Optičko prepoznavanje karaktera

Hopfildova neuralna mreža

Ognjen Zelenbabić

Definicija problema

Problem: Isprojektovati neuralnu mrežu da se ponaša kao asocijativna memorija. Generisati skup binarnih slika, obučiti mrežu da asocira na njih i izvršiti analizu rada mreže u uslovima postojanja različitih tipova šuma.

1. Veštačke neuralne mreže

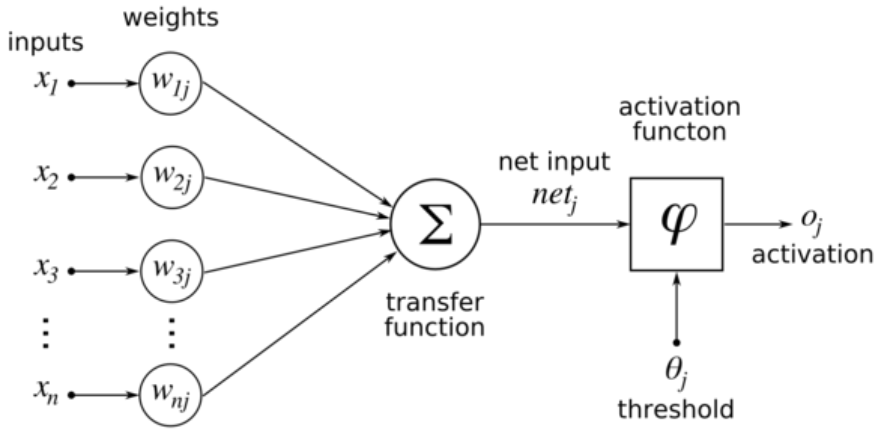
Veštačke neuralne mreže (ANNs – Artificial Neural Networks) su sistemi koji su konstruisani tako da koriste organizacione principe koji podsećaju na principe ljudskog mozga. ANNs su dobre za zadatke kao što su uparivanje obrazaca, klasifikaciju, aproksimaciju funkcija, optimizaciju, kvantizaciju vektora i za klasterizaciju podataka. One imaju veliki broj veoma povezanih procesirajućih elemenata (čvorova ili jedinica) koji obično rade paralelno.

Kolektivno ponašanje ANNs-a, kao i ljudski mozak, pokazuje mogućnost učenja, prisećanja i generalizacije iz obrazaca za treniranje ili podataka. Inspirisane su modelom mreža neurona u ljudskom mozgu pa se zbog toga procesirajući elementi nazivaju veštački neuroni (ili samo neuroni).

Kao što su neuroni u ljudskom mozgu sačinjeni iz tri celine (telo ćelije, dendriti i akson) tako je i veštački neuron simuliran na sličan način. Kod simuliranja veštačkog neurona definišemo tri različite kategorije:

1. Procesirajući element (telo neurona),
2. Konekcije neurona (dendriti),
3. Pravila obučavanja neurona.

Matematički model neurona je prikazan na sledećoj slici (slika 1).



Slika 1: Matematički model neurona.

Kao što se vidi sa slike 1, svaki neuron ima dve funkcije koje ga opisuju, pored ulaznih i izlaznih vektora. Te dve funkcije su:

1. Integralna funkcija,
2. Aktivaciona funkcija.

Integralne funkcije mogu biti različitog tipa, kao na primer:

1. Linearna integralna funkcija,
2. Kvadratna integralna funkcija,
3. Sferna integralna funkcija,
4. Polinomska integralna funkcija.

Ja sam za rešavanje svog problema odabrao najjednostavniju, linearnu, integralnu funkciju koja je data sledećim izrazom:

$$f_j \square net_j = \sum_{i=1}^{n} w_{ij} x_i - \theta_j .$$

Indeksi u datom izrazu odgovaraju prikazanoj slici.

Aktivaciona funkcija je funkcija koju koristimo da na izlaz izvedemo izlaznu informaciju dobijenu kao funkciju mrežnih ulaza.

Neke uobičajene aktivacione funkcije su:

1. Step funkcija,
2. Funkcija praga,
3. Funkcija rampe,
4. Unipolarna sigmoidna funkcija,
5. Bipolarna sigmoidna funkcija.

Za rešavanje svog problema sam odabrao funkciju praga kao aktivacionu funkciju koja je data sledećim izrazom:

$$o_j = \text{sgn}(f_j) = 1, \text{ ako je } f_j \geq 0$$
$$o_j = \text{sgn}(f_j) = -1, \text{ inače.}$$

2. Asocijativne memorije

Asocijativne memorije su tip neuralni mreža koje asociraju ulazne vektore sa izlaznim vektorima. Ovaj postupak se naziva obučavanje (treniranje) neuralne mreže. Obučavanje neuralne mreže je omogućio Hebb koji je napisao algoritm skladištenja:

$$w_{ij} = \sum_{k=1}^{p} x_i^k x_j^k ,$$

gde je sa w_{ij} označena matrica težina konekcija, k je broj ulaznih vektora (k=1,2,...,p) a x_i^k predstavlja i-ti ulaz k-tog vektora.

U svom programu sam pored Hebovog pravila obučavanja koristio pravilo evolucije za svaki čvor:

$$y_i^{(k+1)} = \text{sgn}\left(\sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} y_j^{(k)} + x_i - \theta_i \right).$$

Hebovo pravilo obučavanja sam koristio za memorisanje binarnih slika koje sam doveo na ulaz mreže i to tako što sam napravio matricu težina koja mi je predstavljala memoriju mreže. Drugo pravilo (pravilo evolucije) sam koristio za dobijanje izlaznih informacija kada bi na ulaz mreže doveo neku narušenu sliku (sliku sa velikim uticajem šuma).

Potrebno je napomenuti da sam svoj napor fokusirao na realizovanje diskretne rekurzivne autoasocijativne memorije (Hopfildove memorije).

3. Pristup rešavanju problema

Za početak je bilo potrebno da generišem skup nekoliko binarnih slika koje ću koristiti za obučavanje mreže. Prvobitno sam počeo da eksperimentišem na malom skupu slika malih dimenzije. Koristio sam četiri slike koje su bile kodirane sa 10 bita.

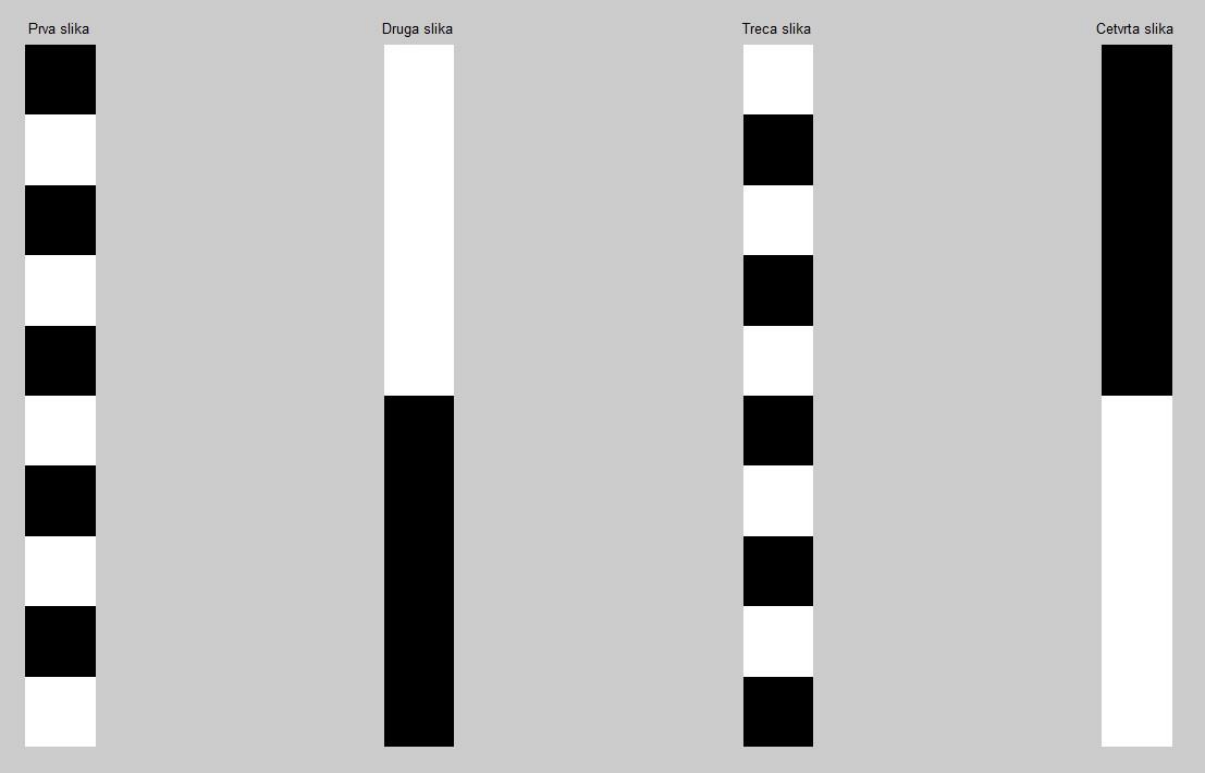
Iako sam algoritam počeo pisati za slike malih dimenzija napisao sam ga tako da je na ulaz projektovane mreže moguće dovesti slike bilo kojih dimenzija. Jedino o čemu se treba voditi računa je kapacitet Hopfildove memorije.

Pošto definisanje slika preko nula i jedinica može biti poprilično naporno, posle prvobitnog skupa malog broja slika koje su bile unapred definisane u mom algoritmu, uveo sam opciju biranja monohromatskih slika preko GUI-ja (Graphical User Interface). Ovo mi je dosta olakšalo testiranje mreže na različite tipove slika. Sada sam mogao napraviti slike u bitmap formatu i uvoziti ih u moj program.

Slike malih dimenzija koje sam koristio u početku sam definisao na sledeći način:

```
% Prva slika
slika_1=[-1;  1; -1;  1; -1;  1; -1;  1; -1;  1];
% Druga slika
slika_2=[ 1;  1;  1;  1;  1; -1; -1; -1; -1; -1];
% Treca slika
slika_3=[ 1; -1;  1; -1;  1; -1;  1; -1;  1; -1];
% Cetvrta slika
slika_4=[-1; -1; -1; -1; -1;  1;  1;  1;  1;  1];
```

Grafički prikaz dobijenih slika je prikazan na sledećoj slici (slika 2).



Slika 2: Grafički prikaz slika malih dimenzija.

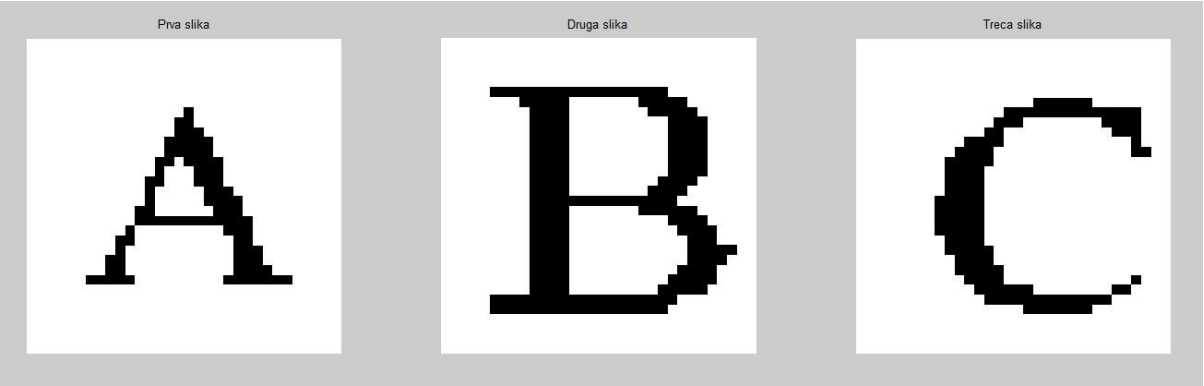
Algoritam koji sam koristio za pozivanje slika iz nekog direktorijuma sa hard diska preko GUI-ja je sledeći:

```
%% Ucitavam prvu sliku
n=32; % Ogranicavam dimenzije slike na 32x32
[filename,pathname]=uigetfile; % Biram sliku
ulazna_slika_1=imread(filename); % Usnimavam sliku u zeljenu promenljivu
ulazna_slika_1=imresize(ulazna_slika_1,[n n]); % Prilagodjavam dimenzije slike
```

Kao što možemo primetiti, u algoritmu sam ograničio slike na matrice dimenzije 32x32. U početku sam na ulaz mreže dovodio tri slike ovih dimenzija ali sam naišao na interesantan problem.

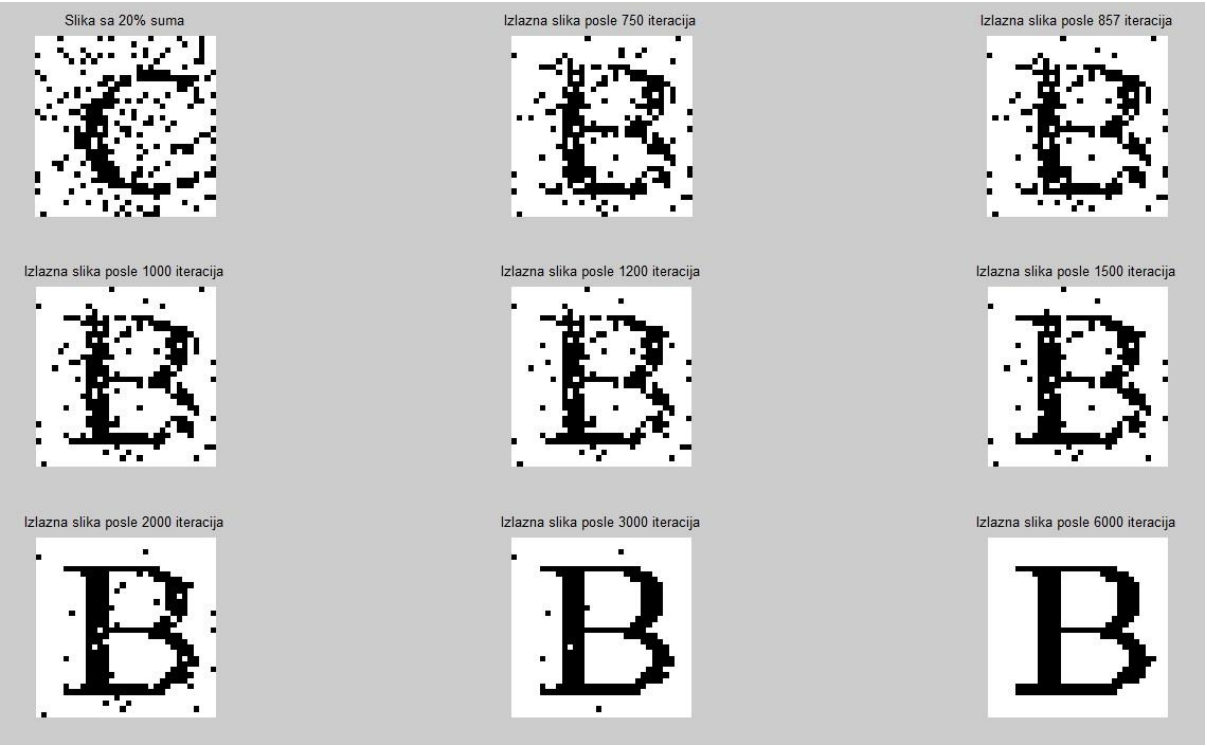
Jasno mi je bilo da kapacitet Hopfildove mreže dosta varira u zavisnosti od broja neurona koji se koriste. Takođe, u Hopfildovoj mreži je broj ulaznih vektora identičan broju neurona. Analizirajući kapacitet mreže empirijski, shvatio sam da zavisnost memorije od broja neurona nije linearna, odnosno veći broj ulaznih vektora ne uslovljava uvek i veću memoriju. A to je upravo ono što se meni desilo. Kada sam izabrao slike dimenzija 32x32, tj. slike su kodirane sa 1024 bita, ispostavilo se da mogu uspešno da memorišem samo dve slike!

Neke velike slike pomenutih dimenzija koje sam koristio su prikazane na sledećoj slici (slika 3).



Slika 3: Slike dimenzija 32x32 sa kojima sam trenirao Hopfildovu mrežu.

Neispravan rad projektovane mreže nastao usled prekoračenja memorije prikazan je na sledećoj slici (slika 4).



Slika 4: Primer neispravnog asociranja mreže prilikom preopterećenja memorije.

Nakon uvoza slike, sledeći korak je bio konvertovanje binarnih podataka slike u matricu koju mi je lako koristiti. Kako su uvezene slike bile dimenzija 32x32 konvertovao sam te matrice u matrice dimenzije 1024x1. Dakle, imao sam matrice ulaznih vektora sačinjene od 1024 vrste i 1 kolone. Ovo sam uradio da bih mogao da proračunam matricu težina. Algoritam je isti za sva tri ulazna vektora sa jedinom razlikom u indeksima. Mesta na kojima sam menjao indekse sam naznačio crvenom bojom.

```
%% Pretvaram prvu sliku u niz brojeva
slika_1=zeros(1024,1);
k=0;
for i=1:n
    for j=1:n
        k=k+1;
        if (ulazna_slika_1(i,j)==0)
            slika_1(k,1)=-1;
        elseif (ulazna_slika_1(i,j)==1)
            slika_1(k,1)=1;
        end
    end
end
```

Kao što se može videti iz algoritma, bio je potreban još jedan dodatni uslov prilikom konvertovanja slika. Naime, bitmap slika je kodirana tako da 0 predstavlja crni piksel dok 1 predstavlja beli piksel. Kako Hopfildova mreža radi na skupu [-1 1] a bitmap slika na skupu [0 1] bilo je potrebno da preslikam bitmap skup na skup koji koristi Hopfildova mreža.

Nakon uvoza i konvertovanja svih slika u matrice određenih dimenzija napravio sam samo jednu matricu ulaznih vektora koja ima onoliko vrsta koliko postoji ulaza (1024 u mom slučaju) i onoliko kolona koliko postoji ulaznih vektora (2 u slučaju kada mreža radi ispravno). Dalje je prikazan ovaj algoritam za slučaj da imam tri ulazna vektora koja sam u gore navedenom tekstu opisao.

```
% Spajam ulazne vektore u jednu matricu
x=horzcat(slika_1,slika_2,slika_3);
```

4. Matrica težina konekcija

Nakon dobijanja potrebnih ulaznih podataka za obučavanje mreže napisao sam algoritam za proračunavanje matrice težina konekcija. Kao što sam već naveo, koristio sam Hebovo obučavanje.

```
%% Formiram matricu tezina konekcija (pamtim podatke) za sve ulazne vektore
for i=1:1024 % Broj ulaza
    for j=1:1024
        if (i==j)
            w(i,j)=0; % Elementi na glavnoj dijagonali su 0
        elseif (i~=j)
            suma=0;
            for k=1:3 % Broj ulaznih vektora
                suma=suma+x(i,k)*x(j,k);
                w(i,j)=suma;
            end
        end
    end
end
```

Sa ovim korakom sam završio uvoz slika i obučavanje Hopfildove mreže. Sada sam imao mrežu koja je zapamtila ulazne slike, odnosno asocirala ih je direktno (autoasocijacija) sa izlaznim vektorima. Ako bih hteo da optimizujem kod kako bi se algoritam brže izvršavao, mogao sam iskoristiti osobinu simetričnosti matrice težina konekcija. Tako bih računao, na primer, samo gornju dijagonalu matrice težina pa bih postavio uslov da je $w_{ij} = w_{ji}$. Međutim, ipak sam odabrao da to ne uradim kako bi jasnoća mog algoritma bila veća.

Jedino što mi je preostalo kod projektovanja Hopfildove mreže je da napišem algoritam koji će mreža koristiti da se „priseti“ neke od slika iz svoje memorije kada na njen ulaz doveden sliku sa proizvoljnim uticajem šuma. Kako sam već naveo, to sam uradio koristeći pravilo evoluiranja (*evolving ili update rule*).

5. Ažuriranje izlaza

Pre nego što sam mogao da pokrenem algoritam ažuriranja izlaza, morao dovesti na ulaz mreže jednu od memorisanih slika. Zadržao sam nasumično biranje ulazne slike koje sam napisao za prvobitni slučaj kada sam imao četiri ulazna vektora.

```
% Biram nasumicno jednu od dve ulazne slike
m=fix(1+2*rand(1)); % Broj izabrane slike
```

Kada sam nasumično odabrao sliku, doveo sam tu sliku na ulaz Hopfildove mreže.

```
% Izabranu sliku dovodim na ulaz Hopfildove mreze
for i=1:1024
    ulazni_sloj(i,1)=x(i,m); % Pocetni vektor je vektor m
end
```

Posle usnimavanja nasumično odabrane slike u matricu *ulazni_sloj* uveo sam šum koji može biti proizvoljan. Mrežu sam testirao na različit uticaj šuma i ispostavilo se da mreža asocira ispravno i kada ulazna slika ima čak 90% šuma! Šum sam uveo na sledeći način:

```
% Uvodim sum, odnosno, nasumicno menjam bitove ulaznog vektora
% Uvodim odredjeni procenat suma (proc_suma)
proc_suma=20;
sum=fix(1024*(proc_suma/100));
while (sum>0)
    m=fix(1+1024*rand(1));
    if (ulazni_sloj(m,1)==-1)
        ulazni_sloj(m,1)=1;
    elseif (ulazni_sloj(m,1)==1)
        ulazni_sloj(m,1)=-1;
    end
    sum=sum-1;
end
```

Šum sam uveo tako što sam invertovao određeni broj vrednosti cifara koje čine matricu ulaznog vektora. Odnosno, mesto -1 sam pisao 1, a mesto 1 sam pisao -1. Broj vrednosti koje se invertuju odgovara procentu uvedenog šuma. U prikazanom kodu, procenat šuma je 20%.

Algoritam za ažuriranje izlaza sam započeo tako što sam podesio da izlazni vektor ima identične vrednosti kao ulazni vektor.

```
% Inicijalizujem pocetnu vrednost izlaznog sloja
for i=1:1024
    izlazni_sloj(i,1)=ulazni_sloj(i,1);
end
```

Računanje izlaza sam ugnezdio u *while* petlju tako da se izlazi ažuriraju definisan broj puta. Da budem iskren, na ovom koraku sam izgubio više vremena nego što bih voleo da priznam. Jer, iako mi sada algoritam izgleda prosto, zapravo sam najveći problem imao sa razumevanjem indeksa i pravilnom realizacijom petlji. Problem je nastao zbog toga što sam u početku pogrešno shvatio indeksiranje u izrazu za pravilo evolucije. U trenutku kada sam zaista shvatio šta koji indeks predstavlja, kod sam završio u vrlo kratkom roku.

Ta ”famozna” petlja je sledeća:

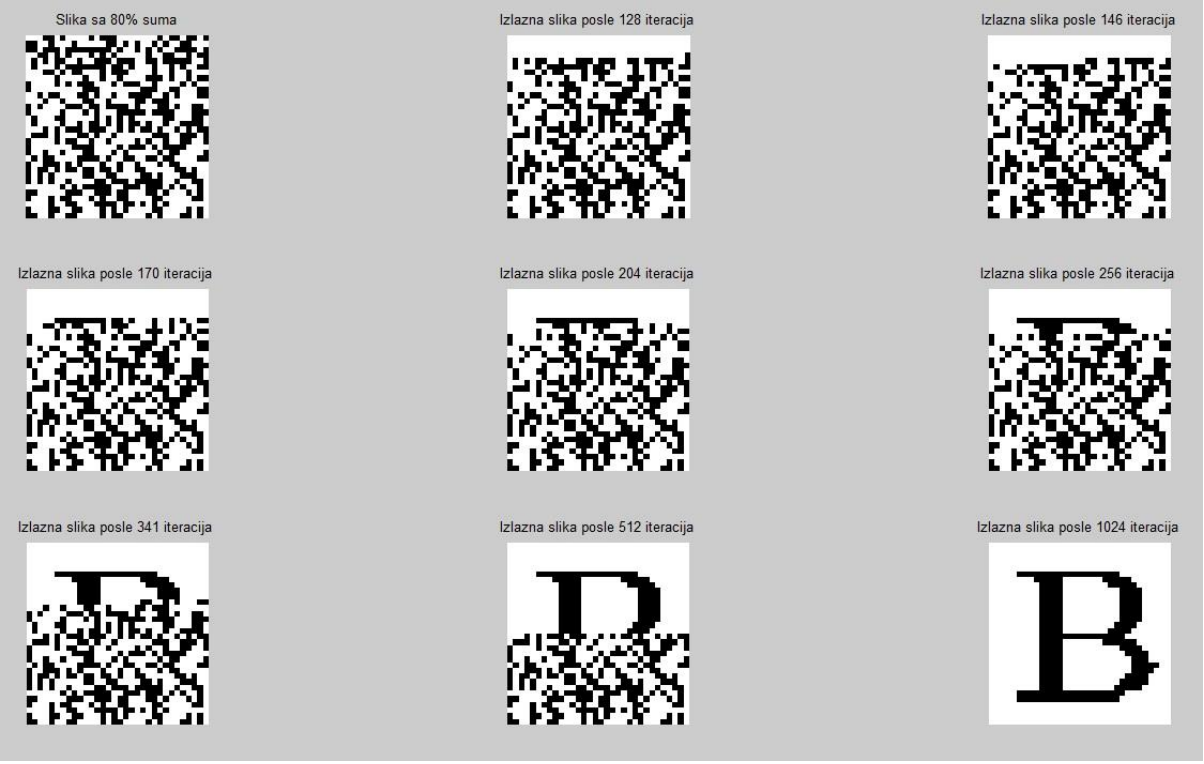
```
k=1;
while (k<6000)
    % Nasumicno biram jednu vrednost ulaznog vektora
    i=fix(1+1024*rand(1)); % Nasumicni broj od 1 do 1024
    %i=k;
    % Racunam izlaz za izabranu vrednost
    suma=0;
    for j=1:1024
        if (j~=i)
            suma=suma+w(i,j)*izlazni_sloj(j,k);
        end
    end
    privr=sign(suma+ulazni_sloj(i,1)-teta);
    izlazni_sloj(:,k+1)=izlazni_sloj(:,k);
    izlazni_sloj(i,k+1)=privr;

    k=k+1;
end
```

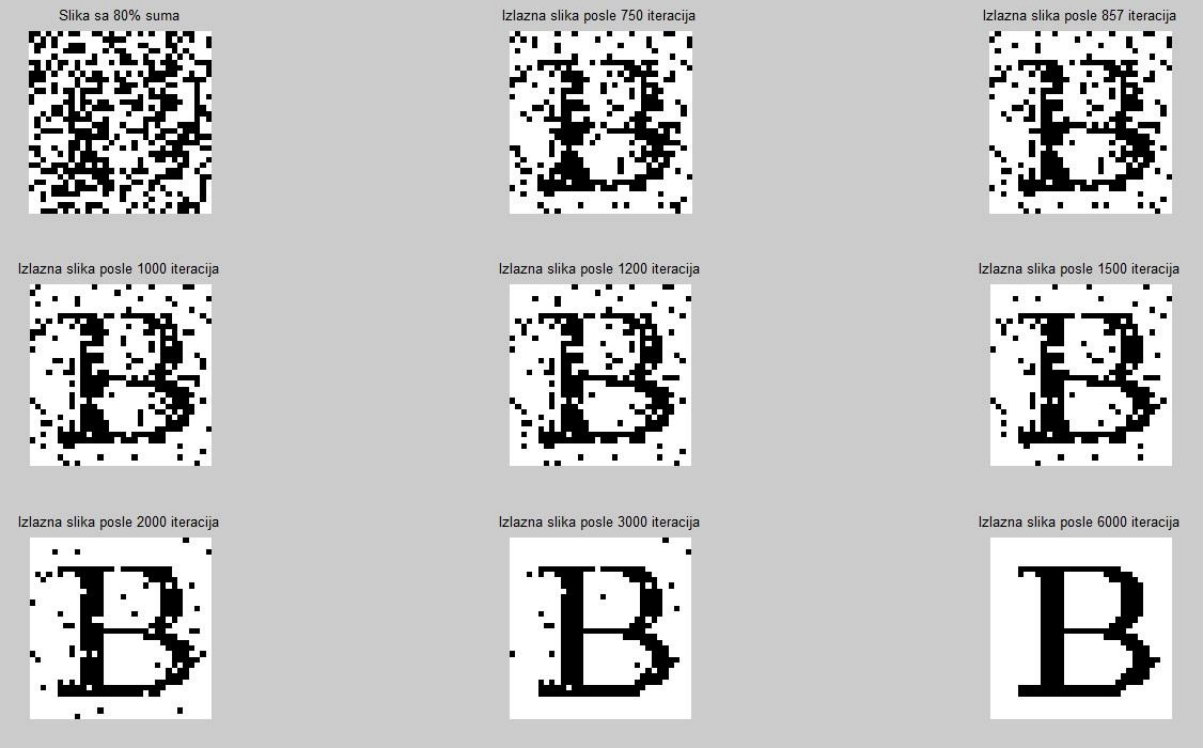
Mogao bih i da kažem da je dodatni problem nastao tako što sam u različitoj literaturi nailazio na različite realizacije ovog istog pravila. Otuda je nastala i moja zabuna.

Kako bih ispoštovao teoriju Hopfildove mreže naveo sam da se sledeći ulaz bira stohastički (nasumično), međutim tada je petlji potrebno nekoliko puta više iteracija nego da sam sledeći ulaz birao po redu. Dešava se da i posle 6000 iteracija mreža ipak ne uspe da prizove i asocira sve ulaze iz učitane slike. Kada stavim da se ulazi biraju po redu, tada mi je potrebno svega 1024 iteracije i mogu biti siguran da će svi ulazi ”doći na red”.

Ipak, kada posmatram rešenje dobijeno stohastičkim i rednim putem mogu da kažem da stohastičko rešenje izlgeda ”lepše”. Stohastičko i redno rešenje za memorisane dve slike i sa drugom slikom (slika B), sa 80% šuma, dovedenom na ulaz je prikazano na sledećim slikama (slike 5 i 6).



Slika 5: Slika na izlazu iz mreže dobijena rednim ažuriranjem izlaza.



Slika 6: Slika na izlazu iz mreže dobijena stohastičkim ažuriranjem izlaza.

Iako je memorija mreže bila manja kada sam koristio slike većih dimenzija, preciznost asociranja je bila nepogrešiva. Dok se, kada sam koristio manje slike, dešavalo ponekad i da na izlazu dobijem sliku koja nije identična ulaznoj slici.

Za projektovanje Hopfildove mreže nisam morao da koristim zasebne funkcije jer je kod relativno jednostavan i dosta je kraći od kodova koje sam iskucao za druga dva problema koja sam rešavao. Zapanjujuće je kako je vrlo jednostavno, algoritamski, rešen problem memorije mreže i asociranja ulaznih vektora sa izlazom. Kada bi takav zadatak bio stavljen pred mene, pomislio bi da je u pitanju ”nemoguća misija”, a opet, ljudi su taj problem rešili veoma elegantno i sa svega dva izraza.

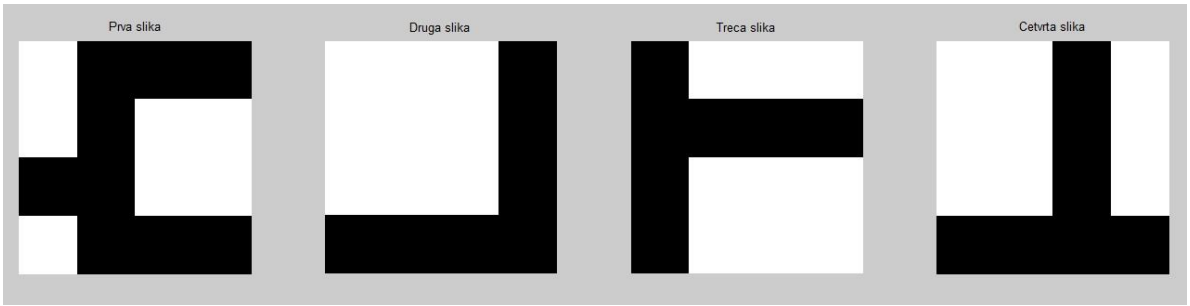
6. Dobijeni rezultati

U dosadašnjem tekstu sam već pokazao izgled izlaza iz Hopfildove mreže kada sam na ulaz dovodio slike dimenzija 32x32. U ovom delu teksta ću da testiram memoriju i asocijativnost mreže na slike različitih dimenzija i različitog broja ulaznih slika.

Počeću sa slikama koje su kodirane matricom dužine 16 bita (4x4).

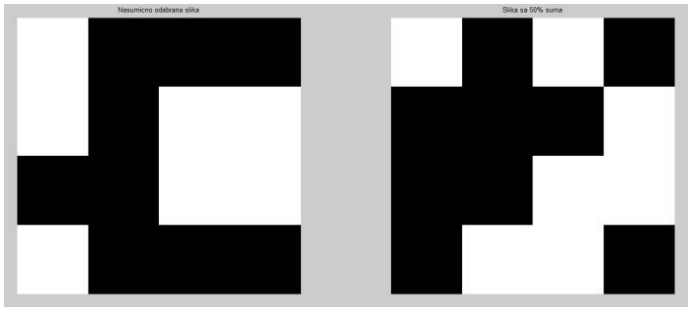
6.1. Slike dimenzija 4x4 bita

Generisao sam četiri slike dimenzija 4x4 bita (piksela). Slike koje sam u ovom slučaju koristio za testiranje mreže su prikazane na sledećoj slici (slika 7).



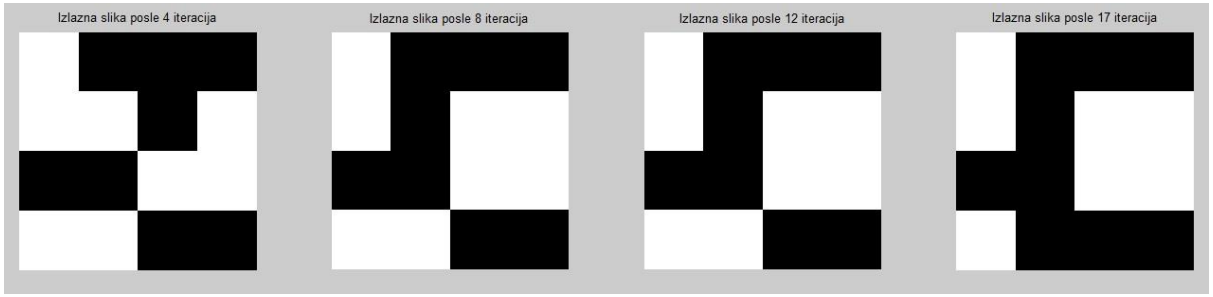
Slika 7: Slike dimenzija 4x4 sam kojima sam obučavao mrežu.

Slike sam na ulaz mreže birao nasumično a dovodio sam ih sa 50% šuma. Nasumično odabrana slika bez šuma i sa 50% šuma su prikazane na slici 8.



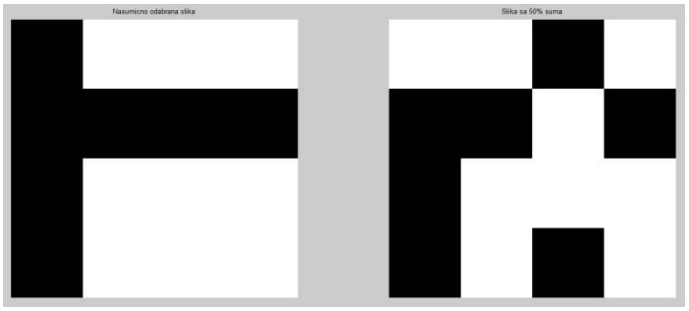
Slika 8: Nasumično odabrana slika bez šuma i sa 50% šuma.

Nakon izvršenog asociranja dobijena slika na izlazu je sledeća (slika 9).



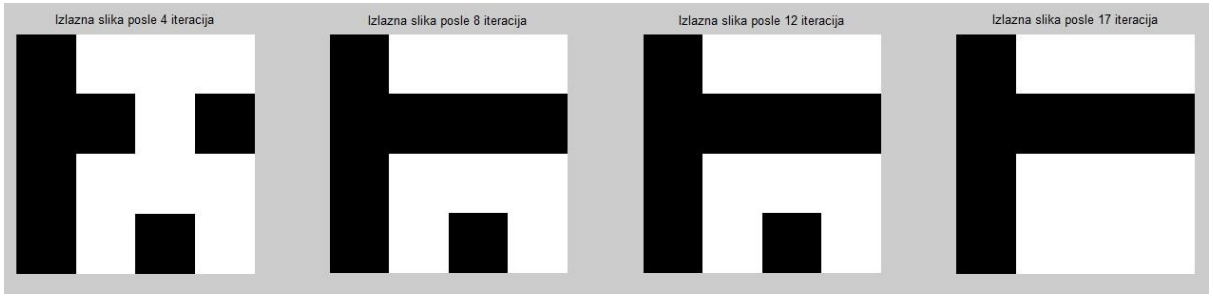
Slika 9: Slika na izlazu mreže nakon asociranja.

Sledeća slika bez šuma i sa 50% šuma (slika 10).



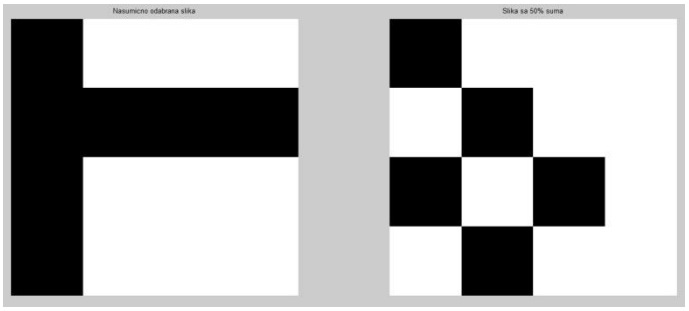
Slika 10: Nasumično odabrana slika bez šuma i sa 50% šuma.

Nakon izvršenog asociranja dobijena slika na izlazu je sledeća (slika 11).



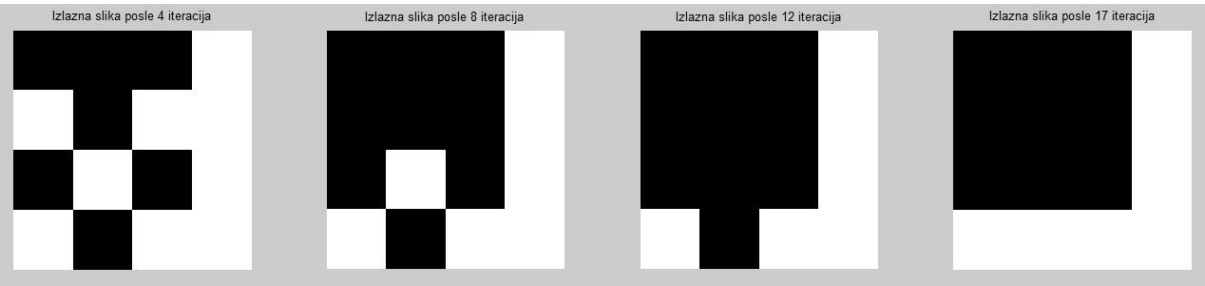
Slika 11: Slika na izlazu mreže nakon asociranja.

Iako izgleda da mreža korektno asocira ulaze, dešava se da mreža asocira izlaz sa slikom koja se nije pojavila na ulazu. Ovo se dešava zbog toga što je slika sa šumom, koja je dovedena na ulaz mreže, sličnija sa jednom od drugih slika iz memorije (slika 12).



Slika 12: Slika sa šumom koja više podseća na jednu od drugih slika iz memorije.

Slika sa šumom, naizgled, nije slična ni jednoj drugoj slici iz memorije. Međutim, ako se ima u vidu da se kod Hopfildove mreže komplement slike takođe posmatra kao i sama slika vidimo da druga slika sa slike 12 podseća neodoljivo na komplement druge slike sa slike 7. Nakon izvršenog asociranja na izlazu je dobijeno sledeće rešenje (slika 13).



Slika 13: Pogrešno asociranje.

Teorija Hopfildove mreže ukazuje na dva bitna nedostatka ove arhitekture neuralnih mreža. Prvi je mali kapacitet memorije, što se pokazalo u slučaju kada sam obučavao mrežu na slike dimenzija 32x32. Drugi problem je upravo problem na koji sam sada naišao. Taj problem je što mreža konvergira ne samo ka rešenju već i ka komplementu rešenja. Komplement rešenja mreža takođe tretira kao stabilno stanje a to upravo i dovodi do greške (kao u malopre prikazanom slučaju). Ovaj vid rešenja se naziva lažno stabilno stanje.

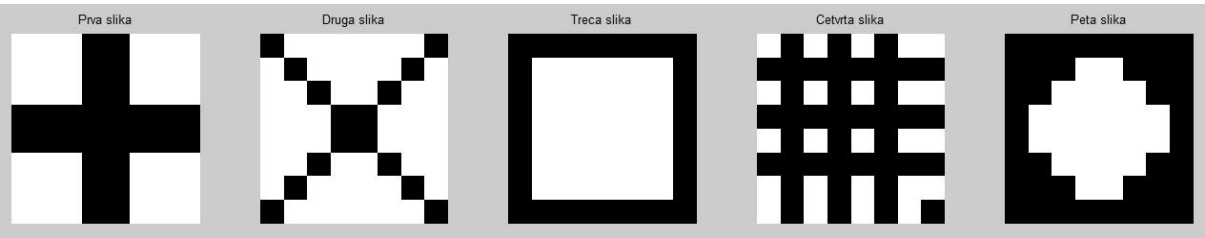
Pri izboru dimenzija slike, odnosno broja neurona Hopfildove mreže treba takođe voditi računa i o prostoru koji je potreban za definisanje nezavisnih slika. Naime, u mom slučaju, neke od slika sa slike 2 imaju pojedine delove identične. Usled toga i pri određenom procentu šuma, dešava se da mreža konvergira ka pogrešnom rešenju. Dakle, ako se biraju male dimenzije slike tada treba voditi računa da slike budu u što većoj meri nezavisne (da nemaju delove skupa koji se preklapaju).

Iako se javljaju izvesne mane u ovom primeru, ipak mogu da primetim da je kapacitet mreže veći. Mreža je uspela uspešno da memoriše veću količinu slika. Problem bi bio manje uočljiv da sam slike drugačije definisao. To ću upravo da uradim sa sledećim testom.

U sledećem primeru koji ću da testiram, definisaću slike dimenzija 8x8 pri čemu će mi svaka od pet definisanih slika biti u što većoj meri različita od ostalih. To jest, pokušaću da slike učinim što nezavisnijim pa da testiram kapacitet mreže i korektnost njene asocijativnosti.

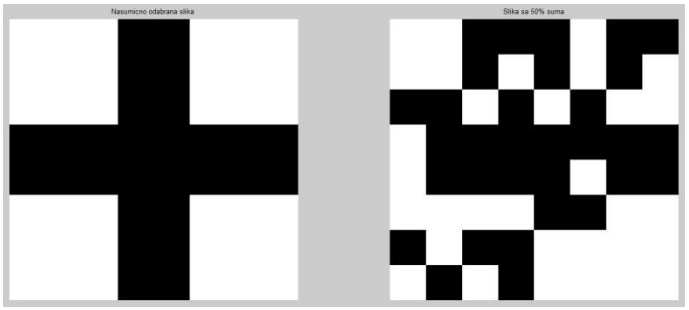
6.2. Slike dimenzija 8x8

Skup od pet binarnih slika koje sam definisao na samom početku programa su prikazane na slici14.



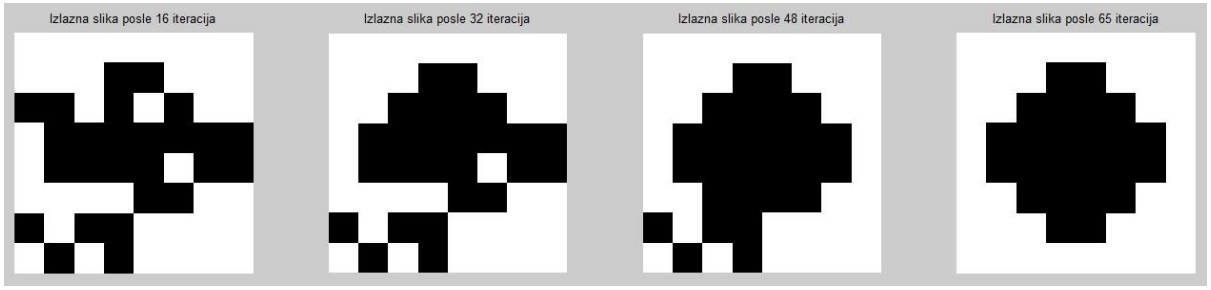
Slika 14: Pet binarnih slika dimenzija 8x8.

Nasumično odabrana slika bez šuma i sa 50% šuma izlgeda ovako (slika 15):



Slika 15: Nasumično odabrana slika.

Nakon asociranja dobijam sledeće slike (slika 16):

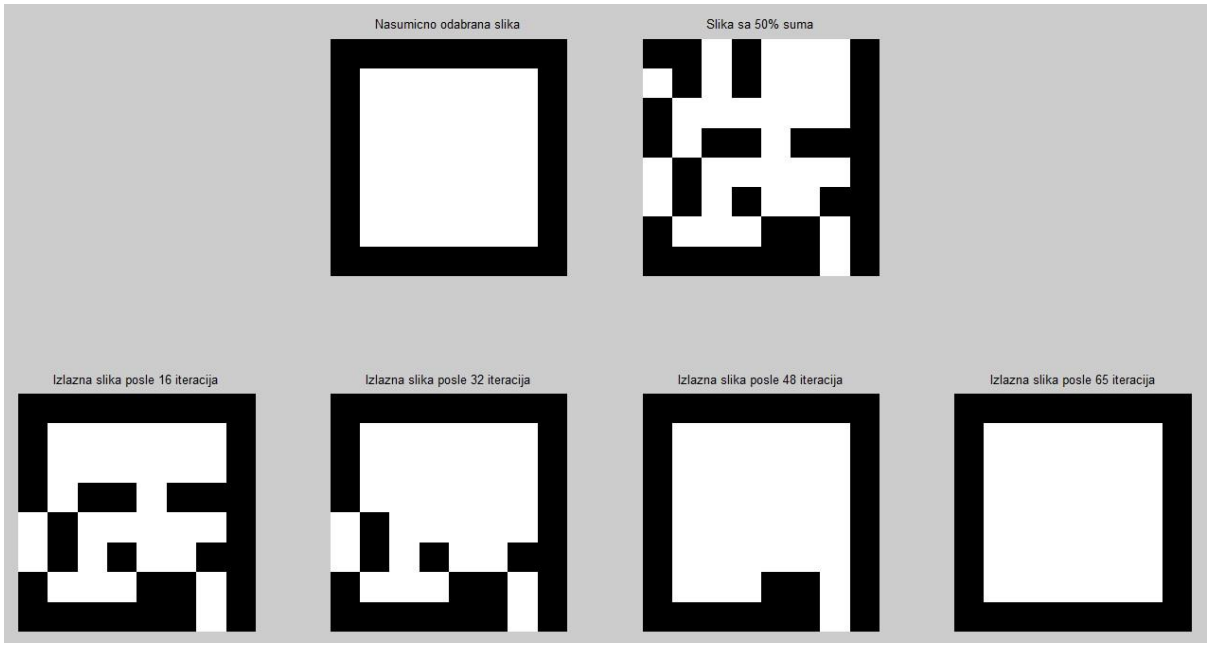


Slika 16: Dobijena slika na izlazu Hopfildove mreže.

Zaključak je da i u ovom slučaju, sa skup od 5 generisanih binarnih slika, Hopfildova mreža dobro memoriše ali usled postojanja lažnog rešenja opet dolazi do greške prilikom asociranja.

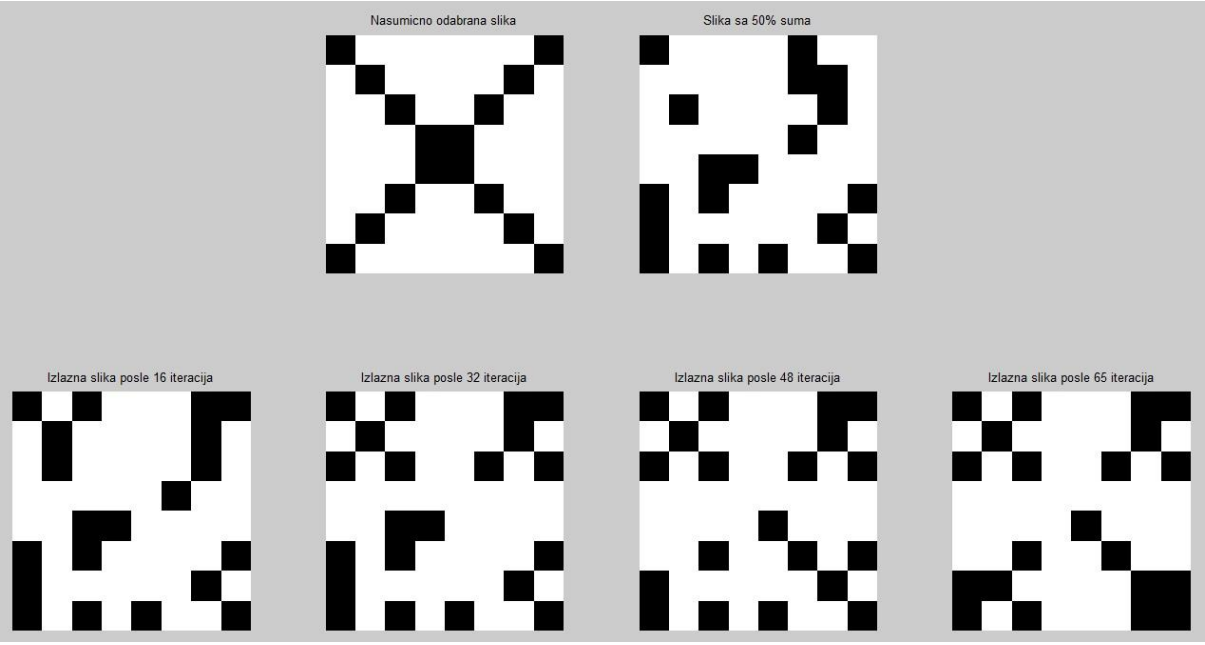
Dobijeni rezultat u poslednjem primeru ne odgovara ulaznoj slici, već predstavlja komplement pete memorisane slike koja je data na slici 14.

Primer ispravnog asociranja je dat na sledećoj slici.



Slika 17: Mreža je ispravno asocirala ulaznu sliku.

U velikom broju slučajeva mreža asocira korektno. Dešava se, međutim, kao što sam već napomenuo, da mreža konvergira ka lažnom rešenju ali se dešava i da se na izlazu mreže pojavi slika koja ne odgovara ni jednoj od ulaznih slika! Takav primer je dat na sledećoj slici.



Slika 18: Nepostojeća slika na izlazu mreže.

7. Zaključak

Iako je koncept Hopfildove mreže od izuzetnog značaja, praktična vrednost nije velika. Ova teorija nam je omogućila da znanja stečena analizom Hopfildove mreže primenimo na druge tipove asocijativnih memorija u kojima su otklonjeni nedostatci diskretne, rekurzivne, autoasocijativne memorije. Tako da, upravo te druge vrste asocijativnih memorija imaju veću upotrebnu vrednost (kao na primer, bidirekcione asocijativne memorije).

Nedostatak Hopfildove memorije koji je karakterisan konvergencijom mreže ka lažnom stabilnom stanju donekle i može da se otkloni ukoliko se koriste slike velikih dimenzija. Ove slike bi trebale biti značajno različite i u malom broju kako ne bi dolazilo do neispravnog asociranja.

Ispostavilo se da je u mom slučaju za slike 32x32 mreža mogla upamtiti više slika ali ih nije mogla i ispravno asocirati sa velikim procentom uspešnosti. Broj od samo dve memorisane slike koji sam ja odabrao za datu veličinu slika nije veliki, ali sa sigurnošću mogu da tvrdim da mreža u tom slučaju ispravno asocira u skoro 100% slučajeva.

Dodatak

Sledi kompletan algoritam kojim sam projektovao Hopfildovu mrežu.

```
%% Citam *.BMP sliku i pretvaram je u niz brojeva
clc
clear all
close all

%% Ucitavam prvu sliku
n=32; % Ogranicavam dimenzije slike na 32x32
[filename,pathname]=uigetfile; % Biram sliku
ulazna_slika_1=imread(filename); % Usnimavam sliku u zeljenu promenljivu
ulazna_slika_1=imresize(ulazna_slika_1,[n n]); % Prilagodjavam dimenzije slike

% Iscrtavam odabranu sliku
figure(1)
subplot(121)
imshow(ulazna_slika_1);
title('Prva slika');

%% Pretvaram prvu sliku u niz brojeva
slika_1=zeros(1024,1);
k=0;
for i=1:n
    for j=1:n
        k=k+1;
        if (ulazna_slika_1(i,j)==0)
            slika_1(k,1)=-1;
        elseif (ulazna_slika_1(i,j)==1)
            slika_1(k,1)=1;
        end
    end
end

%% Ucitavam drugu sliku
n=32; % Ogranicavam dimenzije slike na 32x32
[filename,pathname]=uigetfile; % Biram sliku
ulazna_slika_2=imread(filename); % Usnimavam sliku u zeljenu promenljivu
ulazna_slika_2=imresize(ulazna_slika_2,[n n]); % Prilagodjavam dimenzije slike

% Iscrtavam odabranu sliku
subplot(122)
imshow(ulazna_slika_2);
title('Druga slika');

%% Pretvaram drugu sliku u niz brojeva
slika_2=zeros(1024,1);
k=0;
for i=1:n
    for j=1:n
        k=k+1;
        if (ulazna_slika_2(i,j)==0)
            slika_2(k,1)=-1;
        elseif (ulazna_slika_2(i,j)==1)
            slika_2(k,1)=1;
        end
    end
end

% %% Ucitavam trecu sliku
% n=32; % Ogranicavam dimenzije slike na 32x32
% [filename,pathname]=uigetfile; % Biram sliku
% ulazna_slika_3=imread(filename); % Usnimavam sliku u zeljenu promenljivu
% ulazna_slika_3=imresize(ulazna_slika_3,[n n]); % Prilagodjavam dimenzije slike
% subplot(143)
% imshow(ulazna_slika_3);
```

```
% title('Treca slika');
%
% %% Pretvaram trecu sliku u niz brojeva
% slika_3=zeros(1024,1);
% k=0;
% for i=1:n
%     for j=1:n
%         k=k+1;
%         if (ulazna_slika_3(i,j)==0)
%             slika_3(k,1)=-1;
%         elseif (ulazna_slika_3(i,j)==1)
%             slika_3(k,1)=1;
%         end
%     end
% end

% Spajam ulazne vektore u jednu matricu
x=horzcat(slika_1,slika_2);
teta=0; % Za sve cvorove je teta jednako 0
w=zeros(1024,1024);

%% Formiram matricu tezina konekcija (pamtim podatke) za sve ulazne vektore
for i=1:1024 % Broj ulaza
    for j=1:1024
        if (i==j)
            w(i,j)=0; % Elementi na glavnoj dijagonali su 0
        elseif (i~=j)
            suma=0;
            for k=1:2 % Broj ulaznih vektora
                suma=suma+x(i,k)*x(j,k);
                w(i,j)=suma;
            end
        end
    end
end

%% Uvodim pravilo azuriranja izlaza
% Setujem pocetno stanje ulaznog sloja na nulu
ulazni_sloj=zeros(1024,1);
% Setujem pocetno stanje izlaznog sloja na nulu
izlazni_sloj=zeros(1024,101);

%% Postavljam neki vektor na ulaz Hopfildove mreze
% % Test na neku od upamcenih slika
% m=2;
% for i=1:1024
%     ulazni_sloj(i,1)=x(i,m); % Pocetni vektor je vektor m
% end
%
% % Konvertujem informacije sa ulaza tako da mogu da ih iscrtam
% test_slika=zeros(32,32);
% k=0;
% for i=1:n
%     for j=1:n
%         k=k+1;
%         test_slika(i,j)=ulazni_sloj(k,1);
%     end
% end
% figure(2)
% subplot(331)
% imshow(test_slika);
% title('Slika dovedena na ulaz mreze');

% Test na sliku sa uticajem suma
% Biram nasumicno jednu od dve ulazne slike
m=fix(1+2*rand(1)); % Broj izabrane slike

% Izabranu sliku dovodim na ulaz Hopfildove mreze
```

```
for i=1:1024
    ulazni_sloj(i,1)=x(i,m); % Pocetni vektor je vektor m
end

% Uvodim sum, odnosno, nasumicno menjam bitove ulaznog vektora
% Uvodim sum*10% suma (sum je neki broj od 1 do 10)
proc_suma=80;
sum=fix(1024*(proc_suma/100));
while (sum>0)
    m=fix(1+1024*rand(1));
    if (ulazni_sloj(m,1)==-1)
        ulazni_sloj(m,1)=1;
    elseif (ulazni_sloj(m,1)==1)
        ulazni_sloj(m,1)=-1;
    end
    sum=sum-1;
end

% Prikazujem sliku sa uticajem suma
distorzija_slike=zeros(32,32);
k=0;
for i=1:n
    for j=1:n
        k=k+1;
        distorzija_slike(i,j)=ulazni_sloj(k,1);
    end
end
figure(2)
subplot(331)
imshow(distorzija_slike);
title(['Slika sa ',num2str(proc_suma),'% suma']);


%% Izlaze azuriram dok god ne dodjem do stabilnog stanja (ekvilibrijuma)
% Inicijalizujem pocetnu vrednost izlaznog sloja
for i=1:1024
    izlazni_sloj(i,1)=ulazni_sloj(i,1);
end

k=1;
while (k<6000)
    % Nasumicno biram jednu vrednost ulaznog vektora
    i=fix(1+1024*rand(1)); % Nasumicni broj od 1 do 1024
    %i=k;
    % Racunam izlaz za izabranu vrednost
    suma=0;
    for j=1:1024
        if (j~=i)
            suma=suma+w(i,j)*izlazni_sloj(j,k);
        end
    end
    privr=sign(suma+ulazni_sloj(i,1)-teta);
    izlazni_sloj(:,k+1)=izlazni_sloj(:,k);
    izlazni_sloj(i,k+1)=privr;

    k=k+1;
    broj=k;
end

% Crtam sliku dobijenu na izlazu Hopfildove mreze
izlazna_slika=zeros(32,32);
k=0;
for i=1:n
    for j=1:n
        k=k+1;
        izlazna_slika(i,j)=izlazni_sloj(k,broj);
    end
end
```

```
subplot(332)
% Crtam sliku dobijenu na izlazu Hopfildove mreze posle broj/8 iteracija
privremeno=zeros(32,32);
k=0;
kor=fix(broj/8);
for i=1:n
    for j=1:n
        k=k+1;
        privremeno(i,j)=izlazni_sloj(k,kor);
    end
end
imshow(privremeno);
title(['Izlazna slika posle ',num2str(kor),' iteracija']);

subplot(333)
% Crtam sliku dobijenu na izlazu Hopfildove mreze posle fix(broj/7) iteracija
privremeno=zeros(32,32);
k=0;
kor=fix(broj/7);
for i=1:n
    for j=1:n
        k=k+1;
        privremeno(i,j)=izlazni_sloj(k,kor);
    end
end
imshow(privremeno);
title(['Izlazna slika posle ',num2str(kor),' iteracija']);

subplot(334)
% Crtam sliku dobijenu na izlazu Hopfildove mreze posle fix(broj/6) iteracija
privremeno=zeros(32,32);
k=0;
kor=fix(broj/6);
for i=1:n
    for j=1:n
        k=k+1;
        privremeno(i,j)=izlazni_sloj(k,kor);
    end
end
imshow(privremeno);
title(['Izlazna slika posle ',num2str(kor),' iteracija']);

subplot(335)
% Crtam sliku dobijenu na izlazu Hopfildove mreze posle fix(broj/5) iteracija
privremeno=zeros(32,32);
k=0;
kor=fix(broj/5);
for i=1:n
    for j=1:n
        k=k+1;
        privremeno(i,j)=izlazni_sloj(k,kor);
    end
end
imshow(privremeno);
title(['Izlazna slika posle ',num2str(kor),' iteracija']);

subplot(336)
% Crtam sliku dobijenu na izlazu Hopfildove mreze posle fix(broj/4) iteracija
privremeno=zeros(32,32);
k=0;
kor=fix(broj/4);
for i=1:n
    for j=1:n
        k=k+1;
        privremeno(i,j)=izlazni_sloj(k,kor);
    end
end
imshow(privremeno);
```

```
title(['Izlazna slika posle ',num2str(kor),' iteracija']);

subplot(337)
% Crtam sliku dobijenu na izlazu Hopfildove mreze posle fix(broj/3) iteracija
privremeno=zeros(32,32);
k=0;
kor=fix(broj/3);
for i=1:n
    for j=1:n
        k=k+1;
        privremeno(i,j)=izlazni_sloj(k,kor);
    end
end
imshow(privremeno);
title(['Izlazna slika posle ',num2str(kor),' iteracija']);

subplot(338)
% Crtam sliku dobijenu na izlazu Hopfildove mreze posle fix(broj/2) iteracija
privremeno=zeros(32,32);
k=0;
kor=fix(broj/2);
for i=1:n
    for j=1:n
        k=k+1;
        privremeno(i,j)=izlazni_sloj(k,kor);
    end
end
imshow(privremeno);
title(['Izlazna slika posle ',num2str(kor),' iteracija']);

subplot(339)
% Crtam sliku dobijenu na izlazu Hopfildove mreze posle broj iteracija
privremeno=zeros(32,32);
k=0;
for i=1:n
    for j=1:n
        k=k+1;
        privremeno(i,j)=izlazni_sloj(k,broj);
    end
end
imshow(privremeno);
title(['Izlazna slika posle ',num2str(broj),' iteracija']);
```