Umjetna inteligencija – Laboratorijska vježba 4

UNIZG FER, ak. god. 2018/19.

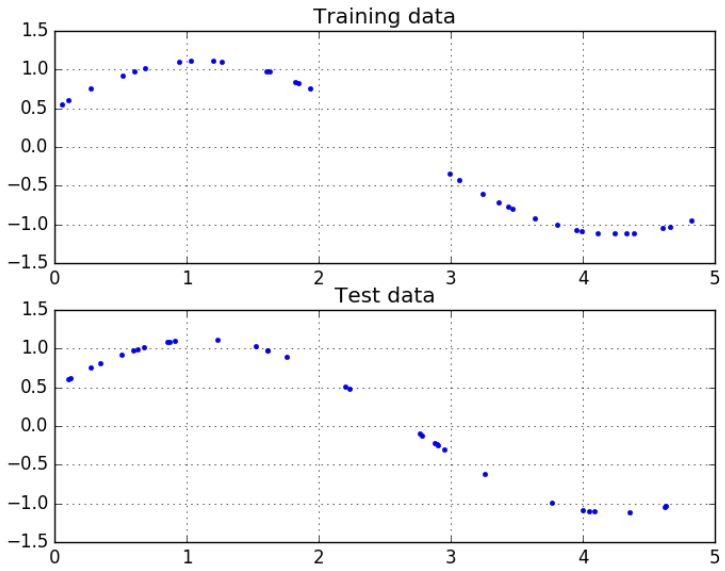
Zadano: 28.5.2019. Rok predaje: 9.6.2019. do 23.59 sati.

1 Uvod

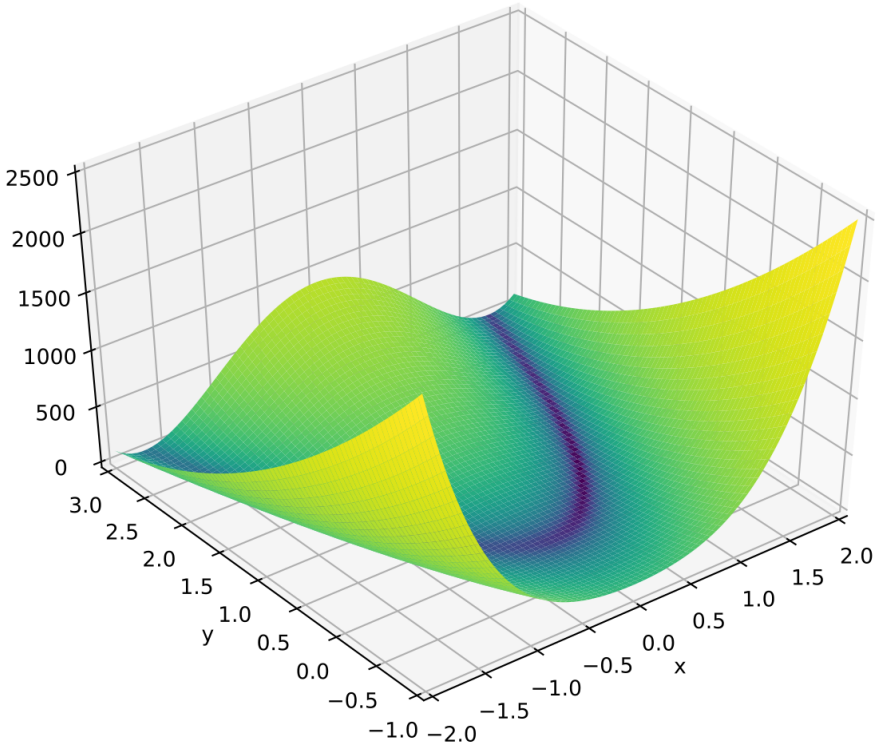
Tema ove laboratorijske vježbe su umjetne neuronske mreže i genetski algoritam. Zadatak vježbe jest napisati program za učenje (treniranje) umjetne neuronske mreže (tj. određivanje njezinih težina) pomoću genetskog algoritma. Neuronska mreža kakva se koristi u ovom zadatku tipično se uči (trenira) algoritmom propagacije pogreške unazad (*engl. backpropagation algorithm*). Medutim, 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 te vrijednosti funkcije (takoder poznato kao regresija). Prva funkcija koja vam je dana za aproksimaciju je sinusoidna krivulja, $f(x) = sin(\alpha x + \phi) \cdot \beta + \gamma$, na intervalu vrijednosti od 0 do 5.

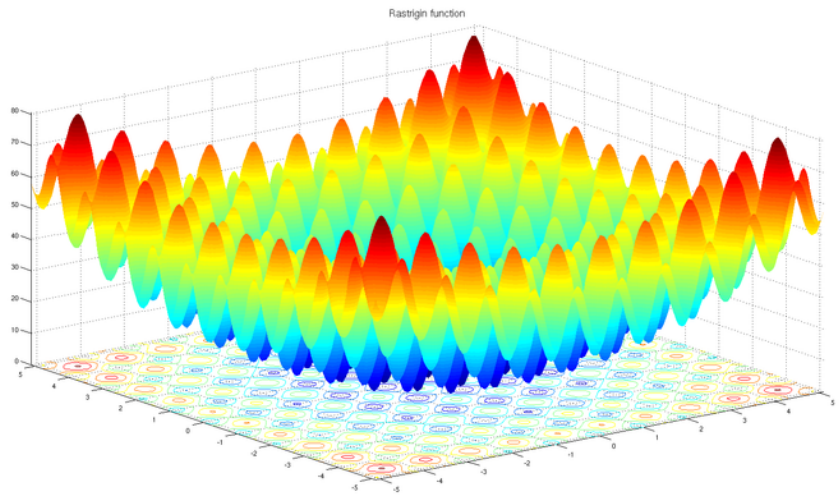
Graf uzoraka sinusoide iz skupa za treniranje te skupa za testiranje možete vidjeti u nastavku:



Ostale funkcije koje se aproksimiraju su Rosenbrockova funkcija, prikazana u nastavku:



te Rastriginova funkcija, takoder prikazana niže:



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

2 Organizacija koda

Kao i u prethodnim vježbama, dostupan je kostur prethodno napisanog koda. U slučaju ove laboratorijske vježbe, taj kod se nalazi u 7 datoteka koje reprezentiraju različite module konačne implementacije rješenja.

Postoje dva pomoćna modula koja možete ignorirati - jedan od njih je dataLoader.py, koji služi za učitavanje podataka iz tekstualnih datoteka u formatu za laboratorijsku vježbu (SSV - space separated values).

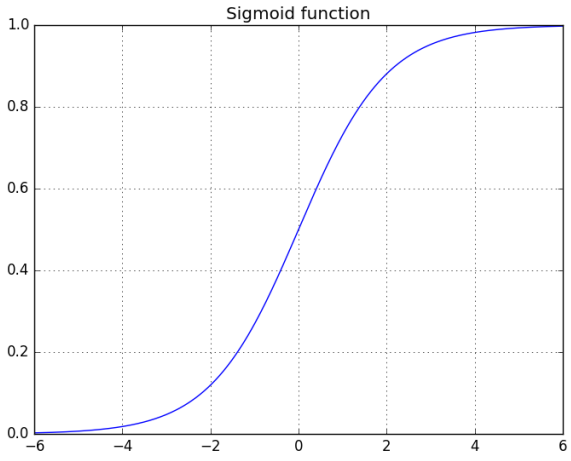
Modul *dataLoader* sadrži samo jednu metodu - *loadFrom(source)*, koja prima putanju do datoteke s podacima u ssv formatu, te vraća učitane podatke podijeljene na varijable i vrijednost funkcije. U slučaju jednodimenzionalne funkcije, poput sinusoide u prvome zadatku - postoji samo jedna varijabla, dok za Rastriginovu i Rosenbrockovu funkciju postoje dvije varijable. Metoda za učitavanje pretpostavlja da je vrijednost funkcije na zadnjoj lokaciji, te da se predvida vrijednost samo jedne funkcije - dimenzionalnost varijabli je nebitna.

Drugi modul koji se može ignorirati je plotter.py, koji koristi biblioteku matplotlib za iscrtavanje funkcija te trenutnih estimacija dobivenih neuronskom mrežom. Ukoliko nemate instaliran paket matplotlib, dostupan je na idućoj poveznici: matplotlib. No - paket je dio većine standardnih distribucija Pythona, te bi ga već trebali imati instaliranog.

Preostali moduli služe za implementaciju neuronske mreže, genetskog algoritma, te glavnu metodu koja pokreće optimizacijski sustav. Ideja je da su implementacije genetskog algoritma i neuronske mreže potpuno odvojene jedna od druge, te da oboje mogu funkcionirati samostalno za potpuno nevezane probleme. Ovo znači da razredi neuronske mreže i genetskog algoritma niti u jednom trenutku ne znaju za potojanje drugog.

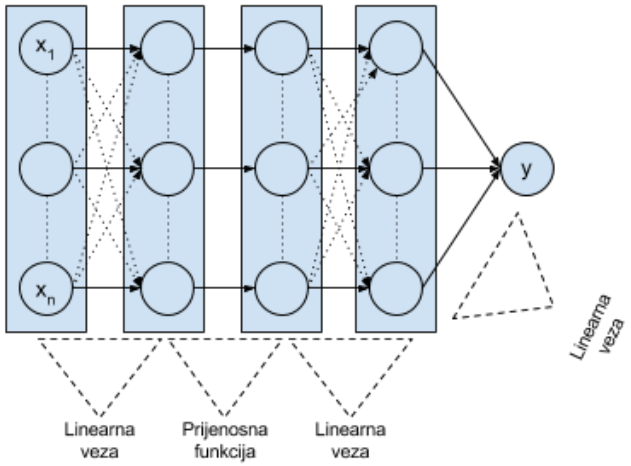
Implementacija neuronske mrže se dijeli kroz tri različite datoteke. U transferFunctions.py, su definirane prijenosne funkcije tipične u neuronskim mrežama. U okviru ove laboratorijske vježbe nije nužno eksperimentiranje s uspješnošću različitih prijenosnih funkcija, iako vas potičemo na to.

Prijenosne funkcije su održavaju dimenzionalnost ulaza ($|\vec{x}| = |f(\vec{x})|$), no izmjenjuju vrijednosti elemenata vektora na nelinearan način. Primjerice, na predavanjima smo obradili sigmoidalnu prijenosnu funkciju: $sigm(x) = \frac{1}{1+e^{-x}}$, čiji je graf:



Potpuno povezana neuronska mreža može se gledati kao niz slojeva, gdje je svaki od

slojeva niz neurona. Primjer ovog možemo vidjeti na idućoj konceptualnoj skici:



Pri čemu su slojevi uzastopno povezani linearnom vezom (sumom umnožaka ulaza i pripadnih težina), te prijenosnom vezom. Broj ovih slojeva može biti proizvoljan, a tipovi nisu ograničeni na linearne i sigmoidalne. U datoteci networkLayers.py zadan je apstraktni razred *NetworkLayer* koji sadrži funkcije koje trebaju implementirati slojevi neuronskih mreža. Kako se u okviru ove laboratorijske vježbe ne služimo algoritmom propagacije unazad (*engl. backpropagation*), zanima nas samo prlolaz unaprijed - koji treba biti definiran u metodi *output*.

Ostatak metoda služi za pojednostavljenje treniranja pomoću genetskog algoritma - neuronska mreža se sastoji od niza težina, te je naš zadatak zapravo pronaći optimalan niz težina koji aproksimira proizvljno zadanu ciljnu funkciju. Implementacija ovoga će se svesti na to da će genetski algoritam optimizirati jedinku -jednodimenzionalni vektor težina koji sadrži sve težine potrebne za neku neuronsku mrežu, te potom evaluirati grešku dobivenu prolazom unaprijed kroz neuronsku mrežu.

Kako bi evaluirali svaku jedinku, potrebno nam je našoj neuronskoj mreži postavljati proizvoljne težine na njene slojeve kako bismo mogli izračunati grešku. Za ovo će nam služiti metoda *setWeights* - koja sloju pridruži njegov vektor težina. Za provjeru veličine svakog sloja služi nam metoda size.

U datoteci neuralNet.py nalazi se kostur neuronske mreže koji trebate dovršiti kako bi neuronska mreža bila potpuno funkcionalna. U datoteci geneticAlgorithm.py nalazi se kostur generacijskog genetskog algoritma koji trebate dovršiti kako biste optimirali težine neuronske mreže u ovisnosti o ciljnoj funkciji pomoću genetskog algoritma. U datoteci runner.py nalazi se glavna funkcija koja pokreće cijeli sustav te gdje možete modificirati hiperparametre genetskog algoritma te definirati arhitekturu vaše neuronske mreže.

Detaljni opis vaših zadataka nalazi se u idućim poglavljima. Osim biblioteke matplotlib, koristi se i biblioteka numpy - iznimno jaka biblioteka za matrično računanje te operacije, randomizaciju te linearnu algebru medu ostalim. Kratki uvod u biblioteku numpy s obješnjenjima dovoljnog dijela funkcionalnosti za rješavanje laboratorijske vježbe dan je u repozitoriju predmeta, kao i na github stranici ovdje. Bitno je držati na umu

da su pretpostavljene vrijednosti svih vektora vrijednosti u vježbi u formatu numpy-evog multidimenzijskog niza.

3 Zadaci

Problem 1: Neuronska Mreža (20% bodova)

Prvi podzadatak laboratorijske vježbe je dovršiti implementaciju neuronske mreže u datoteci neuralNet.py. Metode koje trebate nedopuniti implementiraju osnovnu logiju propagacije unaprijed neuronske mreže te računanja greške u ovisnosti u stvarnim vrijednostima ciljne funkcije.

Podsjetimo se, skup za učenje je podijeljen na niz ulaznih vrijednosti X, te na niz vrijednosti koje predviđamo $y = f(X)$. Prolaz unaprijed je niz operacija kojima ulaze X propuštamo kroz slojeve neuronske mreže kako bismo dobili đpredvideni izlaz $y^{predicted}$. Kod računanja greške uspoređujemo vektor predviđenih vrijednosti s vektorom stvarnih vrijed nosti dobivenih preko skupa za treniranje kako bismo vidjeli u kolikoj mjeri smo pogriješili. U sklopu laboratorijske vježbe, greška koju trebate implementirati je MSE (Mean Square Error, prosječno kvadratno odstupanje). MSE se računa na idući način:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i^{predicted} - y_i^{true})^2$$

Pri čemu je $y^{predicted}$ prethodno navedeni vektor vrijednosti dobiven propagacijom unaprijed kroz slojeve neuronske mreže, dok je vektor y^{true} vektor stvarnih vrijednosti predviđane ciljne funkcije iz skupa za treniranje.

Metoda *forwardStep* objedinjuje propagaciju unaprijed te računanje greške i za zadani vektor ulaznih varijabli X te vektor stvarnih ciljnih vrijednosti Y vraća ukupnu pogrešku (MSE) na skupu podataka.

Problem 2: Genetski algoritam (40% bodova)

Drugi podzdatak laboratorijske vježbe je implementacija algoritma generacijskog genetskog algoritma. U datoteci geneticAlgorithm.py je implementiran konstruktor objekta *GeneticAlgorithm* - no niz kostura metoda čeka vašu implementaciju.

Učenje neuronske mreže treba ostvariti pomoću generacijskog genetskog algoritma s ugradenim elitizmom, koji prenosi najbolju (ili više naboljih) jedinku u sljedeću generaciju. Kao prikaz rješenja koristite numpy polje decimalnih brojeva – svaka jedinka treba biti predstavljena numpy-evim nizom koje sadrži onoliko elemenata koliko neuronska mreža ima težina.

- **Operator križanja** izvedite kao izračun aritmetičke sredine izmedu dviju jedinka.
- **Operator mutacije** izvedite dodavanjem broja nasumično odabranog iz normalne distribucije $N(0, K)$, gdje je K standardna devijacija normalne distribucije, zadana hiperparametrom *mutationScale*. Pri tome mutirajte samo p posto težina odjednom (pri čemu je p proizvoljno odabrana mutacijska konstanta).
- Kao **operator selekcije** koristite selekciju proporcionalnu dobroti (fitness proportional selection).

Problem 3: Optimizacija funkcija i istraživanje hiperparametara (40% bodova)

Nakon uspješne implementacije genetskog algoritma te neuronske mreže, u datoteci runner.py trebate dovršiti isječak koda s definicijom slojeva vaše neuronske mreže. Za početak definirajte neuronsku mrežu koja ima samo jedan skriveni sloj s n neurona (arhitektura $1 \times n \times 1$), pri čemu su vam veličina ulaznog i izlaznog sloja preddefinirane.

U datoteci imate definirane puteve do raznih tekstualnih datoteka sa skupovima za treniranje i testiranje različitih funkcija. Za početak, vaš genetski algoritam bi trebalo relativno brzo i uspješno rješavati problem sinusoide. No, prije nego što uspijete trebate se poigrati s hiperaprametrima (kao i s brojem neurona u skrivenom sloju). Razmislite o tome kako koji od njih utječe na izvođenje te pratite poklapa li se izvodenje s vašim predviđanjem. **Rješenje vježbe koje ne uspjeva zadovoljavajuće aproksimirati sinusoidu neće moći ostvariti bodove za ovaj problem.** Primjer zadovoljavajuće aproksimacije sinusoide, kao i ostalih funkcija može se vidjeti na kraju dokumenta.

Nakon što ste zadovoljavajuće dobro aproksimirali sinusoidu, okušajte se s Rastriginovom i Rosenbrockovom funkcijom, koje imaju dvodimenzionalni ulaz.

- Kako se optimizacija ponaša kada povećavate broj slojeva neuronske mreže?
- Kako se ponaša kada povećavate širinu slojeva neuronske mreže?
- Koja bi, po vama, bila optimalna arhitektura neuronske mreže? Zašto?
- Kako svaki od hiperparametara genetskog algoritma utječe na postupak optimizacije?

Odgovorite na ova pitanja te argumentirajte vaše odgovore na terminu predaje.

Pri definiciji vaše mreže, držite na umu da svaki idući linearni sloj neuronske mreže mora imati jednak broj ulaza broju izlaza prethodnog sloja! Pratite najniže greške koje dobivate na svakoj od zadanih ulaznih funkcija te hiperparametre za koje ste ih dobili kako biste rezultate mogli reproducirati na terminu predaje.

Prilikom treniranja algoritama, povremeno će vam se pokazivati slike na kojima ćete vidjeti vizualizaciju vaše trenutne aproksimacije funkcije i vrijednosti uzoraka iz skupa za treniranje (ili testiranje). Učestalost iscrtavanja ovih slika te ispisa greške možete kontrolirati varijablama *print_every* i *plot_every*. Za nastavak izvodenja programa morate zagasiti skice. Stvarne vrijednosti funkcija označene su zelenom bojom, dok su predvidanja označena plavom bojom ili plavom krivuljom (u 2d slučaju), ili plohom - kojoj boja ovisi o trenutnoj vrijednosti (u 3d slučaju). 3d slike, kad se jednom iscrtaju na vašem računalu se mogu rotirati.

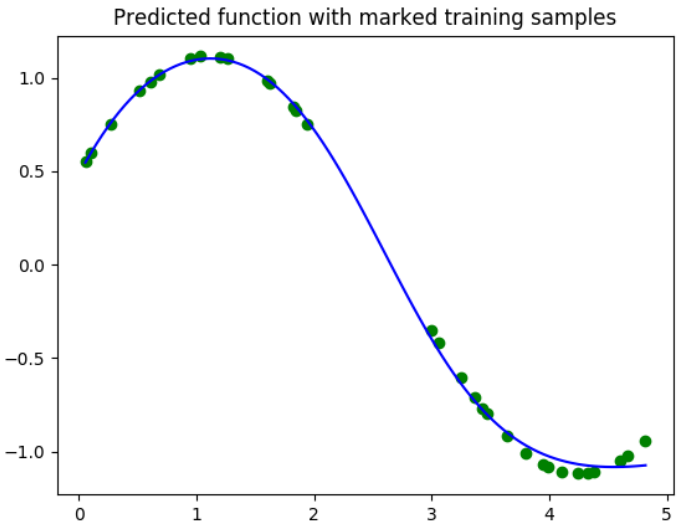
4 Primjeri aproksimacije

Nakon modifikacije hiperparametara, ispis postupka optimizacije te skica aproksimiranih funkcija (i njihovih primjera za treniranje) mogu se vidjeti u nastavku:

Sinusoida

```
Error at iteration 1000 = 0.003803
Error at iteration 2000 = 0.002901
```

```
Error at iteration 4000 = 0.002151
Error at iteration 6000 = 0.001792
Error at iteration 8000 = 0.001636
Error at iteration 10000 = 0.001443
Error on test set:  0.00092272888201
```



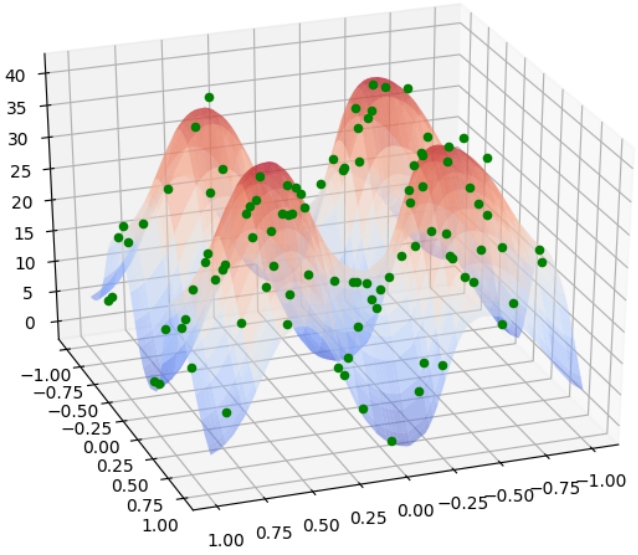
Slika 1: Sinusoida

Rastriginova funkcija

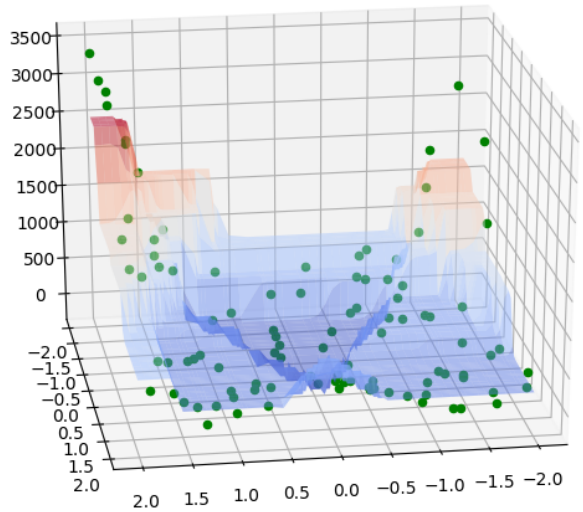
```
Error at iteration 1000 = 56.064648
Error at iteration 2000 = 20.680325
Error at iteration 4000 = 2.596907
Error at iteration 6000 = 1.697376
Error at iteration 8000 = 1.581420
Error at iteration 10000 = 1.457434
Error on test set:  2.06605975288
```

Rosenbrockova funkcija

```
Error at iteration 1000 = 146096.470522
Error at iteration 2000 = 132394.253083
Error at iteration 4000 = 80059.521281
Error at iteration 6000 = 73484.380331
Error at iteration 8000 = 67925.648229
Error at iteration 10000 = 59211.889461
Error on test set:  90720.1330711
```



Slika 2: Rastriginova funkcija



Slika 3: Rosenbrockova funkcija