 Klijentske biblioteke 6
1.3 Spring programski okvir 8

2 Neuronske mreže **11**
2.1 Uvod u neuronske mreže 11
2.2 Treniranje neuronske mreže 17
2.3 Prednosti i nedostaci neuronske mreže 20
2.4 Upotreba neuronskih mreža 20

3 Implementacija softvera **23**
3.1 Implementacija klijentskog dijela softvera 24
3.2 Implementacija serverskog dijela softvera 25

4 Primjer uporabe **29**

Bibliografija **35**

Uvod

U ovom diplomskom radu sam napravio aplikaciju koja trenira i analizira potpuno povezanu neuronsku mrežu feed-forward tipa nad proizvoljnim numeričkim podacima. Neuronske mreže pripadaju području strojnog učenja, te se na njima temelje algoritmi dubokog učenja, grane strojnog učenja. Njihov rad i struktura su inspirirani ljudskim mozgom, oponašajući način na koji biološki neuroni šalju signale jedan drugome. Sastoje se od neurona koji su raspoređeni u slojeve, te međusobno povezani. Neuronske mreže uče na prije pripremljenim podacima za treniranje, te, među ostalim, omogućuju klasifikaciju i klasteriranje podataka. Klasifikacijom se određuje kojoj klasi pripada ulazni podatak, dok se klasteriranjem na većem skupu podataka grupiraju podaci prema sličnim karakteristikama u grupe odnosno klastere.

Danas su jako popularne u svijetu računarstva, te su najviše zastupljene u sljedećim područjima:

- Računalni vid - područje umjetne inteligencije koje trenira računala da procesiraju slike, neki od primjera su sljedeći:
  - vizualno prepoznavanje u autonomnim vozilima kojim oni mogu prepoznati znakove i ostale okolnosti na cestama
  - cenzuriranje fotografija - prepoznavaju i uklanjaju neprikladne sadržaje na fotografijama
  - identificiranje ljudskog lica, dijelova lica kao što su oči, naočale
- Prepoznavanje govora - koriste se za prepoznavanje govora kod čovjeka usprkos naglasku, tonu glasa, te jeziku. Jedan svjetski poznati alat koji se bazira na prepoznavanju ljudskog glasa i tako djeluje je Amazon Alexa
- Preporuke - praćenje aktivnosti korisnika na temelju čega se rade preporuke. Tipičan primjer bi bio web aplikacija za prodaju odjeće koja na temelju kupljenih proizvoda i posjećenih artikala radi pretpostavke za sljedeću kupnju

- Obrada prirodnog jezika (engl. *Natural language processing*) - grana umjetne inteligencije u kojoj se uče računala da razumiju tekst slično kao što ga i ljudi razumiju. Neka područja gdje se koriste su:
  - pretraživanje podataka
  - detekcija neželjenog sadržaja (engl. *spam*)
  - kategorizacija pošte

Rad sam podijelio u tri poglavlja. U prvom ću poglavlju opisati tehnologije i alate koje sam koristio kako bi implementirao ovu aplikaciju, u drugom poglavlju ću opisati neuronske mreže, njihovu strukturu i u kojim su industrijama najzastupljenije, a u trećem ću poglavlju dokumentirati aplikaciju, proći kroz korake njenog razvoja, te kroz primjere njenog korištenja.

Poglavlje 1

Uvod u tehnologije

Kratak uvod

U ovom radu sam napravio web aplikaciju za učenje i interaktivnu analizu neuronske mreže. Web aplikacija se sastoji od klijentskog dijela i serverskog dijela. Klijentski dio sam implementirao koristeći Angular programski okvir s jezikom Typescript, a serverski dio je napravljen u Javi 11 koristeći Spring Boot programski okvir. Komunikacija između Angular aplikacije i Spring Boot aplikacije postignuta je pomoću REST API-ja i HTTP protokola.

1.1 Angular i TypeScript

Programski okvir Angular ([6]) je uz React ([24]) i Vue ([32]) jedan od najpoznatijih i najkorištenijih programskih okvira za izradu klijentskog dijela klijent-server web aplikacije. Angular je razvijen od strane Google-a, te je potekao iz AngularJS-a, odnosno samo ga je zamijenio. Danas je najaktualnija verzija Angular 15 ([5]) koja je puštena u produkciju 18. studenog 2022. Po službenim podacima, trenutno je oko 1.7 milijuna korisnika Angulara. Napisan je u TypeScriptu ([31]), programskom jeziku otvorenog koda osnovanog od strane Microsofta. TypeScript ima iste funkcionalnosti kao i JavaScript, te se kod napisan u TypeScriptu kompajlira u JavaScript. Najbitnija razlika između ova dva jezika je što kod TypeScripta mora eksplicitno biti zadan tip dok kod JavaScripta ne mora. Koristeći TypeScript je lakše uočiti greške jer prevodioc provjerava ispravnost tipova. U TypeScriptu je moguće kreirati apstraktnu klasu i koristiti ključne riječi private i public čega nema u JavaScriptu. Na slikama 1.1 i 1.2 možemo primjetiti pogreške koje javlja prevodioc u TypeScriptu jer funkcija add prima dva parametra tipa number, dok u gotovo analognom kodu napisanom u JavaScriptu prevodioc ne javlja pogrešku jer parametri mogu biti proizvoljnog tipa.

```
1  const add = (x: number, y: number): number => x + y;
2
3  add('1', '1'); //compiler error
4  add(1, '1'); // compiler error
5  add(1, 1); // 2
```

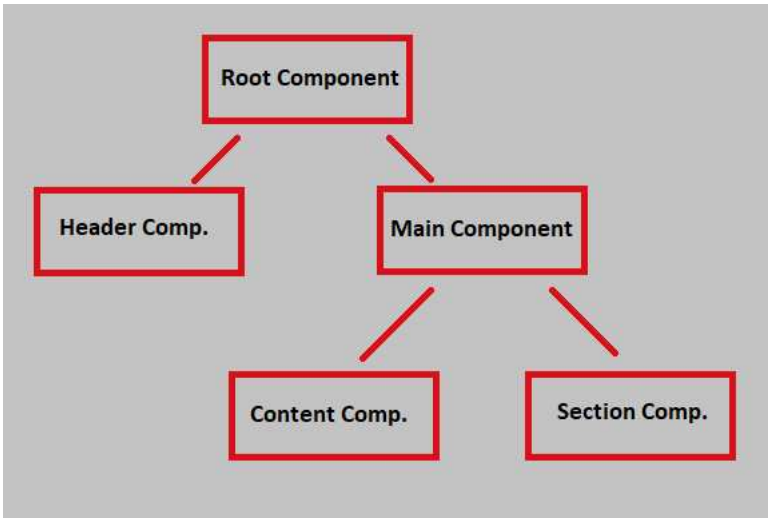
Slika 1.1: Primjer TypeScript koda

```
1  const add = (x, y) => x + y;
2
3  add('1', '1'); // 11
4  add(1, '1'); // 11
5  add(1, 1); // 2
```

Slika 1.2: Primjer JavaScript koda

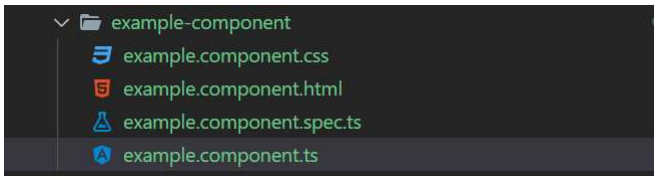
Angular aplikacije, kao i React i Vue, su single-page aplikacije, što znači da se sastoje od samo jedne stranice, jednog html dokumenta čiji se sadržaj dinamički mijenja po potrebi korisnika. Prednosti SPA (Single Page Application [25]) su te što korisnici ne moraju dohvaćati cijele stranice sa servera, nego samo dohvate podatke ako su potrebni, pri čemu smanjujemo latenciju, odnosno vrijeme potrebno da aplikacija reagira na korisnikovu akciju. Nedostatak toga je što moramo sami implementirati navigiranje, ali zato Angular dolazi s paketom @angular/router koji upravo to omogućuje. Uz spomenuti @angular/router, neki od poznatijih Angular paketa su: @angular/forms za rad s formama, @angular/animations za rad s animacijama, @angular/common/http za dohvaćanje http klijenta s kojim radimo pozive prema serveru i ostalim servisima.

Također, Angular aplikacija se bazira na komponentama, odnosno aplikaciju gradimo kao stablo komponenata čiji je korijen osnovna komponenta u koju su njena djeca uključena pa onda djeca od djece i tako dalje. U donjem primjeru jasno vidimo strukturu jedne obične Angular aplikacije čija je korijenska komponenta Root komponenta, te ona sadrži Header komponentu i Main komponentu, te su na kraju u Main komponentu ubačene Content i Section komponente.



Slika 1.3: Primjer strukture Angular aplikacije

Svaka Angular komponenta se sastoji od najmanje dva dijela, a to su html datoteka, koja predstavlja vizualni dio komponente, te typescript datoteka koja predstavlja logičku pozadinu komponente, odnosno dohvaća i radi bilo kakve promjene na podacima potrebnim za tu komponentu. Također, komponenta još može sadržavati dvije ne nužne datoteke, a to su typescript datoteka koja sadrži testove za tu komponentu, te css datoteka koja sadrži stilove za gore napomenutu html datoteku.



Slika 1.4: Primjer strukture Angular komponente

Uz komponente još postoje i moduli kojih mora biti najmanje jedan. Modul je typescript datoteka u kojoj definiramo koje ćemo komponente koristiti u tom modulu koje mogu biti definirane u drugim modulima ili ih možemo izvlačiti iz nekih od instaliranih ovisnosti, odnosno paketa potrebnih za rad aplikacije.

Za lokalno programiranje ([4]) potrebno je instalirati Node.js ([23]) s kojim dolazi node package manager, odnosno npm, koji olakšava instaliranje novih ovisnosti, brisanje starih

ovisnosti, te se brine o tome jesu li verzije kompatibilne s verzijama ostalih paketa. Nakon toga, potrebno je instalirati Angular CLI ([3]), alat za kreiranje Angular aplikacija, kreiranje Angular biblioteka, te stvaranje komponenata. Također, sadrži server na kojem možemo lokalno pokrenuti našu aplikaciju, te koji ponovno pokrene aplikaciju čim dođe do promjene u datotekama. Server se pokrene naredbom za pokretanje aplikacije (*ng serve*), te se tada automatski postavi aplikacija na njega. Angular CLI možemo instalirati sljedećom naredbom, te s naredbom ispod kreirati novu Angular aplikaciju.

```
npm install -g @angular/cli
ng new my-app
```

Nadalje, potrebno je instalirati sve pakete koji su navedeni u package.json datoteci, pod objektom "dependencies". To je datoteka koja vodi brigu o svim ovisnostima koje naša Angular aplikacija koristi, o nazivu i verziji aplikacije, te o skriptama za pokretanje aplikacije, buildanje i pokretanje testova. Svaka Angular aplikacija mora sadržavati package.json datoteku. Nakon što instaliramo sve potrebne ovisnosti za našu aplikaciju, kreira se node_modules mapa u kojoj se nalaze sve potrebne ovisnosti, te nakon toga možemo pokrenuti aplikaciju. To možemo napraviti sljedećim dvjema naredbama koje pokrenemo u korijenskoj mapi naše aplikacije.

```
npm install
ng serve
```

1.2 Klijentske biblioteke

S Angular-om možemo koristiti razne bliblioteke kako bi si ubrzali razvoj aplikacije jer one dolaze s već implementiranim komponentama pa ih ne moramo implementirati iz početka. Takve biblioteke naravno možemo koristiti i s React-om i Vue-om. Neke od najpoznatih biblioteka su Ant Design ([7]) i Material UI ([19]).

Ant Design i Material UI

Ant Design je biblioteka otvorenog koda koju je proizvela Ant Group tvrtka i podržava najpoznatije programske okvire za klijentski dio web aplikacije kao što su React, Angular i Vue. Prvenstveno je napravljena za React programski okvir, ali napravljene su njene inačice koje su prilagođene za Angular i Vue. Tako za Angular postoji NG Zorro ([22]), što je ustvari Ant Design za Angular aplikacije, te Ant Design Vue za Vue aplikacije.

Sve one dolaze s gotovim UI komponentama koje samo ubacimo u naš kod umjesto da ih implementiramo iz početka. Neke od takvih komponenata su: input komponenta, gumb, tablica, kalendar i još mnoge druge. Kako bi koristili NG Zorro, moramo dodati ovisnost ng-zorro-antd u objekt zvan dependencies koji se nalazi u package.json datoteci (vidi sliku 1.5), a ona se nalazi u korijenskoj mapi naše aplikacije.

Slika 1.5: Dodavanje Ant Design ovisnosti

Material UI je također biblioteka otvorenog koda koja implementira Google-ov Material Design ([18]), te je na njoj radilo preko 2200 različitih tvrtki odnosno programera. Kao i Ant Design, dolazi opremljen s raznim komponentama za razvoj sučelja kod klijentskih aplikacija, te je kao i Ant Design prvenstveno napravljen za React, ali također postoje implementacije za Angular, Angular Material UI, te Vuetify za Vue programski okvir.

D3 biblioteka

Za vizualizaciju neuronske mreže koristio sam D3.js biblioteku ([9]). D3 je Javascript biblioteka koja se koristi za kreiranje raznih vizualizacija na proizvoljnim podacima. Koristi HTML ([15]), SVG ([30]) i CSS ([8]). SVG je kratica za Scalable Vector Graphics, odnosno format za slike baziran na XML-u ([34]). Pomoću D3 biblioteke možemo kreirati grafove, dijagrame, mreže sa dinamičkim podacima. Kako bi koristili D3 biblioteku dovoljno je samo staviti sljedeći fragment koda u head element od index.html datoteke.

```
<script src="https://d3js.org/d3.v7.min.js"></script>
```

1.3 Spring programski okvir

Spring framework ([28]) je programski okvir otvorenog koda za razvoj aplikacija baziran na programskom jeziku Java, te se najčešće koristi kao serverska strana web aplikacije. Prvu verziju Springa napisao je Rod Johnson 2002. godine koja je izdana 2003. godine u lipnju pod vlasništvom Apache-a. Nedavno nakon tog izdana je prva produkcijska verzija, odnosno Spring framework 1.0 2004. godine u ožujku ([14]). Trenutno je najnovija stabilna verzija Spring programskog okvira 6.0 izdana u prosincu 2022. godine koja, za razliku od Spring-a 5, zahtjeva korištenje Jave 17 ([17]) ili više.
Osnovne karakteristike Spring programskog okvira su:

- Pojednostavljuje razvoj i testiranje Java aplikacija i omogućava korištenje labave sprege među komponentama (engl. *loose coupling*)
- POJO klase (*Plain Old Java Objects*), odnosno jednostavne Java klase koje nemaju ovisnosti o Spring programskom okviru pa se mogu jednostavno iskoristiti u drugim softverima gdje se ne koristi Spring
- Spring kontejner (engl. *Spring container*) - srž Spring programskog okvira, kreira objekte, povezuje ih, te brine za njih tijekom cijelog životnog ciklusa ([16])
- Injektiranje ovisnosti (engl. *Dependency injection*) - mehanizam Springa koji omogućava da objekti ne moraju kreirati instance drugih objekata o kojima ovise nego ih Spring kontejner ubaci putem konstruktora, seter metoda ili varijabli. Na taj način je moguće imati samo jednu instancu (engl. *singleton*) objekta o kojem drugi ovise jer ga ovako Spring kontejner ubacuje na ostala mjesta
- Aspektno orijentirano programiranje - programska paradigma koja povećava modularnost odvajanjem funkcionalnosti koje su zajedničke za različite dijelove aplikacije. Primjeri takvih funkcionalnosti su logiranje, sigurnost aplikacije
- Spring predložci - jedan od osnovnih dijelova Spring programskog okvira, unaprijed pripremljene funkcionalnosti koje nas liše pisanja istog (engl. *boilerplate*) koda svaki put iznova. Primjer Spring predložaka bi bio pristup bazi podataka pri čemu je potrebno kreirati i zatvoriti vezu s bazom podataka, inicijalizirati resurse i ostalo

Spring Boot

Spring Boot ([27]) je modul Spring programskog okvira koji olakšava i ubrzava izgradnju Spring web aplikacije s minimalnom konfiguracijom. Trenutno je najnovija verzija Spring Boot-a 3.0.4 izdana u prosincu 2022. godine, koja je bazirana na verziji Spring-a 6.0, te

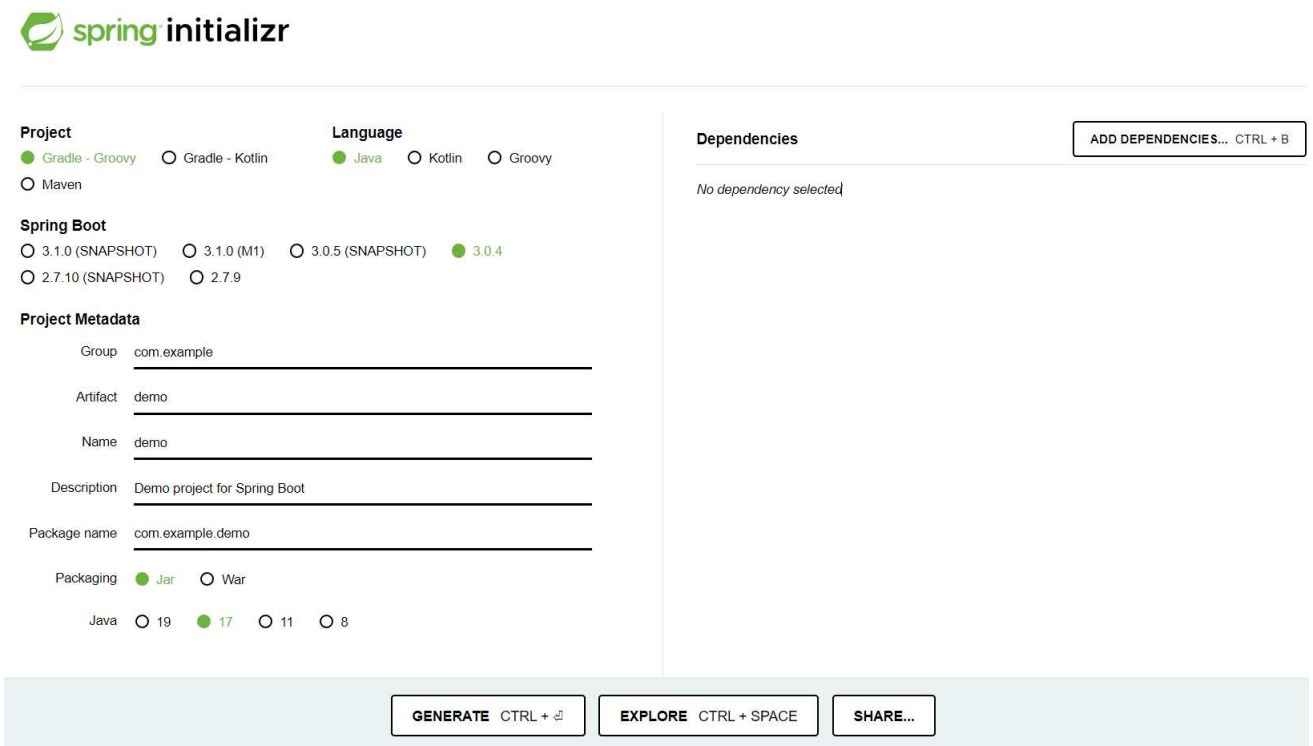
kao i Spring 6.0 zahtjeva korištenje Jave 17 ili više.
Karakteristike Spring Boot-a:

- Stvaranje samostalnih Spring aplikacija
- Ugradi Tomcat, Jetty ili Undertow web server (engl. *embedded server*) u jar datoteku naše aplikacije i kad se pokrene jar datoteka pokrene se server na kojem se automatski pokrene aplikacija. U suprotnom bi trebali instalirati jedan od tih web servera lokalno, te ručno prebaciti jar datoteku naše aplikacije na njega i pokrenuti je, a ovako samo treba pokrenuti jar datoteku kao normalnu java aplikaciju ([10])
- Nema potrebe za XML konfiguracijom
- Omogućuje monitoriranje aplikacije pomoću alata Spring Boot Actuator ([26])

Koristan alat kod kreiranje Spring Boot aplikacije je Spring initializr ([29]) s kojim možemo brzo kreirati novi Spring Boot projekt sa svim potrebnim ovisnostima (slika 1.6). Sa Spring initializr-om potrebno je odabrati alat za izgradnju aplikacije, odnosno maven ([20]) ili gradle ([13]). To su automatizirani alati za izgradnju softvera koji olakšavaju dodavanje novih ovisnosti. Također, treba odabrati jezik koji ćemo koristiti, verziju Spring Boot-a, verziju Jave, u kojem obliku će biti aplikacija kada ju izgradimo (jar datoteka ili war datoteka), te navesti ime aplikacije. Neobavezan dio je dodavanje ovisnosti (*starter dependencies*), a neke od njih su Spring Web koji se koristi za kreiranje web aplikacije, Spring Security za osiguravanje aplikacije, Spring Boot Actuator za monitoriranje aplikacije i mnogi drugi.

Weka

Weka (*Waikato Environment for Knowledge Analysis* [33]) je softver otvorenog koda napisan u jeziku Java koji sadrži algoritme strojnog učenja za dubinsku analizu podataka (engl. *data mining*). Nad podacima omogućuje pripremu, klasifikaciju, klasteriranje, te vizualizaciju. Najnovija verzija Weke je 3.9.6 izdana u siječnju 2022. godine, a zadnja stabilna verzija je 3.8.6. Weku je razvilo sveučilište Waikato iz Novog Zelanda, a prva verzija se počela razvijati 1993. godine.



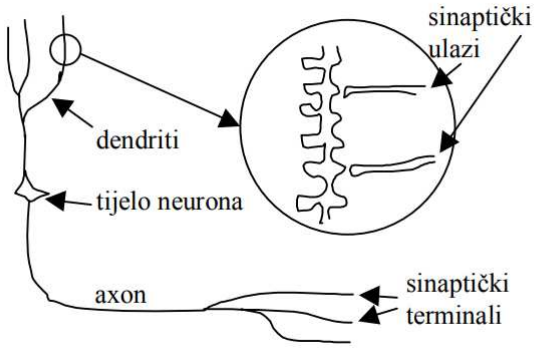
Slika 1.6: Kreiranje Spring Boot aplikacije koristeći Spring initializr

Poglavlje 2

Neuronske mreže

2.1 Uvod u neuronske mreže

Umjetna neuronska mreža ili jednostavno rečeno neuronska mreža ([21]) je skupina povezanih neurona koja pokušava oponašati osnovne principe rada neurona u mozgu, te su motivirane biološkim neuronskim mrežama. Biološke neuronske mreže sastoje se od velikog broja jednostavnih jedinica, neurona, koji primaju i prenose signale jedni drugima. Biološki neuron se sastoji od tijela stanice, dendrita i aksona. Dendriti su produžeci kojima ulazni signali dolaze do neurona, a aksoni su produžeci koji prenose izlazne signale drugim neuronima. Aksoni jednog neurona povezani su na jedan ili više dendrita drugog neurona spojevima koji se zovu sinapse koje omogućuju interakciju između neurona.



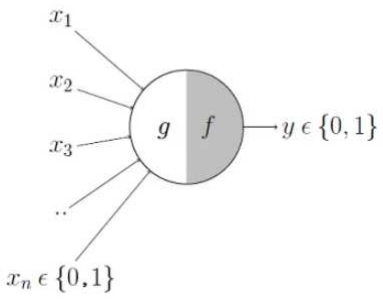
Slika 2.1: Struktura biološkog neurona

Neki od ključnih događaja za razvoj neuronskih mreža su:

- godine 1943. neurofiziolog Warren Sturgis McCulloch i logičar Walter Pitts izdaju rad u kojem opisuju jednostavni model neurona, MCP neuron ([37]), pokušavajući shvatiti kako mozak procesira neke kompleksnije uzorke preko neurona. Njihov model neurona prima ulazne vrijednosti koje mogu biti 0 ili 1, sumira ih, te na temelju toga vraća 1 ako je suma veća od vrijednosti praga θ, a inače vraća 0

$$g(x_1, x_2, x_3, ..., x_n) = g(x) = \sum_{i=1}^{n} x_i$$

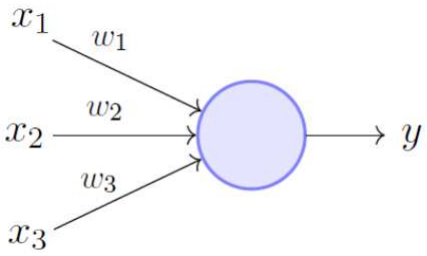
$$y = f(g(x)) = \begin{cases} 1, & g(x) \geq \theta \\ 0, & \text{otherwise} \end{cases}$$



Slika 2.2: MCP neuron

- godine 1958. američki psiholog Frank Rosenblatt izumio je perceptron, model neurona kojim je proširio model od McCullocha i Pittsa tako što je uveo težine ([21]). Perceptron (slika 2.3) prima niz ulaznih vrijednosti pri čemu svaka ima vrijednost težine koja određuje koliko je taj ulaz važan, zatim se ulazne vrijednosti množe sa svojim težinama, te na kraju ako je zbroj tih produkata veći od vrijednosti praga, prije postavljenog broja, izlaz će biti 1, a u suprotnom 0. Također, Rosenblatt je uspio da računalo razlikuje bušene kartice označene s lijeve strane u odnosu na one označene s desne strane

$$y = \begin{cases} 1, & \sum_{i=1}^{n} w_i x_i > \theta \\ 0, & \text{otherwise} \end{cases}$$



Slika 2.3: Perceptron

- godine 1986. su David Rumelhart, Geoffrey Hinton i Stephen Williams usavršili učenje propagacijom unatrag (engl. *backpropagation*)

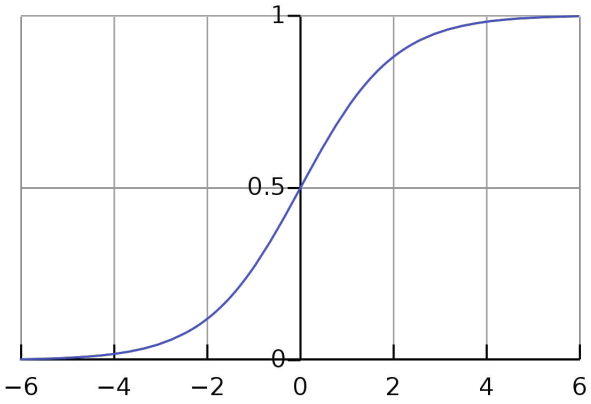
Uz skup ulaza od kojih svaki ulaz ima svoju težinu i sumatora za zbrajanje otežanih ulaza, još jedan bitan element umjetnog neurona je aktivacijska funkcija. Aktivacijska funkcija ili prijenosna funkcija je funkcija koja odlučuje hoće li se neuron aktivirati ili ne. Kao argument prima rezultat sumatora, te rezultat koji vraća predstavlja ulaz sljedećeg sloja. Jedne od najpoznatijih aktivacijskih funkcija su ([1]):

- Funkcija prag - najjednostavnija aktivacijska funkcija koja se ne može koristiti kod problema klasifikacije, te u neuronskim mrežama koje treniramo propagacijom unatrag zbog gradijenta koji je uvijek 0

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

- Sigmoidna funkcija - jedna od najkorištenijih aktivacijskih funkcija, nelinearna funkcija čiji se skup vrijednosti nalazi između 0 i 1, ne uključujući rubove (slika 2.4)

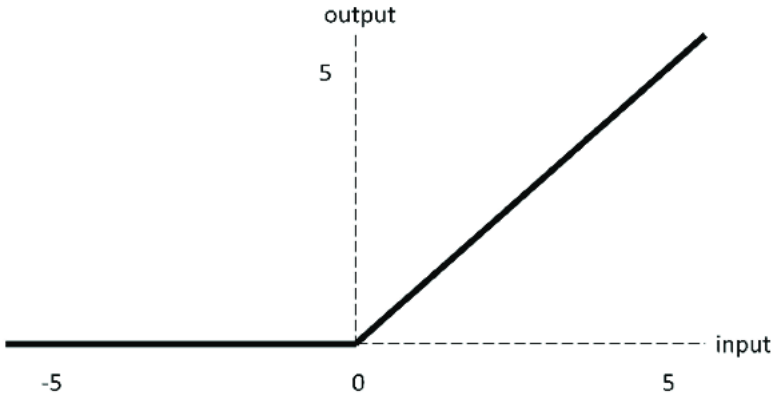
$$f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} = 1 - f(-x)$$



Slika 2.4: Sigmoidna funkcija

- ReLU funkcija (*Rectified Linear Unit*) - često korištena nelinearna aktivacijska funkcija koja se ako je ulaz pozitivan ponaša kao identiteta, a inače vraća 0. Učinkovita je jer nisu svi neuroni istovremeno aktivirani ([1]), te joj je gradijent uvijek 0 ili 1 (slika 2.5)

$$f(x) = \begin{cases} x, & x > 0 \\ 0, & \text{otherwise} \end{cases}$$



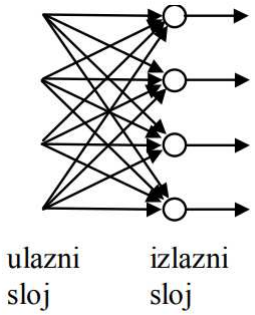
Slika 2.5: ReLU funkcija

Neuronske mreže se razlikuju po arhitekturi, odnosno po načinu na koji su raspoređeni njeni neuroni. Tako postoje četiri vrste neuronskih mreža [36]:

- Jednoslojne mreže bez povratnih veza
- Višeslojne mreže bez povratnih veza
- Mreže s povratnim vezama
- Ljestvičaste mreže

Jednoslojne mreže bez povratnih veza

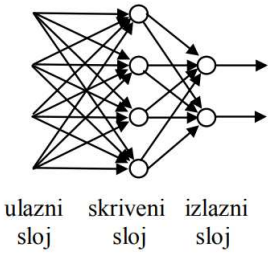
Jednoslojna neuronska mreža se, kako i njen naziv govori, sastoji od samo jednog sloja neurona, njenog izlaznog sloja, te ne sadrži povratne veze od izlaznog sloja prema ulazu. Ulazni sloj neurona se ne ubraja u njenu arhitekturu jer se u ulaznom sloju ne vrši nikakvo računanje. Ulazi u mrežu spojeni su na ulaze neurona čiji izlazi predstavljaju i izlaz mreže.



Slika 2.6: Primjer jednoslojne neuronske mreže bez povratnih veza

Višeslojne mreže bez povratnih veza

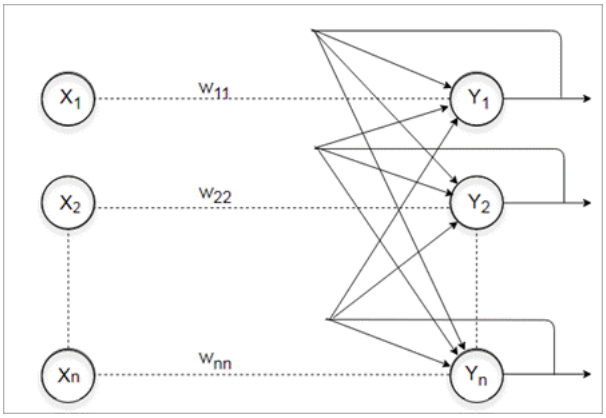
Višeslojne mreže imaju osim ulaznog i izlaznog sloja i jedan ili više skrivenih slojeva neurona. Najčešće su potpuno povezane, a za neuronsku mrežu kažemo da je potpuno povezana ako je svaki neuron u svakom sloju povezan sa svim neuronima u sljedećem sloju. Na doljnoj slici nalazi se višeslojna neuronska mreža s jednim skrivenim slojem koja sadrži četiri ulazna neurona, četiri neurona u skrivenom sloju i dva neurona u izlaznom sloju.



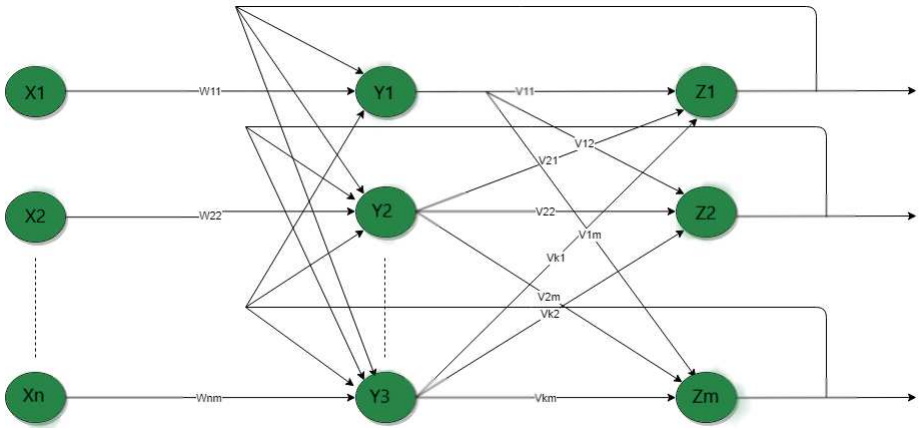
Slika 2.7: Primjer višeslojne neuronske mreže bez povratnih veza

Jednoslojne i dvoslojne mreže s povratnom vezom

Jednoslojne mreže s povratnom vezom su po arhitekturi iste kao i jednoslojne mreže bez povratnih veza jedino što sadrže barem jednu povratnu vezu, dok višeslojne neuronske mreže uz sve to još imaju jedan ili više skrivenih slojeva. U takvim mrežama se izračunata vrijednost koju daje neki neuron može ponovno vratiti tom neuronu ili ostalim neuronima iz tog ili prethodnog sloja. Prisutnost povratnih veza daje dodatnu kvalitetu ovim mrežama i omogućava bolje rezultate, ali s njima se povećava i složenost mreža.



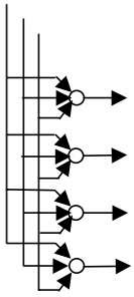
Slika 2.8: Primjer jednoslojne neuronske mreže s povratnim vezama



Slika 2.9: Primjer višeslojne neuronske mreže s povratnim vezama

Ljestvičaste mreže

Ljestvičaste mreže se sastoje od jednodimenzionalnog, dvodimenzionalnog ili višedimenzionalnog polja neurona, te je kod njih svaki ulaz spojen na sve neurone u polju. Ustvari su to mreže bez povratnih veza u kojima su neuroni raspoređeni u polje.



Slika 2.10: Primjer jednodimenzionalne ljestvičaste mreže

2.2 Treniranje neuronske mreže

Kako bi naša neuronska mreža radila korektno, odnosno vraćala ispravne rezultate za većinu ulaznih podataka na kojima ju testiramo, moramo ju prvo naučiti na posebno konstruiranom skupu za treniranje. Neuronska mreža uči kroz iterativni proces namještanja

težina, koje su u početku učenja postavljene na proizvoljne, uglavnom slučajne vrijednosti. Jedni od najpoznatijih i najbržih metoda učenja neuronskih mreža su propagacija unaprijed i unatrag. One se iterativno pozivaju jedna za drugom, pri čemu prvo počinje propagacija unaprijed.

Propagacija unaprijed

Propagacija unaprijed ([35]) je prvi korak učenja neuronskih mreža koji dolazi prije propagacije unatrag, a može se i samostalno koristiti, ali onda nije toliko efektivno. Važna je zato što za zadani ulaz i zadane težine neuronske mreže računa izlaz koji se zatim može koristiti za računanje greške i modifikacije težina u koraku propagacije unatrag.
Cijeli proces propagacije unaprijed možemo razvrstati u dvije veće cjeline:

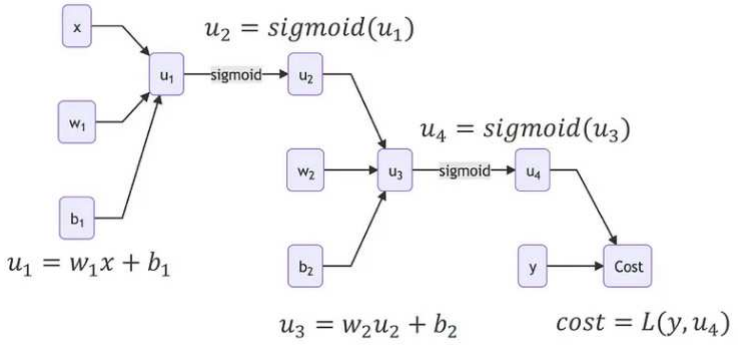
- Zbroj produkta - u ovom dijelu propagacije unaprijed zbrajamo produkt težina s ulaznim vrijednostima i na kraju tome pridodajemo vrijednost pomaka (engl. *bias*). Na donjoj formuli $x_1, x_2, x_3, ..., x_n$ predstavljaju vrijednosti neurona u prethodnom sloju s pripadajućim težinama $w_{i,j}$ prema j-tom neuronu za koji računamo vrijednost sume, a b_j je vrijednost pomaka za taj neuron

$$g(x_1, x_2, x_3, ..., x_n) = \sum_{i=1}^{n} x_i w_{i,j} + b_j$$

- Aktivacijska funkcija - poziva se na vrijednosti koju je vratio prošli korak, te izlaz aktivacijske funkcije predstavlja ulaz u sljedeći sloj. Najpoznatija aktivacijska funkcija je sigmoidna funkcija

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

Gore navedeni koraci se ponavljaju za svaki neuron u svakom sloju, počevši od prvog skrivenog sloja ako postoji, sve do izlaznog sloja, pri čemu je izlazna vrijednost iz svakog sloja, ulazna vrijednost za sljedeći sloj. Povratna vrijednost predstavlja rezultat koji je vratila neuronska mreža, te taj rezultat najčešće oduzmemo od očekivanog rezultata i tako dobijemo funkciju dobrote, a kada funkcija dobrote postane dovoljno mala, onda pretpostavljamo da je mreža dobro istrenirana.
Na donjoj slici možemo vidjeti kako izgleda propagacija unaprijed za neuronsku mrežu koja se sastoji od ulaznog sloja, jednog skrivenog sloja te izlaznog sloja. Oznaka x je vektor ulaznih vrijednosti, w su težine između slojeva, te su b vrijednosti pomaka.



Slika 2.11: Primjer propagacije unaprijed

Propagacija unatrag

Nakon izvršavanja propagacije unaprijed i usporedbe izlaza neuronske mreže s očekivanim rezultatom, slijedi propagacija unatrag ([35]). Propagacijom unatrag popravljamo iznose težina kako bi u sljedećoj iteraciji propagacija unaprijed vratila rezultat još bliži očekivanom rezultatu. Propagacija unatrag za razliku od one unaprijed, kreće od izlaznog sloja prema ulaznom sloju, prilagođavajući težine u svakom sloju.
Osnovna formula po kojoj u svakom sloju, počevši od izlaznog sloja, računamo nove iznose težina je sljedeća:

$$w = w - \alpha \cdot J'(w),$$

pri čemu se od stare vrijednosti težina oduzme produkt stope učenja α s derivacijom funkcije J čija derivacija izgleda ovako:

$$J'(W) = Z \cdot \delta,$$

gdje je Z vrijednost dobivena nakon prethodne iteracije propagacije unaprijed, iz aktivacijske funkcije, a δ je specifična za svaki sloj i na početku, na izlaznom sloju, je jednaka funkciji dobrote, razlici između dobivenog rezultata propagacije unaprijed i očekivanog. U ostalim slojevima δ se dobije iz vrijednosti delte iz prošlog sloja.

$$\delta = w \cdot \delta' \cdot f'(Z),$$

gdje su δ' vrijednost delte iz prijašnjeg koraka i f' derivacija sigmoidne funkcije pozvana na vrijednosti Z, rezultatu sigmoidne funkcije u prošloj iteraciji propagacije unaprijed.

$$f'(x) = f(x) \cdot (1 - f(x))$$

2.3 Prednosti i nedostaci neuronske mreže

Neuronske mreže dolaze s raznim prednostima, ali i nedostacima. Neke prednosti neuronskih mreža su:

- Paralelno procesiranje - ubrzava proces treniranja neuronske mreže jer može istovremeno računati propagaciju unaprijed ili unatrag za više neurona neuronske mreže. Možda nećemo vidjeti razlike na maloj količini podataka, ali najčešće su ti podaci ogromni kako bi se mreža čim bolje istrenirala
- Rad nad podacima s nedostajućim vrijednostima (engl. *data with missing values*) - neuronske mreže ignoriraju nedostajuće podatke postavljajući težine na 0, te predviđaju pomoću ostalih
- Prilagodljivost - uz modifikacije se mogu trenirati nad raznim tipovima podataka

Nedostaci neuronskih mreža:

- Izabrati pravi model - ne postoji točno pravilo kako odrediti strukturu neuronske mreže za neki posebni problem, ali do ispravne odluke dolazi se iskustvom
- Ovisnost o hardveru - paralelni rad neuronskih mreža zahtijeva od računala procesore koji omogućuju paralelni rad, u suprotnom vrijeme treniranja traje iznimno dugo
- Numeričke vrijednosti - neuronske mreže omogućuju rad s podacima koji su isključivo numeričkih vrijednosti pa je zato u suprotnom potrebno uvesti neki reprezentacijski mehanizam kojim bi pretvorili podatke u numeričke
- Rad s velikom količinom podataka - kako bi neuronska mreža davala čim preciznije rezultate moramo joj za treniranje omogućiti ogromnu količinu podataka

2.4 Upotreba neuronskih mreža

U posljednje vrijeme neuronske mreže su se značajno razvile iz jednostavnijih modela do sada sve kompleksnijih, te se nalaze u mnogim softverima, našim pametnim telefonima, tabletima, a da toga nismo ni svjesni. Najpoznatija područja u čijim pozadinama leže neuronske mreže su softveri za prepoznavanje lica, softveri za prepoznavanje govora, softveri za prepoznavanje rukopisa, te se počinju primjenjivati u medicini i predviđanju vremenske prognoze. Kod predviđanja vremenske prognoze se još uvijek uglavnom koriste tradicionalne metode.
Usprkos raznim prednostima i nedostacima koje donose neuronske mreže, ne može se opovrgnuti činjenica da su ekstremno bitna inovacija u svijetu tehnologija, te su već sad

drastično promijenile način na koji računala funkcioniraju i uvele potpuno novi pogled na tehnologije.

Poglavlje 3

Implementacija softvera

U ovom diplomskom radu napravio sam web aplikaciju za učenje i analizu neuronske mreže. Mogućnosti ove aplikacije su:

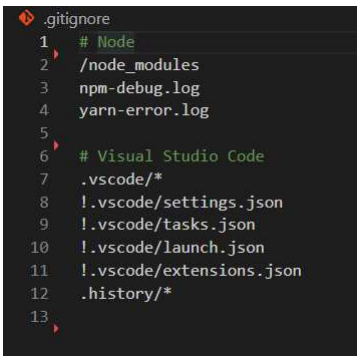
- učenje feed-forward potpuno povezane neuronske mreže nad podacima koji se pošalju u obliku arff datoteke, uz ručno postavljanje broja skrivenih slojeva i koliko će koji sloj imati neurona
- predviđanje ciljne labele jedne ili više instanci, te se u drugom slučaju podaci moraju poslati u obliku tekstualne datoteke gdje su oni odvojeni zarezom. Preduvjet za stvaranje predikcija je da je neuronska mreža natrenirana
- mijenjanje težina, te raditi predikcije s novim težinama
- klasteriranje neurona istog sloja na temelju njihove aktivacijske vrijednosti nakon što je neuronska mreža natrenirana. Klasteriranje se vrši algoritmom aglomerativnog hijerarhijskog klasteriranja pri čemu se udaljenost između dva klastera računa kao minimalna udaljenost elemenata iz tih klastera (*single linkage*)[2]
- vidjeti distribuciju po klasama nakon predikcije jednog podatka

Pri razvoju softvera koristio sam git, besplatan sustav otvorenog koda za upravljanje revizijama izvornog koda ([12]). Za pohranjivanje koda koristio sam poslužitelj github ([11]). Napravio sam dva repozitorija, jedan za klijentski dio aplikacije rađen u programskom okviru Angular i drugi repozitorij za serverski dio aplikacije implementiran u programskom okviru Spring.

3.1 Implementacija klijentskog dijela softvera

Klijentski dio softvera implementiran je pomoću Angular ([6]) programskog okvira uz jezik TypeScript ([31]). Verzija programskog okvira Angular koju sam koristio je 13, a verzija jezika TypeScript 4.6.2. Od pomoćnih biblioteka sam koristio alat D3 ([9]) za vizualizaciju neuronske mreže, verzije 7.6.1, Ng Zorro ([22]), Angularovu implementaciju Ant Designa, verzije 13, te biblioteku ngx-toastr za prikazivanje poruke kada je treniranje neuronske mreže završeno kako bi korisnik mogao raditi predikcije. Ng Zorro i ngx-toastr sam instalirao komandama *npm install ng-zorro-antd* i *npm install ngx-toastr –save*, dok sam za D3 unio ovisnost u package.json datoteku. Na slici 1.5 vide se sve ovisnosti potrebne za rad aplikacije.

S obzirom da je node_modules mapa izrazito velika (ima skoro 400MB, te sadrži oko 66000 datoteka), praksa je da se ona ne pohranjuje na git repozitorije jer se nakon kloniranja projekta može lako rekreirati pomoću *npm install* komande. Zato sam je stavio u .gitignore datoteku kao što se može vidjeti na slici 3.1. Uz node_modules mapu, u *.gitignore* datoteci nalaze se i postavke za Visual Studio Code koji sam koristio tijekom implementacije.



Slika 3.1: Datoteka .gitignore

Za komunikaciju sa serverskim dijelom softvera putem HTTP protokola, koristio sam HttpClient klasu iz osnovnog Angularovog paketa *@angular/common/http*. Sve pozive prema serverskom dijelu softvera stavio sam u jednu datoteku *src/app/provider/neural-net-provider.ts*. Na slici 3.2 vidi se korištenje HttpClienta za slanje POST metode prema serveru. Ova funkcija prihvaća arff datoteku s podacima za treniranje i listu integera koji predstavlju skrivene slojeve, te njih šalje na server kako bi server istrenirao neuronsku mrežu sa svim potrebnim podacima. Također, uz podatke koji se šalju mora se definirati i URL na kojem server prima zahtjeve. URL se razlikuje ovisno o tome je li aplikacija u produkciji ili nije, te u tome pomažu varijable okoline (engl. *environment variables*) koje

se definiraju u mapi *src/environments* u datotekama environment.*.ts (*environment.ts* se koristi za lokalni razvoj, a *environment.prod.ts* za produkciju). Na slici 3.3 vidi se primjer varijabli okoline za lokalni razvoj.

```
export class NeuralNetProvider {
  private readonly url = environment.apiUrl + '/api/neural-network/';

  constructor(
    private readonly http: HttpClient
  ) { }

  public trainNeuralNetwork(file: any, hiddenList: any): Observable<ITrainDTO> {
    const formData: FormData = new FormData();

    formData.append('file', file);

    const blobOverrides = new Blob([JSON.stringify({
      hiddenList,
    })], {
      type: 'application/json',
    });

    formData.append('properties', blobOverrides);

    return this.http.post<ITrainDTO>(
      this.url + 'train',
      formData,
    );
  }
```

Slika 3.2: Datoteka neural-net-provider.ts

```
src > environments > environment.ts > ...
1  export const environment = {
2    production: false,
3    apiUrl: 'http://localhost:8080'
4  };
5
```

Slika 3.3: Varijable okoline za lokalni razvoj

3.2 Implementacija serverskog dijela softvera

Serverski dio softvera implementiran je pomoću Spring Boot ([27]) modula programskog okvira Spring verzije 2.7.3 uz jezik Java verzije 11 ([17]). Spring Boot sam koristio zbog jednostavne izrade web aplikacija, a za učenje neuronske mreže sam koristio alat Weka ([33]) verzije 3.8.0. Ovisnost za alat Weka sam dodao u *build.gradle* datoteku s obzirom

da sam koristio Gradle ([13]) za izgradnju aplikacije. Na slici 3.4 se vide sve verzije i ovisnosti koje sam koristio, pri čemu je spring-boot-starter-web ovisnost za izradu web aplikacije.

```
plugins {
    id 'org.springframework.boot' version '2.7.3'
    id 'io.spring.dependency-management' version '1.0.13.RELEASE'
    id 'java'
}

group = 'com.pmf'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

repositories {
    mavenCentral()
    maven {
        url "https://mvnrepository.com/artifact/nz.ac.waikato.cms.weka/weka-stable"
    }
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation group: 'nz.ac.waikato.cms.weka', name: 'weka-stable', version: '3.8.0'
}

tasks.named('test') { Task it ->
    useJUnitPlatform()
}

```

Slika 3.4: Datoteka build.gradle

Struktura Spring Boot aplikacije se sastoji od dva dijela, *controller* mapa i *service* mapa. *Controller* mapa sadrži klase koje su odgovorne za prihvaćanje HTTP zahtjeva, te se u njima definiraju rute na kojima Spring Boot aplikacija prihvaća HTTP zahtjeve. Nakon što se u *controller* mapi primi zahtjev, pozivaju se klase iz *service* mape koje onda obrađuju zahtjev. Uz *controller* i *service* mape, napravio sam *command* mapu gdje sam stavio klase koje opisuju podatke koji dolaze s HTTP zahtjevima prema aplikaciji, *DTO* mapu (engl. *Data transfer object*) gdje se nalaze klase koje opisuju podatke koji idu natrag s HTTP zahtjevom prema klijentskom dijelu aplikacije kao odgovor na zahtjev, te *model* mapu u kojoj su pomoćne klase u kojima se uči i testira neuronska mreža pomoću Weke.

```java
@RestController
@CrossOrigin
@RequestMapping("/api/neural-network")
public class NeuralNetworkController {
    private final NeuralNetworkService neuralNetworkService;

    public NeuralNetworkController(NeuralNetworkService neuralNetworkService) {
        this.neuralNetworkService = neuralNetworkService;
    }

    @PostMapping(value = "/train", consumes = { MediaType.APPLICATION_JSON_VALUE, MediaType.MULTIPART_FORM_DATA_VALUE })
    public TrainDTO train(@RequestPart("properties") TrainCommand properties,
                          @RequestPart("file") MultipartFile file) throws Exception {
        return neuralNetworkService.train(file, properties);
    }

    @PostMapping("/predict")
    public PredictDTO predict(@RequestBody PredictCommand command) throws Exception {
        return neuralNetworkService.predict(command.getInputList());
    }

    @PostMapping("/predict-testing-file")
    public ArrayList<PredictTestingFileDTO> predictTestingFile(@RequestParam("file") MultipartFile file) throws Exception {
        return neuralNetworkService.predictTestingFile(file);
    }

    @PostMapping("/modify-weights")
    public void modifyWeights(@RequestBody ModifyWeightsCommand command) throws Exception {
        neuralNetworkService.modifyWeights(command);
    }
}
```

Slika 3.5: Klasa NeuralNetworkController

Na slici 3.5 se vidi jedina *controller* klasa u kojoj su definirane metode koje obrađuju HTTP zahtjeve na temelju URL-a na koji je poslan HTTP zahtjev. Svaka metoda poziva servisni sloj u kojem se koristeći Weka obrađuje zahtjev. U slučaju metode *train* i *predict* u servisnom sloju se pozivaju klase *ModelGenerator* i *ModelClassifier* s pomoćnim metodama. Na slici 3.6 je metoda *loadDataset* klase *ModelGenerator* koja se koristi u metodi *train* za dohvaćanje inicijalnih podataka iz arff datoteke pomoću Weke.

```java
import weka.classifiers.evaluation.Evaluation;
import weka.classifiers.functions.MultilayerPerceptron;
import weka.core.Instances;
import weka.core.SerializationHelper;
import weka.core.converters.ConverterUtils.DataSource;

public class ModelGenerator {

    public Instances loadDataset(InputStream inputStream) {
        Instances dataset = null;
        try {
            dataset = DataSource.read(inputStream);
            if (dataset.classIndex() == -1) {
                dataset.setClassIndex(dataset.numAttributes() - 1);
            }
        } catch (Exception ex) {
            Logger.getLogger(ModelGenerator.class.getName()).log(Level.SEVERE, msg: null, ex);
        }

        return dataset;
    }
```

Slika 3.6: Metoda *loadDataset* klase *ModelGenerator*

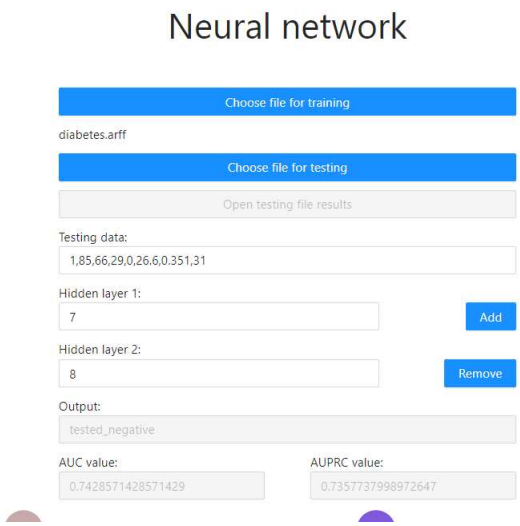
Poglavlje 4

Primjer uporabe

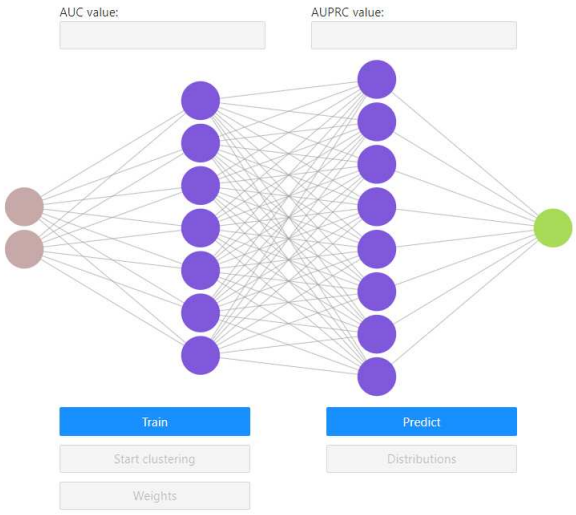
Za uporabu aplikacije (slika 4.1 i 4.2), nužno je imati arff datoteku s podacima za treniranje neuronske mreže. Može se preuzeti postojeća ili napraviti nova s proizvoljnim podacima. Ja sam za ovaj primjer preuzeo arff datoteku s podacima za dijabetičare (slika 4.6). Neki od atributa su tjelesna masa, godine, koliko puta je osoba bila trudna, a klase su *tested positive* i *tested negative*. Kada je arff datoteka spremna, korisnik klikne na gumb *Choose file for training* (slika 4.1) pri čemu se otvori prozor za učitavanje datoteke. Prije nego treniranje počne, korisnik može postaviti koliko želi skrivenih slojeva u neuronskoj mreži i koliko će svaki skriveni sloj imati neurona. Kada je datoteka učitana i postavljene su vrijednosti skrivenih slojeva, korisnik klikne na gumb *Train* (slika 4.2) pri čemu se datoteka zajedno sa podacima za skrivene slojeve pošalje serverskom dijelu softvera, Spring Boot aplikaciji, koja pokreće proces treniranja neuronske mreže s dobivenim podacima.

Kada je treniranje neuronske mreže završeno, na klijentskoj aplikaciji će doći obavijest o tome, te će se popuniti polja *AUC value* i *AUPRC value* (slika 4.1) s odgovarajućim AUC i AUPRC vrijednostima dobivenim na testnim podacima. AUC i AUPRC su mjere evaluacije modela strojnog učenja koje pokazuju koliko je model točan. Sada korisnik može raditi predikcije na treniranoj mreži. Predikcije se mogu raditi na samo jednom testnom podatku tako da se on zapiše u *Testing data* sekciju pri čemu vrijednosti atributa trebaju biti odvojene zarezom, te se klikne na gumb *Predict*. Nakon što je u serverskom dijelu softvera neuronska mreža testirala poslani podatak, *Output* sekcija se popuni s imenom klase kojoj po naučenoj mreži taj podatak pripada. Uz rezultat, klikom na gumb *Distributions* može se vidjeti distribucija testnog podatka za sve klase (slika 4.3). Također, predikcije se mogu raditi na skupini podataka koji se pošalju tekstualnom datotetom klikom na gumb *Choose file for testing* (slika 4.1). Svaki redak predstavlja jedan podatak koji se želi testirati, pri čemu su vrijednosti atributa odvojene zarezom. Kada Spring Boot aplikacija završi s testiranjem, rezultate će vratiti klijentskoj aplikaciji, te će ih korisnik moći vidjeti klikom na

gumb *Open testing file results*.



Slika 4.1: Gornji dio aplikacije



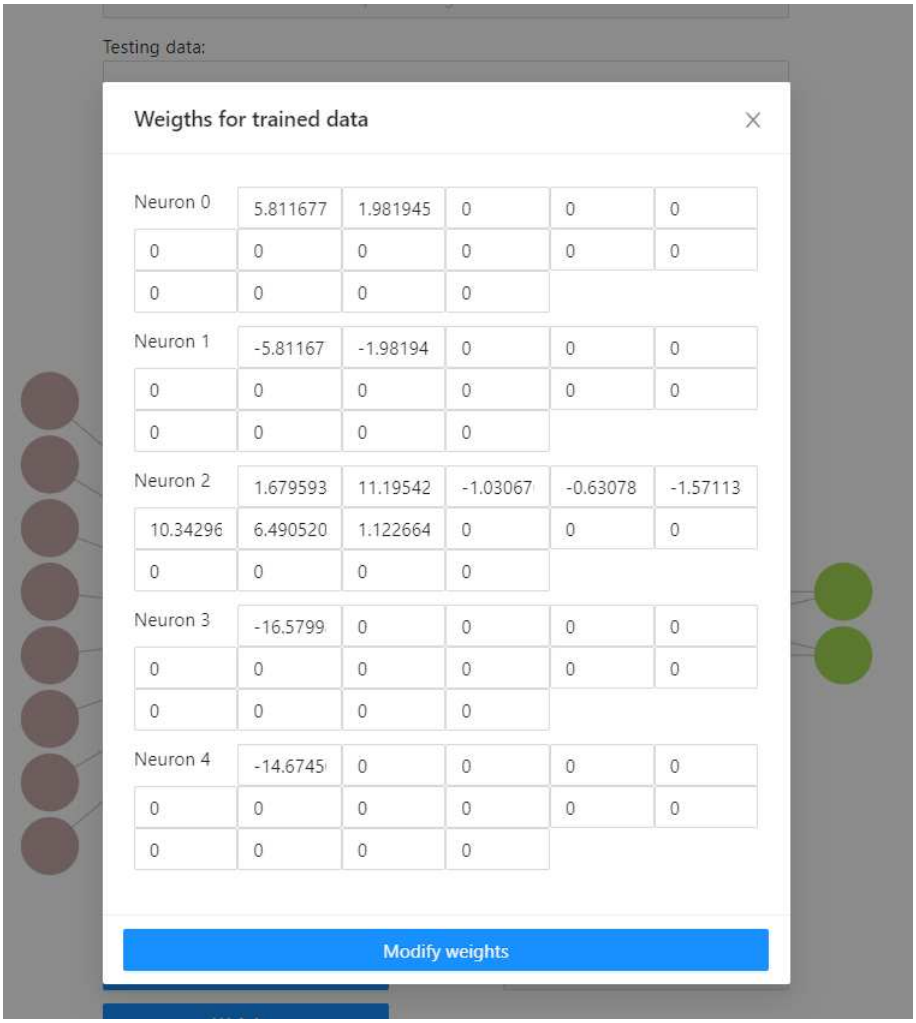
Slika 4.2: Donji dio aplikacije

Nakon što je neuronska mreža naučena, Spring Boot aplikacija će vratiti aktivacijske vri-

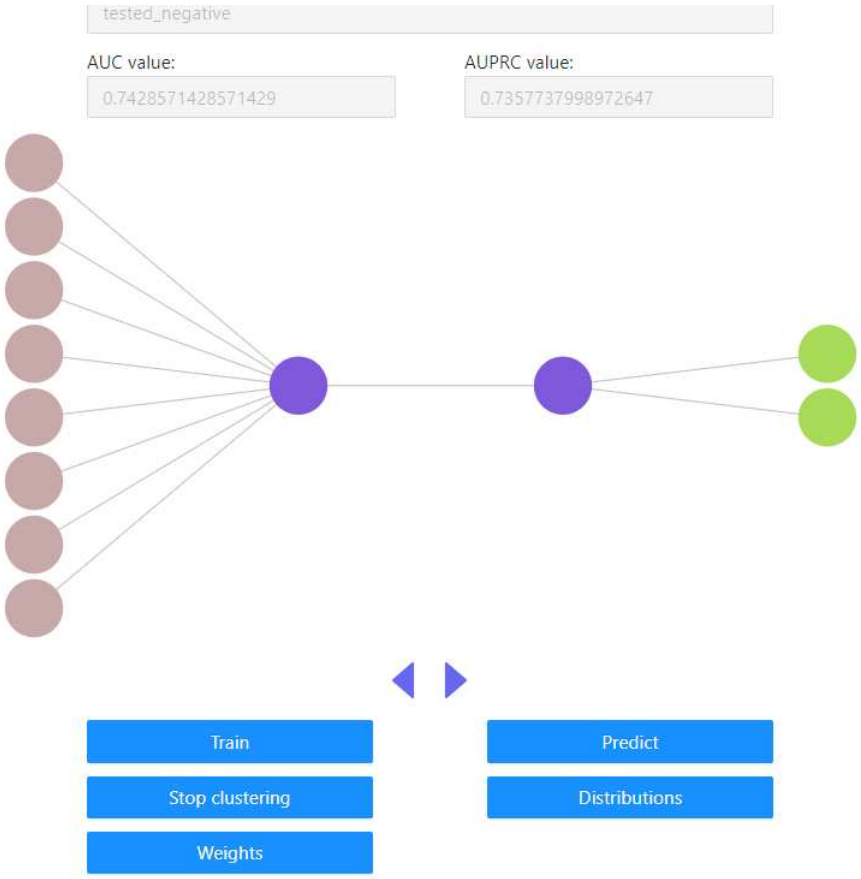
Slika 4.3: Distribucije za testni podatak

jednosti za neurone skrivenih slojeva koje se mogu vidjeti na neuronskoj mreži. Također, uz aktivacijske vrijednosti vratit će i vrijednosti težina koje se mogu vidjeti klikom na gumb *Weights* (slika 4.5). Klikom na gumb se otvori prozor za prikaz i promjenu težina, te nakon što korisnik promijeni težine, stisne na gumb *Modify weights* (slika 4.4) čime se serverskom dijelu softvera šalju nove vrijednosti težina. Nakon što prođe zahtjev za promjenom težina, moguće je raditi predikcije s novim težinama.
Na slici 4.4 se vidi prozor za prikaz i mijenjanje težina. Za svaki neuron, osim za neurone ulaznog sloja, prikazane su vrijednosti težina za njegove veze s neuronima iz prethodnog sloja. Oznaka neurona može se vidjeti kada se mišem prijeđe preko neurona.

Nakon što je neuronska mreža naučena korisnik može klikom na gumb *Start clustering* (slika 4.5) raditi klasteriranje neurona na temelju njihove aktivacijske vrijednosti. Zatim, klikom na strelice korisnik može mijenjati razine klasteriranja, te kada želi završiti s klasteriranjem klikne na gumb *Stop clustering* (slika 4.5). Na slici 4.5 se vidi klasteriranje neurona, te kada korisnik prijeđe mišem preko pojedinog klastera može vidjeti koji se neuroni nalaze u klasteru preko njihove aktivacijske vrijednosti.



Slika 4.4: Prozor za prikaz i mijenjanje težina



Slika 4.5: Klasteriranje neurona

```
%
% For Each Attribute: (all numeric-valued)
%    1. Number of times pregnant
%    2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
%    3. Diastolic blood pressure (mm Hg)
%    4. Triceps skin fold thickness (mm)
%    5. 2-Hour serum insulin (mu U/ml)
%    6. Body mass index (weight in kg/(height in m)^2)
%    7. Diabetes pedigree function
%    8. Age (years)
%    9. Class variable (0 or 1)
% Relabeled values in attribute 'class'
%    From: 0                       To: tested_negative
%    From: 1                       To: tested_positive
%
@relation pima_diabetes
@attribute 'preg' real
@attribute 'plas' real
@attribute 'pres' real
@attribute 'skin' real
@attribute 'insu' real
@attribute 'mass' real
@attribute 'pedi' real
@attribute 'age' real
@attribute 'class' { tested_negative, tested_positive}
@data
6,148,72,35,0,33.6,0.627,50,tested_positive
1,85,66,29,0,26.6,0.351,31,tested_negative
8,183,64,0,0,23.3,0.672,32,tested_positive
1,89,66,23,94,28.1,0.167,21,tested_negative
0,137,40,35,168,43.1,2.288,33,tested_positive
5,116,74,0,0,25.6,0.201,30,tested_negative
3,78,50,32,88,31,0.248,26,tested_positive
10,115,0,0,0,35.3,0.134,29,tested_negative
2,197,70,45,543,30.5,0.158,53,tested_positive
8,125,96,0,0,0,0.232,54,tested_positive
4,110,92,0,0,37.6,0.191,30,tested_negative
10,168,74,0,0,38,0.537,34,tested_positive
10,139,80,0,0,27.1,1.441,57,tested_negative
1,189,60,23,846,30.1,0.398,59,tested_positive
5,166,72,19,175,25.8,0.587,51,tested_positive
7,100,0,0,0,30,0.484,32,tested_positive
0,118,84,47,230,45.8,0.551,31,tested_positive
7,107,74,0,0,29.6,0.254,31,tested_positive
1,103,30,38,83,43.3,0.183,33,tested_negative
```

Slika 4.6: Podaci o dijabetičarima pomoću kojih sam testirao aplikaciju

Bibliografija

[1] *Activation functions in neural networks*, https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf.

[2] *Agglomerative clustering*, https://online.stat.psu.edu/stat508/lesson/12/12.6.

[3] *Angular CLI*, https://angular.io/cli.

[4] *Angular: Setting up the local environment and workspace*, https://angular.io/guide/setup-local.

[5] *Angular versioning and releases*, https://angular.io/guide/releases#actively-supported-versions.

[6] *Angular*, https://angular.io/guide/what-is-angular.

[7] *Ant Design*, https://ant.design/.

[8] *CSS*, https://developer.mozilla.org/en-US/docs/Web/CSS.

[9] *D3.js*, https://d3js.org/.

[10] *Embedded Web Servers*, https://docs.spring.io/spring-boot/docs/2.1.9.RELEASE/reference/html/howto-embedded-web-servers.html.

[11] *GitHub*, https://github.com/.

[12] *Git*, https://git-scm.com/.

[13] *Gradle*, https://gradle.org/.

[14] *History of Spring and the Spring Framework*, https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html#overview-history.

[15] *HTML*, https://developer.mozilla.org/en-US/docs/Web/HTML.

[16] *The IoC container*, https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html.

[17] *Java*, https://www.java.com/en/.

[18] *Material Design*, https://m3.material.io/.

[19] *Material UI*, https://mui.com/.

[20] *Maven*, https://maven.apache.org/.

[21] *Neural network*, https://www.ibm.com/topics/neural-networks.

[22] *NG Zorro*, https://ng.ant.design/docs/introduce/en.

[23] *Node.js*, https://nodejs.org/en.

[24] *React*, https://react.dev/.

[25] *SPA (Single-page application)*, https://developer.mozilla.org/en-US/docs/Glossary/SPA.

[26] *Spring Boot Actuator*, https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html.

[27] *Spring Boot*, https://spring.io/projects/spring-boot.

[28] *Spring Framework*, https://spring.io/projects/spring-framework.

[29] *Spring initializr*, https://start.spring.io/.

[30] *SVG*, https://developer.mozilla.org/en-US/docs/Web/SVG.

[31] *Typescript*, https://www.typescriptlang.org/.

[32] *Vue.js*, https://vuejs.org/.

[33] *Weka: Machine learning software in Java*, https://www.cs.waikato.ac.nz/ml/weka/.

[34] *XML*, https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction.

[35] Bojana Dalbelo Bašić, Marko Čupić i Jan Šnajder, *Umjetne neuronske mreže*, Fakultet elektrotehnike i računarstva.

[36] Sven Lončarić, *Neuronske mreže: Uvod*, Fakultet elektrotehnike i računarstva.

[37] Beeior Rov-Ikpah, *Neural network modeling for brain visual cortex*, 2016.

Sažetak

U ovom diplomskom radu napravio sam softver za učenje i analizu neuronske mreže. Ovim softverom je moguće:

- naučiti potpuno povezanu feed-forward neuronsku mrežu nad proizvoljnim tabularnim podacima za treniranje koji se pošalju u obliku arff datoteke
- testirati neuronsku mrežu na jednom testnom podatku ili na skupini podataka koji se pošalju u obliku tekstualne datoteke
- mijenjati težine i raditi predikcije koristeći promijenjene težine
- klasterirati neurone istog sloja na temelju njihove aktivacijske vrijednosti

U prvom poglavlju opisao sam tehnologije i alate s kojima sam implementirao softver, a neke od njih su:

- Angular - programski okvir za izradu klijentskog dijela web aplikacije
- Spring Boot - modul programskog okvira Spring za jednostavniju izradu web aplikacija
- Weka - alat koji sadrži algoritme strojnog učenja za dubinsku analizu podataka

U drugom poglavlju opisano je što su to neuronske mreže, kako trenirati neuronsku mrežu, te koje su im prednosti i nedostaci. U zadnjim poglavljima opisao sam kako sam implementirao softver i kako se koristiti njime.

Summary

In this thesis I made software for training and analyzing neural networks. With this software user can:

- train fully connected feed-forward neural network on arbitrary tabular training data which is sent as an arff file
- test neural network on one instance or on set of instances in which case they should be sent as a text file
- modify weights and make predictions on modified neural network
- cluster neurons of same layer based on their activation value

In the first chapter I described technologies and tools which helped me develop this software. Some of them are:

- Angular - application framework for building client side of web application
- Spring Boot - Spring framework module for building web applications with minimum effort
- Weka - tool which contains machine learning algorithms for data mining tasks

In second chapter I described what are neural networks, how they can be trained and which are their advantages and disadvantages. In last chapters I described how I implemented this software and how it can be used.

Životopis

Rođen sam 28. svibnja 1997. godine u Zagrebu i živim u Samoboru. Pohađao sam Osnovnu školu Bogumila Tonija u Samoboru koju sam upisao 2004. godine. Nakon završetka osnovne škole, upisao sam opću gimnaziju u Samoboru. U srednjoj školi me sve više počela zanimati matematika pa sam tako odlučio nakon srednje škole upisati preddiplomski studij matematike na Prirodoslovnom matematičkom fakultetu u Zagrebu. Na prvoj godini me počelo zanimati računarstvo, te sam odmah znao da se s time želim baviti. Nakon završenog preddiplomskog studija matematike, upisao sam diplomski studij Računarstvo i matematika. Za vrijeme 4. godine, u veljači 2021. godine, počeo sam raditi u Hivetechu kao softverski inžinjer. Tamo sam bio skroz do rujna 2022. godine kada sam u tvrtki Axilis kao softverski inžinjer započeo svoj prvi radni odnos.