ONE-STEP ERROR PROBABILITY

Table of Contents

Initial setup

```
clear, clc
N = 120;
number_of_trials = 1e5;
p_array = [12,24,48,70,100,120];
```

With zero diagonals

```
error_probability_array_zero_diag = zeros(1,size(p_array,2));
for i = 1:size(p_array,2)
    p = p_array(i);
    errors = 0;
    for trial = 1:number_of_trials
        patterns = generate_patterns(p, N);
        weight_matrix = generate_weight_matrix_zero_diag(patterns);
        pattern_number = randi([1 p]);
        neuron_number = randi([1 N]);
        old_neuron_value = patterns(neuron_number,pattern_number);
        new_neuron_value =
 signum(weight_matrix(neuron_number,:)*patterns(:,pattern_number));
        errors = errors + double(old_neuron_value ~=
 new_neuron_value);
    end
    error_probability_array_zero_diag(i) = errors / number_of_trials;
end
disp("The one step error probabilities with zero diagonals are:")
disp(num2str(error_probability_array_zero_diag))
```

```
The one step error probabilities with zero diagonals are:
0.00035     0.01147     0.05612     0.09402     0.13625     0.15764
```

Without zero diagonals

```
error_probability_array = zeros(1,size(p_array,2));
for i = 1:size(p_array,2)
    p = p_array(i);
    errors = 0;
    for trial = 1:number_of_trials
```

```
        patterns = generate_patterns(p, N);
        weight_matrix = generate_weight_matrix(patterns);
        pattern_number = randi([1 p]);
        neuron_number = randi([1 N]);
        old_neuron_value = patterns(neuron_number,pattern_number);
        new_neuron_value =
 signum(weight_matrix(neuron_number,:)*patterns(:,pattern_number));
        errors = errors + double(old_neuron_value ~=
 new_neuron_value);
    end
    error_probability_array(i) = errors / number_of_trials;
end
disp("The one step error probabilities without zero diagonals are:")
disp(num2str(error_probability_array))
```

```
The one step error probabilities without zero diagonals are:
0.00022     0.00304     0.01242      0.019     0.02119     0.02308
```

Functions

```
disp('')
function weight_matrix = generate_weight_matrix(patterns)
    weight_matrix = patterns*patterns';
end

function weight_matrix = generate_weight_matrix_zero_diag(patterns)
    weight_matrix = patterns*patterns'/size(patterns,1);
    weight_matrix = weight_matrix - diag(diag(weight_matrix));
end

function sign = signum(x)
    sign = 2*(x >= 0) - 1;
end

function patterns = generate_patterns(p, N)
    % Generates a p by N matrix with p N-bit patterns as columns
    patterns = 2.*randi([0 1],N, p) - 1;
end
```

Published with MATLAB® R2020a

RECOGNISING DIGITS

Table of Contents

Initial setup, loads patterns

```
clear, clc
x1=[ [ -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, 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],[ -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, -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],[ -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, -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],[ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1] ];

x2=[ [ -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, 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],[ -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, 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],[ -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, 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],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1] ];

x3=[ [ 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, 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],[ -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, 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],[ 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, -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],[ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1] ];

x4=[ [ -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, -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],[ -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, 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],[ -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,
 -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],[ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1] ];
```

```
x5=[ [ -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, -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],[ -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, 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],[ -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, -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],[ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1] ];

stored_patterns = [ x1', x2', x3', x4', x5' ];


fed_pattern_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, -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, 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, 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, -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, -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, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1]]';

fed_pattern_2 = [[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, -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],
 [-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, 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], [-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, -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], [-1, -1, 1, 1, 1, 1, 1, 1,
 -1, -1]]';

fed_pattern_3 = [[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, -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],
 [-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, 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], [-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, -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], [-1, -1, -1, -1, -1, -1,
 -1, 1, 1, -1]]';
```

Calculate and classify steady states of patterns

```
clc
weight_matrix = generate_weight_matrix_zero_diag(stored_patterns);

steady_state_pattern_1 =
 update_pattern_until_steady_state(weight_matrix, fed_pattern_1);
```

```
col_index_pattern_1 = classify_pattern(steady_state_pattern_1,
 stored_patterns);
disp(strcat("The steady state of pattern 1 corresponds to digit index
 ", int2str(col_index_pattern_1), "."))

steady_state_pattern_2 =
 update_pattern_until_steady_state(weight_matrix, fed_pattern_2);
col_index_pattern_2 = classify_pattern(steady_state_pattern_2,
 stored_patterns);
disp(strcat("The steady state of pattern 2 corresponds to digit index
 ", int2str(col_index_pattern_2), "."))

steady_state_pattern_3 =
 update_pattern_until_steady_state(weight_matrix, fed_pattern_3);
col_index_pattern_3 = classify_pattern(steady_state_pattern_3,
 stored_patterns);
disp(strcat("The steady state of pattern 3 corresponds to digit index
 ", int2str(col_index_pattern_3), "."))
```

```
The steady state of pattern 1 corresponds to digit index 2.
The steady state of pattern 2 corresponds to digit index 4.
The steady state of pattern 3 corresponds to digit index 5.
```

Functions

```
disp('')
function col_index = classify_pattern(pattern, stored_patterns)
    for i = 1:size(stored_patterns, 1)
        if isequal(pattern, stored_patterns(:,i))
            col_index = i;
            break
        elseif isequal(-pattern, stored_patterns(:,i))
            col_index = -i;
            break
        end
    end
end

function steady_state_pattern =
 update_pattern_until_steady_state(weight_matrix, fed_pattern)
    old_pattern = fed_pattern;
    new_pattern = update_state_asynchronously(weight_matrix,
 old_pattern);
    while ~isequal(old_pattern, new_pattern)
       old_pattern = new_pattern;
       new_pattern = update_state_asynchronously(weight_matrix,
 old_pattern);
    end
    steady_state_pattern = new_pattern;
end

function state = update_state_asynchronously(weight_matrix, state)
    for neuron_number = 1:size(state, 1)
```

```
        state(neuron_number) =
 signum(weight_matrix(neuron_number,:)*state);
    end
end

function weight_matrix =
 generate_weight_matrix_zero_diag(stored_patterns)
    weight_matrix = stored_patterns*stored_patterns'/
size(stored_patterns,1);
    weight_matrix = weight_matrix - diag(diag(weight_matrix));
end

function sign = signum(x)
    sign = 2*(x >= 0) - 1;
end
```

Published with MATLAB® R2020a

STOCHASTIC HOPFIELD NETWORK

Table of Contents

Initial setup

```
clear, clc
beta = 2;
N = 200;
T = 2e5;
number_of_updates = T/N;
number_of_tests = 100;
```

For p = 7

```
p = 7;
m=zeros(1,number_of_tests);
for n_test = 1:number_of_tests
    patterns = generate_patterns(p,N);
    weight_matrix = generate_weight_matrix_zero_diag(patterns);
    initial_pattern = patterns(:,1);
    state = initial_pattern;
    sum_t = 0;
    for t = 1:number_of_updates
        for neuron_number = 1:N
            state = update_neuron_asynchronously(weight_matrix, state,
 neuron_number);
            sum_t = sum_t + 1/N*sum(state.*initial_pattern);
        end
    end
    m(n_test) = sum_t/T;
end
m_average = mean(m);
disp(strcat("The average m_1 for p = 7 is ", num2str(m_average)))
```

The average m_1 for p = 7 is 0.85115

For p = 45

```
p = 45;
m=zeros(1,number_of_tests);
for n_test = 1:number_of_tests
    patterns = generate_patterns(p,N);
```

```
    weight_matrix = generate_weight_matrix_zero_diag(patterns);
    initial_pattern = patterns(:,1);
    state = initial_pattern;
    sum_t = 0;
    for t = 1:number_of_updates
        for neuron_number = 1:N
            state = update_neuron_asynchronously(weight_matrix, state,
 neuron_number);
            sum_t = sum_t + 1/N*sum(state.*initial_pattern);
        end
    end
    m(n_test) = sum_t/T;
end
m_average = mean(m);
disp(strcat("The average m_1 for p = 45 is ", num2str(m_average)))
```

The average m_1 for p = 45 is 0.11416

Functions

```
disp('')
function g = g(b)
    beta = 2;
    g = 1./(1+exp(-2*beta*b));
end

function neuron_value = generate_neuron_value(b)
    x = rand;
    if x < g(b)
        neuron_value = 1;
    else
        neuron_value = -1;
    end
end

function weight_matrix = generate_weight_matrix_zero_diag(patterns)
    weight_matrix = patterns*patterns'/size(patterns,1);
    weight_matrix = weight_matrix - diag(diag(weight_matrix));
end

function patterns = generate_patterns(p, N)
    % Generates a p by N matrix with p N-bit patterns as columns
    patterns = 2.*randi([0 1],N, p) - 1;
end

function state = update_neuron_asynchronously(weight_matrix, state,
 neuron_number)
    b = weight_matrix(neuron_number,:)*state;
    state(neuron_number) = generate_neuron_value(b);
end
```

Published with MATLAB® R2020a