8. domaća zadaća – algoritam diferencijske evolucije

Uvod

U ovoj zadaći razmotrit ćemo još jednu primjenu umjetnih neuronskih mreža: predviđanje vremenskih serija. Vremensku seriju čini niz podataka koji su prikupljeni tijekom određenog vremena. Primjerice, zamislite da ste na izlaz nekog analognog elektroničkog sklopa spojili digitalni voltmetar kojim upravlja računalo i da ste na računalu napisali program koji svakih 1 milisekundu očita mjerenu vrijednost i zapiše je u datoteku. Nakon 10 minuta u datoteci ćete imati prikupljenih 600 000 uzoraka. Uz pretpostavku da ste uzorke pohranjivali slijedno, uzorke možemo zapisati kao: $u(t_0)$, $u(t_0+1)$, $u(t_0+2)$, $u(t_0+3)$, $u(t_0+4)$, ..., , $u(t_0+599999)$.

Razliku između kolekcije od 600 000 uzoraka u klasifikacijskom zadatku i vremenske serije od 600 000 elemenata čini jedna važna pretpostavka: uzorci u klasifikacijskom zadatku međusobno su nepovezani uzročno-posljedičnom vezom; s druge strane, elementi vremenske serije su rezultat opažanja nekog (skrivenog) procesa koji međutim ima određene zakonitosti pa je stoga za očekivati da uzorak u trenutku t_x ovisi o prethodno opaženim uzorcima u trenutcima t_x-1,t_x-2,t_x-3, ...

Primjerice, možete imati kolekciju od 600 000 sličica dviju vrsta leptira i to 300 000 sličica jedne vrste leptira i 300 000 sličica druge vrste leptira. Takav skup nema vremensku dimenziju: pokušate li ga promatrati kao poredan skup sličica, svaka permutacija poretka predstavlja isti skup i isti klasifikacijski zadatak. S druge strane, pretpostavimo sada da je voltmetar iz prethodnog primjera spojen na izlaz elektroničkog sklopa koji predstavlja generator uzlaznog pilastog napona gdje napon raste od 0V do 1V u vremenu od 5 ms. Uzorkovanjem svake milisekunde dobit ćemo vremensku seriju:
0V, 0.2V, 0.4V, 0.6V, 0.8V, 1V, 0.2V, 0.4V, 0.6V, 0.8V, 1V, 0.2V, 0.4V, 0.6V, 0.8V, 1V, ...
Ponaša li se sklop pak kao generator silaznog pilastog napona, dobili bismo sljedeću vremensku seriju:
1V, 0.8V, 0.6V, 0.4V, 0.2V, 1V, 0.8V, 0.6V, 0.4V, 0.2V, 1V, 0.8V, 0.6V, 0.4V, 0.2V, ...

Uočimo da su izmjereni uzorci isti: svi u elementi iz skupa {0V, 0.2V, 0.4V, 0.6V, 0.8V, 1V}; međutim, ne smijemo izmiješati njihov poredak jer je upravo poredak ono na temelju čega se nadamo da ćemo uspjeti modelirati unutrašnje ponašanje procesa koji na svojem izlazu generira signal koji smo uzorkovali.

Kada neuronske mreže koristimo s vremenskim serijama, najčešće ih koristimo kako bi na temelju danih podataka "naučile" modelirati skriveni proces i na taj nam način omogućile da nastavimo vremensku seriju nakon posljednjeg izmjerenog podatka: kažemo da takvu mrežu koristimo za predviđanje (predikciju).

Danas vremenske serije možemo generirati iz mnoštva izvora: evo samo nekoliko primjera.

- Gustoća prometa na autocesti mjerena svakih sat vremena.
- Prosječna mjesečna gustoća prometa na gradskim prometnicama.
- Prosječno dnevno stanje burzovnih indeksa kroz posljednjih 5 godina.
- Prosječna dnevna temperatura za svaki od 12 mjeseci kroz posljednjih 300 godina.
- Brojevi izvučeni na lotu 7/39 za sva kola u proteklih 10 godina.

Zašto je interesantno modelirati skriveni sustav samo na temelju opaženih podataka – evidentno je samo po sebi. Primjerice, *kada će prosječna gustoća vozila na autocesti doseći maksimalni kapacitet autoceste* (što znači da bi do tada trebalo investirati u njezino proširenje), *kakve nas klimatske promjene očekuju kroz sljedećih 50 ili 100 godina* (što kažu podatci: ima li tog zatopljenja ili nema), *koji će brojevi biti izvučeni u sljedećem kolu na lotu* (komentar nepotreban) i slično samo

su neka od pitanja na koja bi bilo vrlo zgodno znati odgovor.

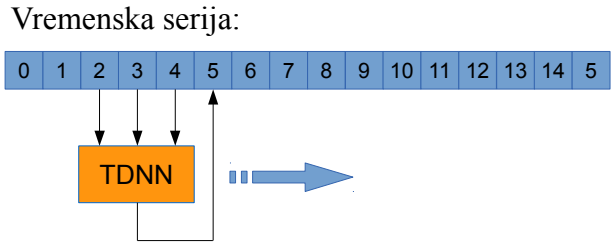
Treba naravno naglasiti da je učenje karakteristika skrivenog procesa samo na temelju podataka već u početku postavljeno na klimavim nogama. Evo relevantnih pitanja iz kojih je vidljivo zašto.

- Imamo li uopće u seriji dovoljno podataka koji su dovoljno reprezentativni da bismo uopće mogli uhvatiti sve karakteristike tog nepoznatog procesa?
- Ima li taj skriveni proces uopće pravilne karakteristike koje bismo mogli naučiti ili je proces kaotičan?
- Postoji li u tom procesu uzročno-posljedična veza koju bismo mogli naučiti ili se uzorci generiraju slučajno?
- Ako se generiraju slučajno, postoji li kakva pravila distribucija koja bi nam ipak omogućila da barem uz malo veću vjerojatnost korektno pogađamo sljedeći element serije ili je ta vjerojatnost naprosto uniformna?

Uz pretpostavku da u procesu postoje pravilnosti koje bismo mogli naučiti, nastavit ćemo daljnje razmatranje s dva tipa umjetnih neuronskih mreža.

Time Delay Neural Network (TDNN)

Pojednostavljeno, TDNN je unaprijedna neuronska mreža koja na ulazu dobiva l slijednih elemenata vremenske serije i koja na izlazu mora generirati korektan (l+1)-vi element serije. Oznakom l ovdje je označena duljina vremenskog prozora koji je dostupan mreži. Ilustracija uz l=3 prikazana je na slici u nastavku.



Primjerice, neka je snimljen sljedeći vremenski niz.
$u(0), u(1), u(2), u(3), u(4), u(5), u(6), u(7)$
Najjednostavnija TDNN je ona za koju je l=1. Takva mreža na ulaz dobiva element vremenske serije u trenutku t i treba pogoditi element vremenske serije u trenutku t+1. Stoga prethodnu vremensku seriju možemo pretvoriti u sljedeći skup uzoraka za učenje oblika (ulazi, izlaz):
(($u(0)$), $u(1)$),
(($u(1)$), $u(2)$),
(($u(2)$), $u(3)$),
(($u(3)$), $u(4)$),
(($u(4)$), $u(5)$),
(($u(5)$), $u(6)$),
(($u(6)$), $u(7)$).
Jednom kada smo na ovaj način konstruirali skup uzoraka za učenje, neuronsku mrežu učimo baš kao i u prošloj domaćoj zadaći. Složeniju TDNN dobit ćemo ako postavimo da je l=2: takva mreža na ulazu dobiva dva prethodna elementa serije i treba pogoditi sljedeći. Uz isti vremenski niz, konstruirani skup uzoraka za učenje bio bi:
(($u(0)$,$u(1)$), $u(2)$),
(($u(1)$,$u(2)$), $u(3)$),
(($u(2)$,$u(3)$), $u(4)$),
(($u(3)$,$u(4)$), $u(5)$),
(($u(4)$,$u(5)$), $u(6)$),

((*u*(5),*u*(6)), *u*(7)).
Uzmemo li TDNN koji ima *l*=3, skup uzoraka za učenje bio bi:
((*u*(0),*u*(1),*u*(2)), *u*(3)),
((*u*(1),*u*(2),*u*(3)), *u*(4)),
((*u*(2),*u*(3),*u*(4)), *u*(5)),
((*u*(3),*u*(4),*u*(5)), *u*(6)),
((*u*(4),*u*(5),*u*(6)), *u*(7)).
Koliki je l potreban da bi neuronska mreža uopće mogla naučiti modelirati skriveni proces ovisi o karakteristikama samog procesa. Međutim, ponekad nam već i sama serija može dati *hint*. Primjerice, ako je serija 0, 0, 1, 1, 0, 0, 1, 1, ... jasno je da s l=1 nećemo daleko dogurati. Naime, takva mreža na temelju jednog uzorka mora pogoditi sljedeći, što kod nas konkretno znači da ako kao ulaz dovedemo 0, mreža bi trebala odrediti koji je sljedeći izlaz: 0 ili 1. No u našoj seriji imamo situacija u kojima nakon 0 dođe 1 i situacija u kojima nakon 0 opet dođe 0. Stoga je očito da l=1 nije dovoljan i takva mreža, ma koliko imala skrivenih slojeva i neurona u njima neće moći dobro naučiti karakteristike procesa koji je generirao ovu seriju. Uzmemo li *l*=2, takva mreža bi morala moći dobro modelirati ovaj proces.

Za ovu neuronsku mrežu ne trebate implementirati ništa novoga: kod koji ste napisali u prošloj domaćoj zadaći može se direktno iskoristiti za ovaj zadatak. Ono što ćete morati napisati nanovo je kod koji generira skup uzoraka za učenje. Pretpostavimo da ste taj kod smjestili u metodu `loadData`. Ona bi morala kao argumente primiti stazu do datoteke s vremenskom serijom (jedan element po retku), parametar *l* te opcionalni parametar *a* koji određuje koliko se elemenata vremenske serije počev od prvog koristi za izgradnju skupa uzoraka za učenje. Primjerice, neka se u datoteci "`serija.txt`" nalazi 1000 elemenata vremenske serije. Poziv:

```
IReadOnlyDataset dataset = loadData("serija.txt", 4, 500);
```

stvorit će skup od 496 uzoraka za učenje:
((e[0], e[1], e[2], e[3]), e[4]),
((e[1], e[2], e[3], e[4]), e[5]),
...
((e[495], e[496], e[497], e[498]), e[499]),

Poziv:

```
IReadOnlyDataset dataset = loadData("serija.txt", 4, -1);
```

stvorio bi skup od 996 uzoraka za učenje (limit od -1 znači da nema ograde).

Treniranje ovakve mreže svodi se na to da se mreži predoče svi uzorci za učenje i da se na uobičajeni način za njih izračuna srednja kvadratna pogreška. Pri tome je potpuno nebitno kojim se redoslijedom mreži predočavaju uzorci jer mreža ne pamti nikakvo stanje. Postupak učenja potom se svodi na optimizacijski problem pronalaska skupa težina uz koji je srednja kvadratna pogreška minimalna.

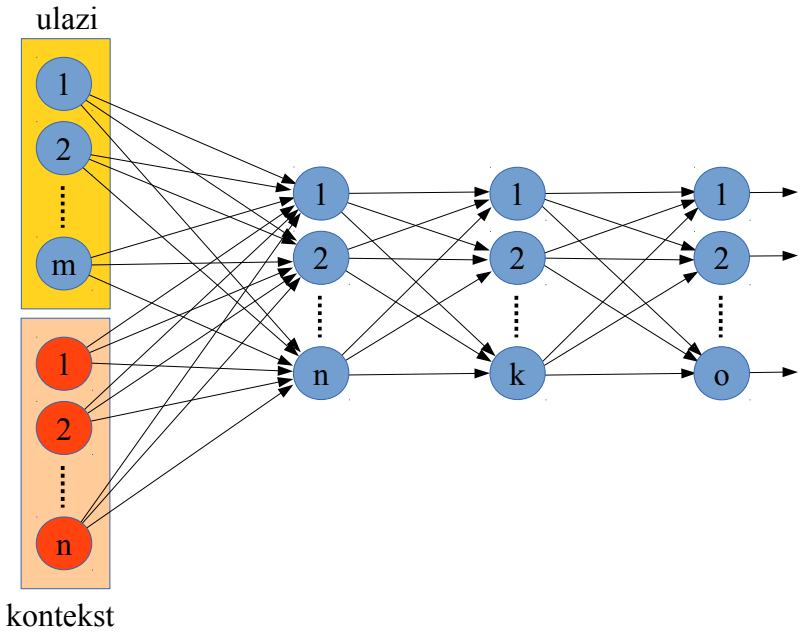
Jednom kada je mreža utrenirana, možemo je iskoristiti za predikciju na sljedeći način. Pretpostavimo da smo mrežu trenirali s 996 uzoraka za učenje uz l=4; to znači da je posljednji uzorak za učenje bio:
(e[995], e[996], e[997], e[998]), e[999]).
Na ulaz mreže stoga ćemo dovesti:
(e[996], e[997], e[998], e[999])
i mrežu ćemo pitati za sljedeći element niza: izlaz mreže označimo s o(1000). To je predikcija mreže za prvi sljedeći element serije. Sada taj element možemo iskoristiti kako bismo stvorili novi ulaz:

(e[997], e[998], e[999], o[1000])
za koji opet pitamo mrežu što će biti sljedeći element: označimo ga s o(1001). Potom stvorimo novi ulaz:
(e[998], e[999], o[1000], o[1001])
za koji opet pitamo mrežu što će biti sljedeći element: označimo ga s o(1002), i postupak možemo nastaviti dalje po potrebi.

Elmanova neuronska mreža

Elmanova neuronska mreža prvi je korak prema općenitim rekurzivnim neuronskim mrežama: neuronskim mrežama kod kojih izlaz ne ovisi samo o trenutno predočenom ulaznom uzorku već i o stanju u kojem se mreža nalazi. Kod Elmanove neuronske mreže to je stanje eksplicitno pohranjeno u tzv. kontekstnim neuronima koji se nalaze u kontekstnom sloju. Ideja je sljedeća. Zamislite klasičnu umjetnu neuronsku mrežu čija je arhitektura 2x5x3x1 (dva ulaza, 5 neurona u 1. skrivenom sloju, 3 neurona u 2. skrivenom sloju te jedan izlazni neuron). S takvom mrežom znamo raditi. Pretpostavite sada da na ulaz mreže dovedete neki podatak i potom izračunate izlaze svih neurona. Promijenite li sada ulaz, mreža će nove izlaze svih neurona računati neovisno o stanju u kojem su bili izlazi njezinih neurona tren prije. Ideja Elmanove neuronske mreže jest arhitekturu mreže promijeniti na način da se kao proširenje ulaznog sloja doda još i kontekstni sloj: u njemu se nalazi upravo onoliko neurona koliko ih postoji i u prvom skrivenom sloju i oni pamte izlaze neurona 1. skrivenog sloja kakvi su bili nakon predočavanja prethodnog ulaznog uzorka. Kontekstni neuroni ništa ne računaju i nemaju ulazne težine. Neuroni prvog skrivenog sloja imaju po jednu težinu do svakog ulaza mreže i do svakog kontekstnog neurona.

Grafički, mrežu možemo prikazati na sljedeći način. Slika ilustrira Elmanovu neuronsku mrežu arhitekture *m* x *n* x *k* x *o*.



Pretpostavimo da su izlazi kontekstnih neurona inicijalno postavljeni na vrijednost 0. Mreži se predočava prvi ulazni uzorak. Mreža izlaze prvog skrivenog sloja računa na temelju danog ulaza i vrijednosti izlaza kontekstnih neurona (i slobodnih težina koje nisu prikazane na slici). Temeljem izlaza neurona prvog skrivenog sloja računaju se izlazi drugog skrivenog sloja i tako dalje sve do izlaza.

Želimo li sada mreži predočiti sljedeći ulazni uzorak, najprije trebamo izlaze prvog skrivenog sloja prekopirati na izlaze kontekstnih neurona (zato njih ima upravo onoliko koliko ima i neurona u prvom skrivenom sloju) i tek tada na ulaz mreže smijemo dovesti sljedeći ulaz. Na ovaj se način

izlazi prvog neurona u prvom skrivenom sloju uvijek računaju temeljem trenutnog ulaza i temeljem prethodnog stanja neurona prvog skrivenog sloja (koje je mreži dostupno kroz kontekstne neurone).

Želimo li na mrežu dovesti sljedeći ulazni uzorak, postupak je isti: naprije trenutne izlaze neurona prvog skrivenog sloja kopiramo u kontekstne neurone i potom dovodimo novi ulaz.

Zahvaljujući činjenici da Elmanova neuronska mreža ima mehanizam za pamćenje stanja, njoj na ulaze nećemo dovoditi *l* slijednih elemenata vremenske serije i pitati je za prvi sljedeći već ćemo na nju doslovno dovoditi element vremenske serije u trenutku *t* i pitati za element vremenske serije u trenutku *t*+1. Drugim riječima, naša Elmanova neuronska mreža imat će jedan ulaz i jedan izlaz. Takva mreža morala bi moći naučiti predviđati i vremenski niz poput 0, 0, 1, 1, 0, 0, 1, 1, ... jer situacije "vidim li na ulazu 0 nakon što sam prethodno vidio 0" ili "vidim li na ulazu 0 nakon što sam prethodno vidio 1" može razlikovati uvidom u stanje zapisano u kontekstnim neuronima. Prilikom postupka učenja očekujemo da će algoritam podesiti težine neurona na način koji će omogućiti zapisivanje upravo ove vrste informacije u kontekstne neurone i njezinu uporabu u generiranju sljedećeg očekivanog izlaza.

Ilustrativan video koji pojašnjava izračun izlaza neuronske mreže postoji i na youtube-u:
http://www.youtube.com/watch?v=e2sGq_vI41s
(možda malo presporo na moj ukus ali ako iz prethodnog objašnjenja nije jasno što i kako računati, ovo može pomoći).

Za ovu neuronsku mrežu trebate napraviti novu implementaciju neuronske mreže. Međutim, ako ste dobro modelirali kod u prošloj zadaći, moći ćete ponovno iskoristiti praktički 95% koda. Morat ćete napisati nanovo i kod koji generira skup uzoraka za učenje. Pretpostavimo da ste taj kod smjestili u metodu `loadData`. Ona bi morala kao argumente primiti stazu do datoteke s vremenskom serijom (jedan element po retku) te opcionalni parametar *a* koji određuje koliko se elemenata vremenske serije počev od prvog koristi za izgradnju skupa uzoraka za učenje. Primjerice, neka se u datoteci "`serija.txt`" nalazi 1000 elemenata vremenske serije. Poziv:

```
IReadOnlyDataset dataset = loadData("serija.txt", 500);
```

stvorit će skup od 499 uzoraka za učenje:
((e[0]), e[1]),
((e[1]), e[2]),
...
((e[498]), e[499]).

Poziv:

```
IReadOnlyDataset dataset = loadData("serija.txt", -1);
```

stvorit će skup od 999 uzoraka za učenje:
((e[0]), e[1]),
((e[1]), e[2]),
...
((e[998]), e[999]).

Dodatni naputci vezani uz mreže

Pri učenju neuronske mreže funkcija kazne s kojom radite i ovdje je srednje kvadratno odstupanje. Za čitav skup podataka izračunajte srednje kvadratno odstupanje prema formuli:

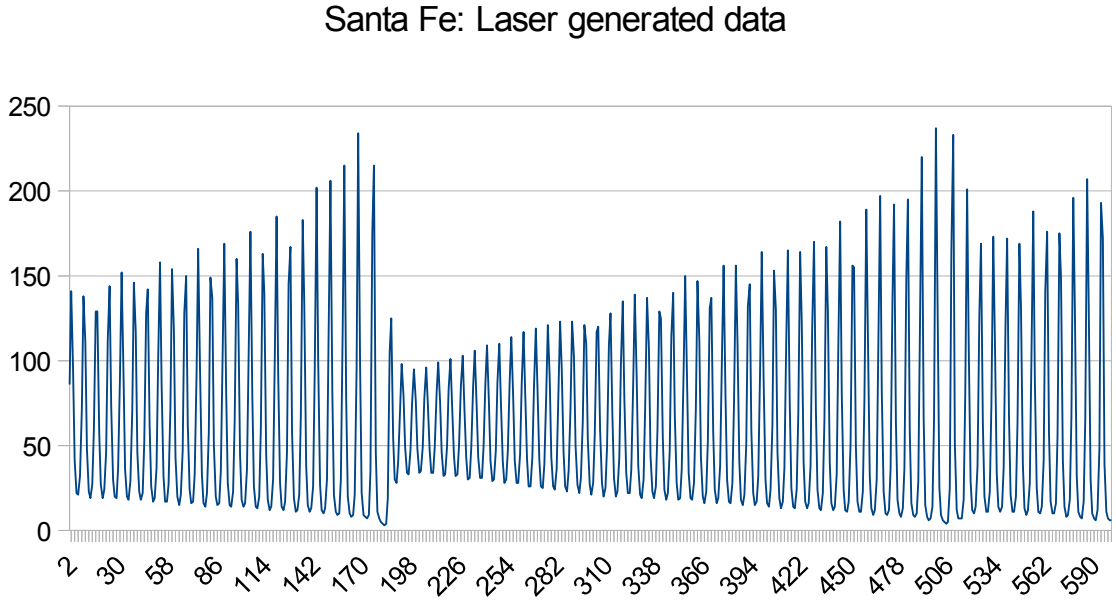
$$Error(dataset, težine) = \frac{1}{N}\sum_{s=1}^{N}\sum_{o=1}^{m}(t_{s,o} - y_{s,o})^2$$

Pri tome je N broj uzoraka za učenje (pa indeks s ide po svim uzorcima) a m broj izlaza pa o ide po svim izlazima (u našem slučaju m će biti 1). Oznakom $t_{s,o}$ označen je podatak koji je u skupu uzoraka za učenje zapisan za o-ti izlaz s-tog uzorka; oznaka $y_{s,o}$ označava o-ti izlaz neuronske mreže koji je mreža izračunala kada joj je na ulaz bio postavljen s-ti uzorak. Kod učenja TDNN redosljed kojim dovodite uzorke nije bitan jer je vremenska komponenta serije "uhvaćena" prilikom pripreme podataka i sama mreža nema stanja. Kod Elmanove neuronske mreže elemente vremenske serije morate dovoditi na mrežu od prvog prema zadnjem.

U okviru ove domaće zadaće prijenosne funkcije u svim slojevima (od prvog skrivenog na dalje) moraju biti tangens hiperbolni, što je zapravo sigmoidalna funkcija skalirana i translatirana tako da ide od -1 do 1 a ne od 0 do 1:

$$y(net) = 2 \cdot sigm(net) - 1 = \frac{1 - e^{-net}}{1 + e^{-net}}$$

Podatci nad kojima ćete trenirati mrežu su *Santa Fe: Laser generated data* (izvorno dostupni na adresi http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html). Kopiju sam stavio na Ferka u repozitorij. Pri tome trebate raditi s prvih 600 elemenata te vremenske serije. Grafički prikaz podataka dan je u nastavku.



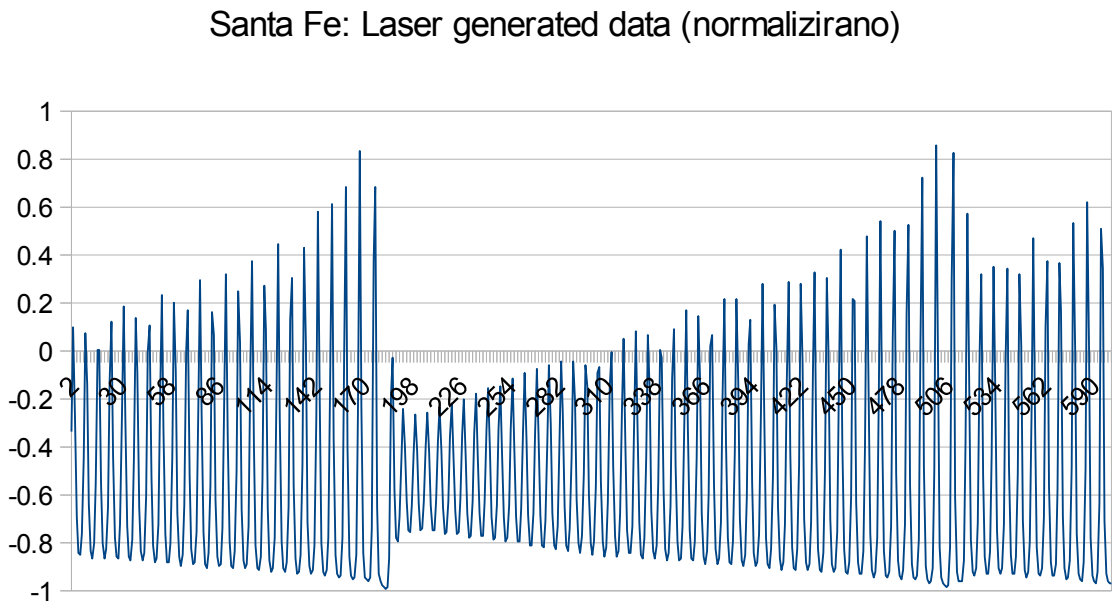
Prilikom uporabe tih podataka za učenje neuronskih mreža najprije napravite njihovu normalizaciju:

1. učitajte sve podatke u memoriju (svih 1000),
2. pronađite minimalni element serije i maksimalni element serije,
3. transformirajte seriju linearno tako da se minimalni element preslika u -1 a maksimalni u 1,
4. od tako dobivene normalizirane vremenske serije uzmite prvih 600 elemenata i na temelju njih izgradite potrebne skupove uzoraka za učenje (ovisno o mreži).

Ako ste dobro napravili normalizaciju, početak serije će biti kao što je prikazano u nastavku.

```
-0.33596837944664026
0.09881422924901195
-0.2648221343873518
-0.691699604743083
-0.841897233201581
-0.849802371541502
-0.7628458498023716
-0.44664031620553357
```

Ovako prilagođena vremenska serija prikazan je na slici u nastavku.



Prilikom učenja TDNN radite s neuronskom mrežom koja na ulaz dobiva 8 prethodnih elemenata vremenske serije (l=8). Imat ćete 592 uzorka za učenje. Za potrebe debugiranja, evo nekoliko podataka za TDNN.

l	Arhitektura	Broj neurona	Broj težina
8	8x10x1	19	101
8	8x5x4x1	18	74
4	4x12x1	17	73
4	4x5x4x1	14	54

Prilikom učenja Elmanove neuronske mreže imat ćete 599 uzoraka za učenje. Za potrebe debugiranja, evo nekoliko podataka za Elmanovu mrežu.

Arhitektura	Broj neurona	Broj težina	Broj parametara
1x10x1	22	131	141
1x5x4x1	16	64	69
1x12x1	26	181	193
1x4x3x5x1	18	65	69

Prilikom učenja TDNN rješenje optimizacijskog problema bit će vektor težina. Prilikom učenja Elmanove mreže rješenje optimizacijskog problema bit će vektor težina i početnih vrijednosti koje treba inicijalno zapisati u kontekstne neurone (nazovimo to sve skupa "parametrima"; dimenzionalnost tog vektora za pojedine arhitekture dana je u prethodnoj tablici). Predlažem da u vektor najprije pohranite težine pa u nastavku početne vrijednosti. Umjesto da prilikom izračuna izlaza na početku kontekstne neurone napunite nulama, u njih zapišite evoluirane početne vrijednosti pa izlaz mreže za prvi uzorak računajte na temelju tog uzorka i evoluiranih početnih vrijednosti.

Zadatak.

Napisati implementaciju algoritma DE koju ćete potom iskoristiti za učenje parametara neuronskih mreža. Algoritam pri tome ne smije ništa znati o konkretnoj neuronskoj mreži koju uči. Ovo ćete postići tako da evaluator modelirate zasebnim sučeljem i potom izvana algoritmu DE date referencu na evaluator koji koristi. Napišite dva evaluatora: jedan koji dobiva referencu na TDNN i skup za učenje i koji zna izračunati dobrotu/kaznu (što vam već treba) te mreže te drugi koji dobiva referencu na Elmanovu neuronsku mrežu i skup za učenje i koji zna izračunati dobrotu/kaznu te mreže.

Zajedničke smjernice

Napišite program `hr.fer.zemris.optjava.dz8.ANNTrainer` koji preko komandne linije prima sljedeće argumente:

- *file*: datoteka koja sadrži izvornu vremensku seriju (svih 1000 uzoraka),
- *net*: oznaka mreže koju je potrebno koristiti,
- *n*: željena veličina populacije,
- *merr*: prihvatljivo srednje kvadratno odstupanje (tj. pogreška),
- *maxiter*: maksimalni broj generacija/epoha.

Kao drugi argument programa potrebno je prihvatiti sljedeće:

- tdnn-*arh*: želimo trenirati TDNN zadane arhitekture; primjeri parametra *arh* su: 8x10x1, 4x10x5x1, itd.; prvi broj ujedno određuje *l* mreže, zadnji mora biti 1.
- elman-*arh*: želimo trenirati Elmanovu neuronsku mrežu zadane arhitekture; primjeri parametra *arh* su 1x10x1, 1x5x4x1 itd.; prvi i zadnji broj moraju biti 1.

Uvjet zaustavljanja optimizacijskog algoritma je prekoračenje maksimalnog broja iteracija ili pak postizanje srednje kvadratne pogreške koja je manja od zadane (npr. 0.02).

Ako pojedini algoritmi imaju dodatnih parametara, njih također izdvojite u konstruktor razreda koji implementira algoritam ali ih postavite na fiksne vrijednosti u glavnom programu.

Pri generiranju početnih rješenja birajte sve težinske faktore uniformno iz intervala [-1, 1].

Isprobajte različite strategije kod DE: predložite neku uz koju postupak učenja radi brzo i efikasno.

Dodatno:

TDNN arhitektura tipa 8x10x1, 8x10x4x1 i slične bi trebao bez problema doći do MSE od 0.01 (pa i bolje). Slično vrijedi i za Elmanovu mrežu.

Napomene:

Rok za predaju Eclipse projekta je nedjelja, 8. prosinca do 23:59.