VISOKO UČILIŠTE ALGEBRA
VISOKA ŠKOLA ZA PRIMIJENJENO RAČUNARSTVO

ZAVRŠNI RAD

IZRADA NEURONSKE MREŽE ZA IZRAČUN AERODINAMIČKOG PROFILA

Jure Fadiga

Zagreb, siječanj 2018.

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

U Zagrebu, 15.01.2018.

Predgovor

Tijekom studiranja sam se susreo sa tematikom svog završnog rada. O neuronskim sam mrežama na predavanjima na fakultetu stekao teoretsko znanje, a na grupnim sam vježbama vidio nekoliko primjera sa gotovim rješenjima. Vjerovao sam kako je završni rad na temu neuronskih mreža moguće vrlo jednostavno pripremiti i izraditi unutar vremenskog roka koji mi je bio zadan. Bio sam u krivu, obzirom na to kako su svi primjeri sa grupnih vježbi bili dobro pripremljeni, uključujući i sve potrebne korake i predradnje, što je značilo kako je bilo potrebno sva dobivena podrješenja samo unijeti u gotovu formulu kako bih dobio konačno rješenje. Kada sam počeo raditi na praktičnom dijelu svog završnog rada, pokušao sam prvo oponašati postupak viđen na vježbama, bez prethodnog čitanja literature vezane za neuronske mreže. Kako su moji pokušaji bili neuspješni, odlučio sam se poslužiti literaturom, te pokušao ponovno izraditi neuronsku mrežu koristeći jednim dijelom metode sa grupnih vježbi, a drugim dijelom metode iz same literature. Konačno rješenje još se uvijek nije naziralo, pa sam pokušao koristiti drugi programski jezik, no kada sam shvatio kako će mi korištenje meni manje poznatog programskog jezika oduzeti previše vremena, odlučio sam ipak se posvetiti izradi neuronske mreže u programskom jeziku koji poznajem. U nastojanju da dođem do konačnih i upotrebljivih rješenja, promijenio sam pet različitih baza podataka u nadi kako je problem u unešenim podacima, njihovom broju ili pak njihovoj kompleksnosti. Na kraju sam uspio izraditi neuronsku mrežu, te nakon niza pokušaja i brojnih zastoja sada pišem ovaj završni rad.

Iskreno se zahvaljujem cijenjenom mentoru Marku Veliću na njegovim brojnim savjetima i vremenu koje je uložio u mene, unatoč činjenici kako mi je ovaj rad pokazao da ova tematika nije nešto u čemu se pronalazim, te bih se u svojoj daljnjoj karijeri htio u čim manjoj mjeri susresti sa ovom granom programiranja.

Original potvrde o prihvaćanju teme završnog rada, umetnuti umjesto ove stranice prilikom uvezivanja rada!

Sažetak

Tema ovog rada biti će neuronske mreže. Prije svega, navest ću pod kojim su utjecajem i kako one nastale, koja im je bila svrha u doba njihova nastanka, a u koju su pak svrhu one korištene danas. Potom ću opisati postupak izrade vlastite neuronske mreže u programskom jeziku C#. Cilj ovog rada je izraditi vlastitu neuronsku mrežu koja će biti korištena u desktop aplikaciji kako bi napravila izračun aerodinamičkih profila.

Ključni pojmovi: Microsoft Excel, Visual Studio, neuronska mreža, obrada podataka, aerodinamički profil, C#

Abstract

The topic of this thesis will be neural networks. First off, I plan to mention how were they invented, and what were the ideas that influenced their discovery, what was their purpose at the time of their invention, and what their purpose is today. After that, I will describe the process of creating my own neural network by using C# programming language. The goal of this thesis is to create a neural network which will be used in desktop application to calculate aerodynamic profiles.

Key terms: Microsoft Excel, Visual Studio, neural network, data processing, aerodynamic profile, C#

Sadržaj

1. Uvod

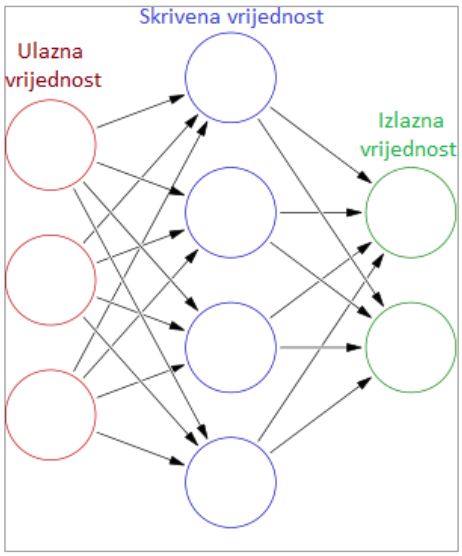
Izrada neuronske mreže za izračun aerodinamičkog profila je tema rada kojim ću predočiti korake i postupke koje sam koristio u izradi vlastite neuronske mreže. Cilj mi je bio dublje se upoznati s granom informatike koju bih mogao susresti u budućnosti, te isto tako svojim radom pomoći drugim kolegama pri izradi njihovih vlastitih neuronskih mreža koristeći moju kao primjer.

Na samom početku izrade svog završnog rada odlučio sam dublje istražiti pojam neuronskih mreža, te proširiti znanje o neuronskim mrežama stečeno do sada na predavanjima. Naučio sam kako se podaci trebaju pripremiti prije same izrade neuronske mreže. Postoji više različith pristupa izradi neuronske mreže, kao i izradi algoritama koji će vršiti izračun nad obrađenim, prethodno pripremljenim podacima, i načinu testiranja istih. Odabrao sam postupak nadgledanog učenja pri izradi neuronske mreže koji mi je omogućio veću sigurnost pri njenoj izradi, jer ovim postupkom korištena baza podataka sadrži ulazne podatke, dakle one koji se predaju neuronskoj mreži, te izlazne podatke koji se mogu usporediti sa izračunatim vrijednostima iz neuronske mreže. Baza podataka koja sadrži potrebne ulazne i izlazne vrijednosti javno je dostupna, a pronašao sam ju na web stranici Sveučilišta u Karolini, https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise# pod nazivom "*Airfoil Self-Noise*". Preuzete podatke sam unio u Excel datoteku te ih tamo obradio i pripremio u oblik spreman za korištenje u izradi algoritma neuronske mreže. Jedan dio podataka bio je namijenjen izradi algortima, a drugi dio testiranju njegove točnosti. Za samu izradu algoritma sam koristio gotov javni *library NeuralNetCore* koji sadrži klase sa metodama u koje sam proslijedio svoje obrađene podatke iz Excel datatoke, kao i druge varijable koje NeuralNetCore traži pri izvršavanju metode. Druge varijable korištene za izradu mog algoritma odabrane su temeljem zaključaka do kojih sam došao čitajući o neuronskim mrežama, te temeljem mnogobrojnih prijašnjih pokretanja te metode koji isprva nisu bili uspješni. Naposljetku sam odabranom i obrađenom bazom podataka te pripremljenim algoritmom uspješno i samostalno izradio desktop aplikaciju koja će zaprimiti ulazne podatke i koristiti neuronsku mrežu za izračun aerodinamičkog profila.

2. O neuronskim mrežama

2.1. Definicija i kratak opis neuronske mreže

Neuronska mreža je računalni koncept koji se temelji na velikom skupu umjetnih neuronskih jedinica, a koji se djelomično modelira prema uzoru na biološki mozak, odnosno način na koji on rješava složene probleme pomoću neurona koji su međusobno povezani tako da tvore neuronsku mrežu. Svaki neuron je povezan sa jednim ili više neurona, kao što je prikazano na slici ispod (Slika 2.1.), koji mogu otežavati ili pomagati u aktivacijskoj funkciji, a može sadržavati funkciju zbroja koja zbraja sve njegove ulazne vrijednosti, te uz to može imati prag čiju vrijednost ulazni signali moraju zadovoljiti kako bi signal određene vrijednosti prešao u drugi neuron. Neuronske mreže se same uče i treniraju, te se koriste kada se ne može lako raspoznati odnos među vrijednostima u tradicionalnom načinu programiranja.



Slika 2.1. Neuronska mreža

2.2. Povijest neuronskih mreža

Godine 1943. Warren McCulloch i Walter Pitts su stvorili model neuronske mreže temeljen na matematici i algoritmima nazvanim logikom praga. Ovaj model je pružio mogućnost istraživanja neuronskih mreža, koja su odvojena u dva različita pristupa. Jedan pristup bio je usmjeren prema istraživanju bioloških procesa u mozgu, a drugi na prema istraživanju

neuronskih mreža za primjenu u umjetnoj inteligenciji. U kasnim 1940-ima psiholog Donald Hebb je stvorio hipotezu o učenju temeljenu na neuronskoj plastičnosti koja se sada naziva *Hebbovim učenjem*. Hebbovo učenje je tipičan primjer nenadziranog učenja i njegovi kasniji modeli se koriste za dugoročno potenciranje (proces koji mjeri koliko su dugo neuroni u mozgu pogođeni za vrijeme kratke elektrostimulacije). Jedan od ovih modela počeo se koristiti kod B tipa računala znamenitog britanskog matematičara, kriptografa, logičara i vizionara, Alana Mathisona Turinga 1948. godine. Belmont Farley i Wesley A. Clark su 1954. godine po prvi puta koristili računalne strojeve kako bi simulirali Hebbijansku mrežu (mrežu temeljenu na Hebbovom učenju) na MIT-u (*Massachusetts Institute of Technology*). Ostala računala s primjenom neuronskih mreža su izradili Nathaniel Rochester, John Henry Holland, L.H. Habit i W.L. Duda 1956. godine.

Frank Rosenblatt je 1958. godine stvorio *perceptron*, algoritam za prepoznavanje uzoraka koji se temelji na dva sloja mrežnog, računalnog učenja pomoću jednostavnih operacija zbrajanja i oduzimanja. Uz pomoć matematičke notacije, Rosenblatt je također opisao logičke sklopove koji nisu dijelom osnovnog perceptrona, poput primjerice *ekskluzivnog ILI* logičkog sklopa, sklopa koji se nije mogao obraditi neuronskim mrežama sve do osmišljanja algoritma povratnog prostiranja izrađenog od strane Paula Werbosa 1975. godine. Istraživanja neuronske mreže stagniraju nakon objave istraživanja o strojnom učenju Marvina Minskya i Seymoura Paperta 1969. godine, koji su otkrili dva ključna problema računalnih strojeva koji obrađuju neuronske mreže. Prvi problem je ukazao na to kako osnovni perceptroni nisu sposobni obraditi *ekskluzivni ILI logički sklop*, odnosno njegovu logičku funkciju. Drugi važan problem jest nedovoljna sofisticiranost tada prisutnih računala, koja nisu bila u mogućnosti obraditi dugotrajne procese (izračune) potrebne za velike neuronske mreže. Istraživanja neuronskih mreža tada bivaju usporena sve do trenutka u kojem su računala postigla bolju sposobnost obrade podataka.

Ključni trenutak napretka nastupio je pojavom algoritma povratnog prostiranja koji učinkovito rješava problem *ekskluzivnog ILI sklopa* i njegove logičke funkcije (Paul Werbos, 1975.).

Sredinom 1980-ih, paralelno distribuirano procesiranje postaje popularno pod imenom konekcionizam (engl. *connectionism*). Priručnik napisan od strane Davida E. Rumelharta i Jamesa McClellanda (Rumelhart i McClelland, 1986.) pružio je puno izlaganje o korištenju konekcionizma u računalima u svrhu simulacija neuronskih procesa.

Neuronske mreže korištene za umjetnu inteligenciju, tradicionalno su se promatrale kao pojednostavljeni modeli neuronskih procesa u mozgu, unatoč tome kako je odnos između ovog umjetnog modela i biološke arhitekture mozga još uvijek temeljem rasprave, budući da nije jasno u kojoj mjeri umjetne neuronske mreže zrcale funkcije mozga.

Metoda potpornih vektora (engl. *support vector machines*) i druge, mnogo jednostavnije metode, poput linearnih klasifikatora, postupno su postale popularnije u strojnom učenju u odnosu na do tada poznate klasične neuronske mreže. Kako se računalna snaga povećavala kroz korištenje *GPU* i distribuiranog računanja, slike i vizualni problemi prepoznavanja došli su u prvi plan, a time su i neuronske mreže ponovno postale upotrebljavane u većim razmjerima. Ova metoda poznata je pod nazivom „duboko učenje“, koje je zapravo jednostavni redizajn neuronskih mreža, s naglaskom na korištenje suvremene paralelne hardverske implementacije.

2.2.1. Moderno doba

Računalni uređaji su izrađeni u CMOS tehnologiji, kako za biofizičke simulacije tako i za neuromorfno računanje. Novija istraživanja približavaju se stvaranju nano uređaja za primjenu u vrlo opsežnim analizama glavnih komponenata i konvolucijama. Ukoliko se ova istraživanja pokažu uspješnima, to će značiti ulazak u novu eru neuronskog računanja, korak dalje od digitalnog računalstva, jer se neuronsko računanje temelji na učenju, a ne programiranju, i u osnovi je analogno, a ne digitalno, iako je moguće kako će se prve izvedbe koristiti sa CMOS digitalnim uređajima.

Između 2009. i 2012. godine, povratne neuronske mreže i duboke neuronske mreže s propagacijom unaprijed, razvijene u istraživačkoj grupi Jürgena Schmidhubera u švicarskom AI Lab IDSIA, pobijedile su u osam međunarodnih natjecanja u raspoznavanju uzoraka i strojnog učenja. Primjerice, višedimenzionalno složeno dugo kratkoročno pamćenje (engl. *Long Short-Term Memory*, Graves & Schmidhuber, 2009.) je pobijedilo u tri natjecanja u prepoznavanju rukopisa 2009. godine na Međunarodnoj konferenciji o Analizi prepoznavanju dokumenata (ICDAR), bez ikakvog prethodnog znanja o tri različita jezika koja je trebalo učiti.

Brze GPU bazirane implementacije ovog pristupa izrađene od strane Dana Ciresana i njegovih kolega na IDSIA, pobijedile su na nekoliko natjecanja raspoznavanja uzoraka, uključujući IJCNN godine 2011. godine, gdje je predmet natjecanja bilo prepoznavanje prometnih znakova, potom ISBI 2012. godine na natjecanju u segmentaciji neuronskih

struktura kod elektronskih mikroskopskih stogova te na drugim natjecanjima koja ovdje nećemo navoditi. Njihove neuronske mreže su također bile prvi umjetni prepoznavatelji uzoraka koji su konkurirali ljudima ili imali bolje performanse od ljudskih, na važnim područjima kao što su prepoznavanje prometnih znakova (IJCNN 2012.) ili MNIST problem ručno zapisanih brojeva (Yann LeCun, NYU).

Duboke, visoko nelinearne neuronske arhitekture poput *neocognitrona* iz 1980. godine, napravljenog od strane Kunihika Fukushime, te „standardne arhitekture vida“ od strane Maximiliana Riesenhubera i Tomasa Poggia, inspirirane jednostavnim i složenim stanicama koje su u primarnom, vizualnom korteksu mozga pronašli David H. Hubel i Torsten Wiesel, mogu se također prethodno trenirati metodama nenadziranog učenja laboratorija Geoffa Hintona na Sveučilištu u Torontu. Tim iz ovog laboratorija je pobijedio na natjecanju 2012. godine organiziranom pod pokroviteljstvom Merck & Co., Inc., za dizajn softvera koji će pomoći pronaći molekule koje bi mogle dovesti do otkrića novih lijekova.

2.3. Modeli neuronskih mreža

Modeli neuronske mreže u umjetnoj inteligenciji obično se nazivaju umjetnim neuronskim mrežama (engl. *Artificial Neural Networks*, skraćeno ANNs). Oni su u biti jednostavni matematički modeli koji definiraju funkciju (1) navedenu ispod;

$$f : X \rightarrow Y \tag{1}$$

ili distribuciju preko X ili oboje X i Y. Ponekad su modeli također usko povezani s određenim algoritmom ili pak pravilom učenja. Većinom je upotreba izraza „ANN model“ zapravo definicija klase takvih funkcija (gdje se dobivaju članovi klase po različitim parametrima, povezanost težina, ili specifičnost arhitekture kao što je broj neurona ili njihovo povezivanje).

2.3.1. Funkcija mreža

Riječ mreža u izrazu „umjetna neuronska mreža“ se odnosi na interkonekcije (međuveze) između neurona u različitim slojevima svakog sustava. Uzmimo za primjer sustav koji ima tri sloja. Prvi sloj ima ulazne neurone koji šalju podatke preko sinapsi na drugi sloj neurona, a zatim preko više sinapsi u treći sloj izlaznih neurona. Složeniji sustavi će imati više slojeva neurona, a neki će imati povećane slojeve ulaznih neurona i izlaznih neurona. Sinapse pohranjuju parametre pod nazivom „težine“ koji manipuliraju podatke u izračunima.

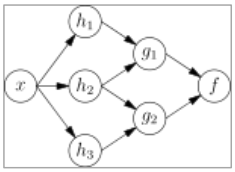
ANN obično definira tri vrste parametara:

1. Dijagram međusobnih veza između različitih slojeva neurona;
2. Proces učenja za ažuriranje težine interkonekcije;
3. Aktivacijska funkcija koja pretvara ulazni podatak neurona u svoju izlaznu aktivaciju.

Matematički, mrežna funkcija za neuron $f(x)$ je definirana kao sastav drugih funkcija $g_i(x)$, koji se dalje mogu definirati kao sastavi drugih funkcija. To može biti prikladno prezentirano kao struktura mreže, sa strelicama koje prikazuju zavisnosti između varijabli. Široko korištena vrsta sastava je nelinearan težinski zbroj, gdje je mrežna funkcija za neuron definirana izrazom navedenim ispod (2):

$$f(x) = K\left(\sum_i w_i g_i(x)\right) \tag{2}$$

gdje je K (obično se naziva funkcija aktivacije) neka unaprijed određena funkcija, kao što je recimo tangens hiperbolni. Ovo će biti prikladno za one situacije koje se odnose na zbirku funkcija g_i kao vektor $g = (g_1, g_2..., g_n)$.



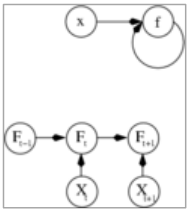
Slika 1.2. Ovisnost među neuronima

Primjer raspada f prikazan je slikom iznad (Slika 2.2.), uz ovisnosti među varijablama označene strelicama. Primjer se može protumačiti na dva načina.

Prvo tumačenje je funkcionalni prikaz: ulazni x se pretvara u trodimenzionalni vektor h, koji se zatim pretvara u dvodimenzionalni vektor g, koji se u konačnici pretvara u f. Ovaj pristup se najčešće susreće u kontekstu optimizacije.

Drugo tumačenje je probabilistički prikaz: slučajna varijabla $F = f\ (G)$ ovisi o slučajnoj varijabli $G = g\ (H)$, koja ovisi o $H = h\ (X)$, koja pak ovisi o slučajnoj varijabli X. Ovakav pristup se najčešće susreće u kontekstu grafičkih modela.

Ova su dva pristupa uglavnom jednaka. U svakom slučaju, za ovu vrstu mrežne arhitekture, komponente pojedinih slojeva su međusobno neovisne (npr. komponente sadržane u vektoru g su međusobno neovisne obzirom na njihov ulazni vektor h). To, naravno, omogućuje paralelnost u provedbi.



Slika 2.3. Primjeri povratne neuronske mreže

Mreže poput one prikazane na prethodnoj slici (Slika 2.2.), obično nazivamo neuronske mreže s propagacijom unaprijed, jer je njihov graf usmjereni aciklički graf. Mreže sa ciklusima obično se nazivaju povratne neuronske mreže. Takve mreže obično su prikazane na način koji je prikazan gornjim primjerom sa slike (Slika 2.3.), gdje f ovisi o samoj sebi. Međutim, implicitna ovisnost o vremenu ovdje nije prikazana.

2.3.2. Učenje

Ono što je privuklo najviše interesa za neuronske mreže je njihova mogućnost učenja. Obzirom na specifičan zadatak koji je potrebno riješiti, te klasu samih funkcija F, učenje ovdje podrazumijeva korištenje skupa opažanja kako bi se pronašlo rješenje za $f^* \in F$, što pruža rješenje zadataka u optimalnom smislu. To podrazumijeva definiranje funkcije troška $C : F \rightarrow \mathbb{R}$ tako da za optimalno rješenje izraza (3) nema rješenja koje ima manji trošak od troška optimalnog rješenja:

$$f^*, \; C(f^*) \leq C(f) \forall f \in F \tag{3}$$

Funkcija troška C je važan koncept u postupku učenja, jer ona pokazuje koliko je daleko određeno rješenje od optimalnog rješenja za zadani problem. Algoritmi učenja pretražuju prostor rješenja kako bi pronašli funkciju koja ima najmanji mogući trošak.

Za aplikacije gdje je rješenje ovisno o nekim podacima, trošak mora obavezno biti funkcija zapažanja, inače ne bismo bili u mogućnosti modelirati ništa vezano za zadane podatke. Često se definira kao statistika kojom je moguće izvesti samo pretpostavke. Kao jednostavan primjer, razmotrimo problem pronalaženja modela f, koji umanjuje $C = E[(f(x) - y)^2]$, za parove podataka (x, y) izdvojene iz distribucije D. U praktičnim situacijama bismo imali samo N uzoraka iz D i na taj način, u navedenom primjeru, samo bismo umanjili sljedeće (4):

$$\hat{C} = 1/n \sum_{i=1}^{n} (f(x_i) - y_i)^2 \tag{4}$$

Dakle, trošak je sveden na minimum pomoću uzorka podataka, umjesto cijelom distribucijom koja sadrži podatke.

Kada je $N \rightarrow \infty$ **,** mora se koristiti oblik pojedinačnog strojnog učenja (engl. *on-line*), gdje je trošak djelomično sveden na minimum pregledom svakog novog primjera. Iako se pojedinačno strojno učenje često koristi u slučajevima kada je D fiksna, ono je veoma korisno i u slučajevima kada se distribucija polako mijenja tijekom vremena. U metodama neuronskih mreža, neki oblik pojedinačnog strojnog učenja često se koristi za konačne skupove podataka.

2.3.3. Odabir funkcija troška

Iako je moguće definirati neku proizvoljnu *ad hoc* funkciju troška, često se upotrebljava prethodno određeni trošak. Razlog tome mogu biti njegova poželjna svojstva (kao što je konveksnost) ili primjerice činjenica da on nastaje prirodnim putem iz određene formulacije problema (primjerice, u probabilističnoj formulaciji stražnja vjerojatnost modela može se koristiti kao inverzni trošak). U konačnici, funkcija troška će ovisiti o željenom zadatku.

2.3.4. Paradigme učenja

Postoje tri glavne paradigme učenja, a svaka odgovara određenom apstraktnom zadatku učenja. To su nadzirano učenje (učenje s nadgledanjem), nenadzirano učenje (učenje bez nadgledanja) te pojačano učenje (učenje pojačavanjem, odnosno podrškom).

2.3.5. Nadzirano učenje

U nadziranom učenju, dobivamo niz primjera parova (x, y), $x \in X$, $y \in Y$, a cilj je pronaći funkciju $f : X \rightarrow Y$ u dopuštenoj klasi funkcija koje odgovaraju primjerima. Drugim riječima, želimo zaključiti mapiranje implicirano podacima; funkcija troška se odnosi na nejednakosti između našeg mapiranja i naših podataka, i ona implicitno sadrži prethodno znanje o području problema.

Najčešće korišten trošak je srednja kvadratna pogreška, koja nastoji na minimum svesti prosječan kvadrat pogreške između izlaza mreže $f(x)$, i ciljane vrijednosti y nad svim primjerima parova. Kada pokušamo na minimum svesti ovaj trošak koristeći gradijentni spust za klasu neuronskih mreža pod nazivom višeslojni *perceptroni*, dobivamo uobičajeni i dobro poznati algoritam s povratnim rasprostiranjem za treniranje neuronske mreže.

Zadaci koji spadaju u paradigmu nadziranog učenja su raspoznavanje uzoraka (također poznato kao klasifikacija) i regresija (također poznata kao aproksimacija funkcije). Paradigma nadziranog učenja je također primjenjiva na sekvenciranim podacima (primjerice, za prepoznavanje govora i gesti). Ona se može shvatiti kao učenje sa „učiteljem“, dakle funkcijom koja osigurava neprekidan priliv povratnih informacija o kvaliteti rješenja koja su do sada dobivena.

2.3.6. Nenadzirano učenje

U nenadziranom učenju, dobiven je određeni podatak x, te funkcija troška koju je potrebno minimizirati, a to može biti bilo koja funkcija danog podatka x i mrežnog izlaza f.

Funkcija troška izravno ovisi o zadatku (o onome što nastojimo modelirati) i našim *a priori* pretpostavkama (implicitna svojstva našeg modela, njegovi parametri i promatrane varijable). Kao trivijalan primjer uzmimo model $f(x) = a$, gdje je a konstanta, a trošak utvrđen izrazom (5):

$$C = E[(x - f(x))^2] \tag{5}$$

Minimiziranjem tog troška dobit ćemo vrijednost a koja je jednaka srednjoj vrijednosti podatka. Funkcija troška može biti i mnogo složenija. Njen oblik ovisi o primjeni; na primjer, u kompresiji bi mogla biti vezana za međusobne informacije između x i $f(x)$, dok bi u statističkom modeliranju ona mogla biti povezana sa stražnjom vjerojatnošću modela ovisno o danom podatku. Bitno je napomenuti kako bi u oba navedena primjera navedene količine bile maksimizirane umjesto minimizirane.

Zadaci koje spadaju u paradigmu nenadziranog učenja su općenito problemi procjene. Koriste se za grupiranje, procjenu statističkih distribucija, kompresiju i filtriranje.

2.3.7. Pojačano učenje

U pojačanom učenju (ili učenju pojačavanjem, odnosno podrškom), podatak x obično nije dobiven, već nastaje kao produkt interakcije agenta s okolinom. U svakom trenutku t, agent izvodi radnju y_t , a okoliš stvara opasku x_t te trenutni trošak c_t , prema nekoj (obično nepoznatoj) dinamici. Cilj je otkriti politiku po kojoj se odabiru akcije koje minimiziraju neku mjeru dugoročnog troška, poput očekivanog kumulativnog troška. Dinamika okoliša i dugoročni trošak za svaku su politiku obično nepoznati, ali mogu biti procijenjeni.

Formalnije okruženje modelirano je kao Markovljev proces odlučivanja (engl. *Markov decision process*, skraćeno MDP) sa stanjima $s_1,..., s_n \in S$ i akcijama $a_1, ..., a_m \in A$, sa sljedećim podjelama vjerojatnosti: trenutna raspodjela troška $P(c_t \mid s_t)$, raspodjela promatranja $P(x_t \mid s_t)$, te prijelaz (tranzicija) $P(s_{t+1} \mid s_t, a_t)$, dok je politika definirana kao uvjetna distribucija (raspodjela) preko akcija ovisno o zapažanjima. Kada oboje uzmemo u obzir, definirali smo Markovljev lanac (engl. *Markov chain*, skraćeno MC). Cilj je otkriti politiku koja minimizira trošak, odnosno Markovljev lanac za koji je trošak minimalan.

Umjetne neuronske mreže (engl. *Artificial Neural Network*, skraćeno ANN) se često koriste u pojačanom učenju kao dio ukupnog algoritma. Dinamičko programiranje su s ANN-om (neurodinamičko programiranje) sjedinili Dimitri P. Bertsekas i John N. Tsitsiklis, te ovu metodu primijenili na višedimenzionalne nelinearne probleme poput onih korištenih za usmjeravanje vozila, upravljanje prirodnim resursima ili primjenu u medicini, budući da ANN ima sposobnost ublažiti gubitke točnosti čak i kada se smanji diskretizacija gustoće mreže radi numeričke procjene rješenja izvornih kontrolnih problema.

Zadaci koji spadaju u paradigmu pojačanog učenja su kontrolni problemi, igre i drugi zadaci sekvencijalng odlučivanja.

2.3.8. Algoritmi učenja

Treniranje modela neuronske mreže u osnovi znači odabir jednog modela iz skupa dozvoljenih modela (ili, u Bayesovim okvirima, određivanje distribucije u odnosu na skup dopuštenih modela) koji minimizira kriterij troška. Postoje brojni dostupni algoritmi za treniranje modela neuronskih mreža, a većina njih se može promatrati kao jednostavna primjena teorije optimizacije i statističkih procjena.

Većina algoritama koji se koriste u treniranju umjetnih neuronskih mreža će korisititi oblik gradijentnog spusta pomoću povratnih izračuna te stvarnih gradijenata. To se postiže jednostavnim uzimanjem derivata funkcije troška u odnosu na parametre mreže, a zatim promjenom tih parametara u smjeru vezanih gradijenata. Algoritmi povratnog prostiranja pogreške treniranja se obično svrstavaju u tri kategorije: najstrmiji spust (uz promjenjivu stopu učenja, uz promjenjivu stopu učenja i zamah, elastičan povrat), kvazi-Newton (Broyden-Fletcher-Goldfarb-Shanno), te Levenberg-Marquardt i konjugirani gradijent (Fletcher-Reeves nadopuna, Polak-Ribiére nadopuna, Powell-Beale algoritam ponovnog pokretanja - restart, skalirani konjugirani gradijent).

Evolucijske metode, programiranje ekspresije gena, simulacija kaljenja, algoritam očekivanja i maksimizacije, neparametarske metode te optimizacija roja čestica su neke od uobičajeno korištenih metoda za treniranje neuronskih mreža.

3. Dohvat podataka iz baze podataka

3.1. Osnove C# jezika

C# je programski jezik višestruke paradigme koji obuhvaća prepoznavanje tipova podataka, imperative, deklarative, funkcije te generičke, objektno i komponentno orijetirane programske discipline. Ovaj široko poznati programski jezik razvijen je od strane Microsofta u okviru njihove .NET inicijative, te je kasnije prihvaćen kao standard od strane Ecma-e i ISO-a. C# je objektno orijentirani programski jezik opće namjene. U siječnju 1999. Anders Hejlsberg je okupio tim s ciljem izrade novog programskog jezika koji je tada bio poznat pod nazivom „Cool“ (engl. *C-like Object Oriented Language*), no kad je jezik javno objavljen u srpnju 2000. godine, promijenio je naziv u C# pod kojim je i danas svima poznat. Naziv je bio inspiriran notnim zapisom gdje simbol # ukazuje na to da na tom mjestu ton treba biti viši za jedan poluton od zapisanog.

3.2. .NET Class Library

3.2.1. Definicija

.NET Class Library je vrsta libraryja koji sadržava klase, sučelja i vrijednosne tipove koji omogućuju pristup funkcionalnosti sustava. Oni su temelji na kojima se grade .NET Framework aplikacije, komponente i kontrole.

3.2.2. NeuronDotNet

NeuronDotNet je *open-source* alat za izgradnju i pokretanje AI aplikacije temeljene na "umjetnim neuronskim mrežama". Napisan je u C# programskom jeziku i kompatibilan je s .NET 2.0 platformom. Ovaj alat iskorištava potencijale objektno orijentiranog dizajna i modularnog programiranja.

3.2.3. Postupak čitanja podataka

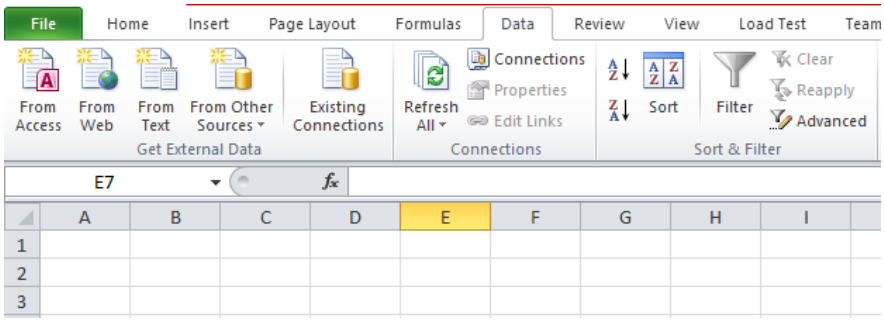
U web repozitoriju na poveznici https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise# se nalazi poveznica za preuzimanje pod nazivom „Data Folder“. Slijedeći ovu poveznicu, otvara se nova web lokacija koja sadrži sljedeću poveznicu koja vodi do tablice sa podacima, https://archive.ics.uci.edu/ml/machine-learning-databases/00291/airfoil_self_noise.dat
Slijedeći ovu poveznicu prikazuje se nova stranica koja ispisuje tablicu sa vrijednostima.

Vrijednosti su u istom retku odnosno različitom stupcu odvojene prazninama. Vrijednosti redaka su zapisane jedna ispod druge. Svi su podaci označeni mišem, te lijevim klikom miša kopirani, pa zatim zalijepljeni u praznu .txt (tekstualnu) datoteku.

4. Pretprocesiranje podataka iz baze podataka

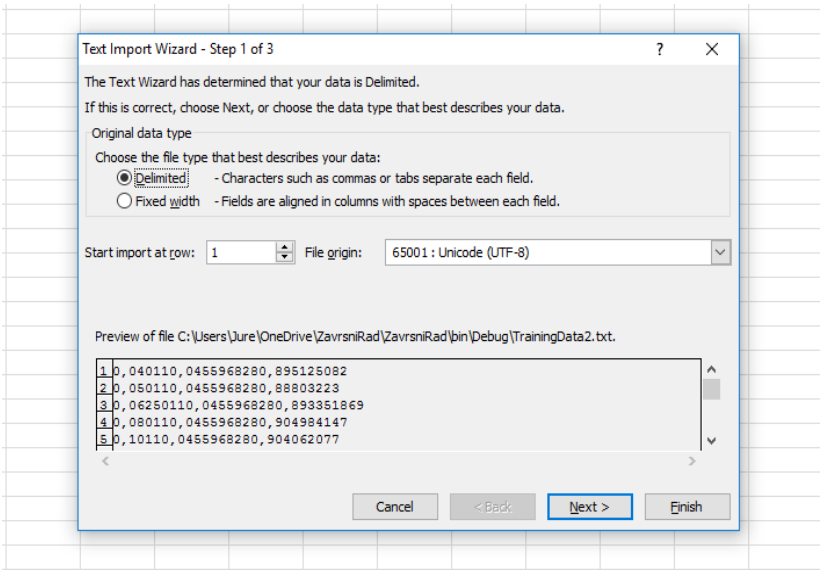
4.1. Postavljanje vrijednosti iz tekstualne datoteke u Microsoft Excel datoteku

Za pretprocesiranje podataka korišten je program Microsoft Excel. Promatrajući sliku ispod (Slika 4.1.), možemo vidjeti kako je otvoren novi Microsoft Excel dokument, a zatim je u traci izbornika odabran izbornik *Data*.



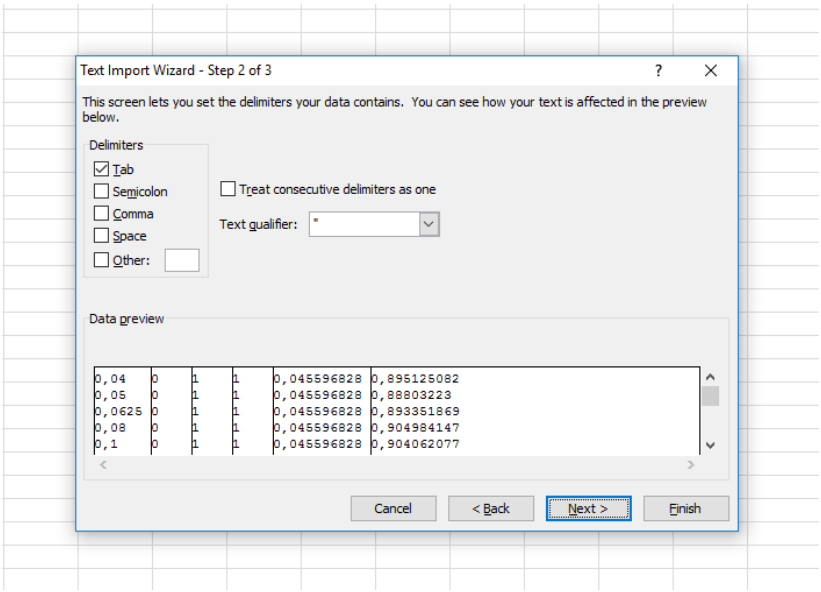
Slika 4.1. *Data* izbornik

U podizborniku *Get External Data* odabrana je opcija *From Text*. U novootvorenom izborniku *Text Import Wizard* je odabrana lokacija tekstualne datoteke, a na slici na sljedećoj stranici (Slika 4.2.) možemo vidjeti odabranu opciju *Delimited*, nakon čega su opcije u ovom koraku (1/3) prihvaćene klikom na *Next*.



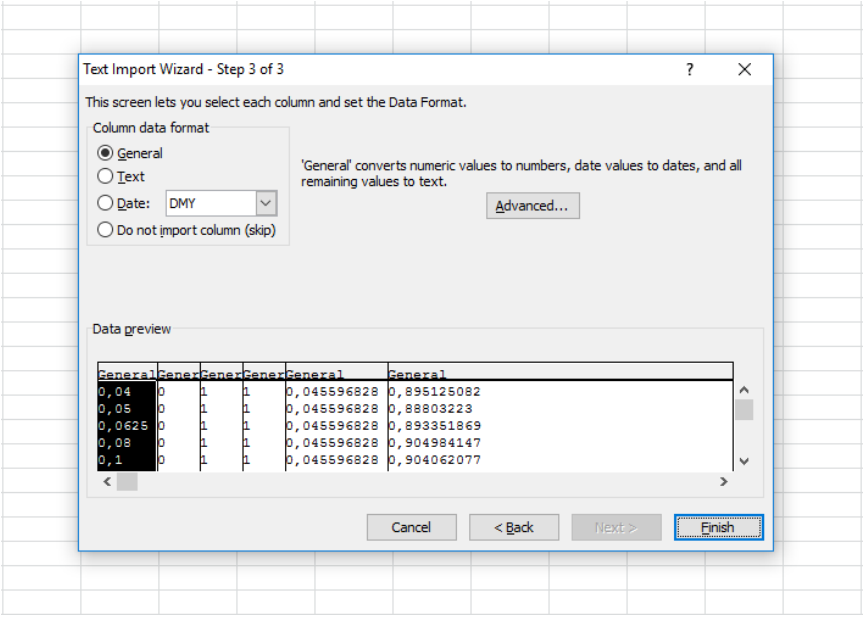
Slika 4.2. „*From Text*“ *pop-up* izbornik (prvi korak)

Sljedeća slika (Slika 4.3.) prikazuje korak 2/3 izbornika *Text Import Wizard*, gdje je označena samo opcija *Tab*, nakon čega su opcije u ovom koraku prihvaćene ponovnim klikom na *Next*.



Slika 4.3. „*From Text*“ *pop-up* izbornik (drugi korak)

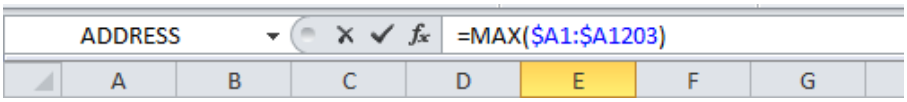
U posljednjem koraku (3/3) u izborniku *Text Import Wizard*, prikazanom na sljedećoj slici (Slika 4.4.), bilo je potrebno označiti opciju *General* te sve dosadašnje odabire potvrditi klikom na *Finish*. Time je dobiven tablični ispis podataka u Microsoft Excel datoteci.



Slika 4.4. *„From Text“ pop-up* izbornik (treći korak)

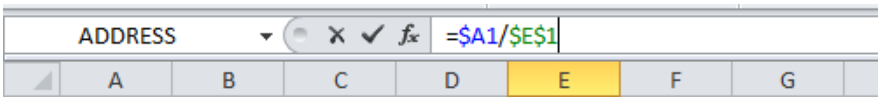
4.2. Normalizacija podataka

U svakom stupcu smo pomoću *MAX* naredbe u Microsoft Excel dokumentu, prikazanoj na slici ispod (Slika 4.5.), pronašli najveću vrijednost tog stupca i zapisali je u zasebnu ćeliju.



Slika 4.5. Funkcija maksimuma

Kada je zapisana svaka najveća vrijednost za pojedini stupac, tada je svaka vrijednost u tom stupcu podijeljena sa najvećom vrijednosti iz istog stupca, kao što je prikazano na sljedećoj slici (Slika 4.6.), a svaka novodobivena vrijednost zapisana je u novom stupcu jedna ispod druge. Taj postupak je ponovljen sve dok nije dobiveno šest novih stupaca, koji imaju upisane vrijednosti između nula i jedan.



Slika 4.6. Funkcija dijeljenja

4.3. Podjela podataka prema njihovoj namjeni

80% podataka uzeto je za treniranje neuronske mreže, dok je njih 20% upotrijebljeno za kasnije testiranje točnosti na nepoznatim podacima.

5. Izrada neuronske mreže

5.1. Postupak korištenja NeuralNetCore libraryja

Izrada neuronske mreže bez pomoći dodatnih alata i libraryja se jako razlikuje od izrade iste uz korištenje NeuralNetCore libraryja. Potrebno je definirati težinske faktore između neurona, uz opcionalno korištenje pomaka (engl. *bias*), zbrajati, a potom množiti sve neurone jednog sloja sa svim neuronima drugog sloja. Sljedeći je korak usporedba dobivenih vrijednosti sa ispravnima, te promjena težinskih faktora i pomaka po potrebi. U NeuralNetCore libraryju sučelje ne omogućuje korisniku jednostavan unos ulaznih vrijednosti na svim mjestima, bez potrebe za ručnim definiranjem istih, no ovaj *library* ipak mnogo toga sam izvršava u pozadini.

Potrebno je definirati broj neurona u svakom sloju, tip vrijednosti koju će neuron primati, broj izvođenja treniranja podataka te tip podataka u sloju. NeuralNetCore iz izvedenih treniranja odabire kombinaciju težinskih faktora i pomake koji daju najbliža rješenja. Osim toga, potrebno je definirati maksimalnu dopuštenu veličinu proizvoljne vrijednosti koja će se dodati rezultatu, te time prouzročiti odstupanje, kao i interval unutar kojeg će se za vrijeme treniranja izvesti spomenuto dodavanje proizvoljne vrijednosti.

5.2. Inicijaliziranje neuronske mreže

Inicijaliziranje neuronske mreže započinje kôdom navedenim na sljedećoj stranici (Kôd 5.1.). U ovom je dijelu postavljen konstruktor klase *NetworkOperationClass*, njegove varijable za svaki sloj (ulazni, skriveni i izlazni sloj), postavljen tip neuronske mreže, te odabran algoritam sa propagacijom unatrag. Također je postavljena i klasa koja će primati podatke za neuronsku mrežu, *TrainingSet* varijabla i metoda *SetHiddenLayer* koja će popuniti vrijednost skrivenog sloja, te postaviti poveznice između slojeva.

```
1.  public class NetworkOperationClass
2.    {
3.      public NetworkOperationClass()     { }      ....
4.
5.      LinearLayer inputLayer = new LinearLayer(5);
6.      SigmoidLayer outputLayer = new SigmoidLayer(1);
7.
8.      SigmoidLayer hiddenLayer;
9.      BackpropagationConnector connector1;
10.     BackpropagationConnector connector2;
11.     BackpropagationNetwork network;
12.     TrainingSet trainingSet;
13.
14.     public void SetHiddenLayer(int hidden)
15.     {
16.       hiddenLayer = new SigmoidLayer(hidden);
17.       connector1 = new BackpropagationConnector(inputLayer, hiddenLayer);
18.       connector2 = new BackpropagationConnector(hiddenLayer, outputLayer);
19.       network = new BackpropagationNetwork(inputLayer, outputLayer);
20.       network.Initialize();          ...
21.     }      ....
22. }
```

Kôd 5.1. Inicijalizacija neuronske mreže (*NetworkOperationClass* uz *SetHiddenLayer*)

Kod inicijalizacije se definira broj neurona u skrivenom sloju u klasi *Program*, kao što je prikazano kôdom ispod (Kôd 5.2.), gdje se također izvršavaju sve naredbe koje su korištene pri izradi neuronske mreže.

```
1.    class Program
2.    {
3.      static void Main(string[] args)
4.      {
5.        NetworkOperationClass class1 = new NetworkOperationClass();
6.        class1.SetHiddenLayer(5);
7.        ....
8.      }
9.    }
```

Kôd 5.2. Inicijalizacija neuronske mreže (definiranje broja neurona u klasi *Program*)

5.3. Unos podataka u neuronsku mrežu

Čitanje podataka za treniranje je postavljeno u zasebnoj klasi *DataManagement* koja se čita u *NetworkOperationClass* klasi, što se može vidjeti kôdom navedenim na sljedećoj stranici (Kôd 5.3.). Prvo se inicijalizira *DataManagement* u klasi *NetworkOperationClass*, čime se omogućuje postavljanje varijabli u kojima će se čuvati podaci iz tekstualne datoteke prije njihovog unosa u samu neuronsku mrežu.

```
1.  public NetworkOperationClass()    {
2.
3.      DataManagment dt = new DataManagment();
4.      ....
5.      double[][] training_data;
6.      double[] training_result_data;
7.      double[][] test_data;
8.      double[] test_data_result;
9.      ....
10. }
```

Kôd 5.3. Unos podataka u neuronsku mrežu (klasa *DataManagement*)

Sljedeći kôd naveden ispod ovog teksta (Kôd 5.4.), prikazuje definiranje novog trening seta ili grupe podataka za treniranja, koji će imati prvu vrijednost jednaku broju neurona u ulaznom sloju i drugu vrijednost jednaku broju neurona u izlaznom sloju, a koja je definirana u samoj klasi. Zatim se vrijednosti pune prvo u *TrainingSample* klasu koja se pridodaje u *TrainingSetu*. U *TrainingSample* klasu dodajemo pet ulaznih vrijednosti i jednu izlaznu vrijednost. Potrebno je pet ulaznih vrijednosti obzirom na to da je definiran ulazni sloj koji ima pet neurona, te jedna izlazna vrijednost jer izlazni sloj neuronske mreže sadrži samo jedan neuron.

```
11.     public NetworkOperationClass()    {
12.     ....
13.
14.     public void SetHiddenLayer(int hidden)
15.     {
16.        ....
17.        training_data = dt.GetTrainingData();
18.
19.        training_result_data = dt.GetTrainingResults();
20.
21.        trainingSet = new TrainingSet(5, 1);
22.
23.        for (int i = 0; i < 1203; i++)
24.        {
25.           TrainingSample sample = new TrainingSample(new double[]    training_data[i][0],
26.           training_data[i][1], training_data[i][2],  training_data[i][3], training_data[i][4] },
27.           new double[] { training_result_data[i] });
28.           trainingSet.Add(sample);
29.        }
30.     }
31. }
```

Kôd 5.4. Unos podataka u neuronsku mrežu (dodavanje *TrainingSample* u *TrainingSet for* petljom)

Kôd ispod (Kôd 5.5.) prikazuje kako je u *DataManagement* klasi čitana tekstualna datoteka "ZavršniRad.txt„ , te su definirane varijable u koje se spremaju podaci za treniranje. Ti su podaci podijeljeni u dvije varijable. U jednoj su spremani ulazni podaci, a drugoj samo izlazni podaci. Prva varijabla obuhvaća polje brojeva unutar ćelija, a druga samo polje brojeva. Klasa ima konstruktor koji osim što inicijalizira vrijednosti varijabli, također poziva funkciju koja čita podatke iz tekstualne datoteke i sprema ih u spomenute dvije varijable, a sama metoda naziva se *ReadTrainingData*.

```
1.  public class DataManagment
2.    {
3.      private double[][] training_data;
4.      private double[] training_data_result;
5.      ....
6.
7.      public DataManagment()
8.      {
9.        training_data = new double[1203][];
10.       training_data_result = new double[1203];
11.       ....
12.       ReadTrainingData();
13.     }
14.     ...
15. }
```

Kôd 5.5. Unos podataka u neuronsku mrežu (metoda *ReadTrainingData*)

Kôd na sljedećoj stranici (Kôd 5.6.) prikazuje kako je u ovoj metodi korištena *StreamReader* klasa preko koje učitavamo tekstualnu datoteku. Instanca te klase čitana je po retku, a ne po stupcu. Uziman je jedan redak za drugim, te je dodavan u listu. Svaki element liste je jedan *string* koji se nadalje dijeli u više manjih *stringova* iz kojih se zatim čitaju vrijednosti iz tekstualne datoteke, te zapisuju u varijable. Ulazne vrijednosti zapisuju se u ulaznu varijablu (koja sadrži polje brojeva unutar ćelija), dok se u drugu, koju čini polje brojeva, zapisuju izlazne vrijednosti.

```
16.  public class DataManagment
17.    {
18.       ...
19.       private void ReadTrainingData()
20.       {
21.          StreamReader streamreader = new StreamReader("TrainingData.txt");
22.          List<string> values = new List<string>();
23.          string line;
24.          while ((line = streamreader.ReadLine()) != null)
25.          {
26.             values.Add(line);
27.          }
28.
29.          double[][] array1 = new double[1203][];
30.          double[] array2 = new double[1203];
31.          int j = 0;
32.
33.          foreach (string item in values)
34.          {
35.             double temp;
36.             string[] stringfield = item.Split('\t');
37.             int counter = 0;
38.             for (int i = 0; i < stringfield.Length; i++)
39.             {
40.                ....
41.                else  {
42.                   temp = double.Parse(stringfield[i].ToString());
43.                   array1[j][counter] = temp;
44.                   counter++;
45.                }
46.             }
47.             j++;
48.          }
49.        ....
50.       } ....
51.     }
52.  }
```

Kôd 5.6. Unos podataka u neuronsku mrežu (korištenje *StreamReader* klase)

5.4. Treniranje i provjera rezultata

Treniranje i provjera rezultata nastupa nakon što je neuronska mreža definirana i napunjena vrijednostima. Treniranje i testiranje (provjera) prikazane su kôdom na sljedećoj stranici (Kôd 5.7.), a pojašnjene u nastavku.

```
1.  public class NetworkOperationClass
2.    {
3.    ....
4.       private void LearningVariatian(int start, int x, double y)
5.       {
6.          network.JitterNoiseLimit = y;
7.          network.JitterEpoch = x;
8.          network.Learn(trainingSet, start);
9.          double[] trainingvalue1 = training_data[0].ToArray();          .....
10.         double trainingresult1 = network.Run(trainingvalue1)[0];    .....
11.         double realtrainingresult1 = training_result_data[0];          .....
12.         double[] testvalue1 = test_data[0].ToArray();                  .....
13.         double testresult1 = network.Run(testvalue1)[0];               .....
14.         double realtestresult1 = test_data_result[0];                   .....
15.      }
16. }
```

Kôd 5.7. Treniranje neuronske mreže

Napravljene su dvije metode za treniranje. Obje metode pozivaju istu podmetodu koja započinje treniranje neuronske mreže. Zatim je izvršen test usporedbom rezultata dobivenih iz vrijednosti kojima je već poznat rezultat (vrijednosti iz tekstualne datoteke) sa onim dobivenim od same neuronske mreže. To testiranje se može nadalje podijeliti na dva manja dijela. U prvom dijelu uspoređujemo rezultate i vrijednosti koje su korištene u treniranju, a drugom dijelu testiramo rezultate i vrijednosti koje nisu bile korištene u treniranju neuronske mreže. Time dobivamo deset vrijednosti od neuronske mreže i deset vrijednosti koje su pročitane iz tekstualne datoteke. Ako je šest ili više vrijednosti bolje od prijašnjih, onda se vrijednosti mijenjaju, dok u protivnom one ostaju iste.

```
17.  public class NetworkOperationClass
18.    {
19.    ....
20.       private void LearningVariatian(int start, int x, double y)
21.       {  .....
22.          if (start == 100 && first == false)
23.          {
24.             lowestTraingingDifference1 = Math.Abs(realtrainingresult1 - trainingresult1);                ....
25.             lowestTestingDifference1 = Math.Abs(realtestresult1 - testresult1);                ....
26.             first = true;
27.          }
28.          else
29.          {
30.             trainingtemp1 = Math.Abs(realtrainingresult1 - trainingresult1);    ....
31.             testingtemp1 = Math.Abs(realtestresult1 - testresult1);   ....
32.             int count = 0;
33.             double[] temp1 = { trainingtemp1, ... , testingtemp1, ... };
34.             double[] temp2 = { lowestTraingingDifference1, .... , lowestTestingDifference1,  .... };
35.             for (int i = 0; i < 10; i++)
36.             {
37.                if (temp1[i] <= temp2[i])
38.                {
39.                   count++;
40.                }
41.             }
42.             if (count >= 6)
43.             {
44.                lowestTestingDifference1 = trainingtemp1;                    ....
45.                lowestTestingDifference1 = testingtemp1;                    ....
46.                current_lowest_epoch_value = network.JitterEpoch;
47.                current_lowest_noise_value = network.JitterNoiseLimit;
48.                current_lowest = start;
49.             }
50.          }
51.       }
```

Kôd 5.8. Usporedba i testiranje rezultata i vrijednosti korištenih pri treniranju

Dvije metode prikazane kôdom iznad (Kôd 5.8.) korištene su tako da su mijenjane gornje navedene vrijednosti, gdje jednu mijenjamo za 5, a drugu za 0.01. Treća se vrijednost mijenja ovisno o metodi - ona vrijednost koja je starija mijenja se za 100, kao što je prikazano u kôdu na sljedećoj stranici (Kôd 5.9.).

```
1.  public class NetworkOperationClass
2.    {
3.     ....
4.       public void AdvenceLearning(int start, int end)
5.       {
6.         while (start <= end)
7.         {
8.           LearningVariatian(start, current_noise_epoch, current_noise_limit);
9.           ...
10.          start = start + 100;
11.        }
12.      }
13.  }
```

Kôd 5.9. Usporedba (starija vrijednost)

Druga, novija vrijednost, mijenja se za 10, kao što je prikazano sljedećim kôdom (Kôd 5.10.). Ova se vrijednost kasnije koristila umjesto one promijenjene za 100 prvom metodom. Prilikom prvog izvršavanja podmetode, kao najbolji, uzima se prvi izračun, a svaki sljedeći izračun će se usporediti sa prethodnim.

```
14. public class NetworkOperationClass
15.   { ....
16.     public void DetailedLearning(int start, int end)
17.     {
18.       while (start <= end)
19.       {
20.         LearningVariatian(start, current_noise_epoch, current_noise_limit);
21.         .....
22.
23.         start = start + 10;
24.       }
25.     }
26. }
```

Kôd 5.10. Usporedba (novija vrijednost)

Kôd na sljedećoj stranici (Kôd 5.11.) prikazuje podatke o broju treninga, maksimalnu veličinu odstupanja vrijednosti u treningu i interval u treningu tijekom kojeg će se provesti funkcija na koju se nadodaje definirano odstupanje. Dobiveni podaci se zapisuju u tekstualnu datoteku "ZavršniRad.txt„, preko *StreamWriter* klase koja zatvara vezu na tekstualnu datoteku nakon unosa, u kojoj će se kasnije među zapisanim podacima tražiti njihov najbolji rezultat.

```
1.  class Program
2.    {
3.        static void Main(string[] args)
4.        {
5.            NetworkOperationClass class1 = new NetworkOperationClass();
6.            class1.AdvenceLearning(2000, 4000);
7.            StreamWriter sw = new StreamWriter("ZavrsniRad.txt");
8.            sw.WriteLine(class1.BestNumberOfEpoch());
9.            sw.WriteLine();
10.           sw.WriteLine(class1.BestNoiseLimit());
11.           sw.WriteLine();
12.           sw.WriteLine(class1.BestNoiseLimit());
13.           sw.WriteLine();
14.           sw.Close();
15.       }
16.   }
```

Kôd 5.11. Korištenje klase *StreamWriter*

5.5. Postupak treniranja neuronske mreže

Neuronska mreža je trenirana tako da je mijenjan broj neurona u skrivenom sloju i broj treninga koji će baza provesti. Granične vrijednosti i odstupanja su se definirale kao niz brojeva koji će se provesti za svaku vrijednost i uzeti najbolja. Pomak granične vrijednosti je za 5 veći ili manji od druge vrijednosti, dok se odstupanje pomicalo za vrijednost 0,01. Ovo treniranje je već jednom provedeno, bazirano na drugim podacima, no veći broj neurona u ulaznom sloju tada je činio treniranje baze težim zbog većeg broja kombinacija u skrivenom sloju. Također, pri tom treniranju nije bilo dovoljno podataka za njihovu optimalnu podjelu na podatke za treniranje i podatke za testiranje.

Najbolji rezultat	Najbolja granična vrijednost	Najbolje odstupanje	Broj neurona u skrivenom sloju
2000 (između 100 i 2000)	58 (između 58 i 88)	0,05(između 0 i 0,6)	5
4000 (između 2100 i 4000)	63 (između 58 i 88)	0,06(između 0 i 0,6)	5
1900 (između 100 i 2000)	63 (između 58 i 88)	0,05(između 0 i 0,6)	4
4000 (između 2100 i 4000)	73 (između 58 i 88)	0,06(između 0 i 0,6)	4
2000 (između 100 i 2000)	88 (između 58 i 88)	0,03(između 0 i 0,6)	6
4000 (između 2100 i 4000)	58 (između 58 i 88)	0,06(između 0 i 0,6)	6

Tablica 5.1. Podaci za treniranje (pomak granične vrijednosti ±5)

Nakon nekoliko prolazaka koja se vide u tablici na prethodnoj stranici (Tablica 5.1.), treniranje je ponovljeno, no uz manji opseg obuhvaćen prilikom treniranja neuronske mreže, te pomak vrijednosti broja treniranja pri izvedbi za deset, a ne za sto kao prije. Budući da bi isti pomak kod nepromijenjene vrijednosti broja treniranja zahtijevao više vremena, uzet je manji opseg te je sam taj opseg obuhvaćao najbolju vrijednost prošlog treniranja koja je zapisana u tekstualnu datoteku. Rezultat se nije ponovio, odnosno pronađena je bolja vrijednost kada se vrijednost broja treniranja promijenila na pomak od deset, u odnosu na rezultat dobiven uz prijašnji pomak vrijednosti broja treniranja. Obzirom na zapažanja proizišla iz ova dva izvođenja treniranja, treniranje je ponovljeno, uz broj neurona u skrivenom sloju koji je davao najbolje vrijednosti u prošlim izvođenjima. Dakle, pri pokretanju treniranja neuronske mreže, u konačnici je uzet broj treniranja sa pomakom od deset. Rezultate ovog treniranja možemo vidjeti u tablici ispod (Tablica 5.2.).

Najbolji rezultat	Najbolja granična vrijednost	Najbolje odstupanje	Broj neurona u skrivenom sloju
1500 (između 100 i 1500)	63 (između 58 i 88)	0,06 (između 0 i 0,6)	5
1990 (između 1510 i 2000)	73 (između 58 i 88)	0,04 (između 0 i 0,6)	5
2500 (između 2010 i 2500)	63 (između 58 i 88)	0,04 (između 0 i 0,6)	5
2980 (između 2510 i 3000)	88 (između 58 i 88)	0,06 (između 0 i 0,6)	5
3500 (između 3010 i 3500)	58 (između 58 i 88)	0,05 (između 0 i 0,6)	5
4000 (između 3510 i 4000)	58 (između 58 i 88)	0,04 (između 0 i 0,6)	5
4300 (između 4010 i 4300)	88 (između 58 i 88)	0,04 (između 0 i 0,6)	5
4600 (između 4310 i 4600)	58 (između 58 i 88)	0,05 (između 0 i 0,6)	5
4900 (između 4610 i 4900)	58 (između 58 i 88)	0,04 (između 0 i 0,6)	5
5000 (između 4910 i 5000)	68 (između 58 i 88)	0,06 (između 0 i 0,6)	5

Tablica 5.2. Podaci za treniranje

Rezultati iz tablice potom su međusobno uspoređeni kako bi bio odabran onaj najbolji. Najbolji rezultat je imao broj treniranja 4000, maksimalna veličina proizvoljne vrijednosti (pomaka) koja će se dodati rezultatu pri treniranju iznosila je 0.04, a interval dodavanja proizvoljne vrijednosti (pomaka) je svaki 58 prolazak u treniranju neuronske mreže.

Kôd naveden ispod (Kôd 5.12.) prikazuje dobivanje najboljeg rezultata u neuronskoj mreži. Po završetku izrade neuronske mreže, rad na aplikaciji za korisničku uporabu mogao je započeti.

```
1.    public class NetworkOperationClass
2.    {
3.      public NetworkOperationClass()    {  }     ....
4.
5.      public void BestResult(List<List<double>> list)
6.      {
7.        for (int i = 0; i < 100; i++)
8.        {
9.          foreach (List<double> item in list)
10.         {
11.           LearningVariatian(Convert.ToInt32(item[0]),
12.           Convert.ToInt32(item[1]), item[2]);
13.         }
14.         Console.WriteLine(string.Format("The best result: epoch number: {0},  jitter number:{1}, jitter noise:{2}",
15.           current_lowest, current_lowest_epoch_value,
16.           current_lowest_noise_value));
17.       }
18.
19.       Console.ReadKey();
20.     }
21.   }
```

Kôd 5.12. Neuronska mreža dobiva najbolje rezultate

6. Izrada aplikacije za korisničku uporabu

Nakon što je napravljena neuronska mreža koja daje približne vrijednosti kod treniranja i testiranja, sljedeći korak je izrada korisničke aplikacije u koju će korisnik moći unijeti vrijednosti, te dobiti rezultat temeljen na radu neuronske mreže. Napravljena je Windows Forms aplikacija koja ima šest tekstualnih polja, od kojih će u njih pet korisnik moći unijeti ulazne vrijednosti, te jedno koje će na dnu ispisati rezultat za upisane vrijednosti.

6.1. Izgled Windows Forms aplikacije

Isječak kôda na sljedećoj stranici (Kôd 6.1.) prikazuje inicijalizaciju elemenata forme. U njemu su vidljive postavke gumba, polja za prikaz teksta i polja za unos ulaznih vrijednosti za obradu u neuronskoj mreži.

```
1.  partial class Form1
2.    {
3.      private System.ComponentModel.IContainer components = null;
4.      protected override void Dispose(bool disposing)
5.      {
6.        if (disposing && (components != null))
7.        {
8.          components.Dispose();
9.        }
10.       base.Dispose(disposing);
11.     }
12.     private void InitializeComponent()
13.     {
14.       this.textBox6 = new System.Windows.Forms.TextBox();  ....
15.       this.button1 = new System.Windows.Forms.Button();
16.       this.label6 = new System.Windows.Forms.Label();  ....
17.       this.SuspendLayout();
18.       this.textBox6.Location = new System.Drawing.Point(134, 274);
19.       this.textBox6.Name = "textBox6";
20.       this.textBox6.ReadOnly = true;
21.       this.textBox6.Size = new System.Drawing.Size(129, 20);
22.       this.textBox6.TabIndex = 23;        .....
23.       this.button1.Enabled = false;
24.       this.button1.Location = new System.Drawing.Point(14, 271);
25.       this.button1.Name = "button1";
26.       this.button1.Size = new System.Drawing.Size(75, 23);
27.       this.button1.TabIndex = 17;
28.       this.button1.Text = "Calculate";
29.       this.button1.UseVisualStyleBackColor = true;
30.       this.button1.Click += new System.EventHandler(this.button1_Click);   ......
31.       this.label6.AutoSize = true;
32.       this.label6.Location = new System.Drawing.Point(131, 258);
33.       this.label6.Name = "label6";
34.       this.label6.Size = new System.Drawing.Size(40, 13);
35.       this.label6.TabIndex = 29;
36.       this.label6.Text = "Result:";
37.       this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
38.       this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
39.     }
40.   }
```

Kôd 6.1. Prilagodba Windows Forms forme za unos ulaznih podataka

Isječak kôda koji možemo vidjeti na sljedećoj stranici (Kôd 6.2.) prikazuje postavke same forme, njezine dimenzije (visinu i širinu), vrijednost naslova te primjer definiranja nekih komponenata unutar same forme.

```
41.  partial class Form1
42.     {           ....
43.         this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
44.         this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
45.         this.ClientSize = new System.Drawing.Size(291, 314);  ....
46.         this.Controls.Add(this.label6);
47.         this.Controls.Add(this.textBox6);  ....
48.         this.Controls.Add(this.button1);
49.         this.Name = "Form1";
50.         this.Text = "Airfoil Self Noise Network";
51.         this.Load += new System.EventHandler(this.Form1_Load);
52.         this.ResumeLayout(false);
53.         this.PerformLayout();
54.
55.       }
56.
57.       #endregion
58.
59.       private System.Windows.Forms.TextBox textBox6;  .....
60.       private System.Windows.Forms.Button button1;
61.       private System.Windows.Forms.Label label6;  ......
62.    }
```

Kôd 6.2. Prilagodba Windows Forms forme (grafička svojstva)

6.2. Priprema neuronske mreže u Windows Forms aplikaciji

Prvo što se definiralo u formi, vidljivo u kôdu ispod (Kôd 6.3.), su parametri koje će forma koristiti u pozadini pri stvaranju i korištenju neuronske mreže.

```
1.   public partial class Form1 : Form
2.     {
3.       private LinearLayer inputLayer;
4.       private SigmoidLayer outputLayer;
5.       private SigmoidLayer hiddenLayer;
6.
7.       private BackpropagationConnector connector1;
8.       private BackpropagationConnector connector2;
9.       private BackpropagationNetwork network;
10.      private TrainingSet trainingSet;
11.      private double[][] data;
12.      private double[] dataResult;
13.      private int numberOfEpochs = 4000;
14.      private int jitterOnEpoch = 58;
15.      private double jitterNoise = 0.04;
16.
17.      public Form1()
18.      {
19.        InitializeComponent();
20.      }
21.      ....
22.  }
```

Kôd 6.3. Definiranje parametara u Windows Forms

Slijedi korištenje definiranih parametara sa zadanim vrijednostima, stvaranje neuronske mreže i pridruživanje vrijednosti na preostale parametre. Sve navedeno izvršeno je u *Form Load* događaju (*Form1_Load*). U nastavku možemo vidjeti isječak kôda iz spomenutog događaja (Kôd 6.4.).

```
23.  public partial class Form1 : Form
24.    {
25.        ....
26.
27.        private void Form1_Load(object sender, EventArgs e)
28.        {
29.            inputLayer = new LinearLayer(5);
30.            outputLayer = new SigmoidLayer(1);
31.            hiddenLayer = new SigmoidLayer(5);
32.
33.            connector1 = new BackpropagationConnector(inputLayer, hiddenLayer);
34.            connector2 = new BackpropagationConnector(hiddenLayer, outputLayer);
35.            network = new BackpropagationNetwork(inputLayer, outputLayer);
36.            trainingSet = new TrainingSet(5, 1);
37.            network.Initialize();
38.            getData();
39.
40.            for (int i = 0; i < data.Length; i++)
41.            {
42.                TrainingSample sample = new TrainingSample(
43.                new double[] { data[i][0], data[i][1], data[i][2],
44.                data[i][3], data[i][4] }, new double[] { dataResult[i] });
45.                trainingSet.Add(sample);
46.            }
47.            network.JitterEpoch = jitterOnEpoch;
48.            network.JitterNoiseLimit = jitterNoise;
49.            network.Learn(trainingSet, numberOfEpochs);
50.
51.            button1.Enabled = true;
52.        }
53.    }
```

Kôd 6.4. Korištenje parametara pomoću *Form Load* događaja

Unutar *Form Load* događaja se poziva na događaj *getData* koji ima istu funkcionalnost kao i *DataManagement* klasa, dakle čitanje podataka iz tekstualnog oblika, uz razliku što su čitani podaci ovdje spremljeni u *resource* datoteku. Zanemarimo li tu razliku, sami kôd je isti, a isječak kôda koji prikazuje pozivanje na događaj *getData* možemo vidjeti na sljedećoj stranici (Kôd 6.5.).

```
54.  public partial class Form1 : Form
55.    {
56.       ....
57.       private void getData()
58.       {
59.          string stream = Resources.TrainingData;
60.          ....
61.       }
62.    }
63.  }
```

Kôd 6.5. Korištenje *getData* događaja

6.3. Kôd za računanje rezultata

U kôdu za računanje rezultata potrebno je koristiti nove parametre koji nisu navedeni u prošlom isječku kôda. U kôdu ispod koji se nastavlja na sljedećoj stranici (Kôd 6.6.) je prikazan primjer provjere vrijednosti unesenih od strane korisnika, njihov unos u neuronsku mrežu, izračun vrijednosti i njezin prikaz u formi.

```
1.   public partial class Form1 : Form
2.     {
3.        ....
4.        private double resultDifference = 140.987;
5.        private double frequencyDifference = 20000;
6.        private double angleOfAttackDifference = 22.2;
7.        private double chordLengthDifference = 0.3048;
8.        private double freeStreamVelocityDifference = 71.3;
9.        private double suctionSideDifference = 0.0584113;
10.       ....
11.
12.       private void button1_Click(object sender, EventArgs e)
13.       {
14.          double[] values = new double[5];
15.          bool ok = true;
16.          double number;
17.          if (this.textBox1.Text != "" && double.TryParse(this.textBox1.Text, out number)
18.             && Convert.ToDouble(this.textBox1.Text) >= 0
19.             && Convert.ToDouble(this.textBox1.Text) <= frequencyDifference)
20.          {
21.             if (Convert.ToDouble(this.textBox1.Text) == 0)
22.             {
23.                values[0] = Convert.ToDouble(this.textBox1.Text);
24.             }
25.             else if (Convert.ToDouble(this.textBox1.Text) > 0
26.                && Convert.ToDouble(this.textBox1.Text) < 1
27.                && this.textBox1.Text.Contains(',') == false)
28.             {
29.                ok = false;
30.                MessageBox.Show("Frequency decimal number value needs to be seperated with , and not with .");
31.                Application.Exit();
32.             }
33.             else
34.             {
```

```
35.                 values[0] = Convert.ToDouble(this.textBox1.Text) / frequencyDifference;
36.             }
37.         }
38.         else
39.         {
40.             ok = false;
41.             MessageBox.Show(string.Format("Frequency needs to a number be between {0} and {1}",
42.                 0, frequencyDifference));
43.             Application.Exit();
44.         }
45.         ....
46.         if (ok)
47.         {
48.             double result = network.Run(values)[0];
49.             result = result * resultDifference;
50.             this.textBox6.Text = result.ToString();
51.         }
52.     }
53. }
```

Kôd 6.6. Rad neuronske mreže po unosu korisnikovih vrijednosti

Zaključak

U ovom sam radu uspio napraviti vlastitu neuronsku mrežu oslanjajući se na programski jezik C# i *open-source library*, te podatke preuzete sa web stranice Sveučilišta u Karolini. Za navedene podatke i pomoćni alat nije bilo potrebno ulagati novac, već samo vrijeme za samu izradu neuronske mreže. Važno je napomenuti kako ovaj rad nije rađen u Python programskom jeziku, već u programskom jeziku C# s kojim sam puno bolje upoznat i koji sam koristio kroz puno duži vremenski period. Ovu sam činjenicu htio posebno naglasiti jer je većina primjera neuronskih mreža koje je moguće pronaći pisana upravo u Python programskom jeziku, što je značilo da sam se prilikom izrade praktičnog dijela ovog završnog rada mogao osloniti na mnogo manji broj primjera koji su mi, bez obzira na njihov mali broj, uvelike pomogli prilikom izrade vlastite neuronske mreže. Izrađena neuronska mreža korištena je za izračun aerodinamičkog profila, te je polučila upotrebljive rezultate, čime sam potvrdio uspješnost praktičnog dijela svog završnog rada.

Popis kratica

MIT	*Massachusetts Institute of Technology* Tehnološki institut Massachusetts
GPU	*Graphic Processing Unit* grafička procesorska jedinica
CMOS	*Complementary Metal–Oxide–Semiconductor* komplementarni metal oksid poluvodič
AI	Artifical Intelligence umjetna inteligencija
IDSIA	*Istituto Dalle Molle di Studi sull'Intelligenza Artificiale* Dalle Molle institut za istraživanje umjetne inteligencije
ICDAR	*International Conference on Document Analysis and Recognition* Međunarodna konferencija za analizu i prepoznavanje dokumenata
IJCNN	*International Joint Conference on Neural Networks* Međunarodna zajednička konferencija za neuronske mreže
ISBI	*International Symposium on Biomedical Imaging* Međunarodni simpozij za biomedicinsko slikanje
MNIST	Mixed National Institute of Standards and Technology Mješoviti nacionalni institut za standarde i tehnologiju
ANN	Artifical Neural Network umjetna neuronska mreža
ECMA	*European Computer Manufacturers Association* Društvo europskih proizvođača računala
ISO	*International Organization for Standardization* Međunarodna organizacija za normizaciju
Cool	*C-like Object Oriented Language* C-nalik objektno orijentiran jezik

Popis slika

Popis tablica

Popis kôdova

Literatura

[1] Neuronska mreža
https://en.wikipedia.org/wiki/Artificial_neural_network
http://neuralnetworksanddeeplearning.com/
[2] C# programski jezik
https://en.wikipedia.org/wiki/C_Sharp_(programming_language)
[3] NET Class Library
https://msdn.microsoft.com/en-us/library/gg145045(v=vs.110).aspx
[4] NeuralNetCore
/NeuronDotNet%20Source/ReadMe.html
https://sourceforge.net/projects/neurondotnet/

Oblikovano: Hiperveza, Font: (Zadano) Times New Roman, Nepodcrtano

Oblikovano: Font: (Zadano) Times New Roman