Upute za laboratorijske vježbe iz predmeta Neuronske mreže

Hrvoje Kalinić, Sven Ločarić

4. studenog, 2011.

Sadržaj

1. Vježba 1: Model neurona i proces učenja 3
   - 1.1. Priprema za vježbu 3
   - 1.2. Model neurona 3
   - 1.3. Mreža s tri neurona 4
   - 1.4. Učenje korekcijom pogreške (delta pravilo) 4
2. Vježba 2: Asocijativna memorija 6
   - 2.1. Priprema za vježbu 6
   - 2.2. Direktno kreiranje korelacijske matrice 6
   - 2.3. Formiranje korelacijske matrice učenjem pod nadzorom . 9
3. Vježba 3: Perceptron 12
   - 3.1. Priprema za vježbu 12
   - 3.2. Perceptron za klasifikaciju uzoraka 12
   - 3.3. Primjer klasifikacije uzoraka s Gaussovom razdiobom . . 15
   - 3.4. Klasifikacija uzoraka pomoću dva perceptrona 15
4. Vježba 4: LMS algoritam za predviđanje cijena dionice 17
   - 4.1. Priprema za vježbu 17
   - 4.2. Kretanje cijene dionice 17
5. Vježba 5: Višeslojni perceptron 20
   - 5.1. Priprema za vježbu 20
   - 5.2. Višeslojni perceptron za klasifikaciju uzoraka 20
   - 5.3. Segmentacija slike višeslojnim perceptronom 23
6. Vježba 6: Radijalne mreže 26
   - 6.1. Priprema za vježbu 26
   - 6.2. Radijalne funkcije 26
   - 6.3. Realizacija XOR logičke funkcije radijalnom mrežom . . 27
   - 6.4. Interpolacija funkcije radijalnom mrežom 28
7. Vježba 7: Rekurzivne mreže 31
   - 7.1. Priprema za vježbu 31
   - 7.2. Hopfieldova mreža 31
   - 7.3. Hopfieldova mreža kao asocijativna memorija za slike . . 34
8. Vježba 8: Samoorganizirajuće mreže 36
   - 8.1. Priprema za vježbu 36
   - 8.2. Ulazni skup podataka 36
   - 8.3. Hebbov zakon učenja bez nadzora 37
   - 8.4. Ojin zakon učenja bez nadzora 37
   - 8.5. Kompetitivno učenje za grupiranje vektora 38

8.6. Problem trgovačkog putnika 38
9. Vježba 9: Genetički algoritmi 40
9.1. Priprema za vježbu 40
9.2. Traženje maksimuma pomoću GA 40
9.3. Treniranje višeslojne neuronske mreže pomoću GA . . . 42
9.4. Rješavanje problema putujućeg trgovca pomoću GA . . . 43

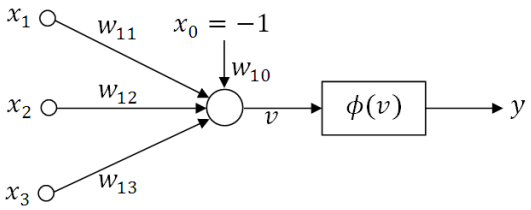
1. Vježba 1: Model neurona i proces učenja

1.1. Priprema za vježbu

Proučiti kratki uvodni tekst "MATLAB upute" koji je dostupan na web stranicama predmeta, i po potrebi pročitati originalnu korisničku dokumentaciju na adresi `http://matlab.zesoi.fer.hr`. Da bi se stekla potrebna teoretska priprema za vježbu potrebno je proučiti materijal "Neuronske mreže: Predavanja", poglavlja "Uvod" i "Proces učenja".

1.2. Model neurona

Napisati MATLAB funkciju za izračunavanje izlazne vrijednosti neurona. Pretpostaviti model neurona na Slici 1. s tri ulaza i pragom. Prag se može interpretirati i kao dodatni ulaz sa fiksnim iznosom -1 i težinom w_{10}. Izlaz napisane funkcije mora odgovarati izlazu neurona.



Slika 1.: Model neurona

Za računanje aktivacije neurona v koristiti skalarni produkt vektora ulaza $[\mathbf{x_0}\,\mathbf{x_1}\,\mathbf{x_2}\,\mathbf{x_3}]$ i vektora težina $[\mathbf{w_{10}}\,\mathbf{w_{11}}\,\mathbf{w_{12}}\,\mathbf{w_{13}}]$. Napisati funkciju tako da ovisno o dodatnom ulazu bira različitu nelinearnu funkciju. Eksperimentirati s nelinearnim funkcijama oblika:

1. jedinični skok (step funkcija)
2. funkcija linearna po odsječcima (rampa)
3. funkcija sigmoidnog oblika definirana izrazom $\phi(v) = \frac{1}{1+\exp(-av)}$, uz a=1.

Odgovorite:

1. Pretpostavite neku proizvoljnu vrijednost vektora težina $\mathbf{w}$. Navedite koje težine ste odabrali te izračunajte odziv neurona na sljedeće ulaze (za sve 3 aktivacijske funkcije):

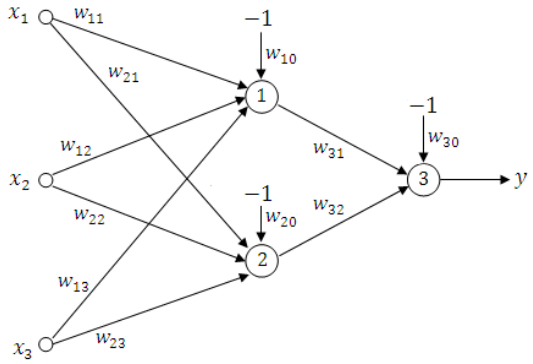
```
x1 = [0.5, 1, 0.7]';
x2 = [0, 0.8, 0.2]';
```

1.3. Mreža s tri neurona

Napisati funkciju za mrežu s tri neurona (Slika 2.) pri tom koristeći funkciju razvijenu u eksperimentu iz poglavlja 1.2.. Pretpostaviti da neuroni koriste sigmoidnu nelinearnost, gdje je $a = 1$, te da su vektori težina za svaki neuron zadani kako slijedi:

```
w1 = [   1,  0.5,    1, -0.4 ]';
w2 = [ 0.5,  0.6, -1.5, -0.7 ]';
w3 = [-0.5, -1.5,  0.6 ]';
```

Napomena: prvi element vektora težina su vrijednosti pragova neurona koji su označeni na Slici 2. kao w_{i0}.



Slika 2.: Mreža s 3 neurona

Odgovorite:

1. Koliki je odziv na ulazni vektor $\mathbf{x} = [\mathbf{0.3, 0.7, 0.9}]$'?
2. Ovisi li izlaz mreže o težinama neurona?

1.4. Učenje korekcijom pogreške (delta pravilo)

Svrha ovog eksperimenta je steći bolje razumijevanje procesa učenja. U ovom eksperimentu realizirat ćemo logičku AND funkciju pomoću jednog neurona s dva ulaza i pragom (vidi Sliku 3.). Koristit ćemo sigmoidnu nelinearnu karakteristiku s parametrom a = 1.

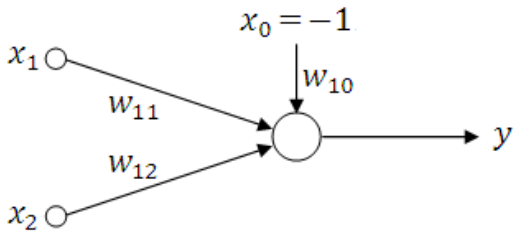
Za fazu učenja neuronske mreže potrebno je definirati sljedeće parove x_i, y_i za učenje logičke AND funkcije: za ulaze **x1 = [-1, 0, 0]', x2 = [-1, 0, 1]'** i **x3 = [-1, 1, 0]'** izlaz **y** treba biti jednak 0. Za ulazni vektor **x4 = [-1, 1, 1]'** izlazna vrijednost **y** treba biti jednaka 1. Prva komponenta svih ulaznih vektora koja ima vrijednost -1 definira prag neurona označen kao w_{10}. Na početku učenja treba postaviti početne težine na neke slučajne vrijednosti. Za učenje treba koristiti delta pravilo:

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n)$$

gdje je

$$e_k(n) = d_k(n) - y_k(n)$$

pri čemu je $d_k(n)$ željeni odziv neurona, a $y_k(n)$ dobiveni odziv neurona. Iterativni proces učenja treba nastaviti sve dok pogreška ne bude zadovoljavajuće mala.



Slika 3.: Mreža s jednim neuronom

Odgovorite:

1. Eksperimentirajte s različitim početnim vrijednostima težina i različitim konstantama brzine učenja. (U slučaju nestabilnosti procesa učenja, ponoviti pokus koristeći male vrijednosti za konstantu učenja, npr. η=0.05.)

2. Prikazati funkciju pogreške u ovisnosti o broju iteracija učenja za nekoliko različitih konstanti učenja.

3. Komentirati rezultate:

   a) Koji je najbolji odabir konstante učenja?

   b) Kako ste definirali "dovoljno malu pogrešku"?

   c) Nakon koliko koraka je pogreška bila dovoljno mala?

4. Možemo li i u ovom eksperimentu koristiti (nemodifciranu) funkciju razvijenu u poglavlju 1.2.? Zašto

2. Vježba 2: Asocijativna memorija

2.1. Priprema za vježbu

Da bi se stekla potrebna teoretska priprema za vježbu potrebno je proučiti materijal s predavanja, posebno poglavlja "Proces učenja" i "Asocijativna memorija".
Prije početka rada skinite sa stranice predmeta komprimiranu datoteku "dodatni materijali-2.vjezba.zip" koja sadrži funkcije potrebne za izvođenje vježbe. Ove funkcije kopirajte u radni direktorij.

Odgovorite:

1. Koju operaciju operator ' (jednostruki apostrof) obavlja nad vektorom?
2. Koji izlaz vraća matlab funkcija *transpose()* ako je ulaz vektor?
3. Kako analitički izražavamo udaljenost između dva vektora?
4. Koje uvjete mora zadovoljiti funkcija da bi se mogla koristiti kao funkcija pogreške?
5. Predložite neku funkciju pogreške.
6. Koju operaciju obavlja matlab funkcija *sse()*?.

2.2. Direktno kreiranje korelacijske matrice

U ovom djelu vježbe koristit ćemo direktni pristup za formiranje korelacijske matrice. Memorija temeljena na korelacijskoj matrici treba zapamtiti parove asocijacija ulaz-izlaz koji su prikazani u obliku vektora odnosno riječi. Za svaki ulazni vektor (ključ) memorija treba zapamtiti pripadni izlazni uzorak tj. vektor u formi ASCII koda. U primjeru se koriste 4-dimenzionalni ulazni i izlazni vektori. Riječi (izlazni vektori) koje asocijativna memorija treba zapamtiti su: 'vrat' , 'kraj' , 'cres' , 'otac'. Vektore bi koji predstavljaju te riječi treba formirati na sljedeći način:

```
b1 = real('vrat')';
b2 = real('kraj')';
b3 = real('cres')';
b4 = real('otac')';
```

2.2.1. Ortogonalni ulazni vektori

Ovaj eksperiment demonstrira način formiranja korelacijske matrice koja predstavlja asocijativnu memoriju. Kao ulazni skup vektora **ai** (ključeva) u ovom eksperimentu koristi se ortonormalizirani skup vektora definiran na sljedeći način:

```
a1 = [1,0,0,0]';
a2 = [0,1,0,0]';
a3 = [0,0,1,0]';
a4 = [0,0,0,1]';
```

Treba formirati memorijsku korelacijsku matricu **M** pomoću ulazno izlaznih parova koristeći sljedeći izraz:

```
M = b1 * a1' + b2 * a2' + b3 * a3' + b4 * a4';
```

Za provjeru je li memorija ispravno zapamtila zadane asocijacije treba izračunati odziv na pojedine ulazne vektore. Npr. odziv na ključ **a1** može se izračunati na sljedeći način:

```
char(M * a1)'
```

Odgovorite:

1. Jesu li svi parovi ispravno zapamćeni ?
2. Koji je odziv za pojedine ključeve.
3. Koliko je parova ispravno zapamćeno ukoliko vektori **ai** imaju normu različitu od 1?

2.2.2. Svojstva korelacijske matrice

Drugi eksperiment ima za cilj demonstrirati kapacitet dobivene asocijativne memorije. U ovom djelu vježbe ćemo pokušati dodati još jednu (petu) asocijaciju ('mrak'). Budući da u 4-dimenzionalnom vektorskom prostoru ima najviše četiri linearno nezavisna vektora, kao ključ za peti izlazni uzorak odabrat ćemo proizvoljni 4-dimenzionalni jedinični vektor, npr. kao:

```
a5 = ( a1 + a3) / sqrt(2);
```

Formirajte vektore **b5** ('mrak') i **a5** na opisani način te ih dodajte u memoriju koristeći izraz:

```
M = M + b5 * a5';
```

Odgovorite:

1. Je li nova asocijacija ispravno zapamćena?
2. Jesu li ostale asocijacije ostale dobro zapamćene?
   (a) Ako nisu - koje to asocijacije nisu dobro zapamćene i zašto?
   (b) Ako jesu - koje to asocijacije jesu dobro zapamćene i zašto?

2.2.3. Parovi riječi kao asocijacije

U ovom djelu vježbe treba formirati asocijativnu memoriju koja pamti asocijacije koje se sastoje od parova riječi. Zadane asocijacije koje treba zapamtiti su: ruka-vrat, kset-kraj, more-cres, mama-otac. Generirati ulazne vektore (ključeve) asocijacija na sljedeći način:

```
a1 = real('ruka')';
a2 = real('kset')';
a3 = real('more')';
a4 = real('mama')';
```

Budući su izlazne asocijacije jednake onima korištenim u prvom dijelu vježbe, vektore **bi** nije potrebno ponovo unositi. Ponovo formirati matricu **M** koristeći isti postupak kao u prvom dijelu vježbe.

Odgovorite:

1. Koji je odziv na pojedine ulazne ključeve.
2. Koje su asocijacije ispravno zapamćene?
3. Koje asocijacije nisu ispravno zapamćene i zašto?
4. Na koji način to možemo ispraviti?

2.2.4. Utjecaj ortogonalizacije ulaznih vektora

U ovom eksperimentu prikazana je asocijativna memorija koja koristi ključeve koji su ortonormalizirani skup vektora. Za ortonormalizaciju koristi se Gram-Schmidtov postupak ortogonalizacije na sljedeći način. Najprije treba formirati matricu **A** pomoću vektora **ai** kako slijedi:

```
A= [a1, a2, a3, a4];
```

Zatim treba provesti postupak ortonormiranja:

```
C = orth(A);
```

Iz matrice **C** zatim treba izlučiti pojedine ortonormalne vektore **ci**:

```
c1 = C(1,:)' , ..., c4 = C(4,:)';
```

U sljedećem koraku treba formirati novu matricu **M** koristeći vektore **ci** umjesto **ai** u izrazu za formiranje matrice **M**. Provjeriti odziv matrice **M** na vektore **ci** korištenjem izraza:

```
char(round(M * c1))'  , ...
```

Odgovorite:

1. Kakav je utjecaj ortonormiranja vektora?
2. Koliko je ispravno zapamćenih parova i zašto?
3. Što možemo očekivati ukoliko samo normiramo vektore?
4. Što možemo očekivati ukoliko samo ortogonaliziramo vektore?
5. Što ukoliko su vektori **ci** linearno nezavisni, no ne i ortogonalni?

2.2.5. Određivanje korelacijske matrice inverznom matricom

Za ranije korištene parove riječi (ruka-vrat, kset-kraj, more-cres, mama-otac) odrediti korelacijsku matricu $\mathbf{M}$ dimenzija 4x4 kao $\mathbf{M} = \mathbf{B}\mathbf{A}^{-1}$, gdje je matrica $\mathbf{B}$ definirana kao:

```
B = [b1, b2, b3, b4]
```

Odgovorite:

1. Jesu li parovi asocijacija ispravno zapamćeni? Napomena: Rezultat treba zaokružiti na cijeli broj prije usporedbe (podsjetite se kako ste to učinili u poglavlju 2.2.4.).

2.2.6. Određivanje korelacijske matrice pseudoinverznom matricom

Pseudoinverzna matrica može se koristiti za određivanje korelacijske matrice u slučaju kada je broj parova asocijacija veći od dimenzije vektora asocijacija. U tom slučaju korelacijska matrica se može odrediti kao $\mathbf{M} = \mathbf{B}\mathbf{A}^{+}$, gdje je $\mathbf{A}^{+}$ pseudoinverzna matrica dana izrazom $\mathbf{A}^{+} = \mathbf{A}^{T}(\mathbf{A}\mathbf{A}^{T})^{-1}$.

Pretpostaviti da su dani vektori **ai** i **bi** definirani ranije (pet asocijacija). Odrediti pseudoinverznu matricu za ovaj slučaj po gore navedenom izrazu.

Odgovorite:

1. Jesu li parovi točno zapamćeni?
2. Ukoliko nisu, kolika je pogreška u numeričkoj vrijednosti pojedinih izlaznih vektora? (Napomena: Koristite funkciju pogreške koju ste predložili u poglavlju 2.1., zadatak 5.)

2.3. Formiranje korelacijske matrice učenjem pod nadzorom

Ovaj dio vježbe demonstrira alternativni način formiranja matrice $\mathbf{M}$ asocijativne memorije – kroz proces učenja pod nadzorom. Konkretno u sljedeća dva eksperimenta koristi se učenje s korekcijom pogreške.

2.3.1. Učenje korekcijom pogreške

Formirati matrice **A** i **B** gdje svaka sadrži po 4 vektora posložene u stupce na način kako je objašnjeno u prethodnim eksperimentima. Provjerite sadržaj matrica **A** i **B** sljedećim instrukcijama:

```
char(A)' , char(B)'
```

Za učenje treba inicijalizirati matricu **M** na neku početnu vrijednost (npr. slučajne vrijednosti uniformno raspoređene na intervalu [-0.5, 0.5]):

```
M = -0.5 + rand (4, 4);
```

Za proces učenja koristi se funkcija *trainlms*, koja predstavlja implementaciju Widrow-Hoff LMS algoritma za učenje, sličnu onomu upotrebljenom u prvoj vježbi. Funkcija se poziva na sljedeći način:

```
[M, e] = trainlms(ni,A,B,M,max_br_iter);
```

gdje je **max_br_iter** broj iteracija učenja, a **ni** konstanta učenja. Varijablu **max_br_iter** pokušajte odrediti eksperimentalno, dok se za ocjenu **ni** možete poslužiti ovom procjenom:

```
ni = 0.9999/max(eig(A*A'));
```

Funkcija *trainlms* provodi učenje dok SSE ne padne ispod 0.02 ili ne odradi broj predviđenih iteracija. Nakon završetka učenja treba pogledati odziv asocijativne memorije opisane korelacijskom matricom **M** na ulazne ključeve **A** matrice:

```
char(round(M*A))'
```

Ako utipkamo:

```
round(M * A)'  ==  B'
```

vidjet ćemo koja su slova ispravno rekonstruirana: na njihovim pozicijama bit će ispisano 1, a na ostalima 0. Višestrukim pozivanjem funkcije *trainlms* (u obliku navedenom ranije) možemo produžiti proces učenja i možda povećati broj korektno zapamćenih slova, no ispravnije je proces učenja produžiti povećavajući **max_br_iter**. Graf ovisnost pogreške o broju iteracija možete iscrtati (u logaritamskoj skali) naredbom:

```
loglog(e)
```

Zadatak:

1. Napravite graf ovisnosti broja zapamćenih slova o broju korištenih iteracija. (*Pripaziti*: Pri izgradnji grafa svaki puta pokrenite simulaciju sa istom početnom matricom.)

2.3.2. Utjecaj većeg broja asocijacija

Ovaj eksperiment demonstrira kapacitet asocijativne memorije. Koliki je kapacitet asocijativne memorije realizirane korelacijskom matricom dimenzija 4×4?

Za dodatni par 'auto'-'mrak' kreirati vektore **a5** i **b5** na način opisan u prethodnom djelu vježbe. Također treba ponovo kreirati matrice **A** i **B**, dimenzija 4 (redaka) × 5 (stupaca) na ranije opisan način. Ponovno inicijalizirati matricu **M** na početne slučajne vrijednosti. Koristiti *trainlms* funkciju za učenje na sljedeći način:

```
[M, e] = trainlms(ni,A,B,M,max_br_iter);
```

Odgovorite:

1. Koliko ste iteracija koristili?
2. Koliko je ispravno zapamćenih slova?
3. Kolika je pogreška SSE?
4. Što se dogodi ukoliko ponovimo poziv funkcije?
5. Koliko je sada ispravno zapamćenih slova, a kolika pogreška? Postoji li razlika i zašto?
6. Je li moguće istrenirati ovu mrežu da zapamti svih pet zadanih asocijacija?
7. Zašto? (objasnite prethodni odgovor)

3. Vježba 3: Perceptron

3.1. Priprema za vježbu

Da bi se stekla potrebna teoretska priprema za vježbu potrebno je proučiti materijal "Neuronske mreže: Predavanja", poglavlja "Proces učenja" i "Perceptron". *Napomena*: Ova vježba nije potpuno kompatibilna s Octave-om.

Prije početka rada sa stranice predmeta iz repozitorija *dodatni materijali* skinite komprimiranu datoteku "vjezba_3.zip" koja sadrži funkcije potrebne za izvođenje vježbe. Ove funkcije kopirajte u radni direktorij.

Odgovorite:

1. Što je hiperravnina?
2. Što radi funkcija *figure*, a što funkcija *plot*?
3. Što radi funkcija *initp*.
4. Koji su ulazi, a koji izlazi funkcija *random* i *trainlms_p*.

3.2. Perceptron za klasifikaciju uzoraka

Da bi demonstrirali temeljne koncepte perceptrona potrebno je prvo definirati ulazne i izlazne podatke. Kao ulazne podatke koristit ćemo N dvodimenzionalnih vektora **ai** organiziranih u matricu **A** dimenzija 2×N (dva retka i N stupaca):

```
A = [ax1, ax2,  ax3, ..., axN; ...
     ay1, ay2, ay3,  ..., ayN];
```

Ovdje je N broj vektora, a **axi** i **ayi** su x i y koordinate i-tog vektora. U ovom primjeru demonstrirat ćemo klasifikaciju vektora u dvije klase. U tom slučaju svaki ulazni vektor može pripadati jednoj od dvije moguće klase, nazovimo ih C_0 i C_1. Njihovu pripadnost pojedinoj klasi definirat ćemo u MATLAB-u pomoću matrice **C** dimenzija 1×N kako slijedi:

```
C = [ c1, c2 ,.. , cN ]
```

Ovdje element **ci** ima vrijednost 0 ako vektor **ai** pripada klasi C_0, odnosno vrijednost 1 ako vektor pripada klasi C_1.

3.2.1. Klasifikacija linearno separabilnih uzoraka u 2D prostoru

U ovom primjeru demonstrirat će se sposobnost perceptrona da separira vektore u dvije klase koje su linearno separabilne. Za ulazne vektore uzmite sljedeće vrijednosti:

```
a1 = [1, 1]', a2 = [1, 1]', a3 = [2, 0]', a4 = [1, 2]', a5 = [2, 1]';
```

s time da vektori **a1**, **a2** i **a3** pripadaju klasi C_0, a ostali klasi C_1. Formirati vektore u matricu **A**, a njihovu pripadnost klasama formirati u vektor **C** na gore opisani način. Iscrtati položaj vektora u ravnini pozivom sljedeće funkcije:

```
plotpv(A,C)
```

Vektori koji pripadaju istoj klasi imaju istu oznaku na grafu. Inicijalizirati perceptron na sljedeći način:

```
[W] = initp(A,C)
```

Ovdje je **W** vektor u kojemu su sadržane težine neuronske mreže. Prvi stupac matrice **W** predstavlja pragove (dakle, **W**(:,1)).

Odgovorite:

1. Segmentira li perceptron opisan matricom **W** ispravno ravninu na klase C_0 i C_1?

2. Hiperavninu iscrtajte naredbom:

   ```
   plotpc(W(:,2:end),W(:,1));
   ```

   (Napomena: Ako ste obrisali graf generiran s *plotpv* funkcijom ponovo ga iscrtajte prije poziva *plotpc* funkcije)

3. Dijeli li pravac ispravno klase C_2 i C_1?

Perceptron trenirajte pozivom funkcije *trainlms_p* dok se ne postigne ispravna segmentacija ravnine u klase.

```
[M, e] = trainlms_p(ni,A,B,M,max_br_iter);
```

Zadnji argument definira broj epoha prije osvježavanja težina i ukupan broj epoha. Prije pozivanja ove naredbe potrebno je matricu ulaza **A** proširiti s pragovima (-1). To možemo jednostavno učiniti naredbom:

```
A1 = [-ones(1,length(A)); A];
```

Odgovorite:

1. Skicirajte ravninu i položaj vektora u njoj te položaj klasifikacijske ravnine perceptrona prije i nakon treniranja.

2. Prikazati pogrešku (segmentacije) ovisno o indeksu iteracije.

3. Sami izmislite neki drugi linearno separabilni slučaj u ravnini te ponovite gornje korake za taj slučaj.

Za one koji žele znati više::

1. Sami izmislite neki drugi linearno separabilni slučaj u trodimenzionalnom prostoru te ponovite gornje korake za taj slučaj.

3.2.2. Slučaj linearno neseparabilnih uzoraka u 2D prostoru

U ovom primjeru pokušati će se istrenirati perceptron da separira dvije linearno neseparabilne klase. Konkretno, pokušati će ga se naučiti da riješi logičku XOR funkciju. Ulazni vektori **ai** će predstavljati argumente funkcije, a klase C_0 i C_1 vrijednost funkcije.

```
A = [0 0 1 1;  0 1 0 1];
C = [0 1 1 0];
```

Pozivom funkcije *plotpv* prikazati vektore **ai**.

Odgovorite:

1. Ponoviti postupak za treniranje iz prethodne točke.
2. Skicirati dobivene rezultate i tok grafa pogreške.
3. Je li perceptron naučio ispravno rješavati XOR problem?
4. Zašto? (objasnite prethodni odgovor)

3.2.3. Klasifikacija linearno separabilnih uzoraka u 3-D prostoru

Ovaj primjer demonstrira klasifikaciju uzoraka u 3-D prostoru. Ulazni vektori su trodimenzionalni i pripadaju u dvije klase koje su linearno separabilne. Ulazni vektori su:

```
a1 = [0  0  0]';
a2 = [0  0  1]';
a3 = [0  1  0]';
a4 = [0  1  1]';
a5 = [1  0  0]';
```

gdje vektori **a1**, **a3** i **a4** pripadaju klasi C_0, a ostali klasi C_1.

1. Ponoviti postupak učenja iz točke 3.2.1. te prikazati dobivene rezultate i graf promjene pogreške.
2. Rezultat se može i rotirati (prva ikona desno od ikone ruke u izborniku iznad slike).
3. Mjenjati pripadnosti vektora **ai** klasama C_0 i C_1 dok se ne dobije slučaj kad su klase C_0 i C_1 neseparabilne. Koji je to slučaj ?

3.3. Primjer klasifikacije uzoraka s Gaussovom razdiobom

Drugi dio vježbe predstavlja upotrebu perceptrona za klasifikaciju uzoraka s Gaussovom razdiobom, kakve često možemo naći u praksi.

Pretpostavimo da imamo dvije klase dvodimenzionalnih vektora koji predstavljaju realizaciju slučajnog vektora čije su komponente slučajne varijable s Gaussovom razdiobom. Uzmimo da prva klasa ima srednju vrijednost $E(C_0) = (10, 10)$ i stanardnu devijaciju $S(C_0) = 2.5$ za svaku od komponenti, dok druga klasa ima očekivanu vrijednost $E(C_1) = (20, 5)$ i standardnu devijaciju $S(C_1) = 2$. Konstruirati po sto vektora za svaku od klasa na sljedeći način:

```
A1 = [random('norm', 10, 2.5, 1,100); random('norm', 10, 2.5, 1,100)]
A2 = [random('norm', 20, 2.0, 1,100); random('norm', 5, 2.0, 1,100)]
```

Zatim konstruirati matricu **A** koja sadrži matrice $\mathbf{A_1}$ i $\mathbf{A_2}$.

```
A=[A1 A2]
```

Nakon toga formirati vektor **C** tako da prvih sto elemenata pripada klasi C_0, a ostalih sto klasi C_1.

```
C = [zeros(1,100) ones( 1,100)]
```

1. Ponoviti postupak treniranja iz prvog dijela vježbe. Skicirati dobivene rezultate?
2. Koliko je neispravno klasificiranih uzoraka ?
3. Ukoliko je ulazni vektor **ai** = (10, 3) kojoj klasi on pripada?
4. Da bi se dobio odziv mreže na vektor **a** (mora biti vektor stupac), koristi se funkcija *run_perc* na sljedeći način:

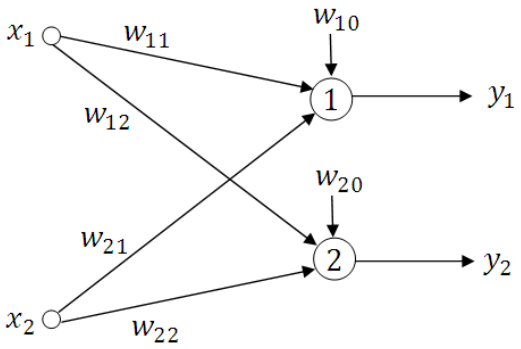
```
run_perc([-1, a], W)
```

3.4. Klasifikacija uzoraka pomoću dva perceptrona

Treći dio vježbe pokazuje mogućnosti korištenja više perceptrona za klasifikaciju ulaznih uzoraka u veći broj klasa. Na slici 4. prikazana je mreža s dva perceptrona kojima je moguće klasificirati uzorke u četiri linearno separabilne klase.

Pretpostavimo da imamo osam ulaznih dvodimenzionalnih vektora definiranih matricom **A**, gdje je svaki stupac matrice jedan ulazni vektor:

```
A = [ 0.1,  0.7,  0.8,  0.8,  1.0,  0.3,  0.0,  -0.3,  -0.5,  -1.5;
      1.2,  1.8,   1.6,  0.6,  0.8,  0.5,  0.2,  0.8,  -1.5,  -1.3]
```



Slika 4.: Dva perceptrona za klasifikaciju u četiri klase (izlazi su binarno kodirani).

Pripadnost ulaznih vektora klasama C_0, C_1, C_2, C_3 definirana je matricom **C** dimenzija 2×N, gdje je N=8 broj ulaznih vektora koje treba klasificirati:

```
C = [1 1 1 0 0 1 1 1 0 0; ...
     0 0 0 0 0 1 1 1 1 1]
```

Svaki stupac matrice **C** je dvodimenzionalni vektor čija dva elementa (dva bita) predstavljaju binarno kodiranu vrijednost klase za odgovarajući ulazni vektor (stupac matrice **A**). Pomoću dva bita možemo binarno kodirati četiri različite vrijednosti koje odgovaraju jednoj od četiri moguće klase: C_0, C_1, C_2, C_3. Npr. ako je izlaz prvog perceptrona jednak 0, a drugoga 1, ulazni vektor pripada klasi C_2 jer je $(10)_2 = 2$. Mreža s dva perceptrona se trenira na identičan način kao i ona s jednim.

1. Trenirati mrežu na opisan način te skicirati rezultate te kretanje pogreške kroz iteracije.

4. Vježba 4: LMS algoritam za predviđanje cijena dionice

4.1. Priprema za vježbu

Proučiti materijal "Neuronske mreže: Predavanja", poglavlje "Višeslojni perceptron". Prije početka rada sa stranice predmeta iz repozitorija *dodatni materijali* skinite komprimiranu datoteku "vjezba_4.zip" koja sadrži funkcije potrebne za izvođenje vježbe. Ove funkcije kopirajte u radni direktorij.

Odgovorite:

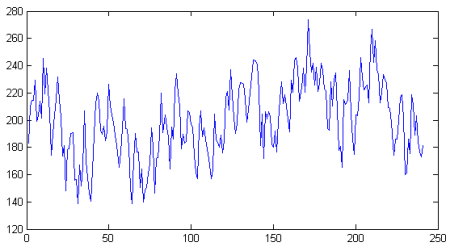
1. Što radi funkcija *trainlms*?

4.2. Kretanje cijene dionice

U ovom eksperimentu ćemo kako možemo korisiti LMS algoritam za predviđanje kretanja cijene neke dionice na burzi, nazovimo je XXX-R-A. Učitajte podatke iz datoteke xxx-r-a u varijablu **xxx** (ili učitajte xxx-r-a.mat datoteku). Brojevi (elementi vektora **xxx**) predstavljaju kretanje srednje dnevne cijene dionice XXX-R-A u razdoblju od gotovo godine dana. Iscratajte varijablu **xxx** naredbom

```
plot(xxx)
```

Trebali biste dobiti isti graf kao i na slici 5.



Slika 5.: Kretanje vrijednosti dionice XXX-R-A

Zadatak ove vježbe je da na temelju nekoliko (recimo **N**) prethodnih dana procijenimo cijenu dionice za današnji dan. To je korisno jer tada možemo dionicu kupiti ili prodati prije nego joj cijena naraste ili padne te na taj način povećati svoju dobit ili zaštititi svoje ulaganje. Također, moramo odabrati i parove ulaz-izlaz na kojem ćemo trenirati našu mrežu. Veličina tog skupa neka bude određena varijablom **i**. Ulazi će biti vektori **ai** posloženi u matricu **A**, a izlazi skalarne vrijednosti poredani u vektor **y**.

Zadaci:

1. Napišite funkciju *sjecanje* koja za određeni dan u godini (indeks vektora xxx) konstruira vektor-stupac **a** čiji su elementi kretanje cijene za **N** prethodnih dana (no ne i današnjeg).

2. Koristeći se funkcijom *sjecanje* napišite funkciju *pamti* koja za zadane ulaze **(xxx, dan, N, i)** konstruira matricu **A** kojoj je svaki stupac vektor **ai** od prethodnog dana. Matrica **A** dakle pamti **i** sjećanja koje ćemo koristiti za učenje (treniranje) neuronske mreže.

Savjet: Matricu **A** konstruirajte u *for* petlji, gdje vektore stupce jednostavno pridružujete naredbom

```
A = [ai,A];
```

gdje je **ai** izračunat kao u prethodnom zadatku

```
ai = sjecanje(xxx, dan-i,N);.
```

Konstruirajte matricu A naredbom:

```
A = pamti(xxx, 150, 100, 50);
```

Vektor željenih vrijednosti (izlaza) konstruiramo naredbom:

```
y = xxx(dan-i+1:dan);
```

Perceptron inicijaliziramo naredbom:

```
W = initp(A,y);
```

Matricu **A** proširimo vrijednostima pragova kao i ranije:

```
A = [-ones(1,length(A)); A];
```

Perceptron treniramo naredbom:

```
[W1, e] = trainlms(ni,A,y,W,max_br_iter);
```

Varijablu **ni** odredite eksperimentalno (koristite se znanjima iz prethodnih vježbi), a mrežu istrenirajte za različite vrijednosti **i, N, max_br_iter**.

Odgovorite:

1. Težine mreža pamte se u matrici **W1**. Zapamtite različite matrice težina (**W1**, **W2**...) ovisno o parametrima: **i** = 30, 50 ili 100; **N** = 20, 50 ili 80; **max_br_iter** = 10000, 50000 ili 500000. Interpretirajte rezultate. Koja mreža ima najmanju pogrešku? Zašto?

Koliko dobro mreža predviđa izlaz **y**, možemo vizualizirati naredbama:

```
plot(1:length(W1*A),W1*A,'b',1:length(y),y,'r')
```

gdje je plavom bojom iscrtana predviđena vrijednost izlaza, a crvenom stvarna vrijednost izlaza.

Ako ne koristimo nikakvu inteligenciju za predviđanje cijena dionice, već pretpostavimo da će ona sutra biti (gotovo) jednaka kao i danas, svoju pogrešku možemo izračunati kao:

```
a = xxx(dan-i:dan-1);
y = xxx(dan-i+1:dan);
err_oo = sum(abs(y -a));
```

dok pogrešku neuronske mreže možemo izračunati:

```
err_nn = sum(abs(y-W1*A1));
```

Ukoliko svaki dan trgujemo dionicama XXX-R-A na burzi, naša pogreška je mjerljiva u novcu. Trgujući dnevno jednom dionicom XXX-R-A, a primjenjujući neuronsku mrežu (umjesto ranije pretpostavke) potencijalnu zaradu možemo izraziti kao:

```
zarada = err_oo - err_nn;
```

5. Vježba 5: Višeslojni perceptron

5.1. Priprema za vježbu

Proučiti materijal "Neuronske mreže: Predavanja", poglavlje "Višeslojni perceptron". Prije početka rada utipkajte u Matlab ljusci *nntwarn off;*.
Prije početka rada iz repozitorija *dodatni materijali* sa stranice predmeta skinite komprimiranu datoteku "vjezba_5.zip" koja sadrži funkcije i slike potrebne za izvođenje vježbe. Ove funkcije i slike kopirajte u radni direktorij.

Odgovorite:

1. Što čine naredbe *newff()*, *train()* i *sim()*?
2. Što iscrtavamo naredbama *mesh()* i *contour()*?
3. Za što koristimo naredbe *fspecial()* i *imfilter()*?
4. Čemu služi naredba *reshape()*?

5.2. Višeslojni perceptron za klasifikaciju uzoraka

Prvi dio vježbe služi za demonstraciju temeljnih koncepata višeslojnog perceptrona pomoću jednostavnog i malog skupa ulaznih uzoraka.

5.2.1. XOR primjer

U prvom dijelu vježbe koristit će se višeslojni perceptron za rješavanje XOR problema.

Prisjetite se:

1. Je li perceptron s jednim neuronom mogao riješiti taj problem?

Formirati matricu **A** koja sadrži ulaze u mrežu i vektor **C** s željenim izlazima na sljedeći način :

```
A = [ 0 0 1 1;  0 1 0 1];
C = [ 0 1 1 0];
```

Kao aktivacijska funkcija mreže koristit će se sigmoidna funkcija (logsig):

$$f(x) = \frac{1}{1 + exp(-x)} \tag{1}$$

Prikazati graf sigmoidne funkcije na sljedeći način:

```
n = -10:0.1:10;
tmp = logsig(n);
plot(n,tmp);
```

Skicirati sigmoidnu funkciju. Nakon toga formirati mrežu s dva ulaza, jednim skrivenim slojem od pet neurona, te izlaznim slojem s jednim neuronom. To možete učiniti naredbom:

```
net = newff([ 0 1; 0 1], [5,1], {'logsig','logsig'},'traingd');
```

Ovdje matrica [**0 1; 0 1**] definira raspon vrijednosti za svaku (dvije) komponentu ulaznog vektora. Primijetite da se sličan rezultat mogle polučiti naredbom *minmax(A)*. Vektor [**5,1**] definira broj neurona u skrivenom i izlaznom sloju mreže, dok su **'logsig','logsig'** redom nelinerne karakteristike neurona u skrivenom i izlaznom sloju. Posljednji parametra postavlja funkciju za popagaciju pogreške kroz slojeve mreže.

Zadaci:

1. Zadati paremetre treniranja mreže:

   ```
   net.trainParam.show = 25;
   net.trainParam.lr = 1;
   net.trainParam.epochs = 10000;
   net.trainParam.goal = 0.02;
   ```

2. Trenirati mrežu pozivom funkcije *train()* na sljedeći način:

   ```
   [net,tr] = train(net,A,C);
   ```

3. Skicirati graf promjene pogreške.

4. Provjerite je li mreža ispravno naučila izlaze za zadane ulaze **A**. Savjet:

   ```
   sim(net,A)}
   ```

5. Kolika je pogreška mreže? Kolika je srednja kvadratna pogreška? Odgovara li to zadanim parametrima mreže?

6. Koliko je otprilike iteracija potrebno da bi mreža konvergirala k rješenju?

7. Provjeriti odziv mreže za sva četiri ulazna slučaja pozivom funkcije **sim(net,A)**.

8. Pogledati kako izgleda odziv za cijeli ulazni prostor na sljedeći način:

   ```
   [X Y] = meshgrid(-0.2 :.03: 1.2);
   Z = reshape(sim(net, [X(:), Y(:)]'), size(X) );
   mesh(X,Y,Z) % ili samo mesh(Z)
   ```

9. Da bi vidjeli kako hiperravnina opisana višeslojnim perceptronom dijeli ulazni prostor koristiti naredbe:

```
cs = contour (X,Y,Z);  % ili contour (Z);
clabel (cs);
```

10. Ako kao graničnu vrijednost izlaznog neurona koristimo vrijednost 0.5 skicirati podjelu ulaznog prostora.

11. Zaključite koliki je izlaz iz mreže ako je ulaz **a=[0.5 0.5]'** ? Na čemu možete temeljiti taj zaključak? Provjerite ispravnost tog zaključka simulirajući mrežu.

5.2.2. Proces učenja

U ovom eksperimentu ćemo pokušat odrediti ovisnost brzine učenja mreže o iznosu faktora učenja. Faktor učenja je u prošlom eksperimentu je iznosio 1.0 (vidi **net.trainParam.lr = 1**;).

Odgovorite:

1. Ponovo inicijalizirajte mrežu, te ju trenirajte s drugim vrijednostima faktora učenja dok ne dobijete najbržu konvergenciju. *Savjet*: Učinite to u *for* petlji. Pripazite da svaka mreža započinje proces učenja s istim vektorom težina.

2. Skicirajte graf ovisnosti brzine konvergencije o faktoru učenja.

3. Koliki je faktor učenja u slučaju najbrže konvergencije?

4. Koliko je iteracija potrebno da mreža konvergira?

5. Skicirati graf pogreške u odnosu na broju iteracija za taj slučaj.

U nastavku eksperimenta promatrati ovisnost brzine treniranja za različiti broj neurona u skrivenom sloju. **Zadatci:**

1. Ponovo inicijalizirati mrežu, no ovaj puta s nekim drugim, manjim brojem skrivenih neurona.

2. Ponoviti postupak treniranja.

3. Skicirajte graf ovisnosti brzine konvergencije o broju neurona u skrivenom sloju.

4. Koliki je minimalni broj neurona u skrivenom sloju potreban da bi mreža konvergirala ka rješenju?

5. Ispitati utjecaj oblika aktivacijske funkcije neurona na proces treniranja. *Savjet*: Ponoviti postupak treniranja iz prve točke uz razliku da se u izlaznom sloju umjesto sigmoidne koristi linearna funkcija: Provjerite paramtear mreže: **net.layers1.transferFcn** i zamjenite **'logsig'** s **'purelin'**. Ukoliko je učenje nestabilno smanjiti stopu učenja (npr. 0.1).

6. Komentirati razlike nastale promjenom aktivacijske funkcije.

7. Što se događa ako se u oba sloja koristi linearna funkcija ?

5.3. Segmentacija slike višeslojnim perceptronom

Ovaj dio eksperimenta objašnjava upotrebu višeslojnog perceptrona za segmentaciju slike koja sadrži objekt na pozadini slike. Segmentacija slike je proces u kojem se slika podijeli na više regija (segmenata) koji su uniformni u nekom smislu. U najjednostavnijem slučaju možemo imati dvije regije od kojih jedna odgovara objektu u slici koji želimo odrediti, a druga regija pozadini objekta. U tom slučaju segmentacija slike može se shvatiti kao problem klasifikacije piksela slike u jednu od dvije klase (objekt ili pozadina). Klasifikacija piksela obično se provodi na temelju same vrijednosti piksela i dodatnih značajki npr. izračunatih na temelju vrijednosti piksela u nekom susjedstvu. U ovom eksperimentu za klasifikaciju koristit ćemo vrijednost piksela slike i usrednjenu vrijednost u okolini tog piksela. U ovom eksperimentu srednja vrijednost u okolici nekog piksela računa se niskopropusnim filtriranjem originalne slike. Dakle neuronska mreža imat će dva ulaza (vrijednost piksela i srednja vrijednost u okolini piksela) i jedan izlaz (objekt ili pozadina).

5.3.1. Učenje mreže

Mrežu je potrebno trenirati da segmentira sliku na objekt i pozadinu. Najprije je potrebno učitati sliku koja sadrži objekt, te segmentiranu sliku koja služi za učenje mreže:

```
[slin, cmap1] = imread('slika1.bmp');
[slout, cmap2] = imread( 'segm1.bmp');
```

Prikazati ulaznu sliku i željenu segmentiranu sliku na sljedeći način:

```
figure;
colormap(cmap1);
image(slin);
figure;
colormap(cmap2);
image(slout);
```

Kao ulazi u neuronsku mrežu koristit će se po jedan piksel iz ulazne slike i korespondentni piksel niskopropusno filtrirane slike koja se dobiva na sljedeći način:

```
filt = fspecial('gaussian', 5, 5);
sllp = imfilter(slin, filt, 'replicate');
```

Potrebno je generirati neki broj (u ovom slučaju 200) slučajnih ulazno izlaznih parova jer bi bilo presporo trenirati mrežu pomoću svih piksela. Parovi se generiraju na sljedeći način:

```
[inN, outN] = getsmpl(200, slout, double(slin)/255, sllp/255);
```

Ulazne vrijednosti se dijele s 255 da bi ulaz u neuronsku mrežu bio skaliran na interval [0,1].

Odgovorite:

1. Generirati neuronsku mrežu s tri sloja (dva neurona u ulaznom sloju, pet u skrivenom i jednim izlazom) po uzoru na prethodni dio vježbe.

2. Trenirati neuronsku mrežu uz slijedeće parametre:

   ```
   net.trainParam.show = 25;
   net.trainParam.lr = 2;
   net.trainParam.epochs = 4500;
   net.trainParam.goal = 0.02;
   ```

   Napomena: Ukoliko mreža ne dostigne željenu pogrešku iteracije ponoviti inicijalizaciju i treniranje.

3. Nakon toga treba segmentirati sliku koristeći istreniranu mrežu. *Savjet*:

   ```
   izlaz = reshape(sim(net, [double(slin(:))/255, ...
   double(sllp(:))/255]'), size(slin));
   ```

4. Izlazne vrijednosti neuronske mreže bit će realni brojevi u intervalu [0, 1]. Da bi ove vrijednosti pretvorili u binarne vrijednosti primijenit ćemo prag. Ako je izlaz mreže veći od praga, piksel je segmentiran kao pozadina, a ako je manji, segmentiran je kao objekt. Kao početnu vrijednost praga uzeti 0.5. *Savjet*:

   ```
   izlaz_seg = (izlaz>0.5);
   ```

5. Prikazati rezultate segmentacije. Eksperimentirati s iznosom praga da bi se dobio dovoljno točan rezultat segmentacije. Savjet:

   ```
   image(izlaz_seg)
   colormap(cmap2)
   ```

6. Prikažite dobivenu sliku. Savršena segmentacija nije nužna jer je manje pogreške je moguće ukloniti u kasnijoj fazi segmentacije slike, na primjer nekom morfološkom operacijom.

7. Prikazati krivulju koja je definirana neuronskom mrežom.
   Napomena: To nam je varijabla **izlaz** koju možemo prikazati na isti način kao i u zadatku 8. iz 5.2.1. dijela vježbe.

8. Prikazati konture krivulje.

9. Skicirati granicu između klasa pozadine i objekta u ravnini.
   Napomena: Odaberite prag konture za koju ste postigli najbolju segmentaciju slike.

5.3.2. Testiranje mreže

U ovom dijelu eksperimenta mreža se testira pomoću slika koje nisu korištene za učenje mreže. Potrebno je upotrijebiti istreniranu mrežu za segmentaciju druge slike novčića, koja se zove **'slika2'**.

Zadaci:

1. Ponoviti postupak učitavanja i prikaza slike iz predhodne točke (pozivi funkcija *imread, colormap* te *image*).
2. Generirati niskopropusno filtriranu verziju ulazne slike na isti način kao i u prvom dijelu eksperimenta, jer je potrebno za ulaz u neuronsku mrežu. *Važno*: Proces učenja (treniranja) mreže se ne ponavlja!
3. Segmentirati novu sliku upotrebom funkcije za segmentiranje mreže kao i u prethodnom zadatku.
4. Koristiti isti prag za binarizaciju izlaza mreže kao i u prvom dijelu eksperimenta.
5. Prikazati dobivene rezultate.
6. Jesu li rezultati kvalitetniji ili slabiji nego u predhodnom slučaju?
7. Objasnite zašto.

6. Vježba 6: Radijalne mreže

6.1. Priprema za vježbu

Proučiti materijal "Neuronske mreže: Predavanja", poglavlje "Radijalne mreže".

Provjerite:

1. Funkcionalnost naredbi *radbas(), solverb(), sim()* i *newrb()*.
2. Što je izlaz funkcije *newrb()*, koje od njih koristi funkcija *sim()*?
3. Što čini naredba *clabel()* te što joj je ulaz, a što izlaz?
4. Funkciju *random()* i njezine ulazne i izlazne parametre.

U prvom dijelu vježbe koristit će se radijalna mreža za rješavanje XOR problema.

Prisjetite se:

1. Je li perceptron mogao riješiti taj problem?
2. Je li višeslojni perceptron mogao riješiti taj problem?

6.2. Radijalne funkcije

Radijalne funkcije su funkcije relane varijable čija vrijednost ovisi samo o udaljenosti od ishodišta. Pišemo dakle: $\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|)$ i kažemo da je svaka funkcija s ovim svojstvom radijalna funkcija, pri tom za normu $(||.||)$ obično uzimamo euklidsku udaljenost. Primjeri radijalnih funkcija su:

1. Eksponencijalna (Gaussova) $\phi(r) = e^{-(\varepsilon r)^2}$
2. Kvadratnog tipa $\phi(r) = \sqrt[n]{1 + (\varepsilon r)^m}$
3. Poliharmonijski krivuljar k-tog reda
   (a) $\phi(r) = r^k,\ k = 1, 3, 5, \ldots$
   (b) $\phi(r) = r^k \ln(r),\ k = 2, 4, 6, \ldots$

Gdje smo koristili notaciju $r = \|\mathbf{x} - \mathbf{x}_i\|$, a m, n, k i ε su proizvoljne konstante, čest odabir su npr. $n \in \{-2, 1, 2\}$, i $m = 2$, te $k \in \mathbb{N}$, a ε ovisi o gustoći uzoraka x_i.

Radijalne funkcije se tipično koriste za aproksimaciju funkcija, i to u sljedećem obliku:

$$y(\mathbf{x}) = \sum_{i=1}^{N} w_i \, \phi(\|\mathbf{x} - \mathbf{x}_i\|),$$

gdje w_i predstavlja težinski faktor, a N broj komponenti (radijalnih funkcija) kojima želimo aproksimirati $y(\mathbf{x})$.

Ako učenje doživljavamo kao aproksimaciju ili interpolaciju funkcije, odnosno težnju da na temelju malog skupa uzoraka (diskretnih podataka) naučimo opće pravilo u skupu podataka (neku funkciju), tada je primjenjivost radijalnih mreža očita. Osim za interpolaciju funkcije, radijalne mreže se često koriste i za klasifikaciju - gdje se klase (npr $\{c_1, c_2, \ldots c_n\}$) doživljavaju kado diskretne vrijednosti funkcije koju aproksimiramo ($c_1 = 1 \cdots c_n = n$) ili predviđanje (npr. vremenskih nizova).

6.3. Realizacija XOR logičke funkcije radijalnom mrežom

U ovoj laboratorijskoj vježbi kao radijalna funkcija neurona u skrivenom sloju koristit će se eksponencijalna funkcija gdje je uzeto $\varepsilon = 1$. Što znači da za svaku točku x_i imamo radijalnu funkciju oblika:

$$\phi_i(x) = exp(-\frac{||x - x_i||}{2\sigma^2}) \tag{2}$$

Gdje je iz izraza 2 vidljivo da je ishodište radijalne funkcije postavljeno u točku x_i. Prikazati funkciju neurona na sljedeći način, te ju skicirati:

```
n = -4:0.1:4;
a = radbas(n);
plot(n,a)
```

Formirati ulaze u mrežu (argumente XOR funkcije) u matricu **A**, a izlaze (vrijednost XOR) funkcije u vektor **C** na slijedeći način:

```
A = [ 0 0; 0 1; 1 0; 1 1]'
C = [ 0 1 1 0]
```

Nakon toga kreirati radijalnu (RBF) mrežu pozivom funkcije *newrb*:

```
net = newrb(A,C,0.02,0.4,2,2);
```

Ovdje je 0.02 je maksimalna dopuštena pogreška mreže, a 0.4 je parametar širine radijalnih funkcija, 2 je maksimalni broj neurona u radijalnoj mreži. Što je posljednji parametar? Provjeriti odziv na ulazne vektore pozivom funkcije *sim*:

```
sim(net,A)
```

Odgovorite:

1. Kolika je pogreška za pojedine ulaze (je li veća ili manja od dopuštene)?
2. Prikazati odziv za cijeli ulazni prostor. *Savjet:*

```
[x,y] = meshgrid(-0.4:0.05:1.4);
z = sim(net,[x(:),y(:)]');
z = reshape(z,size(x));
mesh(x,y,z)
```

Da bi vidjeli kako radijalna mreža dijeli ulazni prostor u dvije klase koristite naredbe:

```
cntr = contour(x,y,z);
clabel(cntr)
```

Zadatak:

1. Skicirati podjelu ulaznog prostora.
   Napomena: za granicu od npr. 0.5 pozvati funkciju s parametrom 0.5, tj.:

   ```
   contour(x,y,z,[0.5]));
   ```

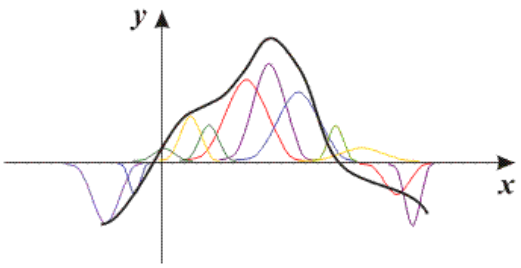
6.4. Interpolacija funkcije radijalnom mrežom

U drugom djelu vježbe radijalna mreža će se koristiti kao interpolator funkcije. Funkcija koja će se interpolirati bit će

$$x(k) = sinc(k) = \frac{sin(\pi * k)}{(\pi * k)}$$

Slika 6.. prikazuje rastav funkcije na više radijalnih funkcija.

Najprije definirati domenu [-10,10] i funkciju koja se interpolira, *x(k)*, na



Slika 6.: Interpolacija funkcije kao sume više radijalnih funkcija Gaussovog oblika

sljedeći način:

```
k = -10:0.2:10;
x = sinc(k);
```

Prikazati dobivene točke funkcije:

```
plot (k,x, '+')
```

Zadaci:

1. Trenirati mrežu pozivom funkcije *newrb()* dok se ne dobije zadovoljavajući odziv.

2. Prikazati graf pogreške.
   Napomena: Tražena naredba je (no ne nužno i prametri!!!):

   ```
   [net,tr] = newrb(k,x,2,0.1,100,10) ;
   ```

   Što je parametar **tr**? Parametar 100 u pozivu funkcije *newrb()* je maksimalni broj neurona. Koliki je stvaran broj korištenih neurona da se postigne željena pogreška (0.0002)?

3. Izračunati odziv te skicirati kretanje pogreške za sve ulazne točke na sljedeći način:

   ```
   y = sim(net,k);
   plot(k, x-y)
   ```

U nastavku vježbe ispitat će se sposobnost mreže da interpolira funkciju na temelju zašumljenog uzorka.

Zadaci:

1. Generirati zašumljeni signal (**xn**) dodajući (normalni) Gaussov šum na ulaznu funkciju koristeći funkciju *random()*. Srednja vrijednost šuma treba biti 0.0, a standardna devijacija 0.04.

2. Trenirati mrežu s tako zašumljenim uzorkom koristeći optimalnu širinu radijalnih funkcija (onu za koju je broj neurona u skrivenom sloju najmanji) iz prethodnog dijela vježbe.

3. Skicirati razliku između izlaza istrenirane mreže i zašumljenog ulaza te razliku između izlaza mreže i originalne funkcije.

Kod zašumljenog uzorka došlo je do tzv. "overfitinga", odnosno mreža je imala previše slobodnih parametara (težina skrivenih neurona) te je zapamtila i šum. Da bi se to izbjeglo koristit će se mreža sa manjim brojem neurona u skrivenom sloju. Ponovo trenirati mrežu pozivom funkcije *newrb* sa manjim brojem neurona u skrivenom sloju. To se postiže promjenom petog parametra funkcije *newrb* koje određuje koliki je maksimalan broj neurona dopušten u istreniranoj mreži.

Odgovorite:

1. Nacrtati graf s ovisnošću pogreške za ulaznu funkciju te za njenu zašumljenu verziju kao funkcije od broja neurona u skrivenom sloju za 3, 5, 10, 20, 30, te 50 neurona.
   Napomena: Suma apsolutne vrijednost pogreške se dobije na sljedeći način:

   ```
   sum ( abs(xn -y)) ili sum ( abs(x -y))
   ```

2. Koji je optimalan broj neurona u skrivenom sloju ?

7. Vježba 7: Rekurzivne mreže

7.1. Priprema za vježbu

Proučiti materijal "Neuronske mreže: Predavanja", poglavlje "Rekurzivne mreže". Sa stranice predmeta skinite datoteku "dodatni_materijali-7.vjezba.zip", raspakirajte je i spremite na svoje računalo u direktorij po izboru. Prije izvođenja eksperimenta potrebno je u programu MATLAB izvršiti naredbu koja dodaje direktorij s funkcijama potrebnim za vježbu u putanju:

```
path(path, '*put_do_direktorija*/dodatni_materijali')
```

Odgovorite:

1. Čemu služe funkcije *setbin()* i *getbin()*?
2. Što su stabilna stanja?
3. Što su lažna stanja?
4. Kako prepoznajemo lažna stanja mreže?
5. Koja lažna stanja unaprijed možemo prepoznati?

7.2. Hopfieldova mreža

U sljedećih nekoliko eksperimenata upoznat ćemo se s Hopfieldovom mrežom koja je jedan od najpoznatijih predstavnika rekurzivnih neuronskih mreža. U prvom eksperimentu koristit ćemo Hopfieldovu mrežu s dva neurona. Pretpostavimo da mreža treba zapamtiti dva vektora:

$$\xi_1 = [-1, 1]^T,\ \xi_2 = [1, -1]^T.$$

Zadaci:

1. Napisati funkciju *'learn_hop'* koja će izračunati matricu težina mreže **W** pomoću izraza:
$$W = \frac{1}{N}\sum_{m=1}^{p} \xi_m \xi_m^T - \frac{p}{N} I,$$
gdje je p=N=2.
2. Hopfieldova mreža s dva neurona može imati četiri različita stanja. Provjeriti jesu li stanja ξ_1 i ξ_2 stabilna te provjeriti stabilnost preostalih dvaju stanja. *Napomena*: Iteracija se računa kao:

```
sign(W*vektor_stanje);
```

Energetska funkcija Hopfieldove mreže u ovom slučaju (kad su pragovi neurona jednaki nuli) definirana je izrazom:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} w_{ji}s_i s_j$$

Odgovorite:

1. Napisati funkciju koja izračunava energiju hopfieldove mreže *'ene_hop'*.
2. Izračunati energije svih četiriju stanja te provjeriti koja stanja pripadaju najnižim energijama.

7.2.1. Hopfieldova mreža kao asocijativna memorija za riječi

U ovom eksperimentu Hopfieldova mreža će se koristiti kao autoasocijativna memorija za riječi. U fazi dohvata mreža će za nepotpunu riječ kao početno stanje rekonstruirati zapamćenu riječ. Svako slovo riječi bit će predstavljeno binarnom verzijom svoga ASCII koda (s tim da je umjesto binarne 0 korišten -1). Svaka od osam bitova ASCII koda bit će kodiran jednim neuronom. Na taj način je za pamćenje riječi od četiri slova potrebna Hopfieldova mreža s trideset i dva neurona. Za prebacivanje u binarni zapis i obratno koristit ćemo funkcije *setbin* i *getbin*.

7.2.2. Učenje Hopfieldove mreže

Za početak ćemo probati spremiti dvije riječi u Hopfieldovu mrežu.

```
s1 = 'stol';
s2 = 'iver';
```

Kreirajmo matricu gdje su zapamćene riječi stupci i izračunajmo vektore težina:

```
w = learn_hop([setbin(s1) setbin(s2)]);
```

Stabilna stanja ćemo ustvrditi tako da odsimuliramo mrežu nakon što je postavljena u neko od stanja **s1** ili **s2**. Rezultanto stanje bi trebao biti jednako početnom.

```
rez = sign(w*setbin(s1));
getbin(rez)
```

Odgovorite:

1. Provjerite isto i za stanje **s2**.

7.2.3. Dohvat netočne verzije zapamćene riječi

U nastavku vježbe rekonstruirati će se cijela riječ od jednog njenog dijela. Konkretno riječ 'stol' od početnog slova 's"što bi trebalo biti dovoljno budući da ta riječ jedina ima 's' na početku. Zadati ulaz u mrežu te željeni odziv koji će služiti za provjeru ispravnosti rezultata na sljedeći način:

```
st = 's...';
tocno='stol';
rez=sign(w*setbin(st));
```

Rezultat možemo ispisati ili usporediti sa željenim odzivom

```
getbin(rez)
sum(rez-setbin(tocno))
```

Pogledajte u kojem stanju će završiti mreža ako se krene od uzorka.

```
st = 'lsto';
rez = getbin(sign(w*setbin(st)));
```

Odgovorite:

1. Je li mreža završila u željenom stanju 'stol'?
2. Pogledajte energije početnih i završnih stanja:

   ```
   ene_hop(w,setbin(st))
   ene_hop(w,setbin(rez))
   ```

3. Što iz toga zaključujemo?

7.2.4. Lažna stanja

Hopfieldova mreža koja ima izlaze samo ± 1 ima svojstvo da pri pamćenju pojedine riječi ujedno se zapamti i njen komplement (+1 je zamijenjen sa -1 i obratno). Isprobajte to.

Provjerite:

1. Jesu li komplementi riječi **s1** i **s2** stabilna stanja.
2. Postoji li još lažnih stanja?

7.2.5. Kapacitet Hopfieldove mreže kao asocijativne memorije

U ovom dijelu eksperimenta demonstrira se ograničeni kapacitet Hopfieldove mreže.

Zadaci:

1. Povećati broj riječi koje mreža treba zapamtiti, tako da dodate određeni broj novih riječi.
2. Ponoviti učenje Hopfieldove mreže za novi skup riječi.
3. Provjeriti odgovara li svaka riječ stabilnom stanju mreže. *Napomena*: Stabilno stanje mreže je ono u kojem mreža ostaje nakon što je inicijalizirana na tu vrijednost.
4. Napisati u izvještaju koliki je eksperimentalno određen kapacitet (i za koje riječi) Hopfieldove mreže korištene u eksperimentu.

7.3. Hopfieldova mreža kao asocijativna memorija za slike

U ovom eksperimentu Hopfieldova mreža će se koristiti kao asocijativna memorija za slike.

Odgovorite:

1. Napisati skriptu koja dohvaća slike 'slika*.jpg'.
2. Naredbom *rgb2gray()* pretvoriti slike u sive slike.
3. Skalirajte slike na veličinu 70x49 piksela i pretvoriti u vektor.
4. Vektor binarizirajte. *Napomena*: binarizaciju radimo na 1 i -1, a ne 1 i 0. Radi jednostavnosti to možete učiniti naredbom:

   ```
   2*(v>=mean(v))-1
   ```

5. Dobivene vektore korititi za učenje Hopfieldove mreže. (Vidi zadatak 1. u poglavlju 7.2.)
6. Na isti način dohvatiti slike 'hop_slike*.tif'.
7. Uz pomoć njih i hopfiledove mreže rekonstruirajte originalne slike.
8. Za svaki ulaz provjerite je li izlaz zadovoljavajući.
9. Prikažite parove ulaz i izlaz (rekonstruirana slika).
10. Što zaključujete? Prokomentirajte rezultate.

11. Koje su energije ulaznih i izlaznog stanja?
12. Koliko je neurona potrebno za rekonstrukciju slika na opisani način?
13. Koliko je kapacitet korištene Hopfieldove mreže?

8. Vježba 8: Samoorganizirajuće mreže

8.1. Priprema za vježbu

U ovoj vježbi bit će predstavljene samoorganizirajuće mreže. Za pripremu treba proučiti materijal "Neuronske mreže: Predavanja", poglavlje "Samoorganizirajuće mreže".

8.2. Ulazni skup podataka

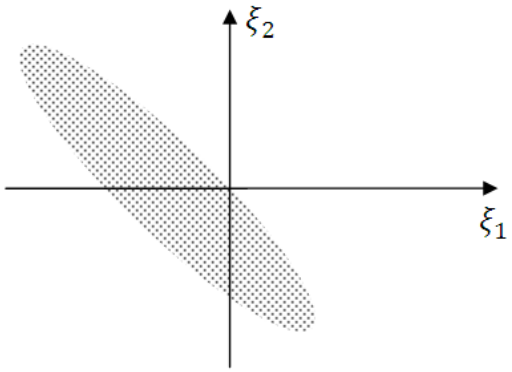
Na početku

```
path(path,'z:\nm\lab');
nntwarn off;
```

U ovom dijelu vježbe generirat ćemo skup slučajnih vektora koji će se koristiti kao ulazni podaci za eksperimente sa samoorganizirajućim mrežama u nastavku. Neka su z1 i z2 slučajne varijable s Gaussovim razdiobama s varijancama $\sigma_1^2 = 10$ i $\sigma_2^2 = 1$. Realizacije slučajnog vektora $z = [z_1,\ z_2]^T$ formirat će u ravnini (z1,z2) nakupinu točaka u obliku horizontalno izdužene elipse. Da bi dobili slučajni vektor $\xi = [\xi_1,\ \xi_2]^T$ distribuiran kao na slici 7., koristit ćemo sljedeći izraz da bi rotirali točke u ravnini (z_1, z_2):

$$\begin{bmatrix} \xi_1 \\ \xi_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

Na ovaj način smo rotirali nakupinu vektora **z** za 45 stupnjeva u smjeru kazaljke na satu.



Slika 7.: Distribucija generiranih slučajnih vektora u ravnini

Konstruirati ulazni skup od 80 vektora $[z_1 z_2]$ te ih prikazati u ravnini na sljedeći način:

```
Z = [random('norm', 0, 10, 1, 80); ...
     random('norm', 0, 1, 1, 80)];
for i=1:80
          E(:,i)=[1 1;-1 1]*[Z(1,i) Z(2,i)]';
end;
plot(E(1,:),E(2,:),'o');
axis([-10 10 -10 10]);
```

Naredba *axis* odreduje koje dio grafa će biti prikazan na ekranu. Ako se to ne učini Matlab ce sam skalirati x i y os na nejednolik način kako bi većinu slike popunio zadanim točkama.

8.3. Hebbov zakon učenja bez nadzora

Za neuronsku mrežu s jednim linearnim neuronom koristiti skup generiranih vektora i Hebbov zakon učenja bez nadzora:

$$\Delta w_i = \eta v \xi_i,$$

gdje je v aktivacija mreže, a ξ_i je i-ta komponenta ulaznog vektora. Aktivacija neurona v definirana je izrazom:

$$v = \sum_{j=1}^{N} w_j \xi_j = w^T \xi = \xi^T w,$$

gdje je w vektor težina neurona.

```
wout = neuropca([0.1 {0.2],E,0.001,'hebb');
```

Početna težina vektora je ovdje nasumce odabrana na [0.1 -0.2], 0.001 je faktor učenja. Prikazati grafom promjenu iznosa vektora težina u ravini (w_1, w_2) te odgovoriti da li vrijednosti težina konvergiraju.

```
plot(wout(1,1),wout(1,2),'bo',wout(:,1),wout(:,2),wout(80,1),wout(80,2),'rx');
```

Plavi O označava početnu vrijednost a crveni X krajnju.

8.4. Ojin zakon učenja bez nadzora

Ponoviti prethodni eksperiment koristeći Ojin zakon učenja bez nadzora definiran izrazom:

$$\Delta w_i = \eta v(\xi_i - v w_i).$$

Koristiti funkciju *neuropca*, ali umjesto 'hebb' staviti 'oja'. Uočite da će rezultantni vektor težina pokazivati u smjeru najveće varijance ulaznih podataka i da mu je veličina jednaka 1. Prikazati grafom promjenu iznosa vektora težina u ravnini (w_1, w_2).

8.5. Kompetitivno učenje za grupiranje vektora

U ovom eksperimentu koristit će se samoorganizirajuća mreža s kompetetivnim učenjem za grupiranje vektora (engl. clustering). U eksperimentu ćemo koristiti drugačije dvodimenzionalne vektore nego u prijašnjim vježbama. Raspodijeliti ćemo ih na 8 grupa i uz pomoć neuronske mreže pokušati naći centre.

Inicijalizirati te trenirati samoorganizirajuću mrežu s kompetitivnim učenjem na sljedeći način:

```
p = nngenc( [0 1; 0 1] , 8, 10, 0.05);
```

Argumenti funkcije *nngenc* su: matrica koja definira granice uzoraka, broj clustera, broj uzoraka po clusteru i na kraju standardna devijacija clustera.

```
plot(p(1,:),p(2,:),'+r');
w = initc(p,8);
```

Funkcija *initc* inicijalizira vektore težina za 8 neurona u ovom slučaju.

```
w = traincomp(w,p,[20 5000 0.01]);
```

Na rezultantnoj slici plavim kružićima su označeni centri grupa kojima je mreža konvergirala. Napredovanje plavih kružića je moguće pratiti i za vrijeme treniranja mreže. U ovom primjeru se koristi 5000 epocha, s konstantom učenja od 0.01 i to možete ručno mijenjati.

Nakon što je faza učenja završena, provjeriti u koju klasu spada ulazni vektor [0.6, 0.4] na sljedeći način:

```
p = [0.6; 0.4];
simuc(p,w)
```

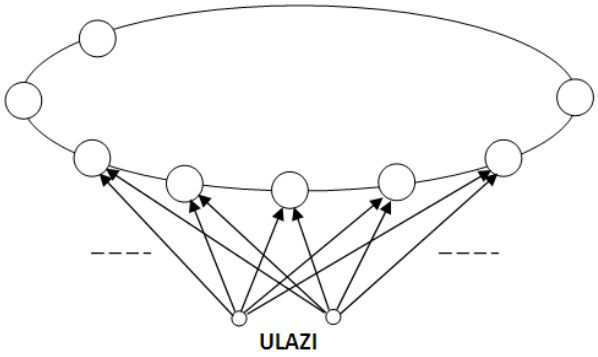
Pogledajte pripadnost još nekih drugih vektora.

8.6. Problem trgovačkog putnika

U ovom eksperimentu potrebno je riješiti optimizacijski problem trgovačkog putnika. Ako trgovac mora obići nekoliko gradova s tim da se u svakom zadrži samo jednom te se na kraju vrati u početni grad, treba naći najkraći put kojim on to može učiniti.

Za rješenje problema koristit ćemo Kohonenovu neuronsku mrežu gdje je jednodimenzionalni niz neurona topološki organiziran u obliku prstena, prema slici 8.. Mreža ima dva ulaza na koje se dovode koordinate gradova.

Broj neurona u mreži jednak je dva puta broj gradova. Na ulaz mreže se naizmjence postavljaju koordinate gradova i taj se proces iterativno ponavlja. Onaj neuron, čiji vektor težina je najbliži koordinatama grada koji je trenutno



Slika 8.: Kohonenova mreža za rješenje problema trgovačkog putnika.

na ulazu je pobjednik u kompetitivnom učenju. Vektor težina pobjedničkog neurona mijenja se u smjeru vektora koordinata grada. Prema Kohonenovom principu, osim pobjedničkog neurona vektore težina mijenjaju i susjedni neuroni, ali s manjim iznosom. Najjače se modificiraju vektori težina najbližih susjeda, a najslabije najdaljih. Promjena iznosa težina susjednih neurona definirana je Gaussovom funkcijom. Rezultat je da će se neuroni u prostoru težina pomicati prema koordinatama gradova. Neurona ima dva puta više nego gradova zbog toga da se višak neurona postavi između dva susjedna grada, u protivnom bi jedan neuron mogao oscilirati između ta dva grada.

Sam algoritam ne garantira da je dobiveni rezultat optimalan tj. moguće je da rezultat predstavlja lokalni a ne globalni minimum. Rezultat ovisi i o početnim vrijednostima težina neurona.

Najprije zadati slučajne koordinate gradova te ih prikazati kako slijedi:

```
t = abs(rands(12,2));
plot(t(:,1), t(:,2), '*b');
```

Zatim generirati Kohonenovu mrežu koja rješava problem trgovačkog putnika:

```
[w, len] = kohtsm(t,300,3,2,2);
```

U ovom slučaju **len** je dobivena optimalna duljina putanje. Parametri funkcije *kohtsm* su redom: ulazni skup gradova, broj iteracija, faktor učenja a zadnji parametar određuje širinu susjedstva Kohonenove mreže. Ponoviti treniranje mreže na istim gradovima za širine 0.1, 0.5, 1, te 6. Nacrtati ovisnost duljine putanje o širini susjedstva.

Eksperiment se može ponoviti i za veći broj gradova, npr. 300.

9. Vježba 9: Genetički algoritmi

9.1. Priprema za vježbu

Proučiti materijal "Neuronske mreže: Predavanja", poglavlje "Genetički algoritmi".

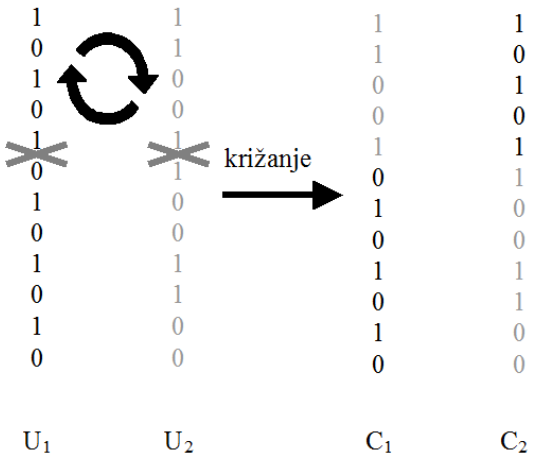
9.2. Traženje maksimuma pomoću GA

Prvi dio vježbe služi za demonstraciju temeljnih koncepata genetičkih algoritama (GA). GA je jedan stohastički optimizacijski algoritam koji simulira proces prirodne selekcije i reprodukcije u prirodi. Potencijalna rješenja optimizacijskog problema kod GA se zovu jedinke. Rješenja mogu biti kodirana na razne načine zavisno od vrste optimizacijskog prostora. U sljedećim eksperimentima bit će prikazano binarno kodiranje i kodiranje pomoću decimalnih brojeva. Nakon pokretanja programa MATLAB potrebno je utipkati sljedeću naredbu kako bi se u putanju dodao direktorij koji sadrži dodatne funkcije potrebne za izvođenje vježbe.

```
path(path,'Z:\NM\LAB\') ;
```

9.2.1. GA s binarnim kodiranjem jedinki

U ovom dijelu vježbe bit će demonstrirani osnovni princip i ideja genetičkih algoritama gdje je svaka jedinka predstavljena nizom bitova. Za križanje koristi se funkcija *'simpleXover'* koja uzima dvije jedinke, dijeli ih na dva dijela prekidanjem na slučajno odabranoj poziciji te stvara dvije nove jedinke zamjenom suprotnih dijelova kao što je prikazano na slici 9.. Za mutaciju se koristi



Slika 9.: Ilustracija operacije križanja

funkcija *'binaryMutation'* koja mijenja bit ulazne jedinke na slučajno odabranoj poziciji. Za selekciju jedinki koje prelaze u sljedeću generaciju koristi se

funkcija *'roulette'* koja vrši proporcionalnu selekciju jedinki. Unijeti naredbu za iscrtavanje funkcije čiji maksimum tražimo na intervalu [0 9]:

```
fplot('x + 10*sin(5*x)+7*cos(4*x)',[0 9])
hold on
```

Slijedi inicijalizacija jedinki (kromosoma) populacije te iscrtavanje položaja svake jedinke na zadanoj funkciji. *'evalFn'* je ime funkcije koja služi za izračunavanje prikladnosti odgovarajuće jedinke.

```
initPop = initializega(10,[0 9],'evalFn',[]);
```

Kreirali smo populaciju od 10 jedinki na intervalu [0..9]. Iscrtajmo ih.

```
plot (initPop(:,1),initPop(:,2),'r+')
```

Jedinke je potrebno pretvoriti u binarni oblik tako da je svaki decimalni broj prikazan s 24-bitovnom preciznošću:

```
initPop = [f2b(initPop(:,1),[0 9],24) initPop(:,2)];
```

Da bi pokazali kako radi križanje izdvojimo dvije jedinke i provedimo ih kroz *simpleXover* algoritam.

```
jedinka1=initPop(1,1:24)
jedinka2=initPop(2,1:24)
[njedinka1 njedinka2]=simpleXover(jedinka1,jedinka2);
[jedinka1' jedinka2' njedinka1' njedinka2'] %za usporedni prikaz
```

Slijedi funkcija za izvršavanje genetičkih algoritama nad početnom populacijom. *'maxGenTerm'* je ime funkcije koja definira uvjete za prestanak rada genetičkog algoritma, što je u ovom slučaju maksimalan broj generacija **50**. *'binaryMutation'* prima kao parametar vjerojatnost mutacije što je ovdje postavljeno na **0.02**. Vjerojatnost pojave križanja je **0.6**. Funkcija *'ga'* tijekom rada ispisuje redni broj generacije i najveću prikladnost ukoliko je ona veća od prikladnosti u prethodnim generacijama.

```
[x endPop bpop trace] =
ga([0 9],'evalFn',[],initPop,[1e-6 0 1],'maxGenTerm',[50],'roulette',[],
'simpleXover',[0.6],'binaryMutation',[0.02]);
```

U gornjoj naredbi **x** je jedinka sa najboljom prikladnošću u svim generacijama, **bpop** je trace najbolje jedinke, **endPop** je završna populacija koju je potrebno pretvoriti u decimalni oblik te iscrtati pozicije tih jedinki na zadanoj funkciji.

```
endPop=[b2f(endPop,[0 9],24) endPop(:,size(endPop,2))];
plot (endPop(:,1),endPop(:,2),'b*', x(1), x(2),'ro')
hold off
```

Sljedećim naredbama prikazuje se najveća (crveno) i srednja (plavo) prikladnost kroz sve generacije.

```
plot(trace(:,1),trace(:,3),'b-', trace(:,1),trace(:,2),'r-')
```

Skicirati i objasniti dobivene rezultate. Ukoliko nije pronađen globalni maksimum ponoviti izvođenje funkcije *'ga'*.

9.2.2. GA sa decimalnim kodiranjem jedinki

U ovom eksperimentu bit će demonstriran GA kod kojeg su jedinke predstavljene pomoću decimalnih brojeva. Za križanje koristi se funkcija *'arithXover'* koja iz dvije ulazne jedinke stvara dvije izlazne jedinke njihovom linearnom kombinacijom. Za mutaciju se koristi funkcija *'unifMutation'* koja mijenja vrijednost slučajno odabranog elementa jedinke (jednog decimalnog broja) uz uvjet da novi element bude unutar određenog intervala vrijednosti.

Kao i u prethodnom dijelu potrebno je iscrtati zadanu funkciju, inicijalizirati jedinke početne generacije te iscrtati njihov položaj na zadanoj funkciji.

```
fplot('x + 10*sin(5*x)+7*cos(4*x)',[0 9])
hold on
initPop=initializega(10,[0 9],'evalFn',[]);
plot (initPop(:,1),initPop(:,2),'r+')
```

Zatim se pokreće funkcija za izvršavanje GA.

```
[x endPop bpop trace] = ga([0 9],'evalFn',[],initPop,[1e-6 1 1],...
                           'maxGenTerm',50, 'roulette',[],...
                           ['arithXover'],[6], 'unifMutation',[2]);
```

Parametar iza imena funkcije *'arithXover'* odnosi se na broj križanja u jednoj generaciji, a broj iza *'unifMutation'* na broj mutacija u jednoj generaciji. Iscrtati završnu generaciju, najbolju jedinku te maksimalnu i srednju prikladnost kroz generacijski ciklus kao i u prethodnom dijelu vježbe.

9.3. Treniranje višeslojne neuronske mreže pomoću GA

U ovom eksperimentu bit će demonstrirana upotreba GA za učenje višeslojne neuronske mreže za rješavanje XOR problema. Učenje pod nadzorom se provodi minimizacijom srednje kvadratne pogreške između dobivenog i željenog odziva mreže. Neuronska mreža korištena u primjeru ima dva ulaza, dva neurona u skrivenom sloju te jedan neuron u izlaznom sloju. Aktivacijske funkcije neurona su sigmoidne. Neuroni u skrivenom sloju imaju jednake težine na oba ulaza tako da se svaka jedinka populacije sastoji od četiri decimalna broja (dva za težine skrivenog sloja i dva za težine neurona u izlaznom sloju). Također će biti demonstrirani i utjecaji promjena osnovnih parametara GA.
Najprije je potrebno definirati raspon svih težina i parove ulaz-izlaz te stvoriti početnu populaciju.

```
rang = ones(4,1)*[-4 4]; % raspon težina
ioPairs = [0 0 0; 0 1 1; 1 0 1; 1 1 0];
initPop = initializega(100,rang,'xorEval',ioPairs);
```

Funkcija *'xorEval'* određuje prikladnost svake jedinke koja se određuje kao negativna vrijednost zbroja kvadrata pogrešaka za dane parove ulaz-izlaz. Pokrenuti izvršavanje GA i obratiti pažnju na promjenu najbolje prikladnosti kroz generacije.

```
[x endPop bpop trace] =
ga(rang,'xorEval',ioPairs,initPop,[1e-6 1 1],'maxGenTerm',[50],
'roulette',[],'simpleXover', [70], 'unifMutation',[5]);
```

Provjeriti odziv mreže te iscrtati promjenu prikladnosti kroz generacije.

```
[a,b,o] = xorEval(x,ioPairs);
plot(trace(:,1),trace(:,3),'b-', trace(:,1),trace(:,2),'r-')
```

Varijabla **o** predstavlja odziv mreže za dane ulazno-izlazne parove. Ukoliko mreža nije dobro naučila parove ulaz-izlaz ponoviti dvije prethodne funkcije.

Skicirati podjelu ulaznog prostora neuronske mreže koja se prikazuje sljedećim naredbama:

```
[i j] = meshgrid(0:0.05:1, 0:0.05:1);
q = [reshape(i,prod(size(i)),1) ...
     reshape(j, prod(size(j)),1) ...
     zeros(prod(size(i)),1)];
[a,b,o] = xorEval(x,q);
z = reshape(o,size(i,1),size(i,2));
cs = contour (i,j,z);
clabel (cs);
```

U nastavku ovog eksperimenta treba ispitati utjecaj vjerojatnosti križanja i mutacije. Uz konstantan broj mutacija potrebno je ispitati utjecaj promjene broja križanja za vrijednosti **90** i **20**. Isto tako uz konstantan broj križanja (**5**) potrebno je ispitati sustav za brojeve mutacija **15** i **50**. Za svaki od gore navedenih slučajeva učenje mreže ponoviti nekoliko puta. Objasniti i zapisati rezultate.

9.4. Rješavanje problema putujućeg trgovca pomoću GA

Problem putujućeg trgovca (engl. travelling salesman problem) sastoji se u odabiru najkraćeg puta za obilazak N gradova i povratak u početni grad. Za svaki grad definirana je njegova lokacija tako da je moguće izračunati udaljenosti među gradovima. Ovaj problem spada u grupu teških problema i ima NP (engl. non-polynomial) složenost.

Zbog posebnosti problema putujućeg trgovca potrebno je koristiti posebne funkcije za križanje i mutaciju, takve da u rezultirajućim jedinkama nema ponavljanja gradova. U ovom eksperimentu križanje se obavlja tako se od prve ulazne jedinke uzme prvi grad te se bira sljedeći između gradova na drugoj poziciji obje ulazne jedinke. Odabire se onaj koji je manje udaljen od početnog grada. postupak se nastavlja za sljedeće gradove, a ukoliko se obadva ponuđena grada već nalaze u jedinki, slučajno se odabire jedan od preostalih gradova. Na isti način se sastavlja i druga izlazna jedinka koja za prvi grad uzima prvi

grad druge ulazne jedinke. Naravno, osim navedenog mogući su i alternativni načini križanja. Mutacija se vrši tako da dva slučajno izabrana grada zamjene mjesta.

Potrebno je stvoriti matricu koordinata gradova te početnu populaciju "pokazatelja" na gradove. Gradova će biti **10**, populacija ima **100** jedinki, a bit će **100** generacija. Raspon **x** i **y** koordinata je [**0, 100**].

```
citNum = 10;                        % broj gradova
rang = ones(citNum,1)*[0 100];      % raspon koordinata za sve gradove
global cit
cit = rand(citNum,2)*100;           % slucajne koordinate gradova
[a,b]=sort(rand(citNum-1,100));  % b drzi 100 različitih lista od 9 gradova
b = [(b+1)' zeros(size(b,2),1)]; % brojevi gradova su od 2 do 10
[initPop initPop(:,citNum)] = distEval(b,cit);
                                    % populacija od 100 jedinki
plot(cit(:,1),cit(:,2),'r.')        % iscrtavanje pozicije gradova
hold on
```

'distEval' je funkcija za određivanje prikladnosti. U ovom slučaju to je negativan iznos ukupnog puta. Tako da se traži maximum. Populaciju sačinjavaju pokazivači na sve gradove osim prvog koji ostaje fiksan što pojednostavljuje proračunavanje.

Pokrenuti izvršavanje funkcije *ga*:

```
[x endPop bpop trace] =
ga(rang,'distEval',cit,initPop,[1e-6 1 1], 'maxGenTerm',100,
'normGeomSelect',[0.04],'tsmXover',[80], 'swapMutation',[10]);
```

Ovdje se koristi funkcija *'normGeomSelect'* za selekciju jedinki, a **0.04** je parametar koji određuje razdiobu vjerojatnosti selektiranja. Naime *'roulette'* selekcija ne podržava mogućnost negativnih funkcija prikladnosti niti mogućnost traženja minimuma.

Iscrtavanje optimalnog puta:

```
y = [1 x(:,1:citNum-1) 1];
plot(cit(y,1),cit(y,2));
hold off
```

Skicirati rezultat kao i promjenu prikladnosti kroz generacije. U nastavku vježbe treba ispitati uspješnost GA za rješavanje problema putujućeg trgovca za veći broj gradova. Ponoviti gornji postupak za **12**, **14** i **16** gradova te zapisati i objasniti dobivene rezultate.