Homework 1: One-step error probability

Problem description:

Write a computer program implementing asynchronous deterministic dynamics [Eq. (1.9) in the course book] for a Hopfield network. Use Hebb's rule with $w_{ii} = 0$ [Eq. (2.26) in the course book]. Generate and store $p = [12, 24, 48, 70, 100, 120]$ random patterns with $N = 120$ bits. Each bit is either +1 or -1 with probability $\frac{1}{2}$.
For each value of p, estimate the one-step error probability $P_{\text{error}}^{t=1}$ based on 10^5 independent trials. Here, one trial means that you generate and store a set of p random patterns, feed one of them, and perform one asynchronous update of a single randomly chosen neuron. If in some trials you encounter $\text{sgn}(0)$, simply set $\text{sgn}(0) = 1$.
List below the values of $P_{\text{error}}^{t=1}$ that you obtained in the following form: $[p_1, p_2, \ldots, p_6]$, where p_n is the value of $P_{\text{error}}^{t=1}$ for the n-th value of p from the list above. Give four decimal places for each p_n.

Hebb's rule

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} x_i^{(\mu)} x_j^{(\mu)} \quad \text{for} \quad i \neq j, \quad w_{ii} = 0, \quad \text{and} \quad \theta_i = 0. \qquad (2.26)$$

Asynchronous deterministic updates

$$s_i(t+1) = \begin{cases} g\left(\sum_j w_{mj} s_j(t) - \theta_m\right) & \text{for } i = m, \\ s_i(t) & \text{otherwise.} \end{cases} \qquad (1.9)$$

Initial conditions:

```
clear
p = [12 24 48 70 100 120]; % 24 48 70 100 120 amount of stored patterns
N = 120; % amount of neurons
nTrials = 10e5; % numver of trials
```

Q1: Hebb's rule setting the diagonal weights to zero

```
% loop through all given numbers
diagonal_zero = true;
P_error_list = [];
for i = 1:length(p)
    P_error_list(end+1) = P_err_cal(p(i), N, nTrials, diagonal_zero);
end
```

```
disp("P_error with diagonal elements are 0: ")
```

```
P_error with diagonal elements are 0:
```

```
out=['[' sprintf(' %.4f, ', P_error_list(1:end-1) ) num2str(P_error_list(end)) ']'];
```

```
disp(out)
```

```
[ 0.0005,  0.0114,  0.0560,  0.0947,  0.1363, 0.15848]
```

Q2: Hebb's rule without setting the diagonal weights to zero

```
diagonal_zero = false;
P_error_list_dz = [];
for i = 1:length(p)
    P_error_list_dz(end+1) = P_err_cal(p(i), N, nTrials, diagonal_zero);
end
```

```
disp("P_error with diagonal elements not set to 0: ")
```

```
P_error with diagonal elements not set to 0:
```

```
out=['[' sprintf(' %.4f, ', P_error_list_dz(1:end-1) ) num2str(P_error_list_dz(end)) ']'];
disp(out)
```

```
[ 0.0001,  0.0031,  0.0124,  0.0179,  0.0222, 0.022872]
```

One-step error calculation function

```
function P_error = P_err_cal(p, N, nTrials, diagonal_zero)
    error = 0;
    for trial = 1:nTrials
        % Generate random patterns 120*p
        patterns_generated = 2 * randi([0, 1], N, p) - 1;
        % Initialize weight matrix NxN with Hebb's rule
        W = zeros(N);
        for i = 1:p
            W = W + patterns_generated(:,i)*patterns_generated(:,i)'/N;
        end

        % set diagonal elements to 0
        if diagonal_zero == true
            for i = 1:N
                W(i,i) = 0;
            end
        end

        % choose a random pattern in the stored patterns 120*1
        pr = patterns_generated(:,randi(p));
        % choose a bit
        nr = randi(N);

        b_nr = W(nr,:)*pr;
        if b_nr == 0
            x_nr = 1;
        else
```

```
            x_nr = sign(b_nr);
        end

        % check if the updated bit equals the origin bit
        if x_nr ~= pr(nr)
            error = error + 1;
        end
    end
    P_error = error/nTrials;
end
```