

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6950

Klasifikacija uporabom umjetnih neuronskih mreža

Darijo Brčina

Zagreb, svibanj 2020.

Zagreb, 12. ožujka 2020.

Polje: **2.09 Računarstvo**

ZAVRŠNI ZADATAK br. 6950

Pristupnik: **Darijo Brčina (0036506587)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Klasifikacija uporabom umjetnih neuronskih mreža**

Opis zadatka:

Klasifikacijske probleme moguće je rješavati mnoštvom različitih pristupa. Jedan od često korištenih modela su umjetne neuronske mreže. Danas postoji mnoštvo arhitektura umjetnih neuronskih mreža, pri čemu su različite arhitekture pogodne za rješavanje različitih vrsta problema.

U okviru ovog završnog rada potrebno je proučiti unaprijedne višeslojne potpuno povezane neuronske mreže primijenjene na klasifikacijske probleme te algoritme njihova učenja neuronskih mreža koji se temelje na gradijentu funkcije pogreške. Potrebno je napisati programsku implementaciju algoritma učenja, pri čemu treba podržati i učenje kroz minigrupe te vizualizaciju decizijske granice tijekom učenja u slučaju kada su ulazni uzorci iz 2D prostora. Algoritam učenja treba podržati mreže s različitim prijenosnim funkcijama. U okviru rada potrebno je razmotriti i utjecaj različitih prijenosnih funkcija na oblik decizijske granice.

Radu je potrebno priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 13. ožujka 2020.

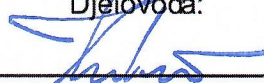
Rok za predaju rada: 12. lipnja 2020.

Mentor:



Doc. dr. sc. Marko Čupić

Djelovoda:



Izv. prof. dr. sc. Tomislav Hrkać

Predsjednik odbora za
završni rad modula:



Doc. dr. sc. Marko Čupić

Zahvaljujem mentoru Marku na slobodi odabira teme za ovaj rad i na mnoštvu ukazanih literatura.

SADRŽAJ

1. Uvod	1
2. Pregled područja	2
2.1. Umjetna inteligencija	3
2.1.1. Simboličko učenje	3
2.1.2. Strojno učenje	3
3. Nadzirano učenje	6
3.1. Učenje modela	6
3.2. Problemi nadziranog učenja	8
3.2.1. Regresija	9
3.2.2. Klasifikacija	11
3.3. Algoritmi	13
4. Umjetne neuronske mreže	14
4.1. Motivacija	14
4.2. Biološki neuron	14
4.3. Umjetni neuron	15
4.3.1. Povijesni pregled	15
4.3.2. Perceptron	19
4.4. Općenito o umjetnim neuronskim mrežama	22
4.5. Unaprijedna neuronska mreža	24
4.5.1. Aktivacijske funkcije	28
4.5.2. Inicijalizacija i normalizacija	33
4.6. Učenje unaprijedne neuronske mreže	34
4.6.1. Gradijentni spust	34
4.6.2. Algoritam propagacije pogreške unatrag	35
5. Rezultati	40

6. Zaključak	45
Literatura	46

1. Uvod

Projektiranje sustava koji samostalno uče je osnovni problem moderne računarske znanosti. Sam postupak učenja ne predstavlja toliko problem koliko modeliranje raznih modela koje je potrebno naučiti te odabira dobrog algoritma iz palete mnoštva. Često se moramo pitati što točno želimo da sustav čini te koje onda postupke umjetne inteligencije možemo iskoristiti. Najbanalniji primjer je prepoznavanje sadržaja slike na kojoj se može nalaziti mnoštvo sadržaja. Npr. prepoznavanje rukom pisanih brojeva, razlikovanje ljudi od životinja i neživih stvari itd.

U ovom radu ćemo se dotaknuti teorijske podloge jednog od algoritama umjetne inteligencije, točnije nadziranog učenja, u svrhu kreiranja jednog jednostavnog klasifikatora koji će razviti svojstvo raspoznavanja razreda na temelju 2D koordinata.

U drugom poglavlju dan je kratak pregled područja umjetne inteligencije i strojnog učenja. U trećem poglavlju detaljnije je prikazana teorijska podloga nadziranog učenja. Četvrto poglavlje opisuje algoritam umjetne neuronske mreže kroz povijest do danas te njegova razna svojstva i primjene. U petom poglavlju su prikazani rezultati programske implementacije koji potkrepljuju teoriju četvrtog poglavlja.

2. Pregled područja

Pitate li se ikada što je to inteligencija te čemu nam služi. To pitanje postavljeno je još za vrijeme začetka filozofije kao znanosti kada su se tadašnji filozofi pitali kako i na koji način je ljudsko razmišljanje, učenje i pamćenje ostvareno. Ni dan danas ne postoji jednoznačan odgovor na to pitanje jer ljudski mozak i dalje predstavlja jednu veliku nepoznanicu koja vjerojatno nikada ili ne tako skoro neće biti razriješena. No znanost je podosta napredovala i shodno tomu se razvila želja da se ljudska inteligencija pokuša pretočiti u nekakvu vrstu inteligencije strojeva.

Početak ovakvog razmišljanja datira od 50-tih godina dvadesetog stoljeća kada Alan Turing u članku *Computing Machinery and Intelligence* časopisa *Mind* postavlja pitanje: Mogu li strojevi misliti? (engl. *Can machines think?*) na koje pokušava odgovoriti kroz tzv. igru imitacije (engl. *imitation game*). Sudionici igre su tri igrača: igrač A, igrač B i igrač C gdje su igrači A i B ispitanici a igrač C ispitivač. Cilj igrača C je utvrditi spol ispitanika postavljanjem pitanja, cilj igrača B je pomoći ispitivaču C a cilj igrača A je navesti ispitivača C na pogrešnu identifikaciju. Što će se dogoditi ako stroj uzme mjesto igrača A? Ako broj pogrešaka igrača C bude gotovo jednak u oba slučaja, onda je stroj inteligentan. (Turing, 1950). Ovakav princip testiranja se često naziva Touringov test (engl. *Turing test*).

Nedugo nakon Touringov eksperimenta, 1956. se održava konferencija u Dartmouthu (Dartmouth workshop, 2020) na inicijativu John McCarthy-a, tadašnjeg mladog profesora matematike na fakultetu u Dartmouthu, koji okuplja oko sebe nekolicinu znanstvenika i prijatelja kako bi pokušali koncepte ljudske inteligencije preslikati u inteligenciju strojeva. Cilj je pokazati kako strojevi koriste jezik, kako zaključuju i stvaraju apstraktne koncepte i kako vremenom postaju sve prilagodljiviji na predložene probleme baš kao i ljudi. Inicijalna ideja je bila da se neki od navedenih problema može dokazati uz manju skupinu dobrih znanstvenika i kroz period od jednog ljeta (McCarthy et al. 1955). To naravno nije bilo moguće. Time se formalno uvodi pojam *umjetna inteligencija*.

2.1. Umjetna inteligencija

Kao što je anticipirano ranije, umjetna inteligencija (engl. *artificial intelligence*) je laički rečeno inteligencija strojeva a znanost koja se bavi proučavanjem umjetne inteligencije naziva se računarska znanost (engl. *computer science*).

Primjena umjetne inteligencije danas je izrazito rasprostranjena kroz gotovo svaku industriju. Pronalazimo ju u medicini, automobilske industriji, robotici, pa i u sportskoj i industriji igara. Jedan od poznatijih događaja koji prikazuje primjenu umjetne inteligencije dogodio se 2016. kada je računalo naziva *AlphaGo* u igri *Go* uspjelo pobijediti svjetskog prvaka Lee Sedolu rezultatom 4:1 te time ostvario velik uspjeh u svijetu umjetne inteligencije kao i pažnju javnosti (Moyer, 2016).

Umjetnu inteligenciju je dakako potrebno trenirati i učiti pa je tako učenje podijeljeno na dvije veće cjeline:

1. simboličko učenje te
2. strojno učenje.

2.1.1. Simboličko učenje

Simbolička umjetna inteligencija (engl. *symbolic artificial intelligence*) je izraz koji definira skup istraživačkih metoda koje se temelje na ljudima lako čitljivim simbolima (engl. *human-readable simbol*) koji modeliraju probleme i logiku. Jedan od najboljih primjera jesu *ekspertni sustavi* koji se temelje na skupu pravila. Pravila su modelirana na sličan način kao i Ako-Onda rečenica (engl. *If-Then statement*) koja je u ljudskoj komunikaciji svakodnevno u upotrebi. Također, razne vrste logika poput propozicijska (engl. *Propositional logic*), često referirana kao Booleova algebra, logika prvog reda (engl. *First order logic*), poznatija kao predikatna logika (engl. *Predicate logic*), neizrazita logika (engl. *Fuzzy logic*) pripadaju upravo simboličkoj umjetnoj inteligenciji. Ovakav način učenja bio je popularan početkom 1950. sve do kraja 1980 (Symbolic artificial intelligence, 2020).

2.1.2. Strojno učenje

Strojno učenje (engl. *machine learning, ML*) predstavlja niz metoda i algoritama koji sustavima pružaju stjecanje novog znanja kroz modeliranje obrazaca koje onda kasnije mogu iskoristiti za predviđanje novih podataka ili sličnih (Čupić, 2020). Glavna ideja

je da sustavi uče iz iskustva, empirijski, bez da se programska implementacija mijenja što znatno olakšava manipulaciju istih.

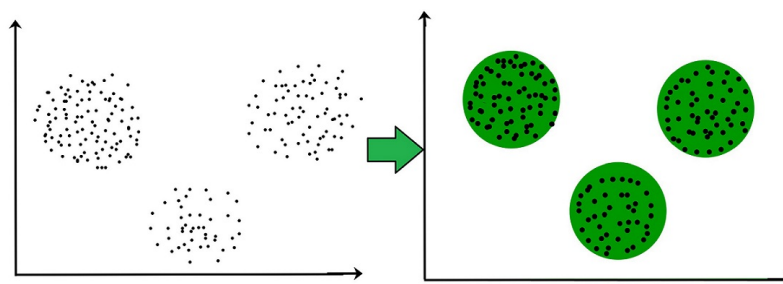
Danas postoji nezgrapno puno podataka koje je moguće i koje je potrebno iskoristiti za učenje pa je cilj konstruirati sustave koji mogu iskoristiti baš te podatke za neka korisna ponašanja poput predviđanja i raspoznavanja raznih uzoraka (Čupić, 2020). Podatci se svrstavaju u dvije grupe: *numerički* i *kategorički* podatci. Numerički podatci su svi oni podatci nad kojima je moguće izvršiti aritmetičke operacije. Npr. unos dnevnih kalorija, broj slobodnih bacanja na košarkaškoj utakmici, isplata plaća i sl. Kategorički podatci se dodatno dijele u dvije podgrupe: nominalni i ordinalni podatci. Nominalni podatci su imenovani podatci koji nisu numerički, tj. podatci nad kojima nisu definirane aritmetičke operacije. Npr. podatci o spolu jedinke, osjećaju raspoloženja poput "tužan" ili "veseo" i sl. Ordinalni podatci također nemaju definiranu aritmetiku, no imaju definiran prirodan poredak i mogu se uspoređivati. Npr. mišljenje jedne osobe može biti "jako zadovoljan" dok druge "zadovoljan" što možemo usporediti i konstruirati prirodan poredak.

Strojno učenje dijelimo na četiri područja:

1. nadzirano učenje,
2. nenadzirano učenje,
3. polu-nadzirano učenje te
4. podržano učenje.

Nadzirano učenje je tema ovog rada pa će biti obrađeno detaljnije u narednom poglavlju.

Nenadzirano učenje (engl. *unsupervised learning*) je vrsta učenja u kojem skup podataka predstavljaju samo ulazni podatci bez znanja o tome kako bi isti trebali biti tabelirani. Zbog ovakvog pristupa, učenje je i dobilo ime nenadzirano učenje, tj. učenje bez prisustva učitelja (engl. *supervisor*). Najpoznatiji postupak nenadzirano učenja je postupak grupiranja (engl. *clustering*). Cilj grupiranja je na temelju danih podataka pokušati pronaći sve podatke koji imaju slična svojstva te ih zatim grupirati u odvojene razrede. Npr. neka skup ulaznih podataka sustava bude mješavina slika ljudi i slika automobila. Kao konačni rezultat, moraju se stvoriti dva razreda: razred ljudi te razred automobila. Na slici 2.1 prikazan je primjer grupiranja.

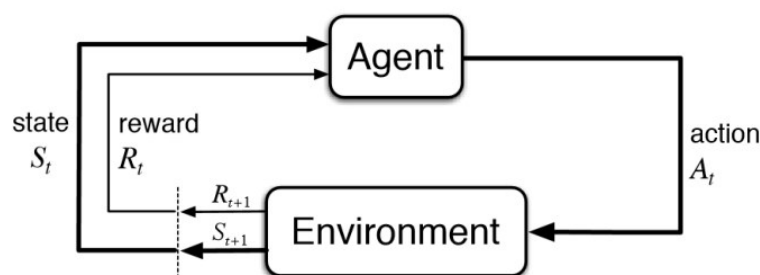


Slika 2.1: Primjer grupiranja¹

Uz grupiranje poznati postupci su i postupci smanjenja dimenzionalnosti (engl. *dimensionality reduction*), postupci otkrivanja stršećih ili novih vrijednosti (engl. *outlier detection*, *novelty detection*) i drugi.

Polu-nadzirano učenje (engl. *semi-supervised learning*), kao što samo ime nalaže, je učenje u kojem se isprepliću koncepti nadzirano učenje s konceptima nenadziranim učenjem. Ono pokazuje dosta veće uspjehe nego navedeni, no izvedba je izrazito zahtjevnija (Semi-supervised learning, 2020).

Podržano učenje (engl. *reinforcement learning*, *RL*) je vrsta učenja koja se potpuno razlikuje od svih do sada navedenih jer ono ne očekuje nikakve tabelirane ulazne ili izlazne podatke već je glavna ideja optimizacija ponašanja računalnih agenata. Razmatra se interakcija agenta s okolinom (engl. *environment*) kroz niz akcija za koje isti može biti nagrađen ili kažnjen ovisno o ishodu akcije. Željeni cilj agenta je maksimizirati mogući dobitak nagrade (Reinforcement learning, 2020). Na slici 2.2 prikazan je model podržanog učenja.



Slika 2.2: Model podržanog učenja²

¹Preuzeto sa <https://www.geeksforgeeks.org/clustering-in-machine-learning/>. [Pristupljeno 10-Maj-2020].

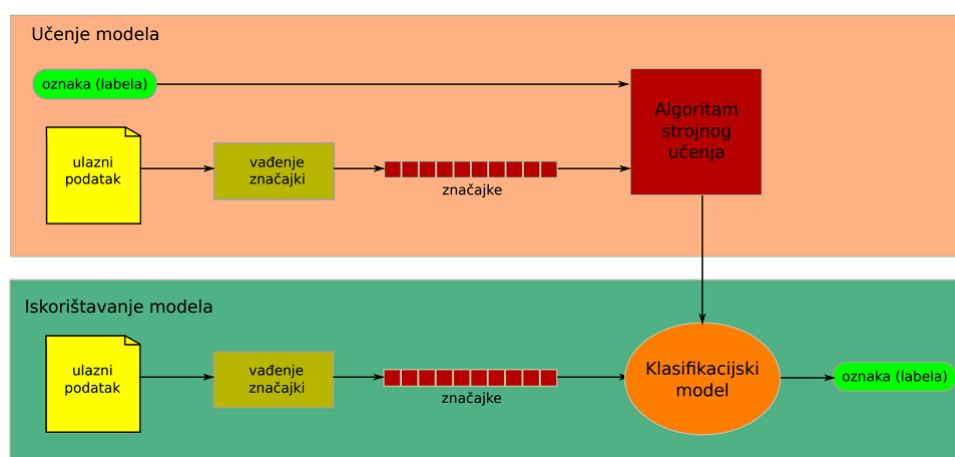
²Preuzeto sa <https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html/>. [Pristupljeno 10-Maj-2020].

3. Nadzirano učenje

Nadzirano učenje (engl. *supervised learning*) je vrsta učenja čiji je cilj sve uzorke koji dođu na ulaz sustava preslikati u izlaz koji točno odgovara predanim ulazima. Dakle, prije samog učenja poznati su ulazi, koji se često modeliraju vektorima, te izlazi, koji su pridruženi tim ulazima, stoga se tijekom učenja u svakom trenutku zna željeni izlaz te se za krive rezultate daje određena povratna informacija (engl. *feedback*) o tome koliko sustav griješi. Upravo zbog navedenog koncepta je učenje i dobilo ime nadzirano učenje jer se proces učenja izvršava uz prisustvo učitelja. Razlikujemo dvije faze strojnog modela prikazane na slici 3.1:

1. faza učenja modela te
2. faza iskorištavanja modela.

3.1. Učenje modela



Slika 3.1: Model strojnog učenja¹

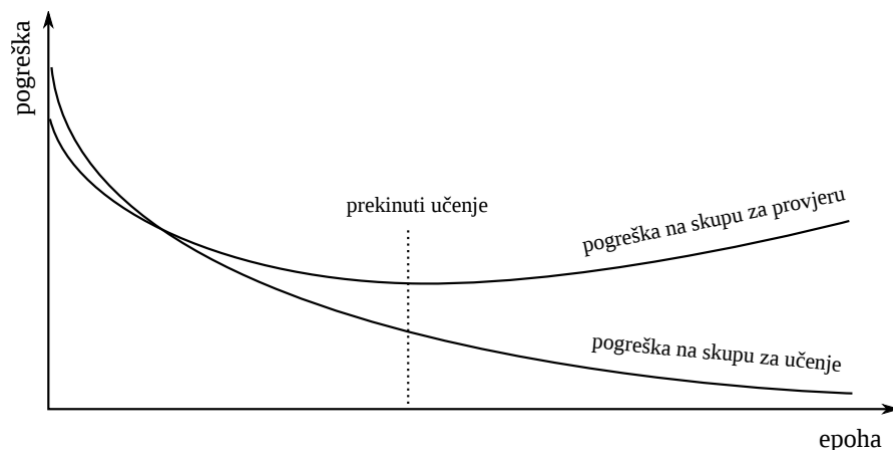
¹Preuzeto iz literature (Čupić, 2020).

Tijekom faze učenja, modelu se predaje skup uzoraka za učenje (engl. *training set*). Svaki uzorak skupa možemo notirati kao $(\mathbf{x}, y)_i$ što predstavlja i -ti uzorak iz skupa za učenje gdje \mathbf{x} predstavlja vektor ulaznih podataka, a y vrijednost pridružena ulaznom vektoru, odnosno oznaku (engl. *label*). Ulazni podatak često može biti kompleksne naravi što znači da ga je potrebno "razbiti" u manje cjeline. Takav postupak se naziva vađenje značajki (engl. *feature extraction*). Rezultat navedenog je kolekcija značajki koja se zatim predaje određenom algoritmu strojnog učenja. Npr. ulazni podatci mogu biti razne gume poput gume automobila, gume bicikla, gume traktora i sl. a značajka bi onda mogla biti volumen gume, oblik gume, boja gume i sl. Algoritam strojnog učenja će na temelju predanih značajki pokušati optimizirati parametre odabranog modela kako bi se isti mogao koristiti u sljedećoj fazi.

Faza iskorištavanja modela predstavlja fazu u kojoj se treba utvrditi radi li naučeni model s uzorcima nad kojima nije učio, tj. je li razvio sposobnost generalizacije (engl. *generalization*). Ako model dodijeli točnu oznaku za većinu ulaznih podataka, onda možemo reći da je razvio svojstvo generalizacije. No što ako u velikoj mjeri ne uspije točno povezati izlaze s ulazima? Ako dođe to takve pojave, kažemo da je model pretreniran (engl. *overfitting*). Do navedene pojave dolazi kada model nauči previše nad skupom uzoraka za učenje, tj. uzorke za učenje nauči savršeno raspoznavati dok za nove uzorke ne uspijeva dati željene rezultate. Postoji nekoliko načina kako se može utjecati da model razvije svojstvo generalizacije a spomenut ćemo samo jedan od njih, *unakrsna provjera*.

Unakrsna provjera (engl. *cross-validation*) je postupak koji se koristi tijekom faze učenja modela. Ideja je da se dio uzoraka iz skupa za učenje razdijeli u novi skup, skup za provjeru (engl. *validation set*). Skup za provjeru često sadrži od 20% do 30% svih uzoraka iz skupa za učenje (Čupić, 2020). Tijekom faze učenja modela, model će i dalje učiti samo nad skupom za učenje, dok ćemo mu povremeno dati i podatke iz skupa za provjeru čisto da se vidi koliko griješi nad njima. Važno je napomenuti da model nad uzorcima iz skupa za provjeru neće učiti, tj. neće optimizirati parametre već će biti prisiljen da nauči generalizirati. Postavlja se pitanje do kada će faza učenja modela onda trajati i odgovor je vrlo jasan. Trajat će do trenutka kada pogreška nad skupom za provjeru počinje rasti. Dijagram na slici 3.2 prikazuje kako model uči kroz epohe².

²Epoha je ništa drugo nego jedna iteracija učenja, ali u strojnom učenju se često koristi izraz epoha pa ćemo se držati konvencije.



Slika 3.2: Kretanje pogrešaka pri učenju strojnog modela³

Također postoji mogućnost da neki od parametara modela utječu na njegovu složenost. Shodno tomu se uvodi još jedan skup, skup za testiranje (engl. *testing set*). Ideja je sljedeća: skup podataka razdijelimo u sva tri do sada navedena skupa i normalno provedemo fazu učenja modela uz malu promjenu. Umjesto da se zadovoljimo samo jednim naučenim modelom, pokušavamo pronaći onaj model koji će najbolje minimizirati pogrešku nad skupom za testiranje (Čupić, 2020).

3.2. Problemi nadziranog učenja

Zadatak nadziranog učenja je sljedeći: pretpostavimo da imamo skup za učenje koji se sastoji od N različitih parova ulaza i izlaza;

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), \dots, (\mathbf{x}_N, y_N)$$

gdje \mathbf{x}_i predstavlja i -ti vektor ulaznih podataka dok y_i i -ti željeni izlaz koji je generiran nekom nepoznatom funkcijom $y = f(x)$.

Potrebno je pronaći takvu funkciju h koja će najbolje aproksimirati funkciju f .

Funkcija h naziva se hipoteza (engl. *hypothesis*) i pripada nekom skupu svih hipoteza \mathbf{H} koje donekle aproksimiraju zadanu funkciju f , neke bolje, neke lošije. U ovom trenutku uključujemo skup za testiranje u proces učenja jer je potrebno pronaći onu hipotezu za koju će aproksimacija biti najbolja, tj. ukupna pogreška generirana tijekom učenja modela bit će minimalna. Time će se osigurati da model poprimi svojstvo generalizacije. Često se pokazuje da je funkcija f stohastička (engl. *stochastic*), što

³Preuzeto iz literature (Čupić, 2016).

znači da ne ovisi uvijek samo o varijabli x pa se mora koristiti i uvjetovana vjerojatnost $P(Y|x)$ tijekom faze učenja modela (Russell i Norvig, 2009)⁴.

Kao što je rečeno ranije, y_i predstavlja i -ti željeni izlaz za i -ti ulaz te može poprimiti vrijednost broja ili vrijednost iz nekog konačnog skupa podataka. Ako je slučaj da je izlaz broj, radi se o problemu koji se naziva *regresija*, a ako je slučaj da je izlaz vrijednost iz nekog konačnog skupa podataka, onda se radi o problemu koji se naziva *klasifikacija*.

3.2.1. Regresija

Regresija (engl. *regression*) je tehnika, tj. algoritam nadziranog učenja koji se koristi kada je potrebno predvidjeti određeni iznos neke funkcije u odnosu na određene parametre. Dakle, dane podatke potrebno je aproksimirati nekom proizvoljnom funkcijom h koja je često polinomijalna. Postoje različite vrste regresije pa spomenimo neke od njih.

Prije svega, postoji regresija s jednom varijablom (engl. *regression with one variable*) i regresija sa više varijabli (engl. *multivariate regression*). Razlika je očita, no pokažimo to na kratkom primjeru. Recimo da želimo modelirati cijene obiteljskih kuća u okolini Zagreba. Za to su nam potrebni neki parametri na temelju kojih ćemo moći vršiti predikcije kao npr. broj kvadratnih metara (m^2), broj spavaćih soba, blizina vrtića i škola, povezanost sa javnim prijevozom i sl. U slučaju regresije s jednom varijablom, za modeliranje bi uzeli samo jedan od navedenih parametara, dok bi u slučaju regresije sa više varijabli uzeli sve ili dio navedenih. Dakle, očigledno je da je regresija s jednom varijablom konkretan slučaj regresije sa više varijabli.

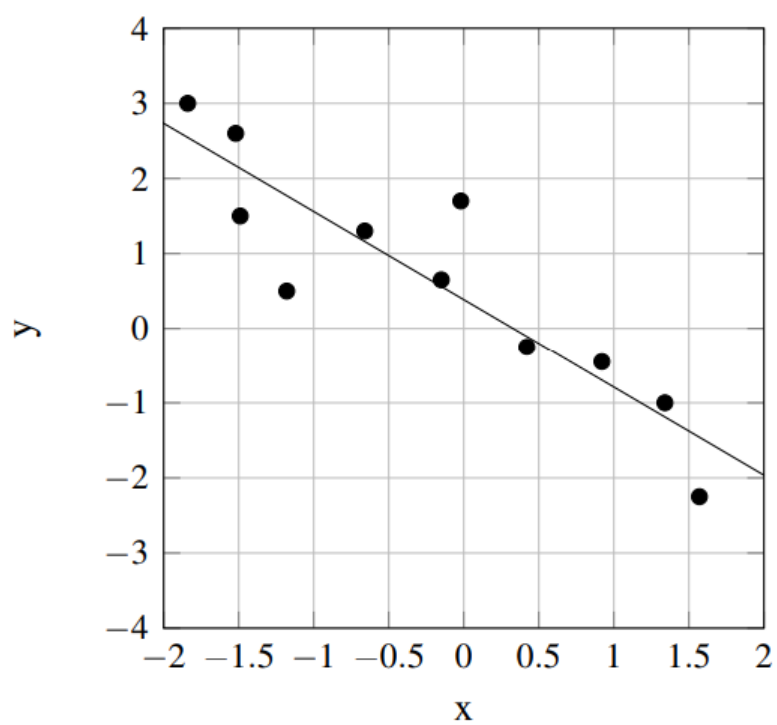
Linearna regresija (engl. *linear regression*) je tip regresije koji koristi linearnu krivulju (pravac), tj. polinom prvog stupnja kao aproksimacijsku krivulju.

Polinomijalna regresija (engl. *polynom regression*) je tip regresije koji koristi polinom reda r kao aproksimacijsku krivulju.

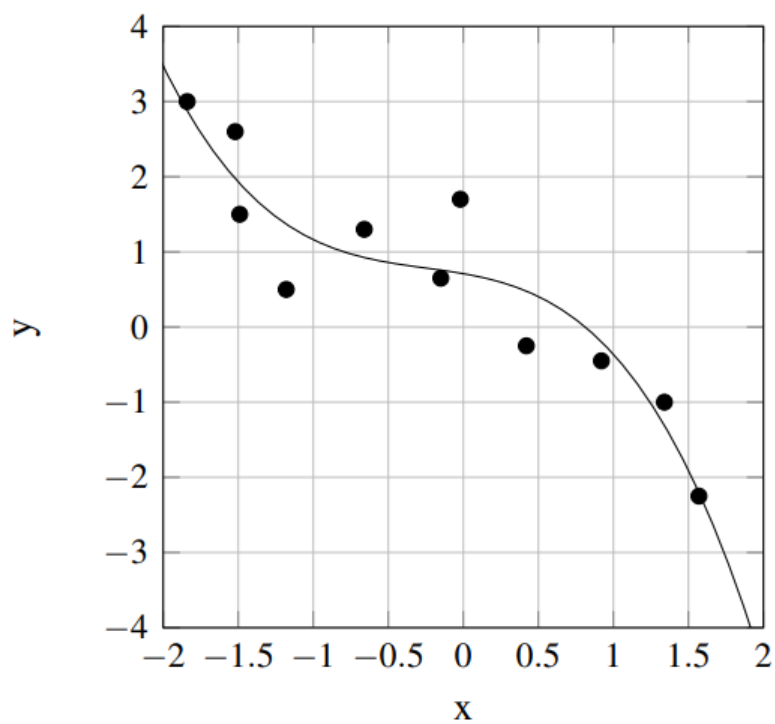
Na slikama 3.3 i 3.4 su prikazani navedeni tipovi regresija.

Dakle, regresija je metoda koja se koristi za predviđanje numeričkih ishoda.

⁴Iz poglavlja 18.2 Supervised learning.



Slika 3.3: Primjer linearne regresije⁵



Slika 3.4: Primjer polinomijalne regresije sa kubnim polinomom⁶

⁵Preuzeto iz literature (Čupić, 2020).

⁶Preuzeto iz literature (Čupić, 2020).

3.2.2. Klasifikacija

Klasifikacija (engl. *classification*) je tehnika, tj. algoritam nadziranog učenja koji se koristi kada je određen ulazni podatak potrebno smjestiti odnosno klasificirati kao pripadnika određenog razreda. Izlazi tako organiziranog modela često poprimaju diskretne vrijednosti za razliku od regresije i takav model onda nazivamo klasifikator (engl. *classifier*). Razlikujemo nekoliko različitih tipova klasifikatora:

1. binarni klasifikator,
2. višerazredni klasifikator te
3. klasifikator više oznaka.

Binarni klasifikator (engl. *binary classifier*) je tip klasifikatora koji zna odrediti pripada li ulazni podatak jednom od ukupno dva moguća razreda. Npr. sustav kao ulaze prima slike na kojima se nalazi ili mačka ili pas. Jednom naučeni klasifikator bi na temelju predane slike morao moći odrediti što se nalazi na slici, mačka ili pas. Naravno uz uvjet da se preda slika mačke ili psa.

Višerazredni klasifikator (engl. *multi-class classifier*) je tip klasifikatora koji zna odrediti pripada li ulazni podatak jednom od barem tri moguća razreda. Dakle, mora postojati minimalno tri različita razreda kako bi se klasifikator nazivao višerazrednim. U ovom radu ćemo se više fokusirati upravo na navedenom klasifikator koji će naučiti raspoznati tri moguća razreda, no o tome nešto kasnije.

Klasifikator više oznaka (engl. *multi-label classifier*) je tip klasifikatora koji predstavlja generalizaciju višerazrednog klasifikatora. Pogledajmo sljedeći primjer klasifikacije filmova za bolje razumijevanje. Pretpostavimo da film može imati sljedeće oznake: (12+), (16+) i (18+) gdje svaka oznaka predstavlja minimalan broj godina kojih gledaoc filma mora imati. Određivanje oznake za pojedini film je rezultat korištenja višerazrednog klasifikatora jer je potrebno odrediti pripada li film samo jednom od navedene tri oznake. No što ako film poželimo klasificirati pomoću žanrova? Tada nam višerazredni klasifikator neće puno pomoći jer jedan film može imati više oznaka kao npr. "drama" i "komedija" i sl. Tada ćemo koristiti klasifikator više oznaka.

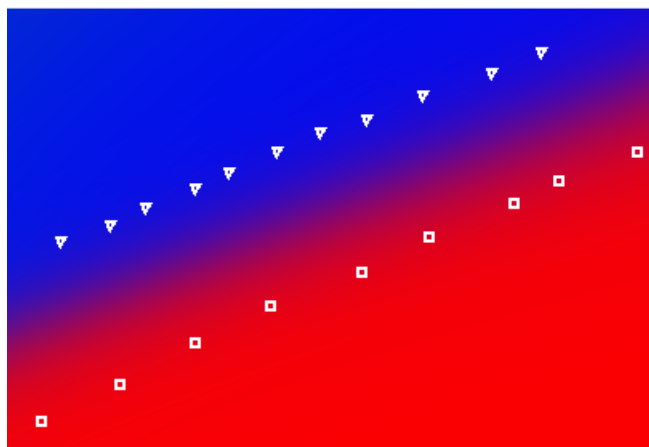
Klasifikaciju, kao i regresiju, možemo podijeliti na *linearnu klasifikaciju* i *nelinearnu klasifikaciju*, odnosno polinomijalnu.

Linearna klasifikacija (engl. *linear classification*) je tip klasifikacije u kojoj se granica između razreda aproksimira pomoću linearna krivulje, tj. pravca.

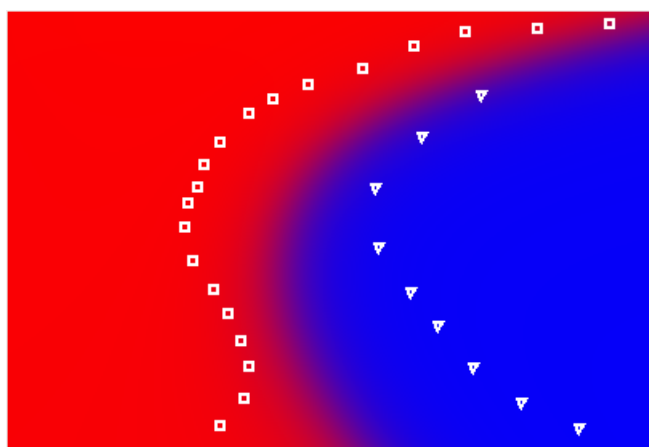
Nelinearna klasifikacija (engl. *non-linear classification*) je tip klasifikacije u kojoj se granica između razreda aproksimira pomoću nelinearne krivulje koja je često polinomijalna.

Ono što je zajedničko u oba slučaja je postojanje granice između razreda koja se u literaturi naziva *decizijska granica* (Čupić, 2016). O oblicima decizijskih granica će biti riječ kasnije.

Na slikama 3.5 i 3.6 su prikazani primjeri klasifikacija.



Slika 3.5: Primjer linearne klasifikacije⁷



Slika 3.6: Primjer nelinearne klasifikacije⁸

⁷Rezultat programske implementacije.

⁸Rezultat programske implementacije.

3.3. Algoritmi

U području nadziranog učenja postoji veliki broj algoritama koji uvelike pridonose boljem učenju. Neki od poznatijih su:

- Stroj potpornih vektora (engl. *support-vector machine, SVM*),
- Stabla odluke (engl. *decision trees*),
- Naivni Bayesov klasifikator (engl. *naive Bayes classifier*),
- Slučajne šume (engl. *random forests*),
- Algoritam k-najbližih susjeda (engl. *k-nearest neighbors algorithm*),
- Umjetna neuronska mreža (engl. *artificial neural network*) i mnogi drugi.

Vidimo da postoji jako velika paleta algoritama od kojih su neki složeniji od drugih, neki daju bolje rezultate od drugih, no niti jedan od algoritama nije u stanju uvijek dati najbolje rezultate za proizvoljan problem nadziranog učenja (Supervised learning, 2020). Zbog navedenog vrijedi tzv. *No free lunch* teorem koji nalaže da je za probleme koji zahtijevaju mnogo računalnih resursa poput pretraživanja prostora stanja i optimizacije parametara složenost⁹ u prosjeku jednaka za sve metode (No free lunch theorem, 2020).

U sljedećem poglavlju ćemo se detaljnije baviti algoritmom *umjetna neuronska mreža*. Pogledat ćemo kako je nastao koncept umjetnog neurona, strukturu umjetnog neurona, arhitekturu unaprijedne neuronske mreže te algoritam kojim unaprijedna neuronska mreža uči.

⁹Često se ovako definirani problemi nazivaju *NP-teškim problemima* (engl. *Non-deterministic polynomial-time problems*).

4. Umjetne neuronske mreže

U drugom poglavlju (*Pregled područja*) dotaknuli smo se razlika između simboličke umjetne inteligencije i strojnog učenja i rekli smo kako simbolička umjetna inteligencija koristi simbolički pristup, tj. sve iskaze pokušava predočiti u mnoštvo pravila pomoću ljudima čitljivim simbolima, dok se strojno učenje bavi raznim algoritmima pomoću kojih se računalni sustav uči. U ovom poglavlju bavit ćemo se algoritmom strojnog učenja koji koristi tzv. konektivistički pristup.

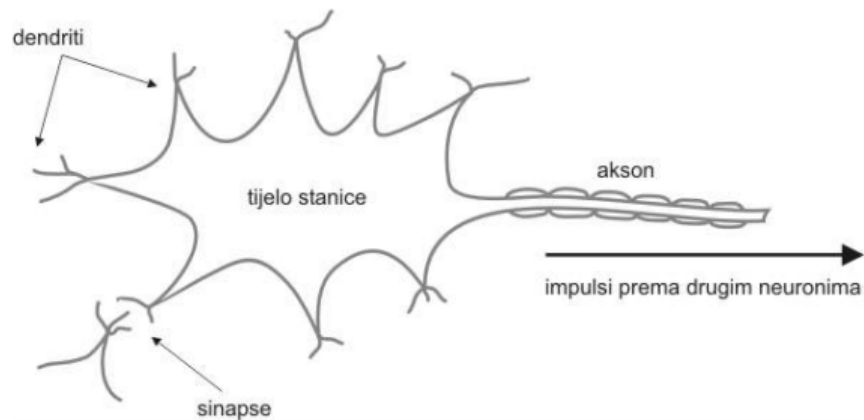
4.1. Motivacija

Ideja za konektivizmom (engl. *connectivism*) potaknuta je na temelju strukture ljudskog mozga te zbog izrazite brzine kojom mozak obrađuje podražaje. Danas je poznato da u ljudskom mozgu postoji oko 10^{11} neurona te 10^{15} međusobnih veza što znači da je svaki neuron u prosjeku povezan sa 10^4 različitih veza (Čupić et al., 2013). To upravo dokazuje da mozak predstavlja jedan kompleksan i paralelan sustav zbog čega je konektivizam dobio ime te čemu algoritam umjetne neuronske mreže teži. Također, koncept ljudskog mozga nije uzet samo radi svoje izrazite brzine prilikom obrade podražaja, već i zbog činjenice da ga se može učiti na temelju velikog broja podataka koji su često ispunjeni raznim šumovima, što je realna situacija, te ga time potaknuti da razvije svojstvo generalizacije. Područje koje se bavi proučavanjem umjetnih neuronskih mreža naziva se neuro-računarstvo (engl. *neuro-computing*) koje je jedno od grana mekog računarstva (engl. *soft-computing*).

4.2. Biološki neuron

Neuroni (engl. *neurons*) su glavne stanice mozga i živčanog sustava. Zadaća im je da primaju podražaje iz okoline, da prenose određene akcije od mozga prema našim mišićima i da vode računa o pretvaranju električnog potencijala između pojedinih neurona u svim koracima procesa vođenja. Neuron je izgrađen od posebne strukture koja

se sastoji od: *tijela stanice* (ili češće *soma*), *dendrita*, *aksona*, i *sinapse*. Struktura biološkog neurona prikazana je na slici 4.1.



Slika 4.1: Biološki neuron¹

Dendriti (engl. *dendrites*) su ulazni dijelovi neurona, tj. strukture koje prihvaćaju impulse od drugih neurona preko veza koje se nazivaju sinapse. Suma svih ulaznih impulsa, tj. potencijala određuje hoće li dotični neuron biti aktiviran ili ne.

Aksoni (engl. *axons*) su izlazni dijelovi neurona, tj. strukture u kojima se generira određeni električni potencijal koji se dalje prosljeđuje sljedećem neuronu preko sinoptičkih veza.

Sinapse (engl. *synapses*) su, kao što je rečeno ranije, veze između dendrita jednog neurona i aksona drugog neurona (Woodruff, 2019).

4.3. Umjetni neuron

4.3.1. Povijesni pregled

MCP neuron

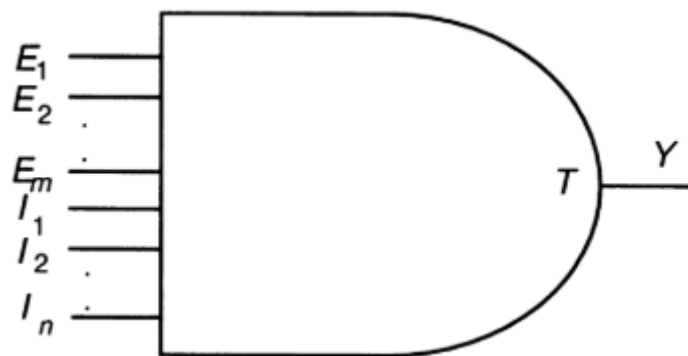
1943. dva znanstvenika, Warren S. McCulloch i Walter H. Pitts, u svojem radu *Logical Calculus of Ideas Immanent in Nervous Activity* definiraju strukturu prvog umjetnog neurona koji se u literaturi često naziva kao *MCP* neuron. Na slici 4.2 prikazana je struktura MCP neurona. Dendriti su predstavljeni ulazima označenih slovima

¹Preuzeto iz literature (Čupić, 2016).

E od riječi ekscitacijski² (engl. *excitatory*) te **I** od riječi inhibicijski³ (engl. *inhibitory*), tijelo stanice je predstavljeno slovom **T** od riječi prag (engl. *threshold*) dok je akson predstavljen izlazom koji je označen slovom **Y** bez posebnog značenja.

Ekscitacijski ulazi uzrokuju da neuron postane aktivan, tj. da se neuron "pali" (engl. *fire*), dok inhibicijski ulazi uzrokuju da neuron postane neaktivan, tj. da se neuron "ne pali". Točnije, ako postoji barem jedan inhibicijski ulaz koji je aktivan, tada će neuron sigurno biti neaktivan neovisno o svim ostalim ulazima. Također, ako niti jedan od inhibicijskih ulaza nije aktivan, onda će neuron postati aktivan samo u slučaju kada je suma svih ekscitacijskih ulaza veća ili jednaka pragu, T (McCulloch i Pitts, 1943). Matematički rečeno (Picton, 2000):

$$Y = \begin{cases} 1, & \sum_{i=1}^n I_i = 0 \text{ i } \sum_{j=1}^m E_j \geq T \\ 0, & \text{inače.} \end{cases} \quad (4.1)$$



Slika 4.2: MCP neuron⁴

MCP neuron mogao je modelirati osnovne Booleove logičke operacije kao što su operacija *i* (engl. *and*), operacija *ili* (engl. *or*) i operacija negacije (engl. *not*) pa se može reći da neuron predstavlja logička vrata (engl. *logic gate*) što je često korištena struktura prilikom konstrukcija uređaja poput računala koji mogu rješavati kompleksne logičke izraze. Shodno tomu se razvilo vjerovanje da se ljudski mozak sastoji upravo

²ekscitacija - uzbuđenje ili aktivirano stanje zbog određenog podražaja (ekscitacija. Hrvatska enciklopedija, mrežno izdanje. Leksikografski zavod Miroslav Krleža, 2020. Pristupljeno 13. 5. 2020. <<http://www.enciklopedija.hr/Natuknica.aspx?ID=17398>>).

³inhibicija - kočenje prijenosa živčanih impulsa (inhibicija. Hrvatska enciklopedija, mrežno izdanje. Leksikografski zavod Miroslav Krleža, 2020. Pristupljeno 13. 5. 2020. <<http://www.enciklopedija.hr/Natuknica.aspx?ID=27450>>).

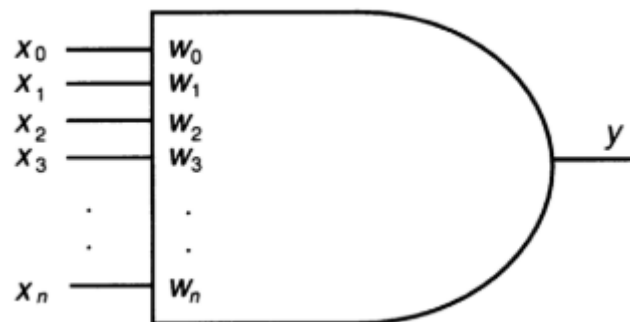
⁴Preuzeto iz literature (Picton, 2000) iz poglavlja 1.4.2 Biologically inspired neural networks.

od mnoštva logičkih vrata, no nisu uzimali u obzir činjenicu da takva konstrukcija ne može biti učena te da nema svojstvo generalizacije (Picton, 2000).

Postoji još jedna bitna stavka oko strukture MCP neurona koja je prešutno korištena, a to su vrijednosti koje poprimaju ulazi i izlaz. Vrijednosti koje mogu poprimiti su iz skupa $\{0, 1\}$ i niti jedne druge, dok vrijednost praga može biti proizvoljan realan broj. To se implicitno moglo zaključiti iz spomenutih primjena na Boolove logičke operacije te iz formule (4.1).

ADALINE

1960. dva znanstvenika, Bernard Widrow i Marcian E. Hoff, u svojem radu *Adaptive Switching Circuits* na temelju MCP neurona definiraju jednu od prvih neuronskih mreža nazvanu *ADALINE* (engl. *ADaptive LINEar Elements*). Na slici 4.3 prikazana je struktura jednog neurona iz neuronske mreže ADALINE. Dendriti su predstavljeni ulazima označenim slovom x , tijelo stanice predstavljeno je težinama označenim slovom w , a akson je predstavljen izlazom označenim slovom y . Dakle, uočljivo je da postoje neke novine za razliku od MCP neurona. Ulogu ekscitacijskih odnosno inhibicijskih ulaza sada imaju težine koje poprimaju vrijednosti iz realnog skupa brojeva, ulazi i izlaz poprimaju vrijednosti iz skupa $\{-1, 1\}$ i niti jedne druge, osim za ulaz x_0 čija je vrijednost uvijek 1, dok prag više nije eksplicitno zadan, već se u definiciju implicitno uključuje simbolom w_0 koji predstavlja konstantan pomak (engl. *bias*) (Picton, 2000).



Slika 4.3: ADALINE neuron⁵

Definiranje težina, koje odgovaraju svaka svom ulazu, revolucionarno je otkriće jer se time pokazalo da se tako definirani neuroni mogu učiti što uvelike olakšava izvedbu računalnih sustava jer više nije potrebna kvantiteta da bi sustav funkcionirao,

⁵Preuzeto iz literature (Picton, 2000) iz poglavlja 1.4.2 Biologically inspired neural networks.

već kvalitetna izvedba. Umjesto praćenja koliko je ekscitacijskih odnosno inhibicijskih ulaza aktivno, definirana je težinska suma *net* koja je opisana sljedećom formulom:

$$net = \sum_{i=0}^n w_i * x_i \quad (4.2)$$

Vrijednost *net*-a je onda transformirana u izlaz *y* pomoću nelinearne funkcije koja pozitivne vrijednosti preslikava u 1, dok negativne vrijednosti ili vrijednosti koje su jednake 0 preslikava u -1. Matematička definicija dana je sljedećom formulom (Picton, 2000):

$$y = \begin{cases} 1, & \text{net} > 0 \\ -1, & \text{net} \leq 0 \end{cases} \quad (4.3)$$

Koncept težina definiran je 1949. kada je Donald Hebb u svome dijelu *The Organization of Behavior: A Neuropsychological Theory* pokušao definirati da kada je akson neurona A dovoljno blizu da aktivira neuron B i to ponavlja veći broj puta dolazi do metaboličkih promjena tako da se povećava efikasnost neurona A u aktiviranju neurona B (Hebb, 1949). Često se navedeno pravilo naziva hebbov princip učenja i može se definirati pomoću ADALINE neurona. Pravilo je dano sljedećom formulom (Picton, 2000):

$$w_i = \sum_{p=1}^p x_{ip} * y_p \quad (4.4)$$

Svaka težina w_i računa se kao težinska suma svih ulaza x_i sa željenim izlazom y i to za svaki uzorak po skupu svih uzoraka koji su označeni slovom p . Ako uzmemo u obzir da ulazi i izlazi poprimaju vrijednosti iz skupa $\{-1, 1\}$, onda je očito da iznos težine w_i raste ako su ulaz za tu težinu x_i i izlaz uzorka y_p aktivni, odnosno smanjuje ako su neaktivni. Iako se čini da je pravilo uspješno, pokazuje se da se njime ne uspijeva dobro naučiti mrežu jer se u izračun ne uzimaju stvarni izlazi neurona, već samo željeni izlazi.

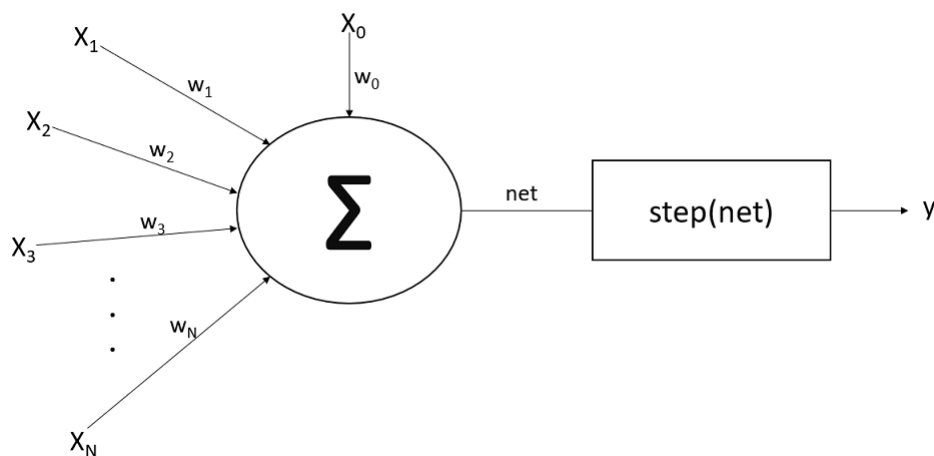
Zbog navedenog nedostatka hebbovog principa, Widrow i Hoff su došli do ideje da bi se težine morale namještati pomoću pogreške (engl. *error*) koja se događa između željenog izlaza i stvarnog izlaza te algoritma gradijentni spust. Navedeni princip nazvan je: *delta pravilo* (engl. *the delta rule*) ili widrow-hoffovo pravilo, a danas je poznato i pod nazivom: pravilo najmanjeg srednjeg kvadrata (engl. *least mean square rule*). Time su uspjeli postići da mreža ADALINE pokazuje jako dobre rezultate tijekom procesa učenja, a to se implicitno potkrepljuje riječju (engl. *ADaptive*) iz imena

što znači prilagodljiv (Picton, 2000). Detaljnije definicije i izvode formula pokazat ćemo u poglavlju gdje se govori o učenju unaprijedne neuronske mreže algoritmom propagacije pogreške unatrag.

Glavni nedostatak ADALINE neuronske mreže je činjenica da se njome uređaji konstruiraju isključivo fizičkim putem što je puno puta nepraktično i skupo za projektirati. U narednim poglavljima reći ćemo nešto o drugačijem izvodu umjetnih neuronskih mreža pomoću računalnih programa koje je puno lakše uklopiti u električne uređaje.

4.3.2. Perceptron

1958. znanstvenik Frank Rosenblatt u svojem radu *The perceptron: a probabilistic model for information storage and organization in the brain* na temelju MCP neurona i hebbovog principa učenja definira strukturu, perceptron⁶, čija je glavna ideja da se računalnim programom pokuša konstruirati sustav koji će moći učiti na temelju danih podražaja, dakle iz iskustva. U to vrijeme je ta ideja bila suluda i izazvala je mnoge kontroverze među pionirima umjetne inteligencije. The New York Times, dnevne američke novine, te iste godine opisuju perceptron kao zametak (engl. *embryo*) računala koji, jednom kada se kreira i usavrši, bi predstavljao prvi računalni sustav koji bi bio u stanju opažati, prepoznati i identificirati svoju okolinu bez ljudskog osposobljavanja ili kontrole⁷.



Slika 4.4: Umjetni neuron perceptron-a⁸

⁶Dolazi od latinske riječi *perceptio* što u prijevodu znači razumijeti (engl. *understand*).

⁷The New York Times, *Electronic 'Brain' Teaches Itself*, stranica 9, srpanj 13, 1958.

⁸Izrađeno pomoću powerpoint-a i na temelju modela umjetnog neurona iz literature (Čupić, 2016).

Struktura perceptrona je, kao što je rečeno ranije, motivirana strukturom MCP neurona. Na slici 4.4 nalazi se prikaz perceptrona koji je predstavljen kao generalizirani MCP neuron gdje je uloga ekscitacijskih, odnosno inhibicijskih ulaza zamijenjena težinama slično kao i kod ADALINE neurona. Novost u strukturi je prikaz funkcije skoka (engl. *step*) koja je implicitno postojala i kod izvornog MCP neurona, no ovdje je navedena eksplicitno jer se takav oblik koristi u literaturi. Također, ovakva struktura se u literaturi često naziva jednoslojni perceptron (engl. *single-layer perceptron*), a kasnije ćemo se susresti i sa višeslojnim perceptronom (engl. *multi-layered perceptron*).

Glavna zamisao koju je Rosenblatt dokumentirao u svom radu bila je izvesti nekakvu vrstu binarnog klasifikatora⁹ i definirati algoritam kojim bi se taj isti klasifikator uspio naučiti (Rosenblatt, 1958). Ideja algoritma je da se težine ažuriraju pomoću pogreške koja se događa između željenog izlaza i stvarnog izlaza te se onda pomnoži sa nekom konstantom koju ćemo definirati u nastavku. Pravilo učenja perceptrona dano je algoritmom 1.

Algorithm 1 Pravilo učenja perceptrona¹⁰

1. Ciklički prolazi kroz svih N uzoraka za učenje, jedan po jedan.
2. Klasificiraj trenutni uzorak.
 1. Ako se klasificira korektno, ne mijenjaj težine i
 1. ako je to N -ti uzastopni uzorak klasificiran korektno, prekini učenje,
 2. inače prijeđi na sljedeći uzorak.
 2. Ako se ne klasificira korektno, korigiraj težine perceptrona prema sljedećem izrazu:

$$w_i(k+1) \leftarrow w_i(k) + \eta * (d - y) * x_i \quad (4.5)$$

U procesu učenja klasifikatora može se uočiti da su spomenuti samo uzorci za učenje, no sada znamo i da postoji još nekoliko mehanizama koji uvelike doprinose da klasifikator razvije svojstvo generalizacije. Neki od tih mehanizama je npr. uporaba skupa za provjeru i skupa za testiranje, ali navedenim algoritmom smo htjeli pokazati elementarni proces učenja pa se time ovdje nismo htjeli zamarati. Ono što nas najviše zanima je interpretacija formule (4.5). Index i predstavlja i -tu vrijednost, tj. w_i je i -ta težina za i -ti ulaz što se može vidjeti i na slici 4.8, k je k -ta iteracija procesa učenja, d

⁹Vidi 3.2.2 Klasifikacija.

¹⁰Iz literature (Čupić, 2016).

predstavlja željeni (engl. *desired*) izlaz neurona, dok y predstavlja stvarni izlaz neurona te η (eta) predstavlja parametar koji se zove stopa učenja (engl. *learning rule*). To je pozitivan realan broj malog iznosa (najčešće između 0.001 i 0.5) kojim se regulira kolikom mjerom će se ažurirati trenutna težina neurona. Ako je η jako mali broj, onda će proces učenja biti prespor, tj. težine će se mizerno malo ažurirati. Ako je η jako veliki broj, onda će proces učenja zasigurno divergirati jer će se težine ažurirati povećim brojevima i vrijednosti će "eksplodirati". Navedeni parametar ima jako bitnu ulogu prilikom optimizacije raznih *NP* teških problema koji koriste inačice algoritma gradijentni spust o kojemu će biti nešto riječi kasnije kada se dotaknemo algoritma propagacije pogreške unatrag.

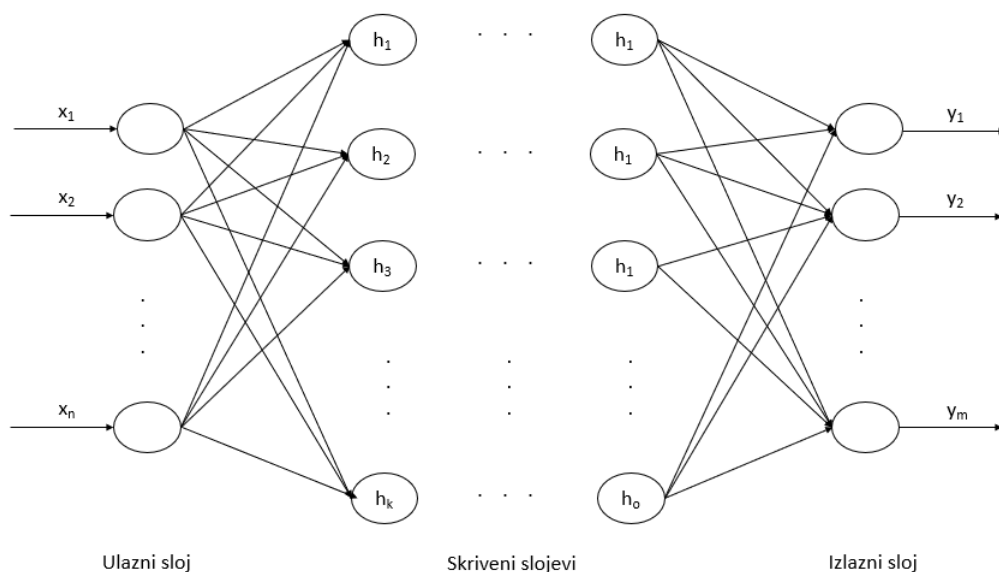
Dakle, Rosenblatt je uspješno uspio iznijeti svoje teze iako su mnogi sumnjali u njih. No, mreže koje su etiketirane kao jednoslojni perceptron imaju jednu veliku manu kao i prethodno navedene ADALINE neuronske mreže, a to je činjenica da uspješno djeluju samo kada je problem linearno interpretabilan, tj. kada su izlazi dotičnih neuronskih mreža linearno odvojivi (engl. *linearly separable*), a ADALINE neuronska mreža to još dodatno potvrđuje i imenom *LINEar Elements*. Ako se kratko vratimo na poglavlje u kojem smo govorili o tipovima klasifikacija, onda možemo vidjeti da smo problem klasifikacije podijelili na linearnu i nelinearnu klasifikaciju. Jednoslojni perceptroni su tipovi linearnih klasifikatora što znači da bi problem na slici 3.5 uspješno uspjeli klasificirati, dok problem na slici 3.6 ne bi jer se radi o nelinearnoj klasifikaciji. Primjeri koji se često vežu uz jednoslojne perceptrone jesu osnovne Booleove logičke operacije i, ili i negacija. Svaka od njih ima linearno interpretabilan izlaz te ih je bilo moguće realizirati. U to vrijeme je to bilo značajno otkriće, kao što je i spomenuto kod opisa primjene ADALINE neuronskih mreža. S druge strane, npr. operacija isključivo ili (XOR) nema linearno interpretabilan izlaz te se ne može realizirati jednoslojnim perceptronom kao takvim, već se mora konstruirati dvoslojni perceptron kako bi realizacija bila korektna ¹¹. No, ideja o višeslojnom perceptronu je tada bila tek u procesu stvaranja jer pravilo učenja perceptrona koje je predložio Rosenblatt je djelovalo samo na jednoslojne perceptrone. 1969. Marvin L. Minsky i Seymour A. Papert u svojem djelu *Perceptrons* jasno iznose nedostatke perceptrona koji je Rosenblatt definirao i to ponajviše o nemogućnostima rješavanja nelinearno odvojivih problema i učenja višeslojnih perceptrona. Time je i započela prva zima u eri umjetne inteligencije (engl. *1st AI winter*) kada se konektivistički pristup na kratko obustavio te se rodio simbolizam. Konektivizam će biti po strani sve negdje do 1986. kada će se po prvi put defini-

¹¹Zanimljivi primjeri mogu se pronaći u skripti Umjetne neuronske mreže (Čupić, 2016).

rati algoritam kojim će višeslojni perceptron moći učiti, a to je algoritam propagacije pogreške unatrag.

4.4. Općenito o umjetnim neuronskim mrežama

Do sada smo se upoznali sa modelom umjetnog neurona i prvim modelima neuronskih mreža, ADALINE i perceptron i vidjeli smo razne nedostatke koji isti sadrže. Mali broj neurona može davati dosta uspješne rezultate, no ograničeni su na jako jednostavne i specifične probleme kao npr. konstrukcija jednostavnih Booleovih logičkih operacija što smo spomenuli ranije. Ljudski mozak je evidentno puno kompleksniji od strukture par neurona te se time došlo na ideju povezati veliki broj neurona u jednu veliku strukturu, tj. algoritam koji će biti vjerni prikaz ljudskog mozga, a takav algoritam je nazvan umjetna neuronska mreža. Na slici 4.5 je prikazana tipična građa jedne unaprijedne višeslojne potpuno povezane umjetne neuronske mreže.



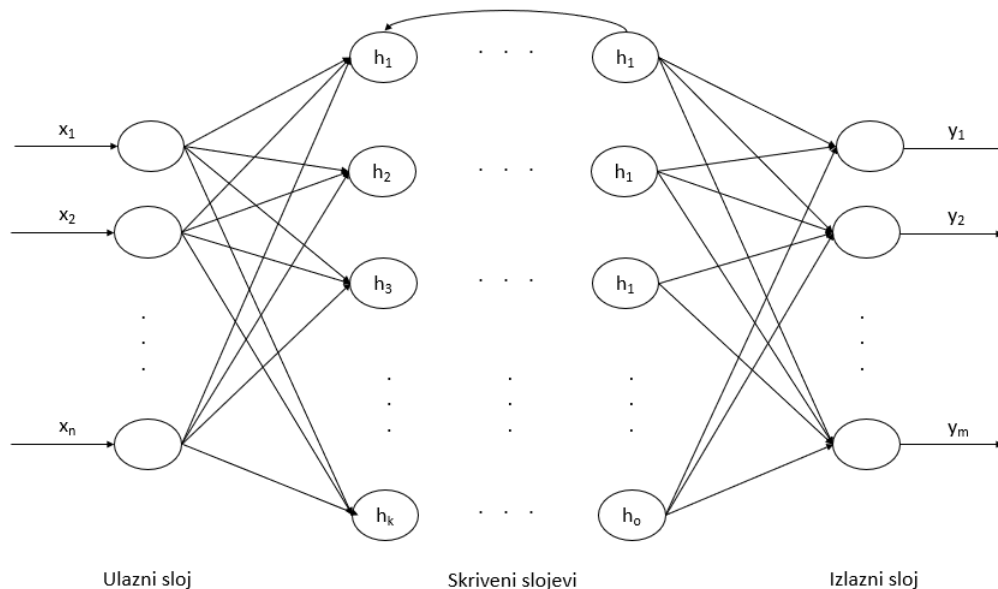
Slika 4.5: Tipična građa umjetne neuronske mreže¹²

Dakle, mreža se sastoji od tri različite vrste slojeva: ulazni, skriveni i izlazni sloj. Ulazni sloj je sloj na koji se dovode uzorci za koje želimo da mreža nešto napravi i ulazi su prikazani slovom \mathbf{x} . Izlazni sloj je sloj na kojemu će se nalaziti izlazi za navedeni uzorak u ulaznom sloju i prikazani su slovom \mathbf{y} . Skriveni sloj je sloj koji jedini nije vidljiv izvana i u kojemu se sva logika umjetne neuronske mreže događa.

¹²Izrađeno pomoću powerpoint-a i na temelju modela umjetne neuronske mreže iz literature (Čupić et al., 2013).

Neuroni skrivenog sloja označeni su slovom **h** i broj slojeva koji se nalaze u skrivenom sloju može biti proizvoljan. Primjetite koji se indeksi koriste kod enumeracije svakog od slojeva. U ulaznom sloju neuronska mreža očekuje n ulaza, u izlaznom sloju se očekuje m izlaza, dok u skrivenom sloju broj neurona može varirati po sloju, stoga su korišteni proizvoljni indeksi označeni slovima **k** i **o**.

U navedenoj strukturi pretpostavili smo da je svaki neuron nekog sloja povezan sa svakim neuronom njemu sljedećeg sloja i to samo tog sloja, osim izlaznog sloja koji je posljednji sloj neuronske mreže. Takve mreže ćemo nazivati slojevitim potpuno povezanim mrežama i njihova je zadaća da za neki fiksni ulaz generiraju neki stabilan izlaz (Čupić et al., 2013). U matematičkom smislu navedena operacija naziva se preslikavanje i to je glavna tema ovog rada kojoj ćemo se uskoro posvetiti detaljnije. Međutim, postoje i različite vrste mreža koje ne moraju biti niti slojevite niti potpuno povezane, a mogu imati i cikluse. Kod takvih mreža neuroni mogu biti povezani i sa neuronima prethodnog sloja te se time postiže da neuronske mreže budu promjenjive kroz vrijeme i da imaju mogućnost čuvanja nekog određenog stanja, tj. možemo reći da razvijaju nekakvu vrstu memorije. Na slici 4.6 je prikazana struktura jedne cikličke umjetne neuronske mreže koja je gotovo identična neuronskoj mreži na slici 4.5, no dodana je jedna povratna veza kojom se cijela ideja mreže mijenja.



Slika 4.6: Tipična građa cikličke umjetne neuronske mreže¹³

Često spominjemo sintagme poput struktura ili građa umjetne neuronske mreže,

¹³Izrađeno pomoću powerpoint-a i na temelju modela umjetne neuronske mreže iz literature (Čupić, 2016).

no u raznim literaturama koristi se sintagma arhitektura umjetne neuronske mreže pa ćemo i mi ubuduće koristiti upravo takav opis. Arhitektura umjetne neuronske mreže definira se na sljedeći način: uvijek započinje brojem neurona ulaznog sloja, zatim se definiraju skriveni slojevi i to svaki sloj je predstavljen brojem neurona tog sloja i na kraju se nalazi broj neurona izlaznog sloja. Npr. jedna od arhitektura mogla bi biti $2 \times 5 \times 4 \times 3$ i nju tumačimo na sljedeći način: mreža se sastoji od četiri sloja; ulaznog, dva skrivena i izlaznog. Ulazni sloj sastavljen je od 2 neurona, prvi skriveni sloj je sastavljen od 5 neurona, drugi skriveni sloj je sastavljen od 4 neurona i izlazni sloj je sastavljen od 3 neurona. Onda bi npr. jednu inačicu rosenblattovog perceptrona mogli definirati kao 3×1 jer on ne sadrži niti jedan skriveni sloj. Kasnije ćemo pogledati na koji način različite arhitekture utječu na uspješnost jedne unaprijedne neuronske mreže tijekom učenja klasifikatora.

4.5. Unaprijedna neuronska mreža

Najčešće korišteni tip umjetne neuronske mreže je unaprijedna višeslojna potpuno povezana umjetna neuronska mreža (engl. *feedforward multilayered fully connected artificial neural network*) čiju smo generalnu arhitekturu već upoznali na slici 4.5. Često se ista u literaturi naziva i samo unaprijedna neuronska mreža zbog lakšeg dokumentiranja rada pa ćemo i mi preuzeti tu konvenciju, a implicitno ćemo biti svjesni punog imena i svojstva. Također, unaprijedna neuronska mreža je pravi primjer višeslojnog perceptrona jer se očigledno sastoji od više slojeva te se može učiti danas već nekolicinom uspješnih algoritama.

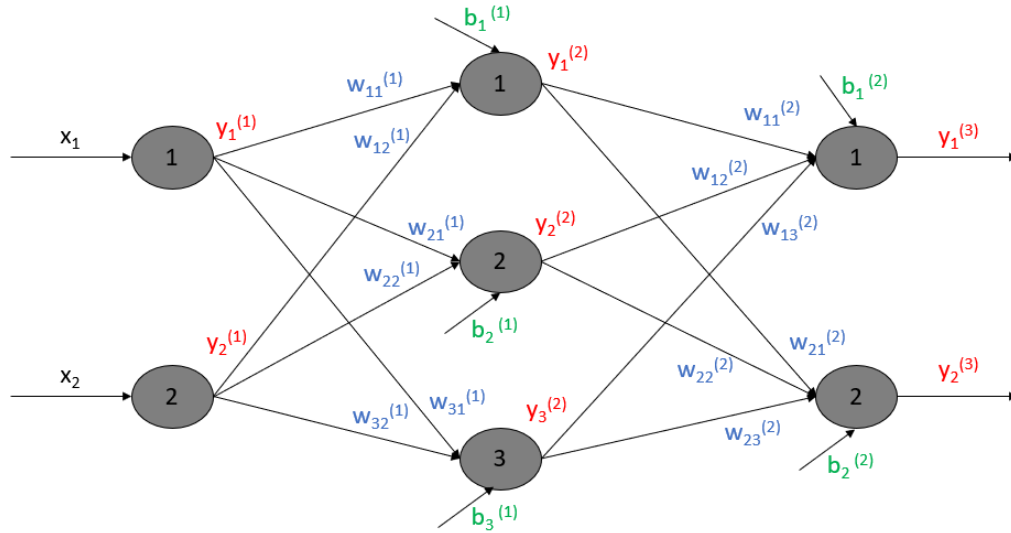
Glavna zadaća unaprijedne neuronske mreže, kao što i samo ime nalaže, je da ulaze propagira kroz sve slojeve sve do izlaznog pri čemu se izvršava operacija mapiranja između ulaza i izlaza mreže na mnoštvo različitih načina. Također, sve operacije koje ćemo susresti se vrlo efikasno mogu prikazati matričnim računima čime se računalna izvedba implementacije i učenja unaprijedne neuronske mreže čini jako jednostavnim, no ponekad ipak to neće biti tako jednostavno jer postoji nekolicina parametara koje je potrebno "nariktati" različitim metoda koje često i nisu zahvalne.

Pogledajmo jednu jednostavnu unaprijednu neuronsku mrežu s arhitekturom $2 \times 3 \times 2$ na slici 4.7. Crnom bojom su označeni ulazi u ulaznom sloju, plavom bojom su označene težine između dva neurona, zelenom bojom je označen bias, a crvenom bojom su označeni izlazi iz neurona pa krenimo redom s notacijama:

$w_{jk}^{(l)}$: težina između k -tog neurona $(l-1)$ -tog sloja i j -tog neurona l -tog sloja.

$b_j^{(l)}$: bias j-tog neurona l-tog sloja.

$y_j^{(l)}$: izlaz j-tog neurona l-tog sloja.



Slika 4.7: 2x3x2 arhitektura unaprijedne neuronske mreže¹⁴

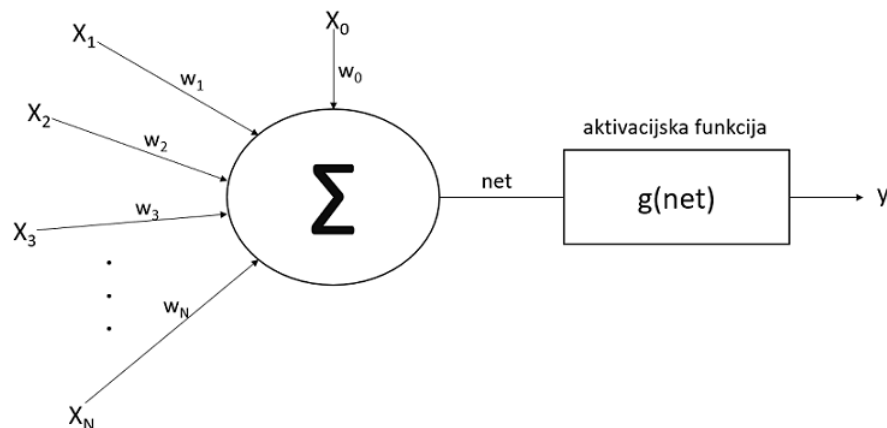
Prije nego se dotaknemo matematičke podloge, definirajmo ponovno strukturu umjetnog neurona.

Do ovog trenutka upoznali smo se samo sa dvije vrste neurona, a to su MCP neuron koji se koristio u rosenblattovu modelu perceptron i ADALINE neuron. Pokazali smo i da se tako definirani neuroni koriste samo kada je problem kojeg rješavamo linearno interpretabilan. To se događa zbog funkcije skoka koja se nalazi na izlazu neurona te zbog nje učenje višeslojnog perceptrona nije moguće jer funkcija skoka nije derivabilna te algoritam propagacije pogreške unatrag neće biti moguće provesti, ali o tome ćemo reći nešto detaljnije kada se dotaknemo učenja unaprijedne neuronske mreže. Zbog navedenog ćemo uvesti jednu generalnu strukturu umjetnog neurona koji ne mora koristiti funkciju skoka kao funkciju izlaza. Na slici 4.8 nalazi se ažurirana struktura umjetnog neurona. Vidimo da je struktura gotovo identična već spomenutom neuronu na slici 4.4 osim oznake za funkciju na izlazu. Funkcije koje se nalaze na izlazu neurona nazivaju se aktivacijske funkcije¹⁵ i njihova je zadaća, kao što i samo ime nalaže, aktiviranje, odnosno deaktiviranje neurona ovisno o svojstvima funkcije i vrijednosti argumenta funkcije, tj. net vrijednosti i takvu funkciju ćemo označavati sa

¹⁴Izrađeno pomoću powerpoint-a i na temelju modela umjetne neuronske mreže iz literature (Nielsen, 2015).

¹⁵U starijim literaturama češći naziv bio je prijenosna funkcija.

slovom g ¹⁶. Dakle, MCP neuron kojeg smo nedavno obrađivali kao aktivacijsku funkciju koristi funkciju skoka. U narednom poglavlju ćemo dati pregled često korištenih aktivacijskih funkcija te njihovih formula kao i derivacija istih, no u ovom trenutku je jedino bitno da smo svjesni da one postoje radi matričnog prikaza unaprijedne neuronske mreže.



Slika 4.8: Umjetni neuron¹⁷

Prisjetimo se formule za izračun net vrijednosti koju smo definirali kada smo pričali o ADALINE neuronu (4.2):

$$net = \sum_{i=0}^n w_i * x_i$$

Primjetite da sumacija u ovom slučaju ide od 0 jer, ako se prisjetite strukture ADALINE neurona, postoji težina w_0 koja predstavlja bias i ulaz x_0 konstante vrijednosti koja iznosi 1. Na slici 4.7 bias pojedinog neurona je eksplicitno naveden slovom **b** te zbog efikasnosti prikaza je izostavljen konstantni ulaz. S obzirom da se ADALINE neuron jedino mogao nalaziti u jednoslojnoj neuronskoj mreži, korišteno je slovo **x** kao ulaz neurona, a u našem slučaju mreža je višeslojna te ćemo ulaze svih slojeva osim ulaznog notirati slovom **y**. Ako se pitate zašto je tomu tako, onda se dobro pitajte jer je ta činjenica prešućena. Naime, ideja samog ulaznog sloja je da prenese dane ulaze na izlaz neurona te će tako izlazi ulaznog sloja biti jednaki ulazima, odnosno ne postoji nikakvo računanje net vrijednosti jer težine nisu prisutne. Također, svaki neuron osim ulaznih sadrži svoju net vrijednost. Uzevši u obzir navedene promjene i navedene notacije, net vrijednost pojedinog neurona možemo definirati na sljedeći način:

¹⁶Općenito, svaki neuron ima svoju aktivacijsku funkciju te zbog toga bi ih trebali indeksirati, ali zbog jednostavnosti ćemo ih sve označavati samo s g .

¹⁷Izrađeno pomoću powerpoint-a i na temelju modela umjetnog neurona iz literature (Čupić, 2016).

$$net_j^{(l)} = \sum_{k=1}^n w_{jk}^{(l)} * y_k^{(l)} + b_j^{(l)} \quad (4.6)$$

gdje $net_j^{(l)}$ predstavlja net vrijednost j -tog neurona l -tog sloja pri čemu l predstavlja sloj koji nije ulazni, a sumacija se proteže do n pri čemu n predstavlja broj neurona u $l-1$ sloju. Nakon izračuna, net vrijednost je potrebno "provući" kroz aktivacijsku funkciju kako bi se dobio izlaz za dotični neuron. Izračun za izlaz dan je sljedećom formulom:

$$y_j^{(l)} = g\left(\sum_{k=1}^n w_{jk}^{(l)} * y_k^{(l)} + b_j^{(l)}\right) \quad (4.7)$$

Evidentno je da se formula može zapisati i matricno po svakom sloju neuronske mreže izuzev ulaznog. Uvedimo još nekoliko notacija:

$\mathbf{W}^{(l)}$: matrica svih težina l -tog sloja gdje su retci težine iz svih neurona $(l-1)$ -tog sloja u jedan neuron l -tog sloja.

$\vec{b}^{(l)}$: vektor svih bias vrijednosti l -tog sloja.

$\vec{net}^{(l)}$: vektor svih net vrijednosti l -tog sloja.

$\vec{y}^{(l)}$: vektor svih izlaza l -tog sloja.

Nakon uvedenih notacija, formule možemo prikazati na sljedeći način:

$$\vec{net}^{(l+1)} = \mathbf{W}^{(l)} * \vec{y}^{(l)} + \vec{b}^{(l)} \quad (4.8)$$

$$\vec{y}^{(l+1)} = g(\mathbf{W}^{(l)} * \vec{y}^{(l)} + \vec{b}^{(l)}) \quad (4.9)$$

Radi lakšeg razumijevanja, pokušajmo pokazati kako će izgledati matrica težina, vektor bias-a i vektor izlaza ako promatramo vezu između prvog i drugog sloja, odnosno ulaznog i skrivenog sa slike 4.7.

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{bmatrix}, \quad \vec{y}^{(1)} = \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \end{bmatrix}, \quad \vec{b}^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}$$

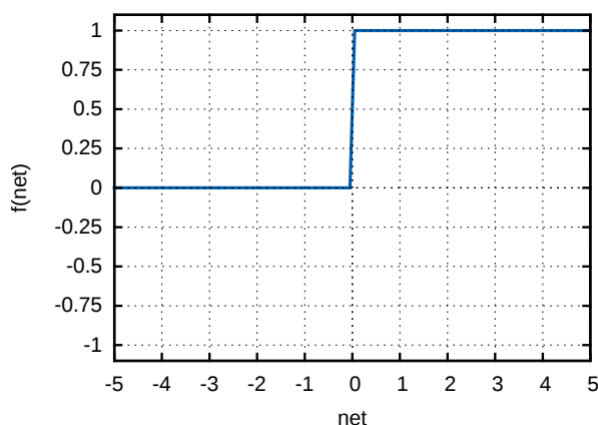
Vidimo da dimenzije svih matrica odgovaraju za operaciju produkta matrica, odnosno operaciju zbroja matrica. Isti postupak ponavljamo za svaki sloj unaprijedne neuronske mreže sve dok ne dobijemo izlaze izlaznog sloja kada stajemo sa računom pri čemu još jednom valja naglasiti da izlazi jednog sloja postaju ulazi onom sljedećem.

4.5.1. Aktivacijske funkcije

Aktivacijska funkcija (engl. *activation function*), kao što smo i spomenuli kod računa unaprijedne neuronske mreže, je funkcija koja se nalazi tik prije izlaza neurona i određuje hoće li dotični biti aktiviran, odnosno deaktiviran, tj. hoće li se "paliti" ili ne. Danas postoji veliki broj aktivacijskih funkcija, no mi ćemo spomenuti one koje se najviše koriste i koje su korištene u ovom radu, a u poglavlju s rezultatima ćemo pokazati kako neke od njih utječu na sam izgled decizijskih granica.

Binarna funkcija skoka

Binarna funkcija skoka (engl. *binary step function*) je funkcija koja se temelji na nekom fiksnom pragu i s njome smo se susreli kod MCP neurona i rosenblattovog perceptrona. Neuroni koji koriste ovu kao aktivacijsku funkciju nazivaju se TLU (engl. *threshold logic unit*) neuroni. Na slici 4.9 nalazi se graf jedne takve funkcije. Nedostatak ovakve funkcije kod učenja višeslojnog perceptrona je što nije derivabilna i što koristi binarne izlaze pa višerazredna klasifikacija nije moguća.



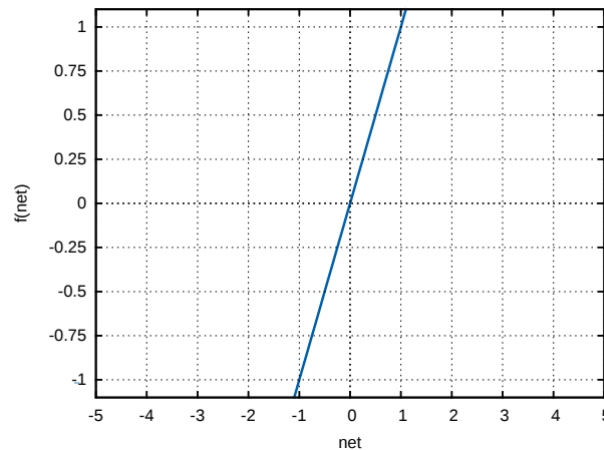
Slika 4.9: Funkcija skoka $f(net) = \text{step}(net)$ ¹⁸

Linearne aktivacijske funkcije

Linearne aktivacijske funkcije su funkcije oblika $y = ax + b$ i one generiraju izlaz koji je proporcionalan ulazu pa su zbog toga nešto efikasnije od funkcije skoka. No, i one imaju dosta nezgodne nedostatke kada govorimo o učenju višeslojnog perceptrona. One jesu derivabilne, ali njihova derivacija je konstanta što za algoritam propagacije

¹⁸Preuzeto iz literature (Čupić, 2016).

pogreške unatrag nikako nije dobro. Također, ako svaki sloj koristi linearnu aktivacijsku funkciju, onda ne postoji nikakva razlika među slojevima jer će na izlazu opet biti linearna aktivacijska funkcija zbog pravila o linearnosti koji kaže da je linearna kombinacija linearnih funkcija također linearna funkcija. Posljedica toga je da linearna aktivacijska funkciju višeslojni perceptron pretvara u jednoslojni te se tada radi o običnom problemu linearne regresije. Na slici 4.10 nalazi se graf jedne takve funkcije.



Slika 4.10: Funkcija identiteta $f(net) = net$ ¹⁹

Nelinearne aktivacijske funkcije

Nelinearne aktivacijske funkcije su najčešće korištene funkcije kada se radi o učenju višeslojnog perceptrona jer uvelike čine unaprijednu neuronsku mrežu ekspresivnijom i ostavljaju mogućnost kreiranja jako dubokih neuronskih mreža po broju skrivenih slojeva. Većina njih je derivabilna što znači da se mogu koristiti u učenju algoritmom propagacije pogreške unatrag te njihove derivacije se mogu zapisati pomoću samih funkcija što znatno olakšava i implementaciju istih.

1. Sigmoidalna/logistička funkcija (prikazana na slici 4.11):

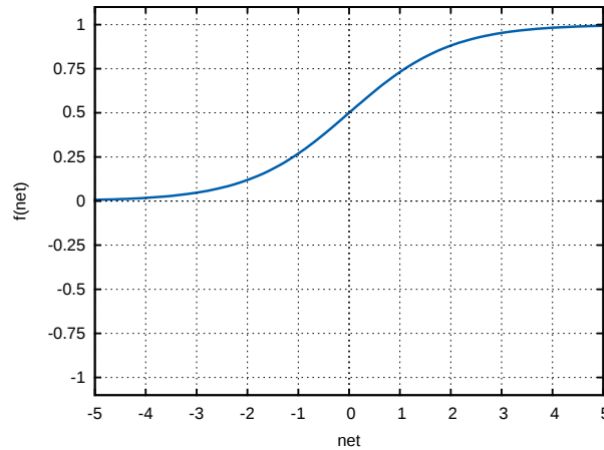
$$\text{sigm}(net) = \frac{1}{1 + e^{-net}}, \quad (4.10)$$

$$\text{sigm}'(net) = \text{sigm}(net) * (1 - \text{sigm}(net)). \quad (4.11)$$

Izlaze mapira na vrijednosti iz intervala (0, 1). Kod ove funkcije se javlja problem iščezavajućeg gradijenta (engl. *vanishing gradient*) što znači da će jako velike ulaze, odnosno jako male izlaze mapirati na vrijednosti oko 1, odnosno

¹⁹Preuzeto iz literature (Čupić, 2016).

oko 0 i time će se učenje mreže znatno usporiti jer će ažuriranje težina gotovo pa stagnirati. Još jedan problem koji se javlja je centriranje svih izlaza oko vrijednosti 0.5 a ne oko 0. To se pokazuje dosta lošim svojstvom kroz literature. Neuroni koji koriste sigmoidalnu funkciju kao aktivacijsku nazivaju se sigmoidalni neuroni. Također, izvođenje ove funkcije je računalno skupo i danas se slabo više koristi.



Slika 4.11: Sigmoidalna funkcija $f(net) = \text{sigm}(net)$ ²⁰

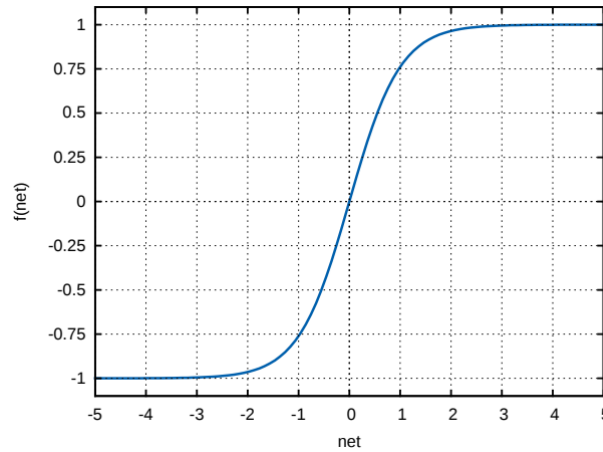
2. Funkcija tangens hiperbolni (prikazana na slici 4.12):

$$\tanh(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}} = 2 * \text{sigm}(2 * net) - 1, \quad (4.12)$$

$$\tanh'(net) = 1 - \tanh^2(net). \quad (4.13)$$

Tangens hiperbolni je zapravo mala modifikacija sigmoidalne funkcije s razlikom u tome da se izlazi centriraju oko 0 što ostavlja više prostora za različite vrste klasifikacija. Ostala svojstva su gotovo identična i tangens hiperbolni se pokazuje nešto bržim za razliku od sigmoidalne funkcije pa se češće koristi u učenjima unaprijedne neuronske mreže.

²⁰Preuzeto iz literature (Čupić, 2016).



Slika 4.12: Tangens hiperbolni $f(net) = \tanh(net)$ ²¹

3. Funkcija zglobnica (engl. *ReLU, Rectified Linear Unit*) (prikazana na slici 4.13):

$$relu(net) = \max(0, net), \quad (4.14)$$

$$relu'(net) = \begin{cases} 1 & net > 0, \\ 0 & \text{inače.} \end{cases} \quad (4.15)$$

Iako izgleda kao linearna funkcija, funkcija zglobnica ima svoju derivaciju koja se može koristiti u algoritmu propagacije pogreške unatrag. Računalno je dosta povoljnija od prethodne dvije te je zbog toga danas sve korištenija i to pogotovo tijekom učenja dubokih neuronskih mreža. No, funkcija ima jedan nezgodan problem a to je da "ubija" sve neurone čiji su izlazi jako mali ili negativni te se time učenje u nekim slučajevima izrazito otežava jer gradijent postaje 0. Tom problemu se doskočilo na način da se ipak propuštaju određeni neuroni koji imaju negativne izlaze. Takva funkcija se onda naziva funkcija propusna zglobnica (engl. *leaky ReLU*) (prikazan na slici 4.14) i ima sljedeću definiciju:

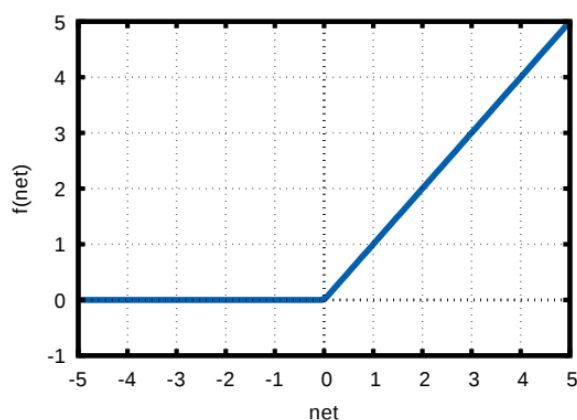
$$lrelu(net) = \begin{cases} net & net > 0, \\ \alpha * net & \text{inače.} \end{cases}, \quad (4.16)$$

$$lrelu'(net) = \begin{cases} 1 & net > 0, \\ \alpha & \text{inače.} \end{cases} \quad (4.17)$$

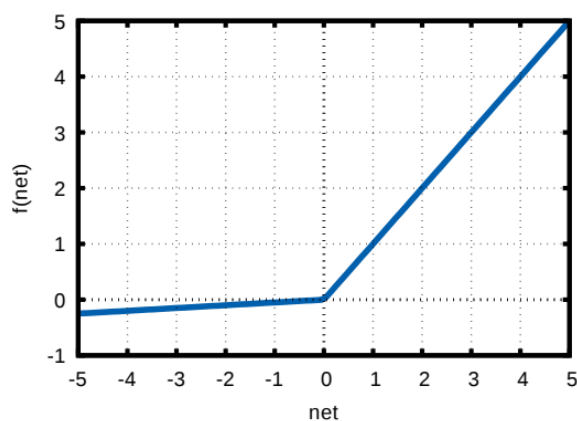
α je parametar koji je često jako mali realan broj i u raznim literaturama poprima vrijednosti poput 0.1 i 0.2. Funkcija propusna zglobnica je zgodna, no pokazuje

²¹Preuzeto iz literature (Čupić, 2016).

se da ne daje konzistentne rezultate kada mora raditi predikcije negativnih vrijednosti.



Slika 4.13: Funkcija zglobnica $f(net) = \text{relu}(net)$ ²²



Slika 4.14: Funkcija propusna zglobnica $f(net) = \text{lrelu}(net)$ ²³

4. Funkcija softmax:

Funkcija softmax je funkcija koja ulaze transformira na vrijednost između 0 i 1 baš kao i sigmoidalna funkcija pa je graf praktički identičan te ga nema smisla ponovno prikazivati. No, ideja softmax funkcije je da vrati postotke u kojima se dotični ulazi nalaze i onda suma svih tih postotaka mora biti stopostotna. Koristi se samo kao aktivacijska funkcija izlaznog sloja i vrlo je pogodna kada se radi o problemima višerazredne klasifikacije jer je onda preko postotaka lako za odrediti koji razred se nalazi na izlazu. Definicija funkcije je malo drugačija

²²Preuzeto iz literature (Čupić, 2016).

²³Preuzeto iz literature (Čupić, 2016).

od ostalih zato što kao ulazni argument prima vektor vrijednosti a ne samo jednu vrijednost i glasi ovako (Bendersky, 2016):

$$S(\vec{a}) : \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} \rightarrow \begin{bmatrix} S_1 \\ S_2 \\ \dots \\ S_N \end{bmatrix} \quad \text{gdje je}$$

$$S_j = \frac{e^{a_j}}{\sum_{k=1}^n e^{a_k}}, \quad (4.18)$$

$$S(\vec{a})' = S(\vec{a}) * (1 - S(\vec{a})). \quad (4.19)$$

Vidimo dakle da je i derivacija softmax funkcije gotovo identična derivaciji sigmoidalne, uzevši u obzir da je ulazni argument vektor.

4.5.2. Inicijalizacija i normalizacija

Do sada smo definirali kako za neke ulaze dobiti nekakvu vrstu izlaza koji će biti u rasponu određenom nekom od navedenih aktivacijskih funkcija, no nigdje nismo rekli kakvi ti ulazi trebaju biti. Kojem rasponu brojeva trebaju pripadati? Koje početne vrijednosti trebaju imati težine svih slojeva? Odgovore na ta pitanja ćemo sada pokušati odgovoriti.

Normalizacija

Normalizacija (engl. *normalization*) je postupak kojim se vrijednosti nekih podataka adaptiraju na neki fiksni interval kao npr. na interval $(0, 1)$. To je potrebno napraviti prvenstveno jer je tako računalu znatno jednostavnije izvoditi operacije s matricama, a i najčešće na izlazima očekujemo vrijednosti između -1 i 1 pa ako su ulazne vrijednosti puno veće od navedenog, onda mreža vrlo vjerojatno neće dobro naučiti ili će se mučiti pronaći prave težine. U implementaciji je korišten tip normalizacije u kojoj stvarnu vrijednost ulaza podijelimo maksimalnom mogućom vrijednosti za ulaz. Npr. točka koja ima ekranske koordinate x i y će se normalizirati na način da ćemo x koordinatu podijeliti širinom prozora, dok ćemo y koordinatu podijeliti visinom prozora.

Inicijalizacija

Inicijalizacija (engl. *initialization*) težina igra još veću ulogu u učenju unaprijedne neuronske mreže od normalizacije ulaza. Zašto nam je uopće potrebna inicijalizacija

težina? Pretpostavimo da su sve težine inicijalno postavljene na 0. U tom slučaju će svi neuroni biti simetrični i naučit će značajke te zbog toga učenje neće biti moguće. Općenito, bilo kakva inicijalizacija nekom konstantom će rezultirati istim rezultatom. Što će se dogoditi ako težine inicijaliziramo jako velikim, odnosno jako malim brojevima? Ako provedemo takvu inicijalizaciju, dogodit će se problem kojeg nazivamo eksplozivajući, odnosno iščezavajući gradijent. Ako nastupi eksplozivajući gradijent, tada će učenje unaprijedne neuronske mreže drastično divergirati, dok će kod iščezavajućeg gradijenta učenje biti izrazito sporo. Zbog navedenih razloga je vrlo jasno da se težine moraju kvalitetno inicijalizirati. Postoje mnoge metode inicijalizacije, no spomenut ćemo samo onu koja je korištena u implementaciji, a to je Xavier inicijalizacija koja vrijednosti težina generira normalnom razdiobom gdje očekivanje, μ , iznosi 0, a standardna devijacija, σ , iznosi $\sqrt{\frac{1}{n(l+1)}}$. Nazivnik predstavlja broj neurona u prijašnjem sloju (Katanforoosh i Kunin, 2019).

4.6. Učenje unaprijedne neuronske mreže

U ovom trenutku trebali bismo biti upoznati sa općenitim načinom rada unaprijedne neuronske mreže, a to je aproksimacija, odnosno predikcija nekog izlaza za neki proizvoljan ulaz. Upoznali smo se i sa normalizacijom ulaznih podataka, inicijalizacijom težina među neuronima i prikazali smo matematičku interpretaciju svega navedenog. Ostalo nam je još objasniti kako se unaprijedna neuronska mreža uči i to ćemo objasniti u ovom poglavlju.

Unaprijedna neuronska mreža može se učiti na nekoliko načina, a najpoznatiji su učenje algoritmima evolucijskog računanja i učenje algoritmom gradijenti spust i algoritmom propagacije pogreške unatrag. U ovom radu fokus je bio na zadnja dva.

Prije nego što krenemo na algoritam propagacije pogreške unatrag, prisjetimo se kratko koja je temeljna ideja algoritma gradijentni spust.

4.6.1. Gradijentni spust

Gradijentni spust (engl. *gradient descent*) je algoritam koji se koristi za minimizaciju funkcija preko njihovih derivacije. Pretpostavimo da postoji funkcija $f(x)$ koju želimo minimizirati. Tada ćemo x za koji je funkcija f minimalna tražiti sljedećom formulom:

$$x \leftarrow x - \alpha \cdot \frac{\partial}{\partial x} f(x) \quad (4.20)$$

pri čemu je α stopa učenja koja je često mali realan broj. Dan je primjer funkcije sa jednom varijablom, no algoritam djeluje i na funkcije s više varijabli. Glavna ideja algoritma je da se po grafu pomičemo u smjeru koji pokazuje na točku gdje će derivacija biti približno jednaka 0, a brzinu kretanja određuje stopa učenja koju smo obradili nešto ranije.

4.6.2. Algoritam propagacije pogreške unatrag

Algoritam propagacije pogreške unatrag (engl. *backpropagation algorithm*) je trenutno možda najkorišteniji algoritam učenja unaprijedne neuronske mreže. 1986. Rumelhart zajedno sa dvojicom prijatelja u djelu *Parallel Distributed Processing* po prvi put definira termin *backpropagation* i učenje višeslojnog perceptrona. Oko te godine konektivistički pristup ponovno dobiva zalet i prva zima umjetne inteligencije polako prolazi. U ovom radu nećemo ulaziti u detalje izvoda samog algoritma, već ćemo dati općeniti pregled gotovih formula, pseudokod algoritma te neke od inačica. Sve notacije koje smo uveli kod definicije unaprijedne neuronske mreže će vrijediti i ovdje.

Pretpostavimo da imamo skup uzoraka koji se sastoji od N različitih uzoraka. Svaki uzorak je predstavljen vektorom ulaza i vektorom očekivanih izlaza i označimo ga kao $(x_{s,1}, x_{s,2}, \dots, x_{s,n}) \rightarrow (d_{s,1}, d_{s,2}, \dots, d_{s,m})$, gdje su ulazi označeni slovom \mathbf{x} , a željeni izlazi slovom \mathbf{d} . Indeks s predstavlja s -ti uzorak, a n i m broj neurona na ulazu, odnosno na izlazu neuronske mreže²⁴. Također, pretpostavimo da je mreža sastavljena od L slojeva gdje je L -ti sloj izlazni sloj. Da bi mreža mogla učiti, moramo definirati određenu funkciju kazne pomoću koje će mreža dobiti povratnu informaciju o tome koliko dobro ili loše uči te na temelju te pogreške korigirati iznose težina. Funkcija kazne za jedan uzorak definirana je na sljedeći način:

$$E(s) = \frac{1}{2} \sum_{i=1}^m (d_{s,i} - y_{s,i}^{(L)})^2. \quad (4.21)$$

S obzirom da želimo učiti nad cijelim skupom uzoraka, onda je potrebno definirati funkciju kazne za cijeli skup:

$$E = \frac{1}{N} \sum_{s=1}^N E(s) = \frac{1}{2N} \sum_{s=1}^N \sum_{i=1}^m (d_{s,i} - y_{s,i}^{(L)})^2. \quad (4.22)$$

Ako se prisjetite formule za izračun izlaza neurona pomoću net vrijednosti i aktivacijske funkcije, onda vrlo lako možemo zaključiti da funkcija kazne ovisi o težinama

²⁴Pogledaj sliku 4.5.

jer svaki izlaz ovisi o težinama i ulazima prijašnjeg sloja, a svaki ulaz je zapravo izlaz prijašnjeg sloja pa onda i on posljedično ovisi o težinama i tako sve do prvog skrivenog sloja:

$$\begin{aligned}\vec{y}^{(L)} &= g(\mathbf{W}^{(L-1)} \cdot \vec{y}^{(L-1)} + \vec{b}^{(L-1)}), \\ \vec{y}^{(L-1)} &= g(\mathbf{W}^{(L-2)} \cdot \vec{y}^{(L-2)} + \vec{b}^{(L-2)}), \\ &\dots \\ \vec{y}^{(2)} &= g(\mathbf{W}^{(1)} \cdot \vec{y}^{(1)} + \vec{b}^{(1)}).\end{aligned}$$

Sada kada to znamo, onda je potrebno minimizirati funkciju, tj. algoritmom gradijentni spust pronaći one težine za koje je funkcija kazne minimalna:

$$w_{jk}^{(l)} \leftarrow w_{jk}^{(l)} - \alpha \cdot \frac{\partial E}{\partial w_{jk}^{(l)}}. \quad (4.23)$$

Možda se sada pitate zašto nam je potreban backpropagation algoritam kada očigledno možemo gradijent spusta direktno primijeniti. Ta pretpostavka je u redu u slučaju kada nemamo mnoštvo uzoraka za učenje i kada arhitektura mreže nije kompleksna, no što ako imamo desetke tisuća uzoraka sa nekoliko skrivenih slojeva gdje svaki ima minimalno deset neurona? Tada bi računanje jedne po jedne težine bilo izrazito računalno zahtjevno a to želimo izbjeći. Upravo zbog navedenog koristimo backpropagation algoritam jer njime uvelike olakšavamo izračun parcijalnih derivacija za sve težine, a time i samo učenje mreže.

Kada smo govorili o ADALINE neuronu, spomenuli smo da se on, kao i MCP neuron, uče delta pravilom. Delta pravilom je moguće učiti samo jednoslojne perceptrone, stoga se ono moralo malo nadograditi tako da se može koristiti u za učenje višeslojnog perceptrona.

Pokažimo kako bi izgledala parcijalna derivacija funkcije kazne u zadnjem sloju prije izlaznog:

$$\frac{\partial E}{\partial w_{jk}^{(L-1)}} = \frac{\partial}{\partial w_{jk}^{(L-1)}} \left[\frac{1}{2N} \sum_{s=1}^N \sum_{i=1}^m (d_{s,i} - y_{s,i}^{(L)})^2 \right] \quad (4.24)$$

$$= \frac{1}{2N} \sum_{s=1}^N \sum_{i=1}^m 2 \cdot (d_{s,i} - y_{s,i}^{(L)}) \cdot (-1) \cdot \frac{\partial y_{s,i}^{(L)}}{\partial w_{jk}^{(L-1)}} \quad (4.25)$$

$$= -\frac{1}{N} \sum_{s=1}^N \sum_{i=1}^m (d_{s,i} - y_{s,i}^{(L)}) \cdot \frac{\partial y_{s,i}^{(L)}}{\partial w_{jk}^{(L-1)}} \quad (4.26)$$

S obzirom da težina $w_{jk}^{(L-1)}$ utječe samo na $y_{sj}^{(L)}$, onda svaka parcijalna derivacija $\frac{\partial y_{sj}^{(L)}}{\partial w_{jk}^{(L-1)}}$ za slučaj da je $i \neq j$ iščezava te vrijedi:

$$\sum_{i=1}^m \frac{\partial y_{si}^{(L)}}{\partial w_{jk}^{(L-1)}} = \frac{\partial y_{sj}^{(L)}}{\partial w_{jk}^{(L-1)}}.$$

Dakle, sada možemo pisati:

$$\frac{\partial E}{\partial w_{jk}^{(L-1)}} = -\frac{1}{N} \sum_{s=1}^N (d_{sj} - y_{sj}^{(L)}) \cdot \frac{\partial y_{sj}^{(L)}}{\partial w_{jk}^{(L-1)}} \quad (4.27)$$

Preostalu parcijalnu derivaciju možemo riješiti pomoću pravila ulančavanja:

$$\frac{\partial y_{sj}^{(L)}}{\partial w_{jk}^{(L-1)}} = \frac{\partial y_{sj}^{(L)}}{\partial net_{sj}^{(L)}} \cdot \frac{\partial net_{sj}^{(L)}}{\partial w_{jk}^{(L-1)}}, \quad (4.28)$$

gdje vrijedi sljedeće:

$$\frac{\partial y_{sj}^{(L)}}{\partial net_{sj}^{(L)}} = \frac{\partial g(net_{sj}^{(L)})}{\partial net_{sj}^{(L)}} = g'(net_{sj}^{(L)}), \quad (4.29)$$

$$\frac{\partial net_{sj}^{(L)}}{\partial w_{jk}^{(L-1)}} = \frac{\partial}{\partial w_{jk}^{(L-1)}} (w_{j1}^{(L-1)} \cdot y_{s1}^{(L-1)} + \dots + w_{jk}^{(L-1)} \cdot y_{sk}^{(L-1)} + \dots) = y_{sk}^{(L-1)}. \quad (4.30)$$

Nakon uvrštavanja u izraz (4.27) dobivamo:

$$\frac{\partial E}{\partial w_{jk}^{(L-1)}} = -\frac{1}{N} \sum_{s=1}^N g'(net_{sj}^{(L)}) \cdot (d_{sj} - y_{sj}^{(L)}) \cdot y_{sk}^{(L-1)} \quad (4.31)$$

$$= -\frac{1}{N} \sum_{s=1}^N \delta_{sj}^{(L)} \cdot y_{sk}^{(L-1)}, \quad (4.32)$$

gdje je $\delta_{sj}^{(L)}$ pogreška j-tog neurona s-tog uzorka na izlaznom sloju i definira se kao:

$$\delta_{sj}^{(L)} = g'(net_{sj}^{(L)}) \cdot (d_{sj} - y_{sj}^{(L)}). \quad (4.33)$$

Sada kada smo definirali parcijalnu derivaciju za proizvoljnu težinu zadnjeg skrivenog sloja, ažuriranje težina na temelju (4.23) glasi:

$$w_{jk}^{(L-1)} \leftarrow w_{jk}^{(L-1)} - \alpha \cdot \left(-\frac{1}{N} \sum_{s=1}^N \delta_{sj}^{(L)} \cdot y_{sk}^{(L-1)} \right) \quad (4.34)$$

$$\leftarrow w_{jk}^{(L-1)} + \eta \cdot \sum_{s=1}^N \delta_{sj}^{(L)} \cdot y_{sk}^{(L-1)} \quad (4.35)$$

gdje smo umjesto $\alpha \cdot \frac{1}{N}$ uveli konstantu η za koju i dalje vrijede svojstva koje smo naveli za stopu učenja.

Postupak koji smo proveli vrijedi kada tražimo parcijalne derivacije za težine predzadnjeg sloja, no procedura za ostale slojeve je gotovo pa identična. Postupak se počinje mijenjati u formuli (4.26):

$$\frac{\partial E}{\partial w_{jk}^{(L-2)}} = -\frac{1}{N} \sum_{s=1}^N \sum_{i=1}^m (d_{s,i} - y_{s,i}^{(L)}) \cdot \frac{\partial y_{s,i}^{(L)}}{\partial w_{jk}^{(L-2)}}, \quad (4.36)$$

gdje navedenu parcijalnu derivaciju opet rješavamo pravilom ulančavanja. Ako detaljnije raspišemo, onda ćemo vidjeti da će biti potrebno "spustiti" se do (L-2)-og sloja. Konačna formula za pogrešku j-tog neurona (L-2)-og sloja i ažuriranje težina će onda glasiti:

$$\delta_{s,j}^{(L-1)} = g'(net_{s,j}^{(L)}) \cdot \sum_{i=1}^m \delta_{s,i}^{(L)} \cdot w_{i,j}^{(L-1)}, \quad (4.37)$$

$$w_{jk}^{(L-2)} \leftarrow w_{jk}^{(L-2)} + \eta \cdot \sum_{s=1}^N \delta_{s,j}^{(L-1)} \cdot y_{s,k}^{(L-2)}. \quad (4.38)$$

Algorithm 2 Backpropagation algoritam²⁵

1. Ciklički prolazi kroz svih N uzoraka za učenje, jedan po jedan.
2. Ponavlja dok nije zadovoljen uvjet zaustavljanja.
3. Za svaki uzorak iz skupa uzoraka za učenje čini:
 1. Postavi uzorak na ulaz unaprijedne neuronske mreže i izračunaj izlaze za sve neurone primjenom formule (4.9).
 2. Izračunaj pogrešku svakog od neurona izlaznog sloja po formuli (4.33).
 3. Vraćaj se sloj po sloj i izračunaj pogreške svakog neurona po formuli (4.37).
 4. Ažuriraj težine po formuli:

$$w_{jk}^{(l)} \leftarrow w_{jk}^{(l)} + \eta \cdot \sum_{s=1}^N \delta_{s,j}^{(l+1)} \cdot y_{s,k}^{(l)}. \quad (4.39)$$

5. Ažuriraj biase po formuli (prisjetite se da su ulazi konstantni):

$$b_j^{(l)} \leftarrow b_j^{(l)} + \eta \cdot \sum_{s=1}^N \delta_{s,j}^{(l+1)}. \quad (4.40)$$

²⁵Iz literature (Čupić, 2016).

Dakle, uvjerali smo se da će izračuni parcijalnih derivacija uz pomoć backpropagation algoritma biti znatno brži, nego da smo ih pojedinačno računali. Za jednu epohu učenja kroz mrežu je potrebno proći svega dva puta: algoritmom unaprijedne neuronske mreže te backpropagation algoritmom što je vrlo efikasno. Također, uvjet zaustavljanja učenja može biti da funkcija kazne poprmi određeni error kojeg korisnik izvana zada ili jednostavno da se učenje završi nakon određenog broja epoha.

Spomenimo još tri glavna tipa učenja backpropagation algoritmom, a to su:

1. Skupno učenje (engl. *batch learning*) - težine se ažuriraju tek kada kroz mrežu prođu svi uzorci, vremenski je dosta zahtjevan, ali je jako precizan.
2. Učenje po manjim grupama (engl. *mini-batches learning*) - težine se ažuriraju nakon što kroz mrežu prođe određen broj uzoraka, vremenski je manje zahtjevan od skupnog, ali je manje precizan.
3. Stohastičko učenje (engl. *stochastic learning*) - težine se ažuriraju nakon svakog uzorka, vremenski je najpovoljnije, ali je nešto neprecizniji od mini-batchesa. Često se naziva i *online* učenje zbog ažuriranja nakon svakog uzorka.

U sljedećem poglavlju ćemo pokazati rezultate programske implementacije.

5. Rezultati

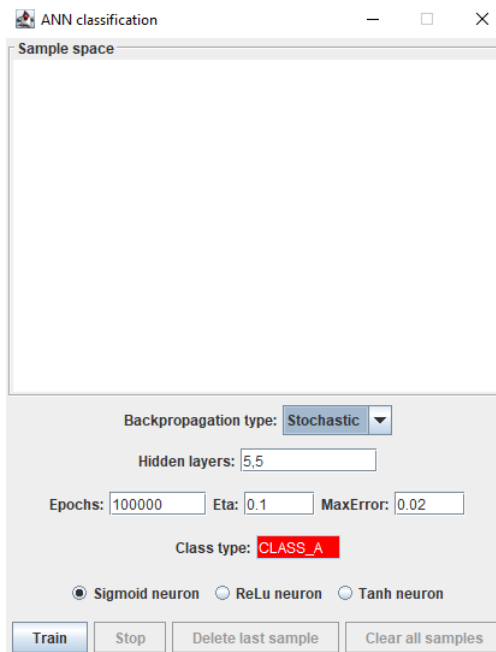
Programska implementacija ovog rada pisana je programskim jezikom *Java* verzije 13 i pomoćnog alata *Maven* verzije 3.6.3. Korištena je samo jedna eksterna biblioteka za rad s matricama¹, dok je sve ostalo pisano standardnim javnim bibliotekama kao npr. *Swing*.

U sklopu implementacije implementirana je unaprijedna neuronska mreža razredom *NeuralNetwork* koji posjeduje metodu *feedForward* zaslužnu za dobivanje izlaza za neki ulaz te metodom *train* koja trenira istu algoritmom backpropagation. U nastavku su prikazane signature metoda.

```
public class NeuralNetwork {  
    ...  
    public double[] feedForward(double[] inputs) {...}  
    public void train(  
        int epochs, double maxError, double eta) {...}  
    ...  
}
```

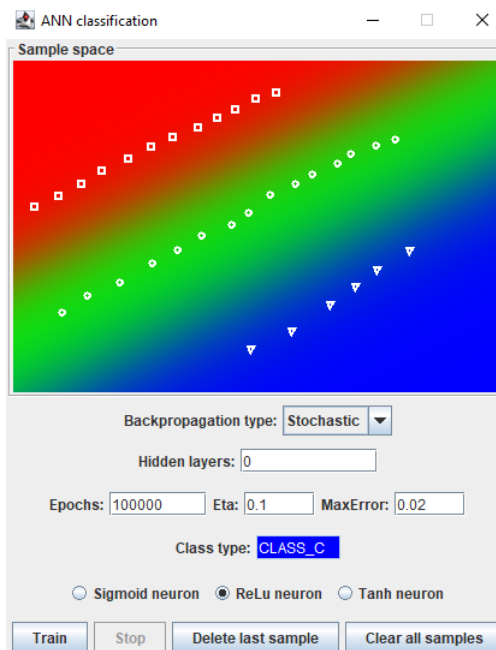
Također, implementirano je i jedno grafičko korisničko sučelje (*GUI*) koje je prikazano na slici 5.1. GUI je dosta samodokumentirajući, ali spomenimo par sitnica. Mreži se za učenje predaju uzorci iz 2D koordinatnog sustava koji se unose klikom lijevog miša, dok se klikom desnog miša mijenja tip razreda kao što je navedeno pod **Class type**. Mogu se izabrati tri tipa backpropagation algoritma koje smo spomenuli ranije. S obzirom da mreža ima fiksne ulaze (2 ulaza: x i y koordinata točke) i fiksne izlaze (3 izlaza jer su omogućene tri vrste razreda), omogućeno je postavljanje samo skrivenih slojeva. Ostale karakteristike sučelja bi trebale biti jasne. Potvrdimo teoriju ovog rada na temelju sljedećih nekoliko primjera.

¹*Apache Commons Math* 3.6.1 <http://commons.apache.org/proper/commons-math/>.



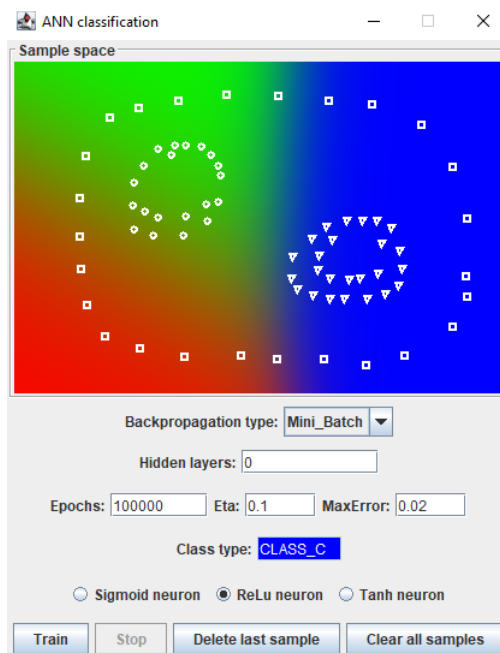
Slika 5.1: Grafičko korisničko sučelje

Krenimo sa jednoslojnim perceptronom i linearnom klasifikacijom prikazanom na slici 5.2. Korištena aktivacijska funkcija u ovom slučaju je ReLU, no sve tri navedene daju donekle sličan rezultat što se tiče oblika decizijske granice jer se radi o linearnoj klasifikaciji gdje nema previše mudrovanja.



Slika 5.2: Linearna klasifikacija - jednoslojni perceptron

Pogledajmo što će se dogoditi ako problem nije linearno interpretabilan ako i dalje koristimo jednoslojni perceptron. Rezultat je prikazan na slici 5.3. Vidimo da je klasifikator neke točke dobro klasificirao, a neke nije te smo time upravo dokazali da jednoslojni perceptron nije u stanju kvalitetno riješiti probleme koji nisu linearno interpretabilni.

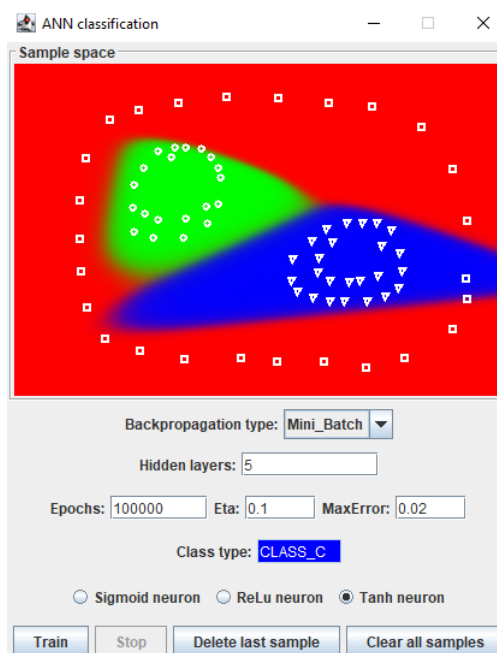


Slika 5.3: Nelinearna klasifikacija - jednoslojni perceptron

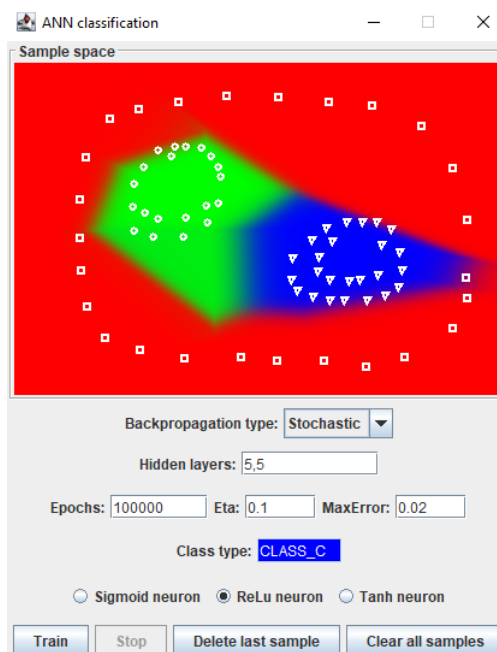
Pogledajmo sada kako bi višeslojni perceptron riješio isti problem. Rezultat je prikazan na slici 5.4. Korišten je tangens hiperbolni kao aktivacijska funkcija jer je davala najbolje rezultate za trenutni problem i korišten je tip učenja u grupama (*mini-batches*), gdje se svaka grupa sastoji od maksimalno 30 uzoraka. Arhitektura mreže sada glasi 2x5x3, što znači da smo dodali jedan skriveni sloj od 5 neurona. No, i dalje nismo zadovoljni kako klasifikator radi pa probajmo mrežu učiniti još ekspresivnijom dodavši joj još slojeva i neurona.

Na slici 5.5 prikazani su nešto bolji rezultati nego ranije jer smo mrežu učinili nešto ekspresivnijom dodavši joj još jedan skriveni sloj od 5 neurona i promijenivši tip backpropagation algoritma u stochastic kao i aktivacijsku funkciju u ReLU. Također, možemo vidjeti kako ReLU funkcija utječe na izgled decizijskih granica čineći grupirane razrede kao pravokutne likove. Pogledajmo možemo li klasifikator još bolje naučiti i ispostavlja se da možemo, no što je mreža ekspresivnija time će se klasifikator pretrenirati samo nad uzorcima za učenje te time neće razviti svojstvo generalizacije i neće biti primjenjiv izvan predstavljenog problema, a to je jako loša praksa. Rezultati

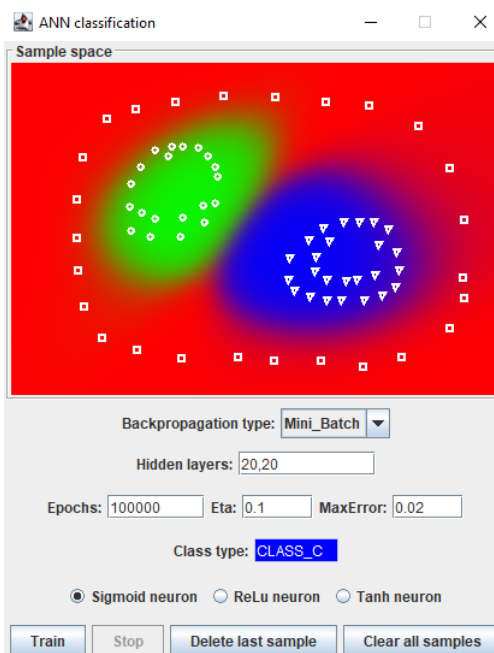
su prikazani na slikama 5.6 i 5.7. Uočite arhitekture neuronske mreže kao i maksimalan error za koji se prekida učenje.



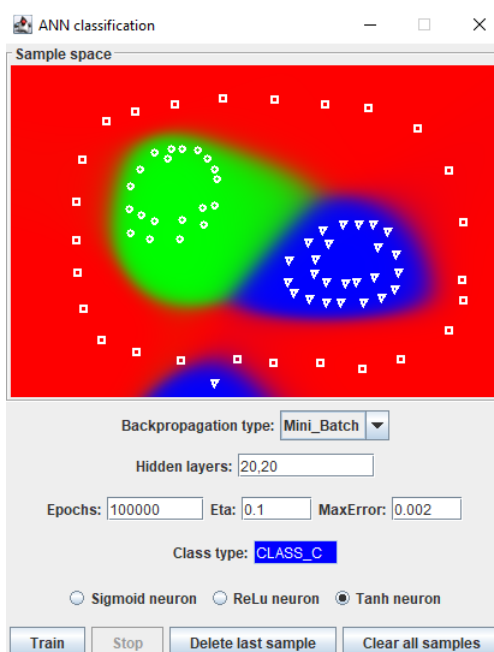
Slika 5.4: Nelinearna klasifikacija - višeslojni perceptron



Slika 5.5: Nelinearna klasifikacija - bolje



Slika 5.6: Nelinearna klasifikacija - još bolje



Slika 5.7: Pretreniranost

6. Zaključak

Kroz ovaj rad obrađen je algoritam unparijedne umjetne neuronske mreže i učenje iste algoritmom propagacije pogreške unatrag.

Najveći problem u realizaciji jednog takvog algoritma predstavlja upravo pametna inicijalizacija težina među neuronima, izbor aktivacijskih funkcija i arhitektura same mreže. Često moramo jako dobro poznavati teoriju koja se krije iza kulisa da bi se mogli suočiti s mnoštvom različitih problema na prihvatljiv način, inače možemo u nedogled vrtjeti algoritam bez kvalitetnih rezultata.

Unatoč svemu, implementacija ovog rada je dosta uspješno realizirana i model klasifikatora za različite aktivacijske funkcije daje dosta dobre rezultate.

Za daljnji nastavak ovog rada bi se moglo proučiti mnoštvo različitih inačica backpropagation algoritma u svrhu postizanja boljih rezultata i upoznati se s dubokim neuronskim mrežama i algoritmima dubokog učenja.

LITERATURA

Eli Bendersky. The Softmax function and its derivative, listopad 2016. URL <https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/>. [Pristupljeno 26-Maj-2020].

Dartmouth workshop. Dartmouth workshop — Wikipedia, the free encyclopedia, ožujak 2020. URL https://en.wikipedia.org/wiki/Dartmouth_workshop. [Pristupljeno 10-Maj-2020].

Donald O. Hebb. *The Organization of Behavior*. New York: Wiley, 1949.

Kian Katanforoosh i Daniel Kunin. Initializing neural networks, 2019. URL <https://www.deeplearning.ai/ai-notes/initialization/>. [Pristupljeno 26-Maj-2020].

Warren S. McCulloch i Walter H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

Christopher Moyer. How Google’s AlphaGo Beat a Go World Champion, ožujak 2016.

Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL <http://neuralnetworksanddeeplearning.com/>. [Pristupljeno 10-Maj-2020].

No free lunch theorem. No free lunch theorem — Wikipedia, the free encyclopedia, maj 2020. URL https://en.wikipedia.org/wiki/No_free_lunch_theorem. [Pristupljeno 12-Maj-2020].

Phil Picton. *Neural Networks*. PALGRAVE, u drugom izdanju, 2000.

Reinforcement learning. Reinforcement learning — Wikipedia, the free encyclopedia, maj 2020. URL https://en.wikipedia.org/wiki/Reinforcement_learning. [Pristupljeno 11-Maj-2020].

- Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65:386–408, 1958.
- S. Russell i P. Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall, u trećem izdanju, prosinac 2009.
- Semi-supervised learning. Semi-supervised learning — Wikipedia, the free encyclopedia, travanj 2020. URL https://en.wikipedia.org/wiki/Semi-supervised_learning. [Pristupljeno 11-Maj-2020].
- Supervised learning. Supervised learning — Wikipedia, the free encyclopedia, maj 2020. URL https://en.wikipedia.org/wiki/Supervised_learning. [Pristupljeno 12-Maj-2020].
- Symbolic artificial intelligence. Symbolic artificial intelligence — Wikipedia, the free encyclopedia, travanj 2020. URL https://en.wikipedia.org/wiki/Symbolic_artificial_intelligence. [Pristupljeno 10-Maj-2020].
- Alan Turing. Computing Machinery and Intelligence. *Mind*, stranice 433 – 460, listopad 1950.
- Alan Woodruff. What is a neuron?, kolovoz 2019. URL <https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron>. [Pristupljeno 13-Maj-2020].
- M. Čupić, B. Dalbelo Bašić, i M. Golub. *Neizrazito, evolucijsko i neuroračunarstvo*, kolovoz 2013. URL <http://java.zemris.fer.hr/nastava/nenr/knjiga-0.1.2013-08-12.pdf>. [Pristupljeno 13-Maj-2020.].
- Marko Čupić. *Umjetne neuronske mreže*, svibanj 2016. URL <http://java.zemris.fer.hr/nastava/ui/ann/ann-20180604.pdf>. [Pristupljeno 11-Maj-2020.].
- Marko Čupić. *Uvod u strojno učenje*, travanj 2020. URL <http://java.zemris.fer.hr/nastava/ui/ml/ml-20200410.pdf>. [Pristupljeno 10-Maj-2020.].

Klasifikacija uporabom umjetnih neuronskih mreža

Sažetak

Raspoznavanje uzoraka te klasifikacija istih je jedan od bitnijih problema računar-ske znanosti. Najveći izazov je konstruirati kvalitetan klasifikator koji je sposoban generalizirati. U ovom radu prikazan je teorijski pogled na algoritam umjetne neuron-ske mreže kroz povijest pa do danas i njegovo učenje backpropagation algoritmom. Također, uz rad je i implementirano jedno takvo učenje kroz grafičko korisničko suče-lje kroz koje korisnik interaktivno unosi točke iz 2D koordinatnog sustava labeliranih različitim razredima i prati koliko kvalitetno klasifikator uči. Reprezentativni rezultati prikazani su i dokumentirani u radu.

Ključne riječi: Nadzirano učenje, klasifikacija, unaprijedna neuronska mreža, algoritam propagacije pogreške unatrag, aktivacijska funkcija.

Classification Based on Artificial Neural Networks

Abstract

Pattern recognition and classification is one of the most important problems in com-puter science. The biggest challenge is to construct a quality classifier that is capable of generalizing. This paper presents a theoretical view of the artificial neural network algorithm through history to the present day and its learning by the backpropagation algorithm. Also, in addition to the work, one such learning was implemented through a graphical user interface through which the user interactively enters points from the 2D coordinate system labeled with different classes and monitors how well the classifier learns. Representative results are presented and documented in the paper.

Keywords: Supervised learning, classification, feedforward neural network, backpro-pagation, activation function.