ACTA MATHEMATICA SPALATENSIA
Series didactica
Vol.6 (2023) 129–146

Zaprimljen: 13. lipnja 2023.
Prihvaćen: 18. rujna 2023.

# Razvijanje plitkih neuronskih mreža pomoću MATLAB-a. Primjena na rješavanje parametarski ovisnih diferencijalnih jednadžbi

Paulina Bakoč, Martin Lazar

**Sažetak**

Cilj ovog rada je objasniti kako se stvara, razvija i koristi plitka neuronska mreža korištenjem programskog alata MATLAB. Kao testni primjer uzet je problem predviđanja izlaznih vrijednosti rješenja parametarski ovisne diferencijalne jednadžbe prvog reda. Dobiveni rezultati potvrđuju sposobnost neuronske mreže da brzo i pouzdano reproducira tražene vrijednosti bez potrebe rješavanja same jednadžbe

*Ključni pojmovi: neuronska mreža, parametarski ovisne obične diferencijalne jednadžbe, nadzirano strojno učenje, logistički rast populacije.*

## 1. Uvod

Zadaća nadziranog strojnog učenja je aproksimacija nepoznate funkcije $f$ koja preslikava ulazne vrijednosti $x$ u izlaze $y$ na osnovu $N$ poznatih podataka $(x_i, y_i = f(x_i))$, $i = 1, \ldots, N$. Ovisno o vrsti ovisne varijable $y$, razlikujemo dvije vrste učenja: klasifikaciju ($y$ poprima vrijednosti unutar konačnog skupa) i regresiju ($y$ poprima kontinuum vrijednosti). Prema ovoj podjeli, $f$ se naziva klasifikator ili regresor za odgovarajući zadatak.

Sama aproksimacija se postiže pomoću matematičkog modela čija izgradnja se zasniva na podacima raspoloživim za učenje. Ti se podaci

mogu podijeliti u tri skupine: podaci za treniranje, podaci za testiranje i validacijski podaci. Prva skupina se koristi za početnu izgradnju modela. Pomoću validacijskih podataka se dodatno podešavaju parametri modela, pritom povečavajući njegovu točnost. Konačno se s podacima za testiranje provjerava njegova uspješnost u radu s novim podacima, te oni ne utječu na samu izgradnju modela.

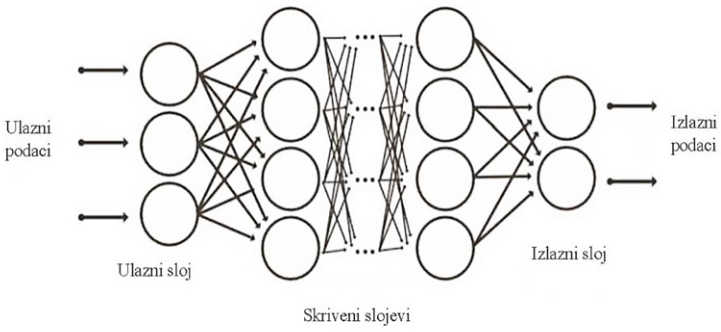
Posebna vrsta strojnog učenja su (umjetne) neuronske mreže. One se sastoje od mnoštva jednostavnih jedinica, neurona, koji za zadaću imaju primanje i prenošenje signala s jednog na drugi neuron. Njihov struktura oponaša rad bioloških neurona u ljudskom mozgu. Neuroni su u mreži razvrstani u slojeve između kojih se odvija prijenos podataka, odnosno signala. Neuronske mreže s dva ili tri sloja poznate su kao plitke neuronske mreže.

U ovom radu opisujemo razvoj plitkih neuronskih mreža kroz programski paket MATLAB. Za primjer je uzet problem predviđanja izlaznih vrijednosti rješenja parametarski ovisne diferencijalne jednadžbe koja modelira logistički rast populacije. U sljedećem poglavlju dajemo kratki opis neuronskih mreža i njihove strukture. Treće poglavlje sadrži opis modela koji koristimo, kao i detaljne upute za kreiranje neuronske mreže pomoću grafičkog sučelja u MATLAB-u. Rad završava sa zaključnim razmatranjima, mogućim dosezima kao i problemima pri korištenju opisanog postupka na općenitije diferencijalne jednadžbe.

## 2. Neuronske mreže

Danas su neuronske mreže jedan od najpopularnijih i najraširenijih algoritama strojnog učenja [4]. U ovom radu ćemo se ograničiti na jednu standardnu klasu neuronskih mreža, takozvane unaprijedne neuronske mreže (eng. feedforward neural networks). U njima razmjena podataka među slojevima ide u samo jednom smjeru, od ulaznog k izlaznom sloju (slika 1), pri čemu se na podacima naizmjenično primjenjuju afine transformacije i nelinearne aktivacijske funkcije.

Kako bismo preciznije opisali njihov rad, koristit ćemo formalnu definiciju iz [7]. Pritom neka je s $(L+1) \in \mathbb{N}$ označen broj slojeva u mreži, te s $N_i \in \mathbb{N}$, $i = 0, \ldots, L$ broj neurona u svakom sloju. Nadalje, kako bismo definirali afine transformacije podataka koristimo težinske matrice $W_i$ tipa $N_i \times N_{i-1}$ te $N_i$-dimenzionalne pomake (eng. biases) $b_i$, za $i = 1, \ldots, L$. Nelinearnost ulazi u model kroz (nelinearnu) aktivacijsku funkciju $\rho\colon \mathbb{R} \to \mathbb{R}$. Pritom s $\rho_n\colon \mathbb{R}^n \to \mathbb{R}^n$ označujemo primjenu funkcije $\rho$ po komponentama na $n$-dimenzionalni vektor. Sad formalno možemo definirati neuronsku mrežu kao funkciju $r_L\colon \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ defi-




Slika 1. Skica unaprijedne neuronske mreže [6]

niranu rekurzivno na sljedeći način

$$\begin{aligned} &r_L(x) = W_L r_{L-1}(x) + b_L, \\ &r_i(x) = \rho_{N_i}(W_i r_{i-1}(x) + b_i), i = 1, \ldots, L-1, \\ &r_0(x) = x. \end{aligned}$$

Najčešće korištene aktivacijske funkcije su ispravljena linearna jedinica (ReLU), sigmoid i tangens hiperbolni. Pritom je ReLU aktivacijska funkcija definirana izrazom $g(x) = \max\{0, x\}$, dok je sigmoid zadan s

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

Na osnovu čuvenih rezultata univerzalne aproksimacije [1, 2], svaka neprekidna funkcija $f\colon \mathbb{R}^p \to \mathbb{R}^N$ se može proizvoljno točno aproksimirati plitkom neuronskom mrežom s diferencijabilnom aktivacijskom funkcijom (npr. sigmoidom ili tangens hiperbolnim). To se postiže učenjem mreže na osnovu prikupljenih podataka $(x_i, f(x_i)) \in \mathbb{R}^p \times \mathbb{R}^N, i = 1, \ldots, n$. Pritom se učenje sastoji od optimalnog izbora težina i pomaka, odnosno minimizacijom srednje kvadratne pogreške

$$MSE = \frac{1}{n} \sum_i \|\Phi(x_i) - f(x_i)\|^2$$

s po svim mogućim izborima matrica $W_i$ te vektora $b_i$, $i = 1, \ldots L$.

Greška se obično minimizira korištenjem odgovarajućih gradijentnih metoda. Pritom se gradijent pogreške $MSE$ može efikasno izračunati korištenjem unatragnog širenja (eng. backpropagation), na primjer [8]. U ovom radu ćemo proces učenja i optimizacije neuronske mreže obaviti korištenjem gotovih programskih paketa dostupnih u MATLAB-u.

# 3. Razvijanje plitkih neuronskih mreža pomoću MATLAB-a

## 3.1. Testni primjer

Razmatramo niz Cauchyevih zadaća za običnu diferencijalnih jednadžbu oblika

$$\frac{dy}{dt} = ky\Big(1 - \frac{y}{M}\Big) \tag{1.1}$$

s početnim uvjetom

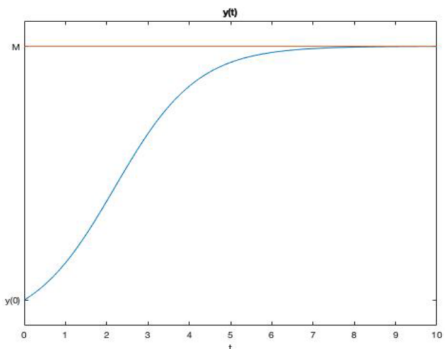
$$y(0) = y_0 > 0. \tag{1.2}$$

Gornja jednadžba modelira logistički rast populacije (eng. *logistic growth model*). Za razliku od eksponencijalnog modela rasta, određenog jednadžbom oblika $y' = ky$, logistički model je nelinearan i ograničava maksimalnu veličinu populacije na konstantu $M$ koja se naziva nosivi kapacitet ([3], §4.4). Vrijednost te konstante predstavlja maksimalan broj jedinki koje mogu nastanjivati neko područje. U praksi se za konkretne primjere njena vrijednost određuje na osnovu prostornog kapaciteta, te dostupnosti svega potrebnog za život. Pozitivni parametar $k$ predstavlja brzinu rasta modela. Što je on veći, to će se veličina populacije prije približiti nosivom kapacitetu $M$.

Rješenje gornje Cauchyeve zadaće dano je funkcijom

$$y(t) = \frac{M}{1 + Ae^{-kt}}, \tag{2}$$

pri čemu je $A = \frac{M - y_0}{y_0}$. Grafička skica rješenja y prikazana je na slici 2. Vidimo da je rješenje rastuća funkcija koja se asimptotski približava nosivom kapacitetu $M$.




Slika 2. Rješenje zadaće (1)

U ovom radu je vrijednost nosivog kapaciteta fiksirana na $M = 1000$, te u daljnjem proučavamo model

$$\frac{dy}{dt} = ky\Big(1 - \frac{y}{1000}\Big)$$

s početnim uvjetom

$$y(0) = y_0.$$

Vrijednosti parametra $k$ i početnog podatka $y_0$ nisu fiksirane, te variraju u rasponu od 1 do 10 (za $k$), te od 10 do 100 (za $y_0$). U daljnjem ćemo radu napraviti odgovarajuću neuronsku mrežu čiji zadatak će biti procjena vrijednost rješenja $y(T)$ u trenutku $T = 1$, za dane vrijednosti parametara $k$ i $y_0$, bez korištenja formule (2) za rješenje početne zadaće (1).

## 3.2. Kreiranje neuronske mreže pomoću grafičkog sučelja u MATLAB-u

Kako bismo mogli stvoriti neuronsku mrežu za rješavanje gore naznačenog problema, moramo najprije definirati podatke (ulazne i izlazne) koji će se koristiti za njeno obučavanje. U našem primjeru su ulazni podaci diskretne vrijednosti parametara $k$ i $y_0$ uvedenih u prethodnom potpoglavlju. U tu svrhu definiramo uniformnu dvodimenzionalnu mrežu točaka, dobivenu diskretizacijom parametra $k$ s korakom 0.5, te parametara $y_0$ s korakom 5. Pripadne izlazne vrijednosti računamo preko formule (2). Konstrukcija navedenih podataka napravljena je pomoću funkcije podatci.m.

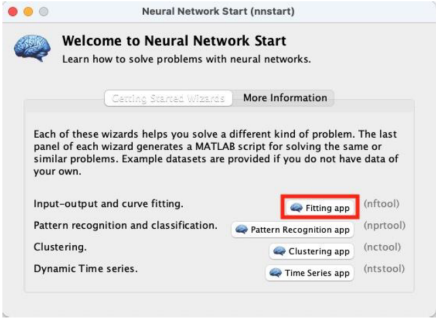
```
function [ulaz,izlaz] = podatci


%funkcija generira podatke za obučavanje neuronske mreže.
%funkcija vraća varijable:
%ulaz - skup ulaznih vrijednosti parametara k i y_0
%izlaz - vrijednosti rješenja y(1) u točkama ulaza


k=1:0.5:10;
y0=10:5:100;
[K,Y0]=meshgrid(k,y0);
Kvec=reshape(K,1,[]);
Y0vec=reshape(Y0,1,[]);
ulaz=[Kvec;Y0vec];
izlaz=1000*Y0vec./(Y0vec+(1000-Y0vec).*exp(-Kvec));
```

Izvršenjem naredbe

```
>>[input,output]=podatci
```

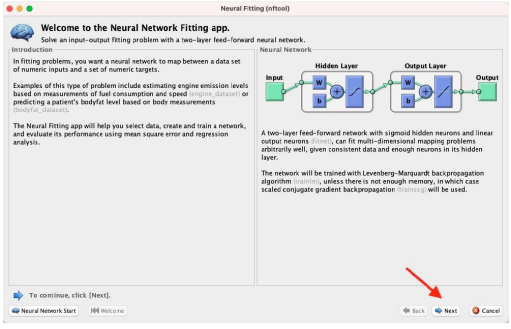
u komandnom prozoru MATLAB-a, ulazni i izlazni podaci se redom pospremaju u varijable *input* i *output*. Pritom je prva varijabla dvoredčana matrica čiji broj stupaca je jednak broju ulaznih parova vrijednosti parametara $k$ i $y_0$ ($19 \times 19 = 361$ u našem slučaju), dok je *output* matrica redak s jednakim brojem stupaca.

U sljedećem koraku, utipkavanjem naredbe *nnstart* u komandnom prozoru MATLAB-a, otvaramo sučelje za kreiranje plitkih neuronskih mreža (slika 3). Za rješavanje našeg problema koristit ćemo opciju *Fitting app*, koja se koristi za rješavanje regresijskih problema, odnosno nalaženje preslikavanja koje najbolje odgovara danim podacima. Osim spomenute opcije mogu se koristi i aplikacije za raspoznavanje uzoraka, klasifikaciju i grupiranje, koje nisu tema ovog članka.



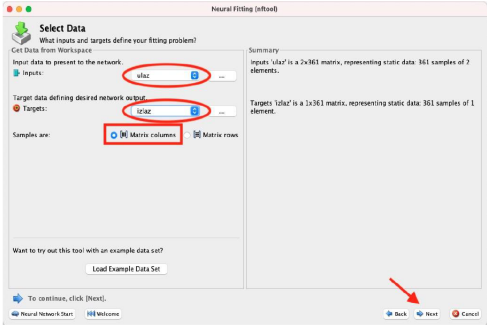

Slika 3. Sučelje dobiveno naredbom „nnstart” i odabir odgovarajuće opcije

Odabirom opcije *Fitting app* otvara se novo sučelje (slika 4) u kojem su nam detaljno objašnjene mogućnosti ove opcije.



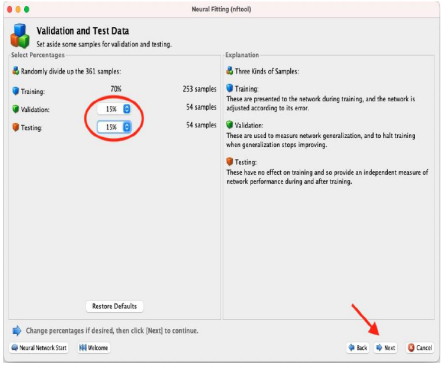

Slika 4. Sučelje dobiveno odabirom *Fitting app* opcije

Odabirom opcije *Next* otvara se sučelje za odabir ulaznih i izlaznih podataka, te se oni specificiraju u prozorčićima zaokruženima na slici 5. Također na istom sučelju moramo odabrati na koji način će naša mreža pristupati čitanju podataka, odnosno da li su oni spremljeni po retcima ili po stupcima. S obzirom da naša funkcija *podatci.m*, kako je već gore objašnjeno, sprema ulazne i izlazne vrijednosti po stupcima, odabiremo *Matrix columns* opciju. Program ispravno prepoznaje podatke, te u desnom dijelu sučelja navodi da je učitao uzorak veličine 361 podatka, pri čemu se svaki sastoji od dva ulazna elementa, te jednog izlaznog. Nakon što smo učitali podatke ponovno izabiremo opciju *Next*.



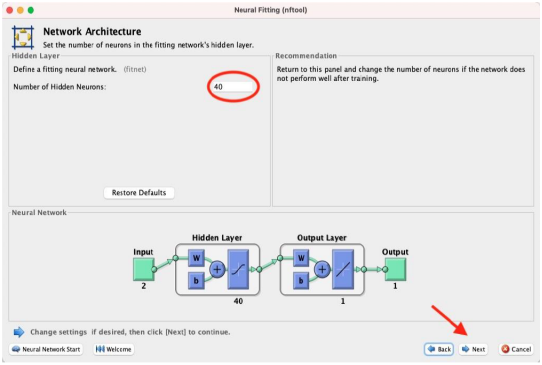

Slika 5. Učitavanje podataka u neuronsku mrežu

Otvara nam se novo sučelje u kojem određujemo koliki postotak učitanih podataka odvajamo za učenje, validaciju i testiranje. O ovim pojmovima već je bilo govora u uvodnom poglavlju. Koristit ćemo ponuđenu standardnu raspodjelu podataka (70-15-15%), te za nastavak izgradnje neuronske mreže biramo *Next* (slika 6).




Slika 6. Odabir broja podataka za validaciju i testiranje

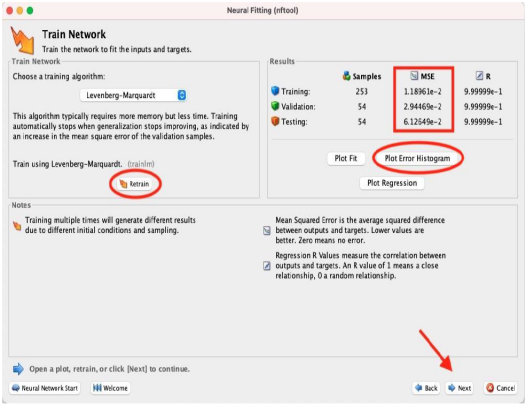
Unosimo broj neurona u skrivenom sloju i nastavljamo dalje. U dnu sučelja možemo vidjeti izgled naše mreže (slika 7). Pritom nažalost nije moguće a priori odrediti optimalan broj neurona. Njihov premali broj rezultirat će u podkapacitiranom modelu koji ne može točno simulirati ulazno-izlaznu funkciju, a prevelik broj dovodi do njegove prekomjerne prilagodljivosti (engl. *overfitting*). Stoga se u izgradnji mreže može krenuti od zadanog standardnog broja neurona (deset), a zatim se on po potrebi modificira ukoliko nismo zadovoljni s postupkom učenja mreže koji se provodi u idućem koraku. Mi ćemo izgraditi mrežu s 40 neurona u skrivenom sloju.



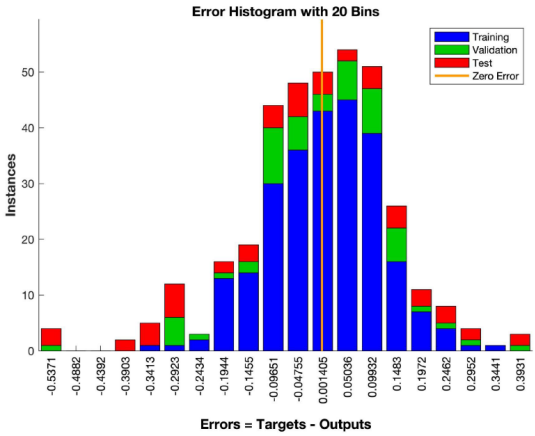

Slika 7. Arhitektura neuronske mreže

Nakon što smo stvorili i konfigurirali mrežu, prelazimo na najvažniji korak, odnosno na treniranje mreže. Odabirom opcije *Train/Retrain*, koju možemo vidjeti zaokruženu na slici 8, pokrećemo postupak učenja. Rezultate tog procesa možemo vidjeti u desnom dijelu slike, gdje su dane vrijednosti srednje kvadratne pogreške MSE, i to odvojeno za svaku od tri skupine podataka (za učenje, validaciju i testiranje). Ukoliko nismo zadovoljni rezultatima, odnosno dobivenom točnošću, postupak učenja se može ponoviti odabirom *Retrain* opcije. Pritom se koriste opet isti podaci, ali ovaj put drugačije raspoređeni među tri gore navedene skupine. Mi smo postupak obučavanja završili nakon što je srednja kvadratna greška na testnim podacima pala ispod $10^{-1}$. Ukoliko se željena točnost ne može dobiti postupkom ponovljenog učenja (*retrain*), potrebno se vratiti na prethodni korak i prilagoditi broj neurona u mreži.

Odabirom opcije *Plot Error Histogram* dobivamo grafički prikaz raspodjele grešaka, opet odvojeno za tri skupine podataka (slika 9). Najmanje vrijednosti grešaka su dobivene na podatcima za učenje, što je logično obzirom da se na njima mreža konfigurira.



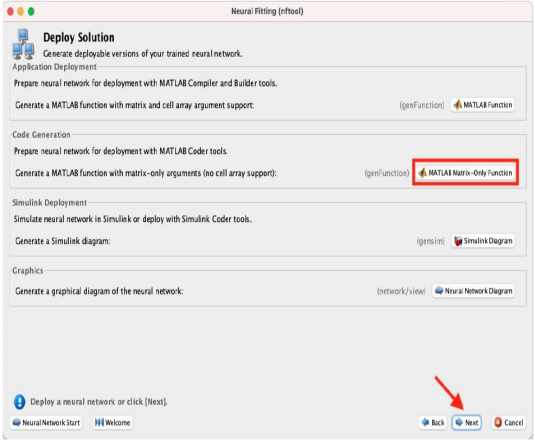

Slika 8. Sučelje za obuku




Slika 9. Dijagram pogrešaka dobivenih tijekom konfiguracije neuronske mreže

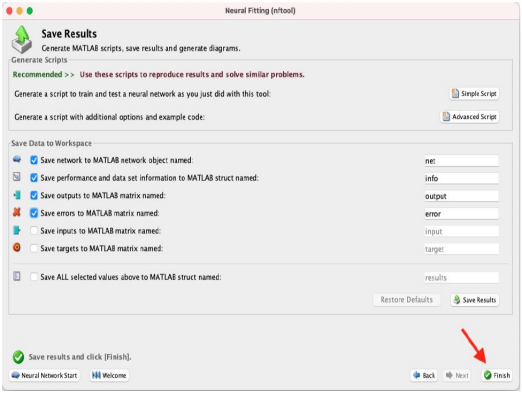
Nakon što smo uspješno konfigurirali mrežu s zadovoljavajućom točnosti, možemo je sad koristit za daljnja predviđanja izlaznih vrijednosti na novim ulaznim podacima. U tu svrhu ćemo u sljedećem prozoru obučenu mrežu pospremiti na način koji će omogućiti njeno efikasno korištenje u budućnosti (slika 10). Generiramo MATLAB funkciju s matričnim zapisom ulaznih podataka (*MATLAB Matrix-only function*) i spremamo je na naše računalo pod imenom *Neuronska_mreza*, te ćemo njenim pozivanjem u bilo kojem trenutku moći koristiti obučenu mrežu. Ulazne vrijednosti te funkcije su parovi vrijednosti parametara $k$ i $y_0$, a izlaz približna vrijednost $y(1)$ rješenja zadaće (1) u trenutku $t = 1$,

izračunata bez korištenja eksplicitne formule za rješenje, već primjenom obučene neuronske mreže. Njezin kod je dan u Dodatku.




Slika 10. Pospremanje obučene mreže

Sam postupak konfiguriranja i obuke mreže završava u idućem prozoru gdje se (opcionalno) mogu pohraniti još neki od podataka generiranih tijekom postupka (slika 11).




Slika 11. Završetak postupka konfiguriranja i obuke mreže

## 3.3. Rezultati

Rad neuronske mreže konfigurirane i obučene u prethodnom poglavlju ispitan je na nizu ulaznih vrijednosti parametara $k$ i $y_0$. U tu svrhu

definiramo vektore testnih parametara

```
>> k_test=1.05:0.3:9.8;
>> y0_test=12.55:2.9:99;
```

svaki dimenzije 30, te pomoću njih konstruiramo uniformnu dvodimenzionalnu mrežu (od 900) točaka. Za svaku točku računamo pripadne vrijednosti rješenja zadaće (1) u trenutku $t = 1$ primjenom obučene neuronske mreže, odnosno korištenjem MATLAB funkcije *Neuronska_mreza* generirane u prethodnom poglavlju. Dobiveni rezultati uspoređeni su s egzaktnim rješenjem $y(1)$ dobivenim primjenom formule (2).

Vrijednosti testnih parametara su izabrane ne način da se ne podudaraju s vrijednostima korištenima u razvoju i obuci mreže. Na taj način testiranje provodimo na, za mrežu, potpuno novim podacima. Sam postupak izračuna i usporedbe provodi se primjenom, u tu svrhu konstruirane MATLAB funkcije *provjera*, čiji kod navodimo u nastavku.

```
function [skg,skrg] = provjera(k_test,y0_test)


%funkcija provjerava rad neuronske mreže (pohranjene u funkciji
Neuronska_mreza) na testnom %skupu ulaznih podataka uspoređujući
izlaze dobivene neuronskom mrežom sa stvarnim %rješenjima
%ULAZI funkcije su testne vrijednosti parametara k i y0
%IZLAZI funkcije su
%skg - srednja kvadratna greška
%skrg - srednja kvadratna relativna greška


[K,Y0]=meshgrid(k_test,y0_test);
Kvec=reshape(K,1,[]);
Y0vec=reshape(Y0,1,[]);

y1_ex=1000./(1+(1000-Y0vec)./Y0vec.*exp(-Kvec));
y1_nn=Neuronska_mreza([Kvec;Y0vec]);
greska=y1_ex-y1_nn;
skg=(mse(greska));
rel_greska=greska./y1_ex;
skrg=(mse(rel_greska));
end
```

Pozivanjem funkcije u komandnom prozoru

```
>> [skg,skrg]=provjera(k_test,y0_test)
```

dobijemo sljedeće rezultate

```
skg = 0.0424
skrg = 4.0072e-06.
```

Prvi od dobiveni brojeva ($skg = 0.0424$) nam daje vrijednost srednje kvadratne greške na novotestiranim podacima. S obzirom da je istovjetna greška na podacima za testiranje korištenim u procesu učenja mreže iznosila 6.1e-2 (slika 8), vidimo da je točnost mreže, nakon provjere na novom i većem skupu podataka (njih 900), ostala na istoj razini, što je u svakom slučaju pozitivan rezultat.

Iako se dobivena greška doima malom i prihvatljivom, prije nego što potvrdimo efikasnost rada neuronske mreže, važno ju je usporediti sa stvarnim veličinama, odnosno naći relativne greške. U ovom primjeru se vrijednosti rješenja $y(1)$ za razmatrani skup parametara kreću u rasponu od 2 do 1000. Relativne se greške, dobivene kao omjer greške za pojedini ulaz i pripadne vrijednosti rješenja, također računaju unutar gore navedene funkcije *provjera*, te je njihova srednja kvadratna veličina dana drugom izlaznom varijablom *skrg*. S obzirom da njena vrijednost iznosi 4.0072e-06, vidimo da su aproksimacije dobivene primjenom neuronske mreže zanemarive u odnosu na stvarne vrijednosti, pa time možemo potvrditi učinkovitost i preciznost rada konstruirane neuronske mreže.

# 4. Zaključak

U ovom radu prikazana je izrada i razvoj neuronske mreže korištenjem programskog alata MATLAB. Mreža je razvijena i testirana na primjeru parametarski ovisne diferencijalne jednadžbe prvog reda. Jednadžba modelira logistički rast populacije (*logistic growth model*), a varijabilni ulazni podaci su vrijednost početnog uvjeta $y_0$, te vrijednost parametra $k$ koji predstavlja brzinu rasta populacije. Izlazni podatak, kojeg je mreža trebala reproducirati, je vrijednost rješenja jednadžbe u trenutku $t = 1$.

U primjeru se vrijednosti parametara $k$ i $y_0$ kreću u rasponu od 1 do 10, odnosno od 10 do 100. Mreža je obučena na 19x19 ulaznih, ravnomjerno raspoređenih podataka. Rezultati dobiveni naknadnim testiranjem na 900 novih podataka potvrđuju točnost rada mreže, te njenu sposobnost da uz malu grešku reproducira traženo rješenje.

Razmatrani testni primjer je školske naravi, budući da promatrana jednadžba ima eksplicitnu formulu za rješenje. No on nam je omogućio provjeru rada razvijene neuronske mreže i usporedbu dobivenih rezultata sa stvarnima. Dobro poklapanje numeričkih i teorijskih rezultata sugerira mogućnost primjene na složenije zadaće koje se ne mogu riješiti egzaktno. Rješavanje takvih zadaća obično zahtijeva primjenu složenih i skupih numeričkih algoritama. Stoga je u slučaju parametarski ovisnih

zadaća, rješavanje problema od *nule* za svaku novu vrijednost parametra neučinkovit način koji ne uzima u obzir spoznaje i podatke prikupljene u prijašnjim rješavanjima sličnih problema. Nasuprot takvom pristupu, u ovom radu smo izradili neuronsku mrežu koja se obučava koristeći rješenja problema za konačan, po mogućnosti mali, skup parametara. Nakon toga mreža može efikasno procijeniti vrijednost rješenja (u našem slučaju brojnost populacije nakon $t = 1$ godine) za proizvoljne vrijednosti parametara, bez da rješava samu jednadžbu.

Opisani postupak može se također smatrati primjerom vanmrežnog/mrežnog (eng. *offline/online*) procesa. U prvom se (vanmrežnom) dijelu takvih procesa rješava konačan, po mogućnosti mali, broj zadaća (određenih npr. vrijednostima jednog ili više parametara) korištenjem standardnih, često vremenski i računalno zahtjevnih algoritama, te se prikupljeni podaci i rezultati spremaju. U mrežnom dijelu se rješava samo jedna zadaća (određena npr. nekom novom vrijednosti parametara), ali se pritom ne koriste standardni algoritmi, već podaci i rezultati prikupljeni u prvom dijelu. Na taj način se omogućuje brzo i efikasno rješavanje zadaće u realnom vremenu. *Offline* dio procesa obično zahtijeva velike memorijske i vremenske kapacitete, ali se izvodi samo jednom. Nasuprot tome *online* dio se primjenjuje višekratno, te su za njegovo izvođenje dostatni manji računalni uređaji, poput PC-a, pametnog telefona i sl.

Rješavanje diferencijalnih jednadžbi na način opisan u ovom radu bi trebalo biti moguće za sve zadaće koje neprekidno ovise o ulaznim parametrima. Naime, rješenje zadaće u određenom trenutku se u tom slučaju može razmatrati kao neprekidna funkcija ulaznih parametara. Na osnovu čuvenih rezultata univerzalne aproksimacije [1, 2], svaka neprekidna funkcija se može proizvoljno točno aproksimirati plitkom neuronskom mrežom s diferencijabilnom aktivacijskom funkcijom (npr. sigmoidom), što daje teorijsku garanciju primjene opisane metode. Ono što ostaje nedorečeno i što je izazov u razvoju neuronske mreže za pojedinu zadaću, jest broj čvorova u pojedinom sloju nužan za postizanje tražene točnosti. Općenito, u postojećoj literaturi postoji malo rezultata koji daju (gornju) ocjenu potrebnog broja čvorova, npr. poput onog za ReLU mreže ([5], Teorem 1). Stoga se u izgradnji mreže može krenuti od nekog zadanog standardnog broja neurona (u MATLAB-u je to deset), a zatim se on po potrebi modificira ukoliko nismo zadovoljni s postupkom učenja i preciznošću dobivene mreže.

Izrada modela neuronske mreže u MATLAB-u moguća je na dva načina. Osim putem grafičkog sučelja, opisanog u ovom radu, u MATLAB-u je moguće stvoriti neuronsku mrežu pomoću komandnog prozora. Konačan rezultat je u osnovi isti, a izbor između dva pristupa je obično uvjetovan osobnim preferencijama. Mi smo se odlučili za izradu putem

grafičkog sučelja jer nam se taj pristup čini didaktički prihvatljivijim, te primjerenijim širem krugu čitatelja, od kojih mnogi možda nisu bliski niti s MATLAB-om niti s neuronskim mrežama.

## Literatura

[1] G. Cybenko, *Continuous valued neural networks with two hidden layers are sufficient, Technical Report,* Department of Computer Science, Tufts University, 1988.

[2] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of Control, Signals, and Systems, 2 (1989); 303–314.

[3] M. Inigo et al. *College Mathematics for Everyday Life*, Coconino Community College, 2021.

[4] Y. LeCun, Y. Bengio, G. Hinton, *Deep learning*, Nature 521 (2015) 436–444.

[5] Z. Lu et al. *The expressive power of neural networks: a view from the width*, Proceedings of the 31st International Conference on Neural Information Processing Systems, (2017) 6232–6240.

[6] MathWorks, Introducing Deep Learning with MATLAB, mrežno, dostupno na: `https://ch.mathworks.com/content/dam/mathworks/ebook/gated/80879v00_Deep_Learning_ebook.pdf?s_tid=wtest_ctent_HTML`, 2021.

[7] P. Petersen, F. Voigtlaender, *Optimal approximation of piecewise smooth functions using deep ReLU neural networks*, Neural Networks 108 (2018) 296–330.

[8] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *Learning representations by backpropagating errors*, Nature 323 (1986) 533-–536.

## 5. Dodatak

U dodatku je prikazan kod MATLAB funkcije Neuronska_mreza, u kojoj je spremljena i za primjenu osposobljena neuronska mreža konfigurirana i obučena u postupku opisanom u 3.2 poglavlju ovog rada. Ulazne vrijednosti te funkcije su parovi vrijednosti parametara $k$ i $y_0$, a izlaz približna vrijednost $y(1)$ rješenja zadaće (1) u trenutku $t = 1$, izračunata bez korištenja eksplicitne formule za rješenje, već primjenom obučene neuronske mreže.

```
function [y1] = Neuronska_mreza(x1)

% Auto-generated by MATLAB, 05-Jun-2023 15:54:46.
%
% [y1] = myNeuralNetworkFunction(x1) takes these arguments:
%   x = 2xQ matrix, input #1
% and returns:
%   y = 1xQ matrix, output #1
% where Q is the number of samples.

%#ok<*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====


% Input 1
x1_step1.xoffset = [1;10];
x1_step1.gain = [0.222222222222222;0.0222222222222222];
x1_step1.ymin = -1;

% Layer 1
b1 = [11.763486958830492668;
14.919052943101727138;1.03461837699119652;
9.5711150173215244763;
5.5715438583400596073;-0.19289399848306348906;
-10.748678477296357059; -4.5406363004804921601;
0.59789221910111789438; -3.9571983648094928299;
-3.3402292300044811313; -4.7104245337462993248;
5.515057175151995672; 2.9018888878687780775;
1.0881572754984594198; 0.82950676991392802151;
0.7133389023335013901; -1.8552229115682830063;
-0.76803907706450824655;
1.5152309836699180412;-0.37481505523251268386;
-0.82220524981190556968; -0.18722455854345418369;
0.07487793831954847600 9; -0.22297596140260478315;
2.6734562536243320352; -5.6100661653666170636;
2.8738188063444325415;- 0.69090319724044790828;
6.6924751846291172797; 5.6041856686634456963;
-1.8127237541298399925; -6.5197711879509006039;
-8.2291270634866222622; 0.04105949448765844334;
1.0725890583715878002; -9.1835554000472896519;
14.534399236059908134; 1.558554297861901361;
-9.2925898046813859565];
```

```
IW1_1 = [-11.267879104777492216
-0.9252370369964382224;-14.298338423522654494
-4.7011513422924711136;-1.2891056286154756183
0.20155608993808246665;0.00094944506837050211823
-10.552552332920770439;-7.5798794791753447697
-0.5217851584244179719;1.4991650240129694716
0.52654165683352271987;2.3556430795284057922
-13.010776496874777663;10.757622586940234655
0.83500282165745565166;-0.89768771082372866843
0.46753950624510698386;7.6071132426519998404
3.6242187324290831008;1.8416017202919015006
-4.4456318139024775249;10.769512458023426049
10.630872039975505672;-4.9420359086109408864
9.4031072069574648253;-13.057434296527723916
2.8444462585901262308;-5.7545473010661067903
3.2335491594630134671;6.8323013232440024112
4.2331239261323698742;2.6585618856295076817
2.1879985692978132228;5.7182521269154511501
8.030912946922935447;-2.6063152885231306577
-1.6762786829000067002;-2.071416868815091572
3.6739758380547069549;0.80955092976040154973
-0.73023731214633136322;-2.9715046739390333919
9.0934498716912646188;8.9575797062714190844
2.3173031920679134643;-7.7929241757890137521
-8.6065129253185617131;-2.0550791927696065997
-0.06045097546481151967 7;3.9833955881785119146
2.9823381436833980018;-6.0985752952776550728
-6.0942821237423414615;4.3381594685478681583
-0.53857946504968645662;8.3556264199034426809
0.74178332325976992223;8.5741309110994414766
2.8134981085323773 62;2.041913720817814859
5.2572357515848393561;-6.1616888474433677558
-3.8280194556399083083;-7.1889713125112049141
8.3639192696616131428;-10.11090809797012291
2.3684629144046707161;1.8067870060906248142
0.23251426662257493327;2.0686644348872200716
0.60588122576033076605;-9.248507688160430007
2.2659951259058068729;10.173210378216255734
11.477204528640596592;-0.72141328265300252998
-0.36243517642566386305;-0.84968349310705282296
-9.0471957859607474717];

% Layer 2
```

```
b2 = -1.8598005469321188254;
LW2_1 = [-0.018794315577047658594
-0.00058405859668117336703 -3.3431215002543068771
-0.0011916127373429018926 0.006723547411321932100 5
-1.5627565552428408502 0.0030675755605112241427
0.0025213758260606007437 5.6219878182223084195
0.0020372746784530952 96 0.02590317654759248869 3
-0.0005437475912830129183 9 -0.0029679189890373263751
0.0006123143455971042986 2 0.0007780605252299695575 1
0.011349564366035160001 0.10000191726188858987
0.0010667462538690019192 0.52022129086710489076
-0.017564378525978769502 1.9407080197789716713
-4.6671962017761094133e-05 -0.0025179069156099158876
-0.0008001674165190299442 9 1.251674739540115544
-0.0500462432908111015 67 0.0081175520202798525 51
0.0259701980208492701 21 -0.0034781600921359855788
0.0029836399103110818921 0.03829318368543335737 7
-0.0227427271318104041539 -0.00066823668108602384593
-0.0041787087809657865595 2.9391052993931192638
1.6860506274658906278 -0.0081032278446239256603
0.0037170509017277328 02 1.9317145736889815844
-0.041631774026710470504];

% Output 1
y1_step1.ymin = -1;
y1_step1.gain = 0.00205577748521312;
y1_step1.xoffset = 26.7236309893952;

% ===== SIMULATION ========

% Dimensions
Q = size(x1,2); % samples

% Input 1
xp1 = mapminmax_apply(x1,x1_step1);

% Layer 1
a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*xp1);

% Layer 2
a2 = repmat(b2,1,Q) + LW2_1*a1;

% Output 1
```

```
y1 = mapminmax_reverse(a2,y1_step1);
end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function
function y = mapminmax_apply(x,settings)
y = bsxfun(@minus,x,settings.xoffset);
y = bsxfun(@times,y,settings.gain);
y = bsxfun(@plus,y,settings.ymin);
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n,~)
a = 2 ./ (1 + exp(-2*n)) - 1;
end

% Map Minimum and Maximum Output Reverse-Processing
Function
function x = mapminmax_reverse(y,settings)
x = bsxfun(@minus,y,settings.ymin);
x = bsxfun(@rdivide,x,settings.gain);
x = bsxfun(@plus,x,settings.xoffset);
end
```

Paulina Bakoč
Odjel za elektrotehniku i računarstvo, Sveučilište u Dubrovniku, Ćira Carića 4, 20 000 Dubrovnik, Hrvatska
*E-mail adresa:* paulina.bakoc@stud.unidu.hr

Martin Lazar
Odjel za elektrotehniku i računarstvo, Sveučilište u Dubrovniku, Ćira Carića 4, 20 000 Dubrovnik, Hrvatska
*E-mail adresa:* mlazar@unidu.hr