Umjetna inteligencija – Laboratorijska vježba 4

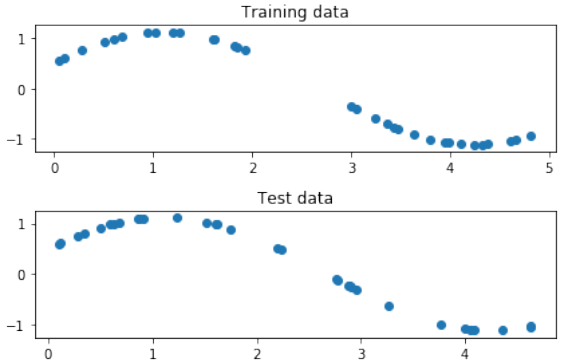
UNIZG FER, ak. god. 2024/25.

Zadano: 5.3.2025. Rok predaje: 12.6.2025. do 23:59 sati.

Neuronska mreža optimizirana genetskim algoritmom (24 boda)

Tema ove laboratorijske vježbe su umjetne neuronske mreže i genetski algoritam. Zadatak vježbe jest napisati program za učenje (treniranje) umjetne neuronske mreže (tj. određivanje njezinih težina) pomoću genetskog algoritma. Neuronska mreža kakva se koristi u ovom zadatku tipično se uči (trenira) algoritmom propagacije pogreške unazad (engl. backpropagation algorithm). Međutim, učenje mreže pomoću genetskog algoritma šire je primjenjivo, a ima i prednosti poput veće širine istraživanja kroz populaciju genetskog algoritma. Osnovna ideja jest oblikovati populaciju neuronskih mreža (svaka jedinka odgovara jednoj inačici neuronske mreže i na sažet način kodira sve njezine težine), a zatim genetskim algoritmom optimirati iznose težina s obzirom na pogrešku mreže na skupu za učenje. Nakon završetka optimizacije, naučena mreža se evaluira na dosad nevidenom skupu za testiranje koji je različit od skupa za učenje.

Vaš zadatak će biti aproksimacija funkcije na temelju zadanog uzorka varijabli i vrijednosti funkcije (također poznato kao regresija). Prva funkcija koju ćemo aproksimirati je sinusoidna krivulja, $f(x) = \sin(\alpha x + \phi) \cdot \beta + \gamma$, na intervalu vrijednosti od 0 do 5. Graf uzoraka sinusoide iz skupa za treniranje te skupa za testiranje možete vidjeti u nastavku:

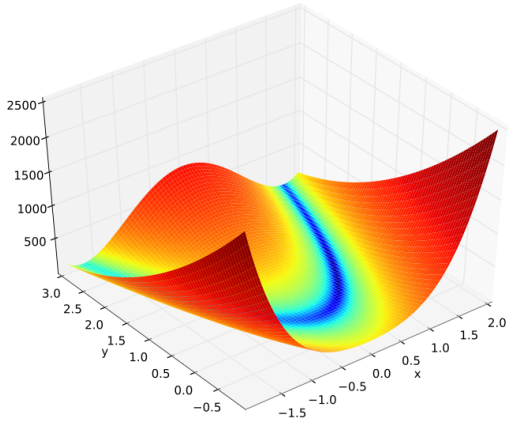


Slika 1: Skup podataka baziran na sinusoidnoj krivulji

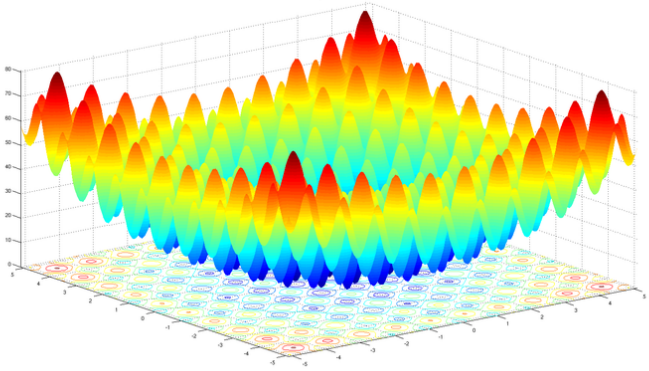
Ostale funkcije koje ćemo aproksimirati su Rosenbrockova funkcija (Sl. 2) te Rastriginova funkcija (Sl. 3).

Naravno, kao ulaz za treniranje nije predvideno cijelo područje definicije prethodno navedenih funkcija, već samo mali podskup vrijednosti dobiven uzorkovanjem.

Iznimno za potrebe ove laboratorijske vježbe, vaš kod **smije koristiti** iduće vanjske biblioteke: **numpy** ukoliko pišete implementaciju u Pythonu, **eigen** ukoliko pišete im-



Slika 2: Rosenbrockova funkcija



Slika 3: Rastriginova funkcija

plementaciju u `c++` i **Apache Commons Math** ukoliko pišete implementaciju u Javi. No, kako se matematičke operacije koje trebate implementirati svode na skalarni produkt i logističku sigmoidu, predlažemo vam da ih probate i implementirati samostalno. Za slučajno uzorkovanje iz normalne razdiobe koristite biblioteke koje su dio standardne biblioteke jezika ili **numpy** u Pythonu. Ako niste sigurni smijete li koristiti neku biblioteku, provjerite je li ona dio standardnog paketa biblioteka za taj jezik.

Provjera koliko dobro vaša implementacija radi provodit će se pomoću *autogradera*. Očekuje se da znate pokrenuti svoje rješenje ispred ispitivača pri predaji laboratorijske vježbe pomoću *autogradera*. "Upute za autograder" pisane su uzevši u obzir da ćemo **mi** pokretati vaše rješenje što ove godine **nećemo** raditi, nego se **vi** morate pobrinuti da se vaše rješenje može pokrenuti i evaluirati *autograderom* ispred ispitivača. Dobit ćete sve testne primjere unaprijed.

Rok za predaju arhive s rješenjem na Moodle **ne uključuje** posljednju minutu u danu, pa se pobrinite da svoju arhivu uploadate **prije** 23:59. Predaja u točno 23:59 ili nakon toga smatrat će se kasnom predajom.

Ukupan broj bodova po podzadacima u ovoj laboratorijskoj vježbi iznosi 24 boda. Broj bodova koji ostvarite prilikom predaje vježbe bit će skaliran na 7.5 bodova.

1. Učitavanje podataka

Kako bi naš algoritam mogli primjeniti na različitim zadacima, prvo ćemo definirati format **datoteke skupa podataka**. Za zapis podataka ćemo ponovno iskoristiti `csv` format iz prethodne laboratorijske vježbe. Datoteke u `csv` formatu sadrže vrijednosti odvojene zarezima. Svaki redak datoteke sadrži jednak broj vrijednosti. U našem slučaju, te vrijednosti su značajke (engl. *features*) za naš algoritam strojnog učenja. Prvi redak datoteke će uvijek biti tzv. `header` koji sadrži naziv značajke koja se nalazi u tom stupcu.

Primjer prva tri retka datoteke skupa podataka za aproksimaciju sinusoidalne funkcije (prvi redak je header):

```
x,y
3.469,-0.795
1.626,0.971
```

Kao što je često i konvencija u strojnom učenju, zadnji stupac će sadržavati ciljnu varijablu. Svi ostali stupci sadržavati će značajke. U okviru ove laboratorijske vježbe ćemo uvijek imati **jednu numeričku ciljnu varijablu**. Sve ulazne značajke će također biti numeričke.

2. Neuronska mreža (12 bodova)

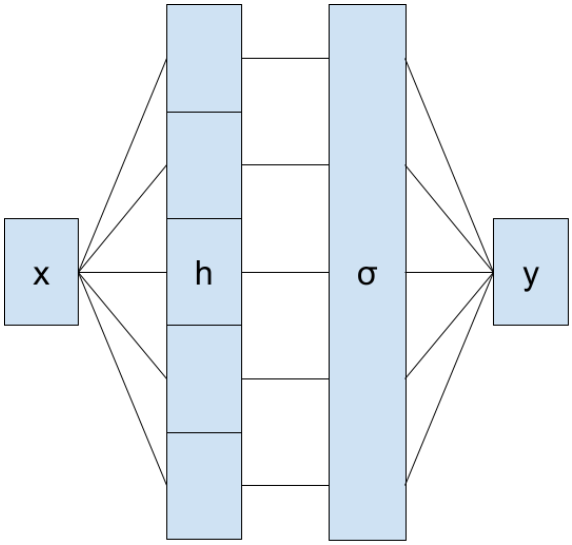
U prvom dijelu laboratorijske vježbe vaš zadatak je implementirati unaprijednu slojevitu neuronsku mrežu i njen unaprijedni prolaz. Algoritam propagacije unatrag ne treba implementirati u okviru vježbe budući da će se težine (parametri) neuronske mreže učiti pomoću genetskog algoritma.

Radi jednostavnosti, za potrebe laboratorijske vježbe nužno je da implementirate samo iduće konfiguracije neuronske mreže:

1. input $\rightarrow$ 5 $\rightarrow \sigma \rightarrow$ output (`5s`)
2. input $\rightarrow$ 20 $\rightarrow \sigma \rightarrow$ output (`20s`)
3. input $\rightarrow$ 5 $\rightarrow \sigma \rightarrow$ 5 $\rightarrow \sigma \rightarrow$ output (`5s5s`)

Pri čemu je *input* dimenzija ulaznih podataka, koju trebate odrediti iz ulazne datoteke (broj stupaca - 1, budući da je zadnji stupac ciljna varijabla), a *output* dimenzija ciljne varijable, koja će uvijek biti 1. Vrijednosti 5 i 20 označavaju dimenzije skrivenog sloja (ili skrivenih slojeva u slučaju (3.)), dok σ označava prijenosnu funkciju logističke sigmoide. Početne vrijednosti svih težina neuronske mreže uzorkujte iz normalne razdiobe sa standardnom devijacijom 0.01. Konfiguracija neuronske mreže koju trebate koristiti biti će predana kao jedan od argumenata putem komandne linije. Jedine konfiguracije koje će se provjeravati autograderom biti će $5s$, $20s$ i $5s5s$.

Mreža (1) se sastoji od dvije linearne transformacije – matrična množenja praćena pribrajanjem vektora pristranosti, te jedne primjene prijenosne funkcije. Ulazni podaci se prvo projiciraju u skriveni sloj dimenzije 5, potom se primjeni prijenosna funkcija, i konačno se pomoću druge linearne transformacije podaci transformiraju u izlaznu vrijednost. Dakle, u posljednjem koraku, nakon primjene posljednje linearne transformacije kako bi se dobila izlazna vrijednost, **ne primijenjuje** se prijenosna funkcija. Vizualiziranu mrežu (1) možete vidjeti na Sl. 4.



Slika 4: Jednoslojna neuronska mreža (5s)

Unaprijedni prolaz vaše neuronske mreže bi za određeni skup ulaznih podataka treba moći izračunati predviđene izlaze

$$\hat{y} = \mathrm{NN}(X), \qquad X \in \underbrace{\{x_1, x_2, \ldots, x_N\}}_{\text{Dataset instances}}$$

pri čemu je N veličina skupa za treniranje ili testiranje.

Osim unaprijednog prolaza, vaša implementacija neuronske mreže treba moći računati i pogrešku – srednje kvadratno odstupanje (engl. *mean squared error*) od točnih vrijednosti Y:

$$\mathrm{err} = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{N}\sum_{i}^{N}(y_i - NN(x_i))^2$$

Vrijednost srednjeg kvadratnog odstupanja koristiti će se pri izračunu dobrote jedinki u genetskom algoritmu.

Iako za potrebe vježbe morate implementirati samo zadane konfiguracije neuronskih mreža, sugeriramo vam da vašu implementaciju oblikujete na način gdje se broj slojeva, kao i njihove skrivene dimenzije mogu proizvoljno definirati kako bi i samostalno mogli istražiti utjecaj konfiguracije mreže na rezultate. Pri implementaciji, preporučamo da dizajnirate vašu neuronsku mrežu na način da možete jednostavno dohvatiti sve njene težine (parametre) i postaviti ih, budući da ćete ih često mijenjati genetskim algoritmom.

3. Generacijski genetski algoritam (12 bodova)

Jednom kad smo implementirali neuronsku mrežu, idući zadatak je implementacija generacijskog genetskog algoritma. U našem slučaju, kromosomi genetskog algoritma sadrže realne vrijednosti – težine neuronske mreže. Dobrotu svake jedinke (neuronske mreže) proizvoljno definirajte kao funkciju srednjeg kvadratnog odstupanja te mreže na skupu podataka za treniranje. Obratite pozornost da dobrota jedinke treba biti veća što je jedinka bolja, dok je obrnuto točno za srednje kvadratno odstupanje.

Komponente vašeg genetskog algoritma trebaju biti implementirane kao u nastavku:

- **Elitizam**: najbolja (ili više najboljih) jedinki se prenosi u iduću generaciju;
- **Križanje**: operator križanja implementirajte kao aritmetičku sredinu;
- **Mutacija**: operator mutacije implementirajte kao Gaussov šum, tako da kromosomu težina pribrojite vektor uzorkovan iz normalne razdiobe sa standardnom devijacijom K. Svaku težinu kromosoma mutirajte s vjerojatnošću p;
- **Selekcija**: za operator selekcije koristite selekciju proporcionalnu dobroti (engl. *fitness proportional selection*).

Genetski algoritami imaju velik broj hiperparametara, koji će biti predani putem komandne linije. Konkretno, hiperparametri genetskog algoritma koji će nas interesirati u sklopu laboratorijske vježbe su: (1) **veličina populacije** (*popsize*) genetskog algoritma, (2) **elitizam** (*elitism*) – broj najboljih jedinki koje će trebati očuvati iz prethodne iteracije, (3) **vjerojatnost mutacije** (p) za svaki element kromosoma genetskog algoritma, (4) **skala mutacije** (K), standardna devijacija Gaussovog šuma i (5) **broj iteracija** (*iter*) genetskog algoritma. Uz putanje do datoteka sa skupovima podataka za treniranje i testiranje te arhitekturu neuronske mreže, ovi argumenti će vam biti definirani putem komandne linije.

Svakih 2000 iteracija (generacija) vaš genetski algoritam treba ispisati srednje kvadratno odstupanje **najbolje** jedinke na skupu podataka za treniranje. Na kraju postupka optimizacije genetskog algoritma (nakon *iter* generacija), ispišite srednje kvadratno odstupanje najbolje jedinke na skupu za testiranje.

Ispis formatirajte kao u nastavku (primjer mreže $5s$ na sinusoidnom skupu podataka):

```
python solution.py --train sine_train.txt --test sine_test.txt --nn 5s
   --popsize 10 --elitism 1 --p 0.1 --K 0.1 --iter 10000
[Train error @2000]: 0.002901
[Train error @4000]: 0.002151
[Train error @6000]: 0.001792
[Train error @8000]: 0.001636
[Train error @10000]: 0.001443
[Test error]: 0.000922
```

Upute za autograder

U nastavku slijede upute o strukturi arhive za upload, a detaljne upute kako pokrenuti *autograder* koristeći zadanu strukturu nalaze se u `autograder.zip` arhivi u datoteci `README.md`.

Struktura uploadane arhive

Arhiva koju uploadate na Moodle **mora** biti naziva `JMBAG.zip`, dok struktura raspakirane arhive **mora** izgledati kao u nastavku (primjer u nastavku je za Python, a primjeri za ostale jezike slijede u zasebnim poglavljima):

```
|JMBAG.zip
|-- lab4py
|----solution.py [!]
```

```
|----neuralnet.py (optional)
|----...
```

Arhive koje nisu predane u navedenom formatu se **neće priznavati**. Vaš kod se mora moći pokrenuti tako da prima iduće argumente putem komandne linije:

1. Putanja do datoteke skupa podataka za treniranje (`--train`)
2. Putanja do datoteke skupa podataka za testiranje (`--test`)
3. Arhitektura neuronske mreže (`--nn`)
4. Veličina populacije genetskog algoritma (`--popsize`)
5. Elitizam genetskog algoritma (`--elitism`)
6. Vjerojatnost mutacije svakog elementa kromosoma genetskog algoritma (`--p`)
7. Standardna devijacija Gaussovog šuma mutacije (`--K`)
8. Broj iteracija genetskog algoritma (`--iter`)

Svi argumenti će se pojavljivati pri svakom pokretanju vašeg rješenja. Zbog stohastičnosti izvođenja genetskog algoritma, vaš ispis se neće provjeravati egzaktno. Autograderom će se provjeravati samo uspjeva li vaše rješenje postići dovoljno nisku grešku kroz opetovana pokretanja, uz visoku razinu tolerancije.

Vaš kod će se pokretati na linux-u. Ovo nema poseban utjecaj na vašu konkretnu implementaciju osim ako ne hardkodirate putanje do datoteka (što **ne bi smjeli**). Vaš kod **smije koristiti** isključivo navedene dodatne vanjske biblioteke. Koristite encoding `UTF-8` za vaše datoteke s izvornim kodom.

Primjer pokretanja vašeg koda pomoću autogradera u Pythonu:

```
>>> python solution.py --train sine_train.txt --test sine_test.txt --nn
5s --popsize 10 --elitism 1 --p 0.1 --K 0.1 --iter 10000
```

Upute: Python

Ulazna točka vašeg koda **mora** biti u datoteci `solution.py`. Kod možete proizvoljno strukturirati po ostalim datotekama u direktoriju, ili sav kod ostaviti u `solution.py`. Vaš kod će se uvijek pokretati iz direktorija vježbe (`lab4py`).

Struktura direktorija i primjer naredbe mogu se vidjeti na kraju prethodnog poglavlja. Verzija Pythona na kojoj će se vaš kod pokretati biti će `Python 3.7.4`.

Upute: Java

Uz objavu laboratorijske vježbe objaviti ćemo i predložak Java projekta koji možete importati u vaš IDE. Struktura unutar arhive `JMBAG.zip` definirana je u predlošku i izgleda ovako:

```
|JMBAG.zip
|--lab4java
|----src
|------main.java.ui
|--------Solution.java [!]
```

```
|--------NeuralNet.java (optional)
|--------...
|----target
|----pom.xml
```

Ulazna točka vašeg koda **mora** biti u datoteci `Solution.java`. Kod možete proizvoljno strukturirati po ostalim datotekama unutar direktorija, ili sav kod ostaviti u `Solution.java`. Vaš kod će se kompajlirati pomoću Mavena.

Primjer pokretanja vašeg koda pomoću autogradera (iz direktorija `lab4java`):

```
>>> mvn compile
>>> java -cp target/classes ui.Solution --train sine_train.txt --test
    sine_test.txt --nn 5s --popsize 10 --elitism 1 --p 0.1 --K 0.1 --iter
    10000
```

Informacije vezano za verzije Mavena i Jave:

```
>>> mvn -version
Apache Maven 3.6.3
Maven home: /opt/maven
Java version: 15.0.2, vendor: Oracle Corporation, runtime: /opt/jdk-15.0.2
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-139-generic", arch: "amd64", family: "unix"

>>> java -version
openjdk version "15.0.2" 2021-01-19
OpenJDK Runtime Environment (build 15.0.2+7-27)
OpenJDK 64-Bit Server VM (build 15.0.2+7-27, mixed mode, sharing)
```

Bitno: provjerite da se vaša implementacija može kompajlirati sa zadanom `pom.xml` datotekom.

Upute: C++

Struktura unutar arhive `JMBAG.zip` mora izgledati ovako:

```
|JMBAG.zip
|--lab4cpp
|----solution.cpp [!]
|----neuralnet.cpp (optional)
|----neuralnet.h (optional)
|----Makefile (optional)
|----...
```

Ako u arhivi koju predate ne postoji `Makefile`, za kompajliranje vašeg koda iskoristiti će se `Makefile` koji je dostupan uz vježbu. **Ako** predate `Makefile` u arhivi (ne preporučamo, osim ako stvarno ne znate što radite), očekujemo da on funkcionira.

Primjer pokretanja vašeg koda pomoću autogradera (iz direktorija `lab4cpp`):

```
>>> make
>>> ./solution --train sine_train.txt --test sine_test.txt --nn 5s
   --popsize 10 --elitism 1 --p 0.1 --K 0.1 --iter 10000
```

Informacije vezano za gcc:

```
>>> gcc --version
gcc (Ubuntu 9.3.0-11ubuntu0~18.04.1) 9.3.0
```

Primjeri ispisa

Uz primjere ispisa biti će priložena i naredba kojom je kod pokrenut. Naredba pokretanja pretpostavlja da je izvorni kod u Pythonu, no argumenti će biti isti za ostale jezike.

1. Sinusoida

```
>>> python solution.py --train sine_train.txt --test sine_test.txt --nn
    5s --popsize 10 --elitism 1 --p 0.1 --K 0.1 --iter 10000

[Train error @2000]: 0.002106
[Train error @4000]: 0.001565
[Train error @6000]: 0.001097
[Train error @8000]: 0.000891
[Train error @10000]: 0.000830
[Test error]: 0.000433


>>> python solution.py --train sine_train.txt --test sine_test.txt --nn
    20s --popsize 20 --elitism 1 --p 0.7 --K 0.1 --iter 10000

[Train error @2000]: 0.003323
[Train error @4000]: 0.002699
[Train error @6000]: 0.001903
[Train error @8000]: 0.001898
[Train error @10000]: 0.001898
[Test error]: 0.002190
```

2. Rastriginova funkcija

```
>>> python solution.py --train rastrigin_train.txt --test
    rastrigin_test.txt --nn 5s --popsize 10 --elitism 1 --p 0.3 --K 0.5
    --iter 10000

[Train error @2000]: 50.640524
[Train error @4000]: 46.177471
[Train error @6000]: 45.925162
[Train error @8000]: 45.578621
[Train error @10000]: 45.539866
[Test error]: 50.544842


>>> python solution.py --train rastrigin_train.txt --test
```

```
   rastrigin_test.txt --nn 20s --popsize 20 --elitism 1 --p 0.1 --K 0.5
   --iter 10000

[Train error @2000]: 62.903413
[Train error @4000]: 15.013800
[Train error @6000]: 1.687832
[Train error @8000]: 1.230139
[Train error @10000]: 0.936814
[Test error]: 1.242374
```

3. Rosenbrockova funkcija

```
>>> python solution.py --train rosenbrock_train.txt --test
   rosenbrock_test.txt --nn 5s --popsize 10 --elitism 1 --p 0.5 --K 4.
   --iter 10000

[Train error @2000]: 287165.467951
[Train error @4000]: 262715.584204
[Train error @6000]: 247707.420406
[Train error @8000]: 230967.797905
[Train error @10000]: 196869.790757
[Test error]: 221803.901110

>>> python solution.py --train rosenbrock_train.txt --test
   rosenbrock_test.txt --nn 5s5s --popsize 20 --elitism 1 --p 0.5 --K 1.
   --iter 10000

[Train error @2000]: 158104.691349
[Train error @4000]: 79784.402381
[Train error @6000]: 63715.907020
[Train error @8000]: 63446.270092
[Train error @10000]: 62710.538447
[Test error]: 94489.025575

>>> python solution.py --train rosenbrock_train.txt --test
   rosenbrock_test.txt --nn 20s --popsize 20 --elitism 3 --p 0.5 --K 10.
   --iter 10000

[Train error @2000]: 149078.216254
[Train error @4000]: 124099.641035
[Train error @6000]: 107335.449190
[Train error @8000]: 98479.245133
[Train error @10000]: 96613.933070
[Test error]: 125773.981642
```