7. domaća zadaća – algoritam roja čestica i algoritam klonske selekcije

Uvod

U ovoj domaćoj zadaći primijenit ćemo algoritam roja čestica te algoritam klonske selekcije na rješavanje problema učenja klasifikatora ostvarenog umjetnom neuronskom mrežom. Problem s kojim ćemo raditi potiče iz područja botanike: na 150 biljaka napravljen je niz mjerenja četiriju karakterističnih veličina kod tri porodice cvijeta iris – porodice su *iris setosa*, *iris versicolor* i *iris virginica* koje su prikazane na slikama u nastavku. Temeljem tih mjerenja pripremljen je skup podataka za učenje klasifikatorskih sustava.

Iris setosa

Iris versicolor

Iris virginica

Ideja je naučiti klasifikatorski sustav da samo na temelju te četiri izmjerene veličine može ispravno razaznati o kojoj se vrsti cvijeta radi. Više o ovom skupu podataka pročitajte na:
http://en.wikipedia.org/wiki/Iris_flower_data_set

Sam skup podataka javno je dostupan na stranici:
http://archive.ics.uci.edu/ml/datasets/Iris

Verziju koja je pripremljena za klasifikaciju neuronskom mrežom stavio sam u repozitorij u dodatak domaćim zadaćama – skinite ga od tamo i radite s njime.

Što je umjetna neuronska mreža pojasnio sam na kraju prethodnog predavanja: podsjetnik je napisan i u knjizi koja prati ovu vještinu, u okviru zadatka A.7. Vaš je zadatak najprije implementirati prikladan objektni model umjetne unaprijedne slojevite neuronske mreže koja u konstruktoru prima podatke o arhitekturi umjetne neuronske mreže. Kao primjer načina uporabe dajem mogući isječak koda.

```
IReadOnlyDataset dataset = LoadData(args[0]);
System.out.println("Imamo uzoraka za ucenje: "+dataset.numberOfSamples());

FFANN ffann = new FFANN(
              new int[] {4,5,3,3},
              new ITransferFunction[] {
                            new SigmoidTransferFunction(),
                            new SigmoidTransferFunction(),
                            new SigmoidTransferFunction()     //f(x)=1.0/(1.0+e^(-x))
                            //new LinearTransferFunction()    // primjer: f(x)=x
                     },
              dataset);
```

Ovdje je neuronska mreža modelirana razredom FFANN koji u konstruktoru prima tri argumenta.

Prvi argument je polje koje govori koliko mreža ima slojeva te koliko je neurona u svakom sloju. U ovom primjeru, budući da polje ima četiri elementa, stvaramo neuronsku mrežu koja ima četiri

sloja: ulazni sloj, dva skrivena sloja te izlazni sloj. Broj neurona po slojevima dan je elementima polja: ulazni sloj ima 3 neurona, 1. skriveni sloj ima 5 neurona, 2. skriveni sloj ima 3 neurona i izlazni sloj ima 3 neurona. Da smo kao argument predali {4, 6, 3}, imali bismo troslojnu neuronsku mrežu s 4 neurona u ulaznom sloju, 6 neurona u jedinom skrivenom sloju i 3 neurona u izlaznom sloju.

Drugi argument govori koje se prijenosne funkcije korisite u svakom sloju (odnosno kako se izračunati *net* transformira u izlaz neurona) – uzimajući u obzir da ulazni sloj samo preslikava podatke i ubacuje ih u mrežu (dakle ništa ne računa), drugi argument je polje referenci na prijenosne funkcije za slojeve mreže počev od drugog sloja – zato je broj elemenata u njemu za jedan manji od broja elemenata u polju koje se predaje kao prvi argument.

Konačno, treći argument je referenca na skup podataka za učenje. Razred čiji se ovdje predaje objekt trebao bi ponuditi metode poput: koliko ima uzoraka za učenje, koliko uzorak ima ulaza, koliko ima izlaza, dohvat *i*-tog uzorka (moguće čak i parcijalno: posebno ulaznog dijela, posebno izlaznog dijela) i slično.

Razred FFANN trebao bi ponuditi metode poput:

```
public int getWeightsCount();
public void calcOutputs(double[] inputs, double[] weights, double[] outputs);
```

Metoda `getWeightsCount` trebala bi vratiti ukupni broj težina koje postoje u neuronskoj mreži (sama mreža valjda zna taj podatak). Metoda `calcOutputs` trebala bi primiti referencu na polje koje sadrži ulaze jednog uzorka, referencu na polje koje sadrži sve težine mreže pohranjene slijedno te referencu na polje u koje treba upisati izlaze koji su izračunati za predani ulaz.

Imate li ovo pripremljeno na ovakav način, onda niti jedan optimizacijski algoritam ne treba ništa znati o mreži. Naime, dovoljno pitati mrežu koliko ima težina – to se koristi kao broj double-ova koji čine jedno rješenje. Algoritam potom stvori populaciju takvih polja, radi nad njima manipulacije (križanja, mutacije odnosno što već algoritam radi) i kada treba evaluirati, mreži preda referencu na trenutno polje koje sadrži težine: mreža sama zna interpretirati na kojoj lokaciji u polju je pospremljena koja težina. Razmislite kako ćete ovo napraviti.

Klasifikatorski problem koji rješavate prirodno definira funkcije kazne: 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 (primjerice, za skup uzoraka za učenje *iris* to je 3) pa o ide po svim izlazima. 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.

U okviru ove domaće zadaće prijenosne funkcije u svim slojevima (uključivo i izlaznom sloju) moraju biti sigmoidalne:

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

Zadaća optimizacijskog algoritma bit će minimizirati srednju kvadratnu pogrešku (odnosno odstupanje). Jednom kada ste mrežu naučili (odnosno kada se zadovolji uvjet za prekid optimizacijskog algoritma), ispitajte koliko uzoraka mreža klasificira dobro a koliko loše: ispišite tu kumulativnu statistiku te za svaki uzorak iz skupa uzoraka za učenje ispišite kako ga mreža klasificira a koja je njegova stvarna klasifikacija. Kod ovog koraka postupite na sljedeći način: postavite uzorak na ulaz mreže i izračunajte izlaz mreže. Potom pogledajte vrijednosti svih triju

izlaza: oni koji su manji od 0.5 postavite ih na 0 a oni koji veći ili jednaki 0.5, postavite ih na 1. Time ćete izlaz mreže pretvoriti u trobitni binarni uzorak. Primjerice, ako na izlazu imate {0.25, 0.62, 0.18}, dobit ćete 010; za {0.11, 0.53, 0.99} dobit ćete 011. U skupu uzoraka za učenje izlazi su već prikazani na taj način: samo je jedan u jedinici. Uzorak je ispravno klasificiran samo ako se binarni uzorak koji dobijete transformacijom izlaza neuronske mreže poklapa s binarnim uzorkom koji je dan u skupu uzoraka za učenje. Napomena: ovo radite samo na kraju kada ispisujete klasifikacije na zaslon; tijekom učenja srednja kvadratna pogreška računa se s pravim izlazima neuronske mreže.

Zadatak 1.

Napisati implementaciju algoritma PSO koja mrežu uči koristeći vremenski promjenjiv utjecaj inercije te:
a) globalno susjedstvo,
b) lokalno susjedstvo kod kojeg su čestice topološki složene u prsten i kod kojeg čestica ima *d* susjeda s lijeve i desne strane.
U varijanti a) ažuriranje brzine *i*-te čestice radi se prema izrazu:

$$\vec{v}_i \leftarrow w(t)\cdot\vec{v}_i + c_1\cdot U_1\cdot(\vec{p_i^{best}} - \vec{x}_i) + c_2\cdot U_2\cdot(\vec{g^{best}} - \vec{x}_i)$$

dok se u slučaju b) ažuriranje brzine *i*-te čestice radi prema izrazu:

$$\vec{v}_i \leftarrow w(t)\cdot\vec{v}_i + c_1\cdot U_1\cdot(\vec{p_i^{best}} - \vec{x}_i) + c_2\cdot U_2\cdot(\vec{l_i^{best}} - \vec{x}_i) \quad .$$

w(t) je težina kojom se inercija uzima u obzir i ovisno o *t* (trenutnoj iteraciji/epohi) mijenja se linearno od neke početne pa prema konačnoj (pazi, ovdje *w* ne označava težine neuronske mreže koje se uče).

Prije početka implementacije preporuča se pročitati rad MonStuBirDor2009tec.pdf koji je dostupan pod tim imenom u repozitoriju na Ferku.

Zadatak 2.

Napisati implementaciju algoritma ClonAlg koja uči neuronsku mrežu.

Prije početka implementacije preporuča se pročitati rad TR02-2007.pdf koji je dostupan pod tim imenom u repozitoriju na Ferku.

Zajedničke smjernice

Napišite program `hr.fer.zemris.optjava.dz7.ANNTrainer` koji preko komandne linije prima sljedeće argumente:

- *file*: datoteka koja sadrži skup podataka iris,
- *alg*: oznaka algoritma koji je potrebno upogoniti,
- *n*: željena veličina populacije,
- *merr*: prihvatljivo srednje kvadratno odstupanje (tj. pogreška),
- *maxiter*: maksimalni broj generacija/epoha.

Kao drugi argument programa potrebno je prihvatiti sljedeće:

- pso-a: želimo pokrenuti varijantu algoritma pso koja je u zadatku opisana pod a);
- pso-b-*d*: želimo pokrenuti varijantu algoritma pso koja je u zadatku opisana pod b) i koja radi sa susjedstvom "poluširine" d; npr: pso-b-1 će rezultirati susjedstvom od 3 čestice (jedna lijevo, sama čestica i jedna desno), pso-b-2 će rezultirati susjedstvom od 5 čestica, itd.;
- clonalg: želimo pokrenuti clonalg algoritam.

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].

Kao pomoć pri debugiranju mreže evo nekoliko testnih podataka. Pretpostavimo da ste dobro učitali iris-skup. Ako stvorite neuronsku mrežu {4,5,3,3} (kao iz primjera na početku dokumenta) i ako sve težine fiksno postavite na 0.1, srednja kvadratna pogreška trebala bi biti `0.8365366587431725` (u toj mreži broj težina je 55). Ako stvorite neuronsku mrežu {4,3,3} i ako sve težine fiksno postavite na 0.1, srednja kvadratna pogreška trebala bi biti `0.8566740399081082` (u toj mreži broj težina je 27); ako sve težine postavite fiksno na -0.2, srednja kvadratna pogreška trebala bi biti `0.7019685477806382`.

Napomene:

Rok za predaju Eclipse projekta je utorak, 3. prosinca do 23:59.