

# Podešavanje težina neuronske mreže upotrebom optimizacionog algoritma

Seminarski rad u okviru kursa  
Računarska inteligencija  
Matematički fakultet

Nikola Stamenić, Lea Petković  
mi16177@alas.matf.bg.ac.rs, mi16163@alas.matf.bg.ac.rs

16. jun 2020.

## Sažetak

Ovaj rad se bavi podešavanjem težina neuronske mreže uz pomoć algoritma optimizacije rojem čestica. Pored osnovnog teorijskog pristupa datim pojmovima, rad sadrži i praktičnu primenu ovakvog pristupa podešavanja težina. Kako bi stekli utisak koliko su rezultati dobri, autori ovog rada svoje rezultate upoređuju sa rezultatima autora rada *Designing Artificial Neural Networks Using Particle Swarm Optimization Algorithms* [3].

**Glavne reči:** PSO, neuronske mreže, ANN, optimizacija rojem čestica, Keras, Iris, Wine, Breast Cancer.

## Sadržaj

<b>1</b>	<b>Uvod</b>	<b>2</b>
<b>2</b>	<b>Neuronske mreže</b>	<b>2</b>
2.1	Keras Pajton biblioteka . . . . .	2
<b>3</b>	<b>Optimizacija rojem čestica</b>	<b>2</b>
3.1	Originalni algoritam PSO . . . . .	2
3.2	Druga generacija PSO algoritma . . . . .	3
3.3	Novi model PSO-a . . . . .	3
<b>4</b>	<b>Podešavanje težina neuronske mreže pomoću optimizacije rojem čestica</b>	<b>3</b>
4.1	Rezultati . . . . .	4
4.1.1	Skup Iris . . . . .	4
4.1.2	Skup Breast cancer . . . . .	4
4.1.3	Skup Wine . . . . .	5
<b>5</b>	<b>Zaključak</b>	<b>5</b>
	<b>Literatura</b>	<b>6</b>
<b>A</b>	<b>Prikaz rezultata</b>	<b>7</b>
<b>B</b>	<b>Kod</b>	<b>7</b>

# 1 Uvod

Podešavanje težina neuronske mreže (eng. *Artificial Neural Network, ANN*) za cilj ima nalaženje optimalnog skupa težina, kako bi čitava neuronska mreža za određene podatke davala što tačnije izlaze, kako u primeni za klasifikaciju, tako i u primeni za regresiju. Neuronska mreža može biti korišćena na jako bitnim istraživanjima, tako da podešavanje težina može igrati jako bitnu ulogu. Jedan od načina na koje možemo podesiti težine jeste algoritmom optimizacije rojem čestica (eng. *Particle Swarm Optimization*) [4]. PSO je algoritam nastao inspirisan prirodom, odnosno kretanjem jata ptica u potrazi za hranom, pa može predstavljati dobar algoritam za podešavanje težina neuronske mreže. Problem sa samim algoritmom može biti to što postoji mogućnost zaglavljivanja u lokalnom optimumu, što u startu ne garantuje nalaženja najboljeg mogućeg rešenja. Problemom podešavanja težina neuronske mreže, pomoću optimizacije rojem čestica, bavili su se mnogi istraživači [3].

## 2 Neuronske mreže

Neuronska mreža je sistem koji vrši mapiranje između ulaza i izlaza problema. Neuronske mreže zapravo predstavljaju parametrizovanu reprezentaciju koja se može koristiti za aproksimaciju raznih funkcija [3]. Matematičkom optimizacijom nekog od kriterijuma kvaliteta vrši se pronalaženje odgovarajućih parametara.

Neuronske mreže uče informacije kroz proces treniranja u nekoliko iteracija. Kada je proces učenja završen, neuronska mreža je spremna i sposobna da klasifikuje nove informacije, predvidi ponašanje, ili aproksimira nelinearnu funkciju problema. Njena struktura sastoji se od skupa neurona, predstavljenih funkcijama, koji su međusobno povezani sa ostalim neuronima organizovanim u slojeve.

Struktura neuronske mreže se razlikuje po broju slojeva. Prvi sloj jeste ulazni sloj, poslednji sloj jeste izlazni, a svi slojevi između se nazivaju skrivenim slojevima. Slojevi su međusobno potpuno povezani. Slojevi komuniciraju zahvaljujući tome što je izlaz svakog neurona, iz prethodnog sloja, povezan sa ulazima svih neurona iz narednog sloja. Jačina veza kojom su neuroni međusobno povezani se naziva težinski faktor (eng. *weight*). Najčešće ima 3 sloja.

Postoje različite vrste neuronskih mreža. Mozemo ih klasifikovati prema: broju slojeva (jednoslojne i višeslojne), vrsti veza između neurona, smeru prostiranja informacija (neuronske mreže sa propagacijom unapred ili unazad) [1], vrsti podataka itd.

Njihove primene su mnogobrojne, obzirom da predstavljaju najčešće primenjivanu metodu mašinskog učenja. Neke od primena su: kategorizacija teksta, medicinska dijagnostika, prepoznavanje objekata na slikama, autonomna vožnja, igranje igara poput igara na tabli ili video igara, mašinsko prevođenje prirodnih jezika, prepoznavanje rukom pisanih tekstova itd.

### 2.1 Keras Pajton biblioteka

Keras je biblioteka za neuronske mreže, napisana u Pajtonu. Keras radi na platformi za mašinsko učenje *TensorFlow*. Razvijena je sa ciljem da omogući brzo eksperimentisanje sa neuronskim mrežama [2], da bude razumljiva, modularna i proširiva.

Pored, već pomenute, *TensorFlow* platforme, ova biblioteka radi i na: *Microsoft Cognitive Toolkit*, *R*, *Theano*, ili *PlaidML* platformama. Nastala je kao deo istraživanja u okviru projekta ONEIROS (eng. *Open-ended Neuro-Electronic Intelligent Robot Operating System*), a njen autor i održavaoc je gugl inženjer - François Chollet.

## 3 Optimizacija rojem čestica

U ovoj sekciji biće objašnjen sam algoritam optimizacije rojem čestica. Najviše vremena biće posvećeno originalnom algoritmu PSO, drugoj generaciji algoritma PSO (eng. *The Secong Generation of PSO*), kao i novom modelu algoritma PSO (eng. *A New Model of PSO*).

### 3.1 Originalni algoritam PSO

Algoritam PSO je metod za optimizaciju neprekidne nelinearne funkcije, koji je predložio Eberhart. Sam algoritam je inspirisan posmatranjem socijalnog i kolektivnog ponašanja u kretanju jata ptica pri potrazi za hranom ili preživaljavanjem. PSO je nadahnut kretanjem najboljeg člana populacije i njegovog iskustva. Metafora govori da se skup rešenja kreće prostorom pretrage sa ciljem da nađe što bolju poziciju, rešenje [3].

Populacijom se smatra grupa čestica  $i$  gde svaka predstavlja poziciju  $x_i \in R^D$ ,  $i = 1, \dots, M$  u višedimenzionom prostoru. Čestice se evaluiraju u posebnoj funkciji optimizacije, kako bi se odredila njihova prilagođenost i sačuvala najbolja vrednost. Svaka čestica se kreće po prostoru pretrage u zavisnosti od funkcije brzine  $v_i$  koja u obzir uzima globalno najbolju poziciju u populaciji ( $p_g \in R^D$  - socijalna komponenta) kao i najbolju poziciju date čestice ( $p_i \in R^D$  - kognitivna komponenta). Čestice će se kretati u svakoj iteraciji na drugu poziciju, dok ne dostignu optimalnu poziciju. U svakom momentu  $t$ , brzina čestice  $i$  se ažurira koristeći:

$$v_i(t+1) = \omega v_i(t) + c_1 r_1 (p_i(t) - x_i(t)) + c_2 r_2 (p_g(t) - x_i(t))$$

gde je  $\omega$  inertna težina i obično je postavljena da varira linearno od 1 do 0 tokom iteracije,  $c_1$  i  $c_2$  su koeficijenti ubrzanja,  $r_1$  i  $r_2$  su slučajni brojevi iz uniformne (0,1) raspodele. Ubrzanje  $v_i$  je ograničeno između  $[v_{min}, v_{max}]$ . Ažuriranjem ubrzanja na ovaj način dozvoljavamo čestici  $i$  da traži najbolju poziciju  $p_i(t)$ , dok se najbolje globalno rešenje računa [3]:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

### 3.2 Druga generacija PSO algoritma

Druga generacija PSO algoritma je unapređenje originalnog PSO algoritma, koja u obzir uzima tri aspekta: lokalni optimum svih čestica, globalno najbolje rešenje, i novi koncept - geometrijski centar optimalne populacije. Autor knjige "*Second Generation Particle Swarm Optimization*" objašnjava da ptice održavaju određenu distancu između centra jata (hrane). Jata ptica uvek ostaju u istom regionu neko vreme, tokom kojeg će centar jata ostati nepomeren u očima čestica. Nakon toga, jato se kreće na sledeći region, tada sve čestice moraju održati određenu distancu sa centrom jata.

### 3.3 Novi model PSO-a

Ovaj algoritam je predložio Garro (eng. *Garro*), bazirao ga je na osnovu ideja drugih autora koji su predlagali unapređenje originalnog PSO algoritma. Shi i Eberhart su predlagali linearno variranje inertnih težina kroz generacije, što je znatno unapredilo performanse originalnog PSO algoritma. Yu je razvio strategiju da kada se kroz generacije globalno najbolje rešenje ne poboljšava, svaka čestica  $i$  biva izabrana sa predefinisanim verovatnoćom, a zatim je dodat slučajni šum svakom vektoru brzine dimenzije  $v_i$  izabrane čestice  $i$ . Bazirano na nekim evolutivnim shemama Genetičkih algoritama, nekoliko efektnih mutacija i ukrštanja su predložene za PSO.

## 4 Podešavanje težina neuronske mreže pomoću optimizacije rojem čestica

Treniranje neuronske mreže, odnosno podešavanje težina, izvršeno je pomoću optimizacije rojem čestica. Napravljena je troslojna neuronska mreža (ulazni, skriveni i izlazni sloj) pomoću Keras biblioteke u Pajtonu, pomoću koje se rešavaju klasifikacioni problemi. Dve trećine podataka korišćene su za trening podatke, a ostatak čini test podatke. Skriveni sloj sadrži 100 jedinica/čvorova, dok je broj izlaza ekvivalentan broju različitih klasa u odgovarajućem skupu podataka. Pri kompiliranju Keras modela korišćen je *adam* optimizator, ali on nema nikakvog uticaja, već je naveden iz sintaksnih razloga. Kao metrika korišćena je preciznost (eng. *accuracy*).

Kao što je ranije napomenuto, ANN trenirana je pomoću PSO algoritma. PSO koristi prvo 30, čestica i prolazi kroz 300 iteracija.. Šalju se trening podaci i neuronska mreža. Težine neuronske mreže predstavljaju pozicije čestica. Inicijalne težine biraju se nasumično unutar intervala  $[-2, 2]$ , a ažuriranje težina se vrši prema formuli:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

objašnjenom u poglavlju 3.1. Parametri  $c_1$ ,  $c_2$  i  $w$  izabrani su u odnosu na vrednosti istih tih parametara u radu [3], odnosno  $w$  je 0.3, dok su  $c_1$  i  $c_2$  za 0.5 manji, tačnije 0.5 i 1.0 redom, jer tako algoritam daje bolje rezultate.

Kao funkcija cilja korišćena je preciznost. Takođe, u cilju izbegavanja zaglavljivanja u lokalnom minimumu algoritam se pokreće nekoliko puta i pozicije najgorih pet čestica se nasumično menjaju. Najbolja čestica je upravo ona sa najvećom preciznošću.

## 4.1 Rezultati

U ovoj sekciji biće predstavljene eksperimentalni rezultati dobijeni nad skupovima: *Iris*, *Breast Cancer* i *Wine*, kao i upoređeni sa rezultatima drugih istraživača koji su se bavili ovom temom i koristili navedene skupove. Stopa prepoznavanja neuronske mreže procenjena je i računata formulom:

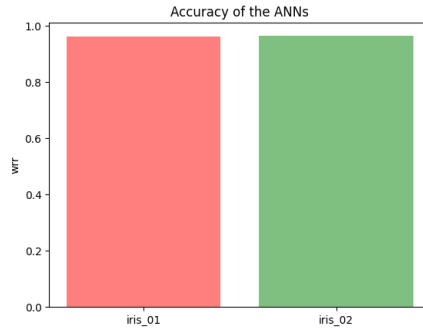
$$wrr = 0.4 * (Tr_{r,r}) + 0.6 * (Te_{r,r}),$$

gde je  $Tr_{r,r}$  preciznost nad trening podacima, a  $Te_{r,r}$  preciznost dobijenu nad test podacima [3]. Korišćeni su faktori 0.4 i 0.6 kako bi se izbegla visoka vrednost  $wrr$  (eng. *weighted recognition rate*) usled visoke preciznosti nad trening podacima (kada to nije slučaj i sa test podacima). U nastavku sledi više o rezultatima na svakom od skupova pojedinačno.

### 4.1.1 Skup Iris

Ovaj skup podataka nalazi se u Pajton paketu za mašinsko učenje: Scikit-learn. Sastoji se od 3 vrste cveta Iris - Setosa, Versicolour, Virginica. Sadrži 150 instanci i ima 5 atributa koji predstavljaju dužinu i širinu latica (eng. *petal length*, *petal width*), dužinu čašičnog listića (eng. *sepal length*, *sepal width*) i vrstu kojoj cvet pripada (eng. *species*).

Procenjena stopa prepoznavanja ( $wrr$ ) neuronske mreže, koja je dobijena na skupu Iris, najveća je od svih testiranih i ona iznosi više od 95%, kako na trening, tako i na test podacima. Na slici 1 može se videti i zaključiti da je dobijena  $wrr$  autora ovog rada i preciznost autora ranije pomenutog rada [3] približno ista.



Slika 1: Preciznost na podacima skupa Iris. Sa leve strane nalaze se rezultati autora, a desno rezultati iz literature [3]

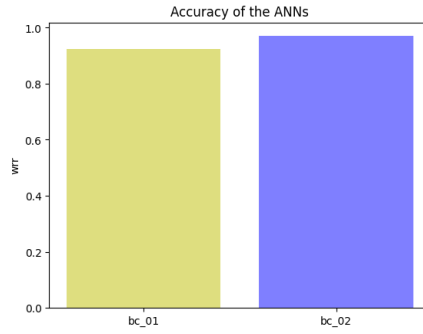
### 4.1.2 Skup Breast cancer

Skup podataka Breast cancer se, takođe, nalazi u Pajton paketu za mašinsko učenje: Scikit-learn. Predstavlja jednostavan skup podataka koji se koristi za binarnu klasifikaciju. Više informacija o skupu dato je u tabeli 1.

Tabela 1: Informacije o skupu Breast Cancer

Klase	2
Instance po klasi	212(M), 357(B)
Ukupno instanci	569
Dimenzija	30
Atributi	real, positive

Rezultati dobijeni na neuronskoj mreži za skup Breast Cancer neznatno su slabiji od rezultata nad skupom podataka Iris. Dobijena  $wrr$  iznosi oko 90-92%. Na slici 2 može se videti poređenje rezultata. Primećuje se da su rezultati autora [3] nešto bolji, ali i dalje su obe preciznosti zadovoljavajuće.



Slika 2: Preciznost na podacima skupa Breast Cancer. Sa leve strane nalaze se rezultati autora, a desno rezultati iz literature [3]

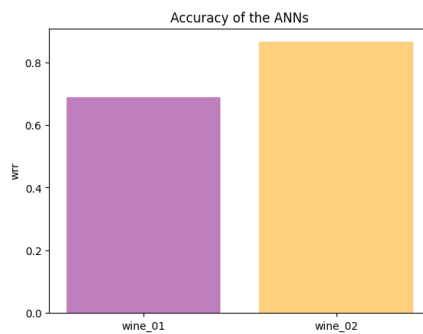
#### 4.1.3 Skup Wine

Wine skup podataka predstavlja klasičan, jednostavan i višeklasni klasifikacioni skup podataka u paketu za Scikit-learn. Više informacija o skupu dato je u tabeli 2.

Tabela 2: Informacije o skupu Wine

Klase	3
Instance po klasi	59,71,48
Ukupno instanci	178
Dimenzija	13
Atributi	real, positive

Najslabiji rezultati dobijeni su upravo na ovom skupu, kako u literaturi, tako i u ovom radu. U radu [3] dobijena je wrt oko 85%, dok su autori ovog rada dobili slabije rezultate - oko 70%, što se može videti na priloženom grafiku 3.



Slika 3: Preciznost na podacima skupa Wine. Sa leve strane nalaze se rezultati autora, a desno rezultati iz literature [3]

## 5 Zaključak

Istraživanjem podešavanja težina neuronske mreže pomoću optimizacije rojem čestica, zaključeno je da rezultati donekle zavise od skupa, što bi moglo sugerisati da je potrebno posvetiti više pažnje pretprocesiranju onih skupova na kojima su rezultati slabiji ili promenu određenih parametara (npr. povećan broj iteracija pri učenju), u cilju veće preciznosti kasnije.

Dalje istraživanje moglo bi da se fokusira na korišćenju drugih optimizacionih algoritama za podešavanje težina neuronskih mreža, kao što su: optimizacija mravljom kolonijom (eng. *Ant Colony Optimization*, ACO), optimizacija kolonijom pčela (eng. *Bee Colony Optimization*,

*BCO*), simulirano kaljenje (eng. *Simulated Annealing, SA*) itd. Drugi pravac istraživanja mogao bi se baviti unapređenjem samog PSO algoritma i njegovih parametara, radi dobijanja boljih rezultata.

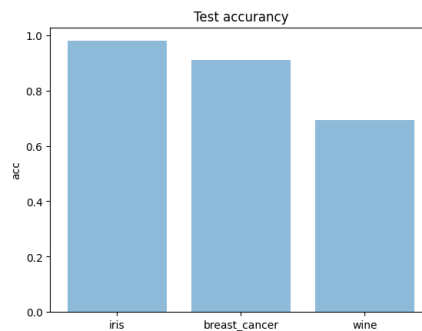
## Literatura

- [1] Internet Archive Wayback Machine. on-line at: <https://web.archive.org/web/20090215055110/http://learnartificialneuralnetworks.com/#Intro>.
- [2] Keras. on-line at: <https://keras.io/>.
- [3] Beatriz A. Garro and Roberto A. Vázquez. Designing Artificial Neural Networks Using Particle Swarm Optimization Algorithms. 2 June 2015.
- [4] J. Yu, L. Xi, and S. Wang. *An improved particle swarm optimization for evolving feedforward artificial neural networks*. Neural Processing Letters vol 26 no 3, 2007.

## A Prikaz rezultata



Slika 4: Uporedni prikaz rezultata dobijenih na trening podacima, za sva tri skupa



Slika 5: Uporedni prikaz rezultata dobijenih na test podacima, za sva tri skupa

## B Kod

```
1000 import pandas as pd                # data processing, CSV file I/O (e.g. pd.read_csv)
1001 import numpy as np                  # linear algebra
1002 from numpy.random import seed
1003 from numpy.random import rand
1004 from sklearn import datasets
1005 from sklearn.model_selection import train_test_split
1006 from sklearn import preprocessing
1007 from keras.models import Sequential
1008 from keras.layers import Dense
1009 from keras import losses, optimizers
1010 from keras.utils import to_categorical
1011
1012 from matplotlib import pyplot as plt
```

```
1000 class Particle:
1001
1002     def __init__(self, weights, velocity, omega, c1, c2,
1003                   lower_bound, upper_bound, particle_id):
1004         self.id = particle_id
1005         self.weights = weights
1006         self.best_weights = weights
1007         self.velocity = np.array(velocity)
1008         self.omega = omega
1009         self.c1 = c1
1010         self.c2 = c2
1011         self.lower_bound = lower_bound
```

```

1012         self.upper_bound = upper_bound
1013         self.best_valuation = 0
1014         self.history = []

1016     def update(self, best_particle):
1017         self.update_velocity(best_particle)
1018         self.check_velocity()
1019         self.update_particle()

1020     return self.weights

1022     def update_velocity(self, bp):
1023         # v = Wv + c1r1(pi - xi) + c2r2(g - xi)
1024         # v - ubrzanje
1025         # W - omega
1026         # c1 i c2 - unapred zadati parametri
1027         # r1 i r2 - slucajni brojevi iz (0,1) uniformne raspodele
1028         # pi - najbolje resenje trenutne jedinke
1029         # g - najbolje globalno resenje
1030         r1 = np.random.random()
1031         r2 = np.random.random()

1032         self.velocity = self.omega * self.velocity + self.c1*r1*(np.add(np.array(
1033             self.best_weights), (-1)*np.array(self.weights))) + self.c2*r2*(np.add(np.array(
1034             bp), (-1)*np.array(self.weights)))

1036     def check_velocity(self):
1037         for i in range(len(self.velocity)):
1038             if(i%2 == 1):
1039                 for j in range(len(self.velocity[i])):
1040                     if(self.velocity[i][j] < self.lower_bound):
1041                         self.velocity[i][j] = self.lower_bound

1042                     if(self.velocity[i][j] > self.upper_bound):
1043                         self.velocity[i][j] = self.upper_bound

1044             else:
1045                 for j in range(len(self.velocity[i])):
1046                     for k in range(len(self.velocity[i][j])):

1047                         if(self.velocity[i][j][k] < self.lower_bound):
1048                             self.velocity[i][j][k] = self.lower_bound

1049                         if(self.velocity[i][j][k] > self.upper_bound):
1050                             self.velocity[i][j][k] = self.upper_bound

1052         return

1054     def update_particle(self):
1055         # xi = xi + vi
1056         self.weights = np.add(np.array(self.weights), self.velocity)

1058     def update_valuation(self, valuation):
1059         self.history.append(valuation)

1060         if(valuation > self.best_valuation):
1061             self.best_valuation = valuation
1062             self.best_weights = self.weights

1064     def get_weights(self):
1065         return self.weights

1067     def set_weights(self, new_weights):
1068         self.weights = new_weights

```

```

1000 class PSO:
1001     def __init__(self, num_particles, num_iters, training_x, training_y, model,
1002         shapes, c1, c2, w, lower_bound, upper_bound):
1003         self.num_iters = num_iters
1004         self.training_x = training_x
1005         self.training_y = training_y
1006         self.model = model
1007         self.best_particle = self.make_velocity(shapes)
1008         self.best_evaluation = 0
1009         self.history = []
1010         self.shapes = shapes
1011         self.lower_bound = lower_bound
1012         self.upper_bound = upper_bound
1013         self.particles = self.make_particles(num_particles, shapes, c1, c2, w,
1014             lower_bound, upper_bound)
1015         self.evaluate_particles()

```



```

1016         self.combo = 0
1017         self.w = w
1018         self.c1 = c1
1019         self.c2 = c2
1020
1021     def make_particles(self, size, shape, c1, c2, w, low, upp):
1022         particles = []
1023
1024         velocity = self.make_velocity(shape)
1025
1026         for i in range(size):
1027             weights = self.make_weights(shape, low, upp)
1028             particles.append(Particle(weights, velocity, w, c1, c2, low, upp, i))
1029
1030         return particles
1031
1032     def make_velocity(self, shapes):
1033         velocity = []
1034
1035         for shape in shapes:
1036             velocity.append(np.zeros(shape))
1037
1038         return velocity
1039
1040     def make_weights(self, shapes, low, upp):
1041         weights = []
1042
1043         for shape in shapes:
1044             if (len(shape) == 1):
1045                 # wght da dobijemo vrednosti iz (0, upp-low)
1046                 wght = rand(shape[0])*(upp-low)
1047                 rec = np.full(shape, low)
1048
1049                 weights.append(np.add(wght, rec))
1050
1051             else:
1052                 wght = rand(shape[0], shape[1])*(upp-low)
1053                 rec = np.full(shape, low)
1054
1055                 weights.append(np.add(wght, rec))
1056
1057         return weights
1058
1059     def evaluate_particles(self):
1060
1061         for particle in self.particles:
1062             self.model.set_weights(particle.get_weights())
1063             train_loss, train_acc = self.model.evaluate(self.training_x, self.
1064             training_y, verbose = 0)
1065
1066             particle.update_valuation(train_acc)
1067
1068             if (train_acc > self.best_evaluation):
1069                 self.best_particle = particle.get_weights()
1070                 self.best_evaluation = train_acc
1071
1072     def isclose(self, a, b, rel_tol=1e-09, abs_tol=0.0):
1073         return abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)
1074
1075     def update_particles(self):
1076
1077         for particle in self.particles:
1078             particle.update(self.best_particle)
1079
1080     def get_particles(self):
1081         return self.particles
1082
1083     def start_pso(self):
1084
1085         for i in range(self.num_iters):
1086             partly_best_solution = self.best_evaluation
1087             self.update_particles()
1088             self.evaluate_particles()
1089
1090             #print("Iteracija: {}\nPreciznost: {}\n".format(i+1, self.
1091             best_evaluation))
1092
1093             if (self.isclose(partly_best_solution, self.best_evaluation, 0.001)):
1094                 self.combo += 1
1095             else:
1096                 self.combo = 0

```

```

1098         if(self.combo == 5):
1100             self.particles.sort(key = lambda x: x.best_valuation)
1102             velocity = self.make_velocity(self.shapes)
1104             weights = self.make_weights(self.shapes, self.lower_bound, self.
upper_bound)
1106             self.particles[0] = Particle(weights, velocity, self.w, self.c1,
self.c2,
self.lower_bound, self.upper_bound,
self.particles[0].id)
1108             weights = self.make_weights(self.shapes, self.lower_bound, self.
upper_bound)
1110             self.particles[1] = Particle(weights, velocity, self.w, self.c1,
self.c2,
self.lower_bound, self.upper_bound,
self.particles[1].id)
1112             weights = self.make_weights(self.shapes, self.lower_bound, self.
upper_bound)
1114             self.particles[2] = Particle(weights, velocity, self.w, self.c1,
self.c2,
self.lower_bound, self.upper_bound,
self.particles[2].id)
1116             weights = self.make_weights(self.shapes, self.lower_bound, self.
upper_bound)
1118             self.particles[3] = Particle(weights, velocity, self.w, self.c1,
self.c2,
self.lower_bound, self.upper_bound,
self.particles[3].id)
1120             weights = self.make_weights(self.shapes, self.lower_bound, self.
upper_bound)
1122             self.particles[4] = Particle(weights, velocity, self.w, self.c1,
self.c2,
self.lower_bound, self.upper_bound,
self.particles[4].id)
1124             self.combo = 0
1126             self.evaluate_particles()
1128             #print("Change bad particles")
1130             self.history.append(self.best_evaluation)

return self.best_particle

```

```

1000 accs = []
1001 print("IRIS DATASET: \n\n")
1002 data1 = datasets.load_iris()
1004 x_train, x_test, y_train, y_test = train_test_split(data1.data,
data1.target,
1006 test_size=0.33)
1008 y_train = to_categorical(y_train)
1010 y_test = to_categorical(y_test)
1012 # Pravljenje neuronske mreze sa 3 sloja - ulazni, skriveni i izlazni
1013 model = Sequential()
1014 model.add(Dense(units=100, input_dim=x_train.shape[1], activation='relu'))
1015 model.add(Dense(units=y_train.shape[1], activation='sigmoid'))
1016 model.summary()
1018 shapes = [i.shape for i in model.get_weights()]
1020 model.compile(optimizer='adam',
loss=losses.categorical_crossentropy,
1022 metrics=['accuracy'])
1024 test_iris_acc = 0
1025 train_iris_acc = 0
1026 best_iris_pso = None
1028 PSOS = []
1029 for i in range(5):
1030     pso = PSO(30, 300,
x_train, y_train,

```

```

1032         model,
1033         shapes,
1034         0.5, 1.0, 0.3, -2, 2)
1036     PSOS.append(pso)
1038     for pso in PSOS:
1040         best_particle = pso.start_pso()
1042         model.set_weights(best_particle)
1044         train_loss, train_acc = model.evaluate(x_train, y_train)
1045         test_loss, test_acc = model.evaluate(x_test, y_test)
1046
1047         if(test_iris_acc < test_acc):
1048             test_iris_acc = test_acc
1049             best_iris_pso = pso
1050             train_iris_acc = train_acc
1052
1053         print("Current PSO train accuracy: " + str(train_acc))
1054         print("Current PSO test accuracy: " + str(test_acc) + "\n")
1056     print()
1057     print("Global best accuracy: " + str(test_iris_acc))
1058     accs.append((train_iris_acc, test_iris_acc))

```

```

1000     print("\n-----\nBREAST CANCER DATASET: \n\n")
1001     data2 = datasets.load_breast_cancer()
1002
1003     x_train, x_test, y_train, y_test = train_test_split(data2.data,
1004                                                         data2.target,
1005                                                         test_size=0.33)
1006
1007     y_train = to_categorical(y_train)
1008     y_test = to_categorical(y_test)
1009
1010     # Pravljenje neuronske mreze sa 3 sloja - ulazni, skriveni i izlazni
1011     model = Sequential()
1012     model.add(Dense(units=100, input_dim=x_train.shape[1], activation='relu'))
1013     model.add(Dense(units=y_train.shape[1], activation='sigmoid'))
1014     model.summary()
1015
1016     shapes = [i.shape for i in model.get_weights()]
1017
1018     model.compile(optimizer='adam',
1019                  loss=losses.categorical_crossentropy,
1020                  metrics=['accuracy'])
1021
1022     test_cancer_acc = 0
1023     train_cancer_acc = 0
1024     best_cancer_pso = None
1025
1026     PSOS = []
1027     for i in range(5):
1028         pso = PSO(30, 300,
1029                  x_train, y_train,
1030                  model,
1031                  shapes,
1032                  0.5, 1.0, 0.3, -2, 2)
1033
1034         PSOS.append(pso)
1035
1036     for pso in PSOS:
1037         best_particle = pso.start_pso()
1038
1039         model.set_weights(best_particle)
1040
1041         train_loss, train_acc = model.evaluate(x_train, y_train)
1042         test_loss, test_acc = model.evaluate(x_test, y_test)
1043
1044         if(test_cancer_acc < test_acc):
1045             test_cancer_acc = test_acc
1046             train_cancer_acc = train_acc
1047             best_cancer_pso = pso
1048
1049         print("Current PSO train accuracy: " + str(train_acc))
1050         print("Current PSO test accuracy: " + str(test_acc) + "\n")
1051
1052     print()
1053     print("Global best accuracy: " + str(test_cancer_acc))

```

```
accs.append((train_cancer_acc, test_cancer_acc))
```

```
1000 print("\n-----\nWINE DATASET: \n\n")
1002 data3 = datasets.load_wine()
1004 x_train, x_test, y_train, y_test = train_test_split(data3.data,
1006                                                     data3.target,
1008                                                     test_size=0.33)
1006 y_train = to_categorical(y_train)
1008 y_test = to_categorical(y_test)
1010 # Pravljenje neuronske mreze sa 3 sloja - ulazni, skriveni i izlazni
1012 model = Sequential()
1014 model.add(Dense(units=100, input_dim=x_train.shape[1], activation='relu'))
1016 model.add(Dense(units=y_train.shape[1], activation='sigmoid'))
1018 model.summary()
1020 shapes = [i.shape for i in model.get_weights()]
1022 model.compile(optimizer='adam',
1024               loss=losses.categorical_crossentropy,
1026               metrics=['accuracy'])
1028 test_wine_acc = 0
1030 train_wine_acc = 0
1032 best_wine_pso = None
1034 PSOS = []
1036 for i in range(5):
1038     pso = PSO(30, 300,
1040               x_train, y_train,
1042               model,
1044               shapes,
1046               0.5, 1.0, 0.3, -2, 2)
1048     PSOS.append(pso)
1050 for pso in PSOS:
1052     best_particle = pso.start_pso()
1054     model.set_weights(best_particle)
1056     train_loss, train_acc = model.evaluate(x_train, y_train)
1058     test_loss, test_acc = model.evaluate(x_test, y_test)
1060     if(test_wine_acc < test_acc):
1062         test_wine_acc = test_acc
1064         train_wine_acc = train_acc
1066         best_wine_pso = pso
1068     print("Current PSO train accuracy: " + str(train_acc))
1070     print("Current PSO test accuracy:   " + str(test_acc) + "\n")
1072 print()
1074 print("Global best accuracy: " + str(test_wine_acc))
1076 accs.append((train_wine_acc, test_wine_acc))
```