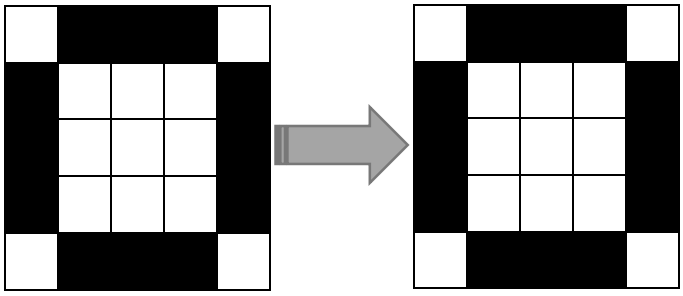
در این تمرین با استفاده از حالات مختلف پاسخ هاپفیلد به ورودی‌های مسئله و تشخیص الگوها، تحلیل صورت گرفته‌است که در ادامه هریک از ورودی‌های مسئله و پاسخ هاپفیلد را بررسی می‌کنیم و به یک پاسخ کلی براساس نتایج بدست آمده میرسیم.

باتوجه به بایپولار بودن ورودی‌های مسئله، برای بررسی راحت‌تر نمونه‌ها از رنگ سفید برای مقدار ۱- و از رنگ مشکی برای نمایش مقدار ۱ استفاده شده‌است. کد متلب پیاده‌سازی شده برای هاپفیلد در انتهای تمرین ضمیمه شده‌است.

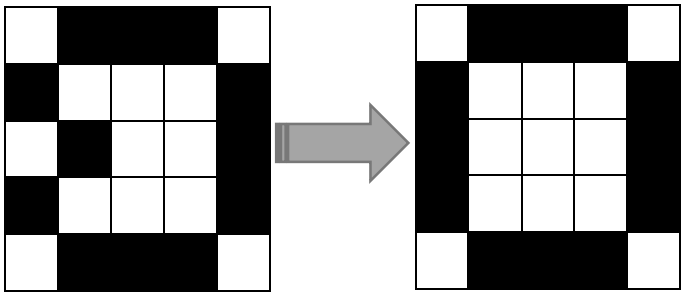
از حالت‌های مختلف الگوی (O) برای بررسی و تحلیل استفاده می‌کنیم.

ورودی شماره ۱)



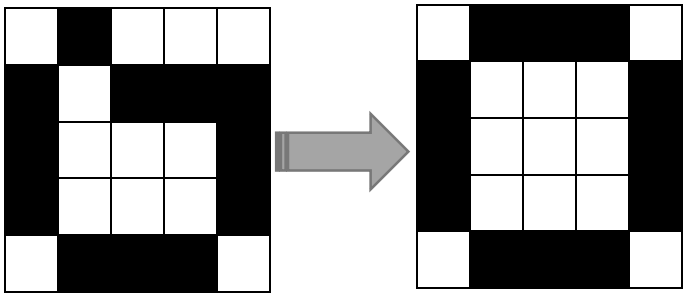
به ازای ورودی بدون نویز و مشابه هریک از الگوهای اولیه خروجی کاملا درستی از خود نشان می‌دهد.

ورودی شماره ۲)



در صورتی یک نویز در یک سطر یا ستون داشته باشیم، باعث تغییر مقدار دو بیت در آن سطر یا ستون می‌شود. اما همچنان شبکه قادر به تشخیص الگوی اصلی می‌باشد.

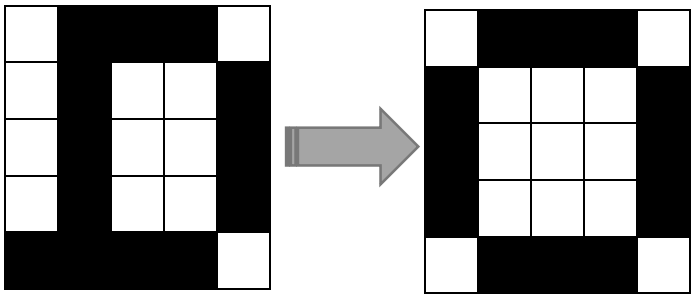
ورودی شماره ۳)



با وجود دو نویز در یک سطر یا ستون، شبکه همچنان بدون مشکل قادر به شناسایی الگوی اصلی می‌باشد.

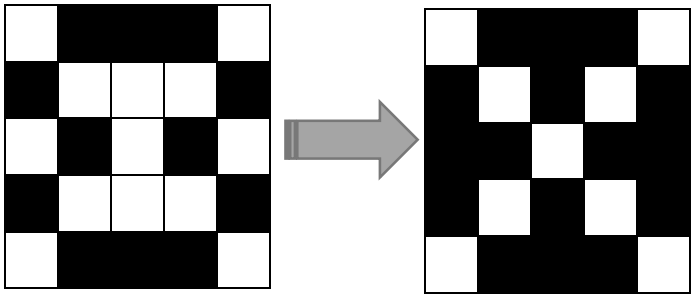
ورودی شماره ۴)

با تفاوت ۳ مقدار در یک سطر یا ستون همچنان شبکه قادر به شناسایی الگو می‌باشد



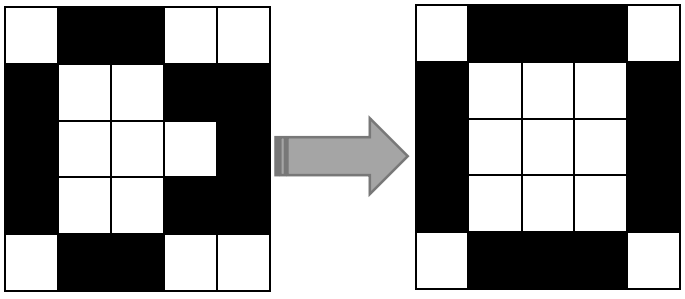
ورودی شماره ۶)

در این شکل همان‌طور که مشاهده می‌شود، با تغییر و جابجایی مقدار دو بیت از سطر سوم، مقدار ۴بیت آن تغییر می‌کند، و شبکه پاسخ کاملا اشتباهی به این الگو داده است.



ورودی شماره ۷)

مشابه ورودی شماره ۶ بر روی ستون مشترک نیز اعمال شد، به طوری که مقدار ۴بیت در یک ستون تغییر پیدا کرد. اما شبکه توانست الگوی اصلی را شناسایی کند.



نتیجه و تحلیل نهایی:

عامل تاثیرگذار در تشخیص پذیری شبکه هاپفیلد، میزان نویز موجود در سطر هاست. که در این مثال خاص، وقتی مقدار نویزها در یک سطر بیش از ۶۰٪ شد، شبکه توانایی خود در تشخیص الگو را از دست داد.

MATLAB Implementation

```
clc;
clear;
close all;

% define X/O/+ patterns
patternX = [
     1, -1, -1, -1,  1;
    -1,  1, -1,  1, -1;
    -1, -1,  1, -1, -1;
    -1,  1, -1,  1, -1;
     1, -1, -1, -1,  1;
    ];
patternO = [
    -1,  1,  1,  1, -1;
     1, -1, -1, -1,  1;
     1, -1, -1, -1,  1;
     1, -1, -1, -1,  1;
    -1 , 1,  1,  1, -1;
    ];
patternPlus = [
    -1, -1, 1, -1, -1;
    -1, -1, 1, -1, -1;
     1,  1, 1,  1,  1;
    -1, -1, 1, -1, -1;
    -1, -1, 1, -1, -1;
    ];

% convert patterns to a single row vector
patternX = reshape(patternX,1,[]);
patternO = reshape(patternO,1,[]);
patternPlus = reshape(patternPlus,1,[]);

% input pattern
x = [
     1, -1, -1,  1,  1;
    -1,  1, -1,  1, -1;
     1, -1,  1, -1, -1;
    -1,  1, -1,  1, -1;
     1, -1, -1, -1,  1;
    ];
x = reshape(x,1,[]);

% initialize weights
w1 = (patternX')*(patternX);
w2 = (patternO')*(patternO);
w3 = (patternPlus')*(patternPlus);
```

```
% set main diameter to zero
for i=1:25
    w1(i,i) = 0 ;
    w2(i,i) = 0 ;
    w3(i,i) = 0 ;
end
w = w1 + w2 + w3 ;

% random-ordered indexes
randomIndex = randperm(25);

% main algorithm
y = x;
y_temp = y;
epochs = 0;

while true
    for i=1:size(x,2)
        sum = 0;
        for j=1:size(x,2)
            sum = sum + (y(j)*w(j,i));
        end

        yin= x(randomIndex(i)) + sum ;
        if yin > 0
            y(i) = 1 ;
        elseif yin < 0
            y(i) = -1 ;
        end
    end

    for j=1:size(x,2)
        if y_temp(1,j) ~= y(1,j)
            run = true;
        end
        if y_temp(1,j) == y(1,j)
            run = false;
        end
    end

    y_temp = y ;
    epochs = epochs +1 ;

    % exit condition
    if run == false
        break;
    end
end

fprintf('number of epochs = %d\n', epochs);
disp(['input = ', mat2str(x)]);
disp(['output = ', mat2str(y)]);
```