Amrita Shah AMS 332 Spring 2023

Unit 5: Neural coding, networks, plasticity and learning.

Project 5 Report

Part A: Poisson spike trains and basic measures of neural activity

1. N = 50 Poisson spike trains were generated.
2. Raster plot

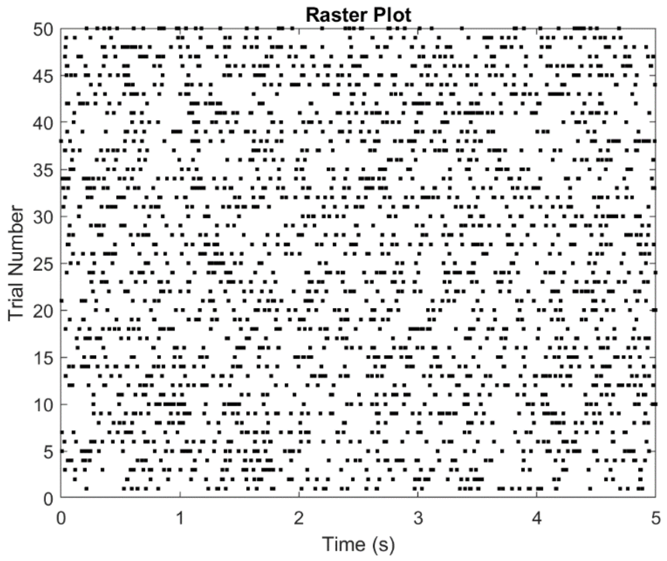


Figure 1.1: A raster plot showing each of the 50 spike trains on a different horizontal line. It shows the spike times in seconds for each trial.

3. The average firing rate across trials was calculated for the whole simulation interval. The total number of spikes was divided by the number of spike trains and the interval duration. The average firing rate was 10.0160 spikes/s. This is very close to the theoretical firing rate λ, which is 10 spikes/s.

4. Peri-stimulus time histogram (PSTH)

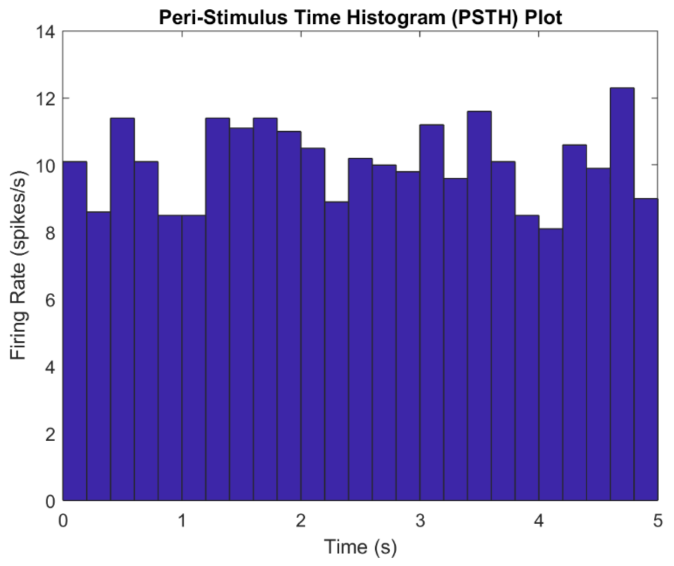


Figure 1.2: PSTH plot showing the firing rate in spikes/s for each 0.2 s time bin. This was calculated by dividing the number of spikes in each time bin across all N trials by N and the duration of each time bin.

5. Coefficient of variability (CV)

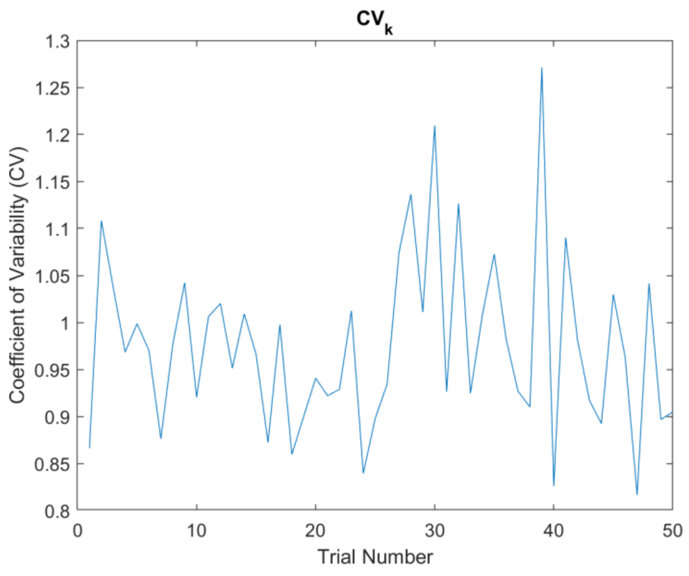


Figure 1.3: Plot of the CV as a function of trial number. The CV of the inter-spike intervals (ISIs) in each trial were calculated by dividing the ISI standard deviation by the mean ISI.

The average CV was calculated to be 0.9984, which is approximately equal to the theoretical CV of 1 for Poisson spike trains.

6. The trial-to-trial variability of the dataset can be quantified by the Fano Factor (FF). The FF is the ratio of the variance of the spike count to the mean spike count. The FF was calculated to be 1.0407, which is approximately equal to theoretical FF of 1 for Poisson spike trains.
7. The CV and FF calculations were repeated for 50-spike train simulations with increasing interval durations: T = {5, 10, 100, 200, 400, 800, 1600, 3200, 6400}.

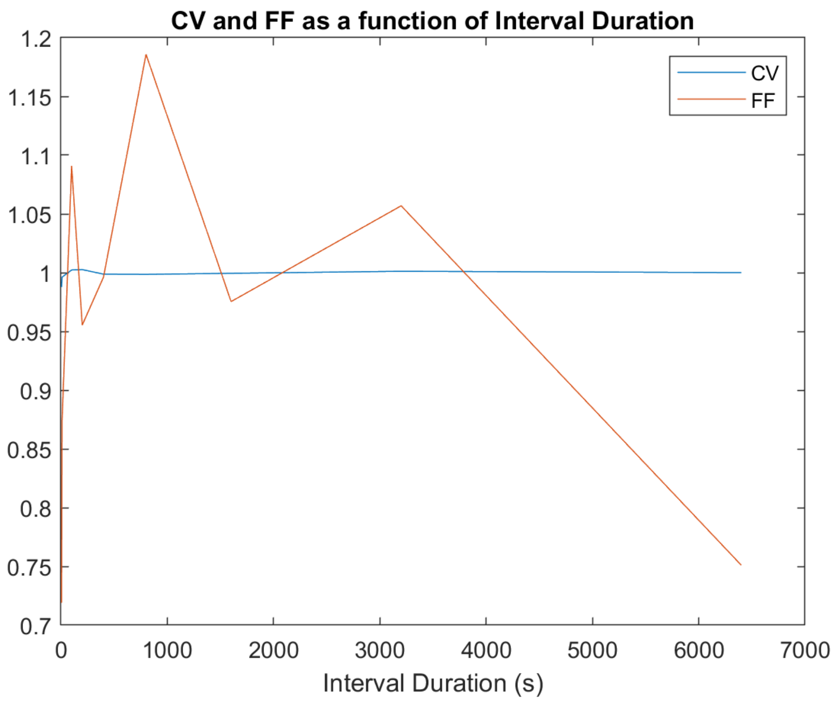


Figure 1.4: CV and FF were plotted as a function of interval duration in seconds.

The coefficient of variability visibly converges to 1 as interval duration increases from 5 to 6400 seconds. The Fano factor does not converge to 1, as can be seen each time the simulation is run.

Part B: Perceptron learning

1. Boolean functions

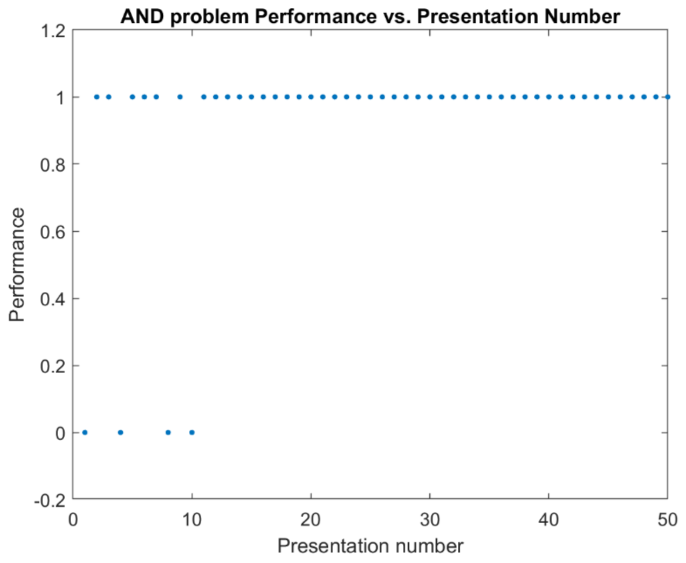


Figure 2.1: The plot of performance as a function of presentation number for the AND problem. A +1 represents a correct classification and 0 represents a misclassification after each pattern presentation.

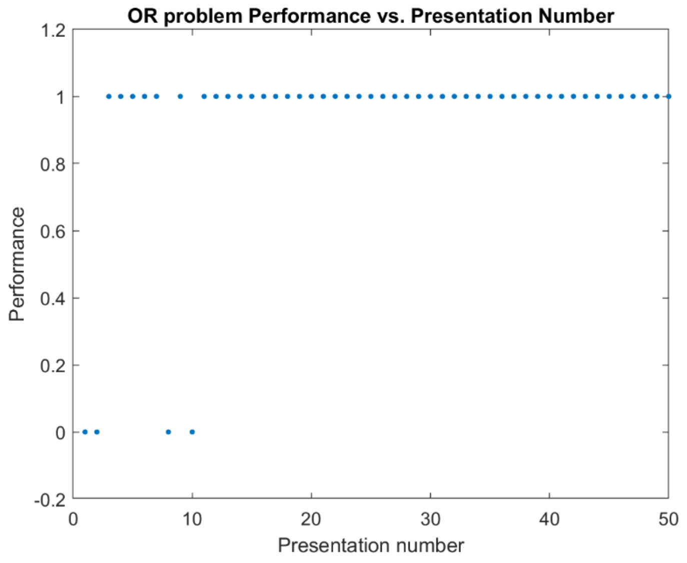


Figure 2.2: The plot of performance as a function of presentation number for the OR problem. A +1 represents a correct classification and 0 represents a misclassification after each pattern presentation.

(i) In a typical simulation, it takes less than 15 steps to converge to the solution for the AND and the OR problems.

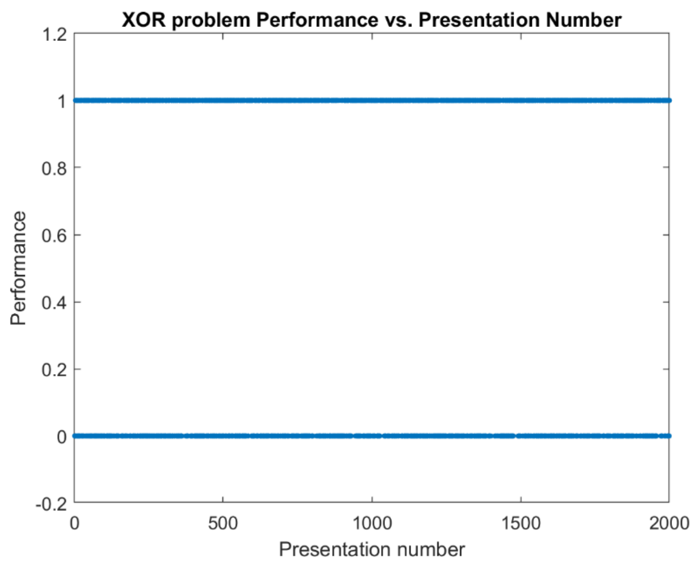


Figure 2.3: The plot of performance as a function of presentation number for the XOR problem. A +1 represents a correct classification and 0 represents a misclassification after each pattern presentation.

(ii) In the case of the XOR problem, the algorithm does not converge to perfect performance. Instead, some input patterns are incorrectly classified and others are not. This set of patterns is not linearly separable under the XOR rule.

2. Random patterns

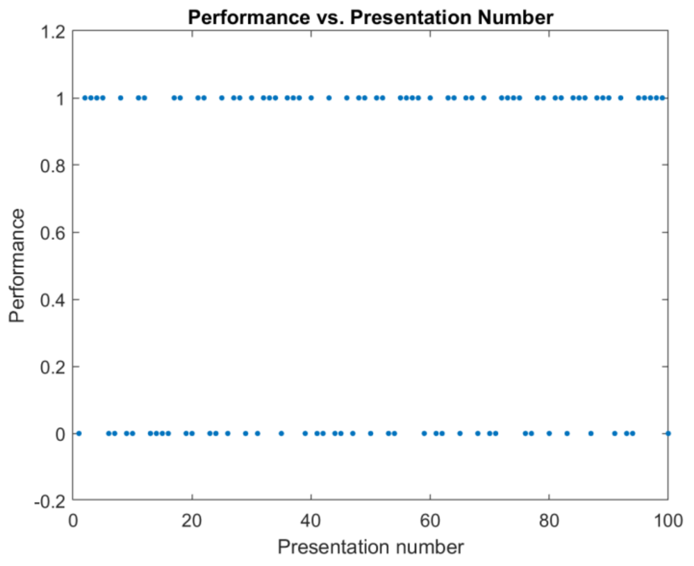


Figure 2.4: Plot of performance as a function of presentation number for separation into classes after 100 steps.

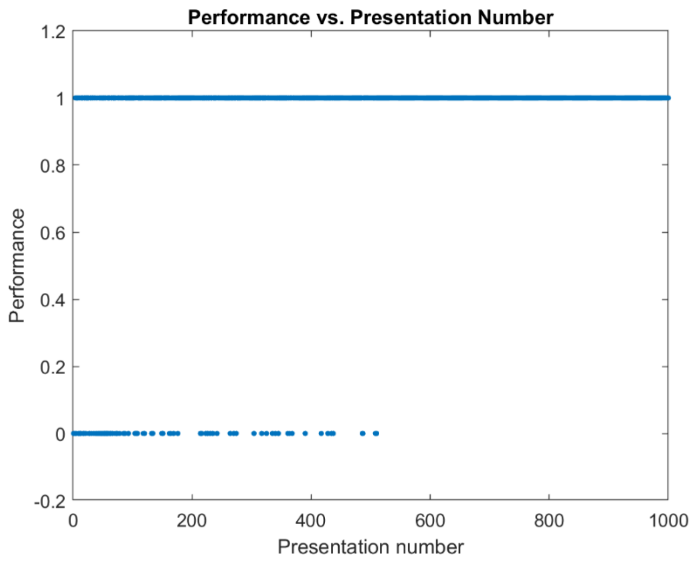


Figure 2.5: Plot of performance as a function of presentation number for separation into classes after 1000 steps.

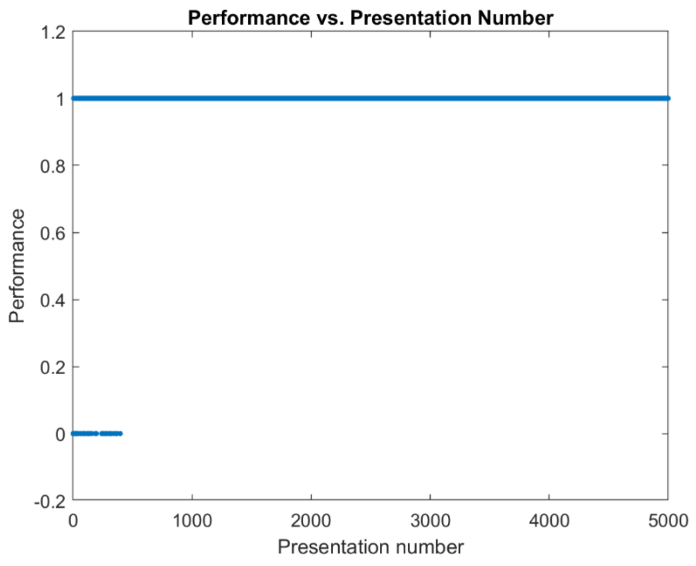


Figure 2.6: Plot of performance as a function of presentation number for separation into classes after 5000 steps.

(i) Based on Figures 2.4, 2.5, and 2.6, the perceptron learned to separate the inputs in the desired classes in roughly 500 steps. 100 steps were not enough, 1000 was an appropriate number, and 5000 were likely too many.

(ii) The random vectors are linearly separable, because it almost always converged to a solution that correctly separated the data points. It generated perfect performance in almost every run.

(iii) If a different learning rate is used, the results change slightly but not much. Generally, comparing learning rates of 0.1 (Figure 2.7) and 10 (Figure 2.8), a slower learning rate of 0.1 caused convergence to occur after more steps. An interesting observation was that one run with a learning rate of 0.1 (not shown) did not converge in 1000 steps, so it is possible that in that run it was not linearly separable.

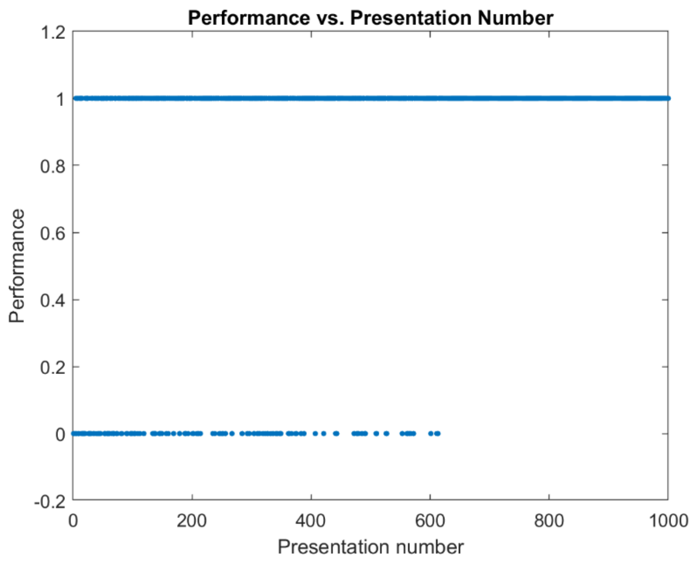


Figure 2.7: Plot of performance as a function of presentation number for separation into classes with a learning rate of 0.1.

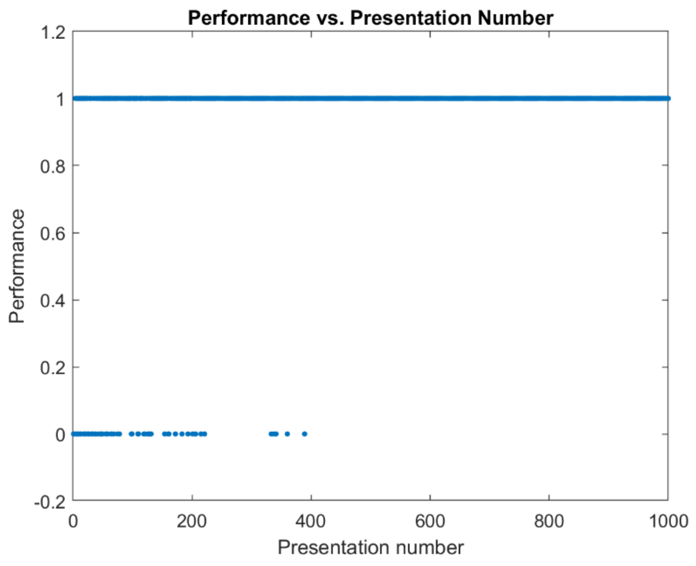


Figure 2.8: Plot of performance as a function of presentation number for separation into classes with a learning rate of 10.

(iv) There were no runs for which convergence was not reached. The average number of pattern presentations until convergence, or $\langle n_{conv} \rangle_{runs}$ was calculated to be 693.4 over 30 runs.

(v) Repeating (iv), $100 \times M$ presentations for $M = 40$ yields a mean number of presentations for each pattern of $\langle n_{conv} \rangle_{runs}/M$ = 20.64. $1000 \times M$ presentations for $M = 90$ yields a mean number of presentations for each pattern of $\langle n_{conv} \rangle_{runs}/M$ = 411.6. It should be noted that of 20 runs, only 12 converged when $M = 90$.

Contents

Part A: Poisson spike trains and basic measures of neural activity

```
clear
close all
```

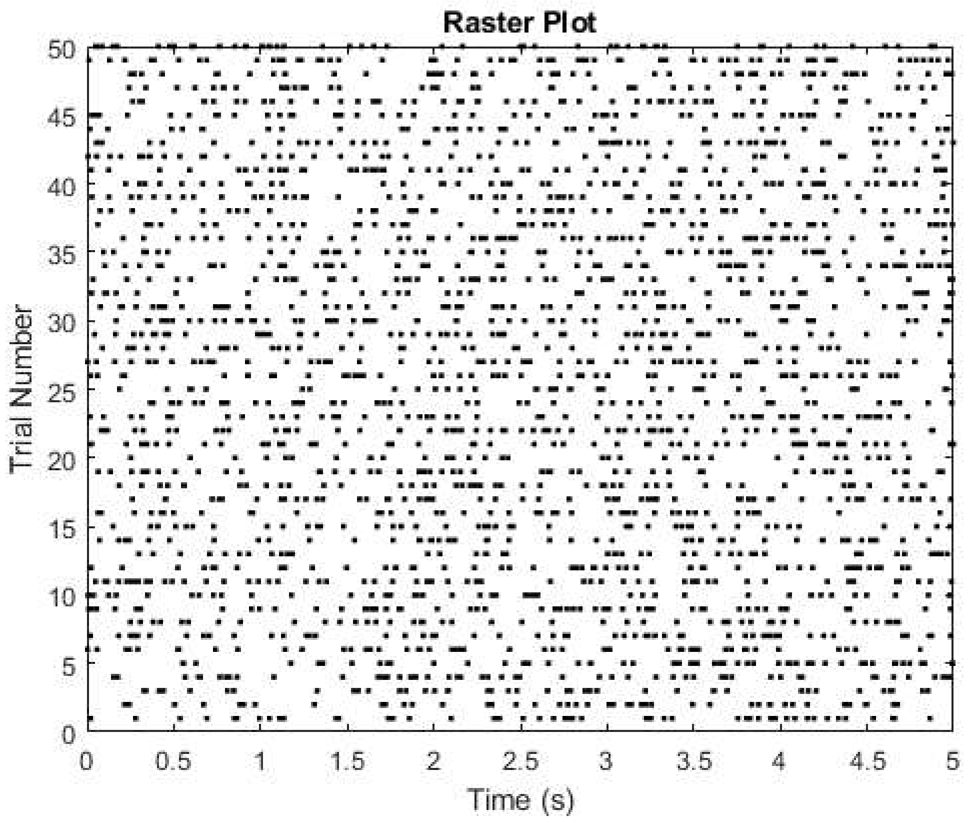
1-6

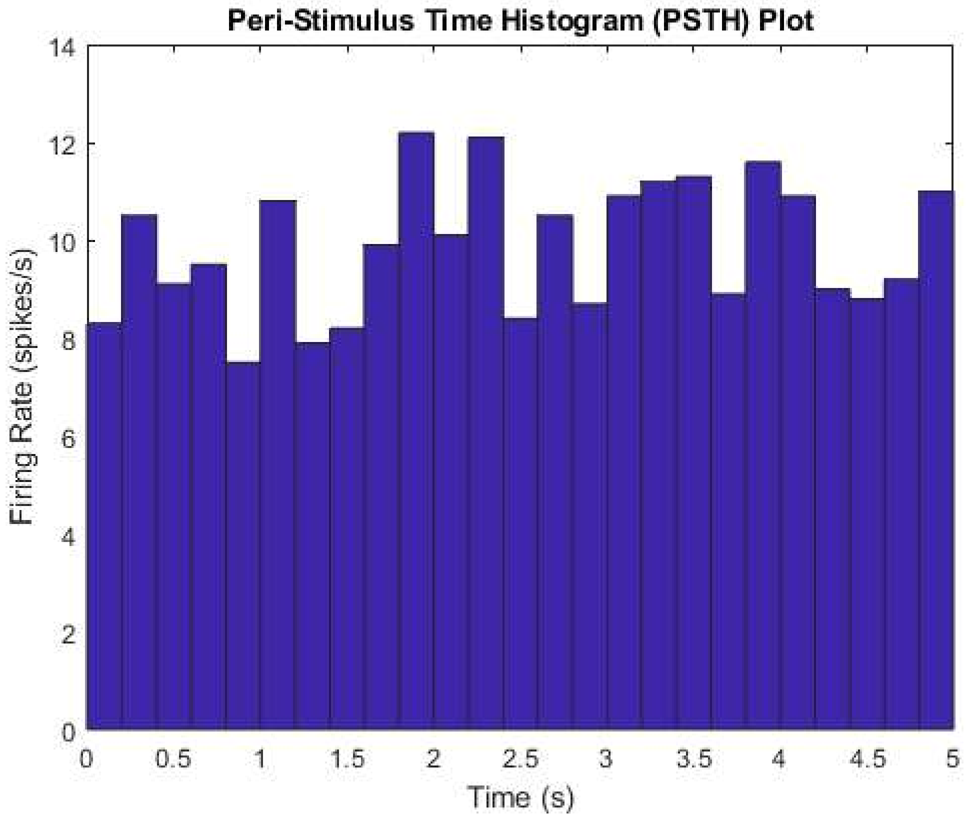
1. Spike trains

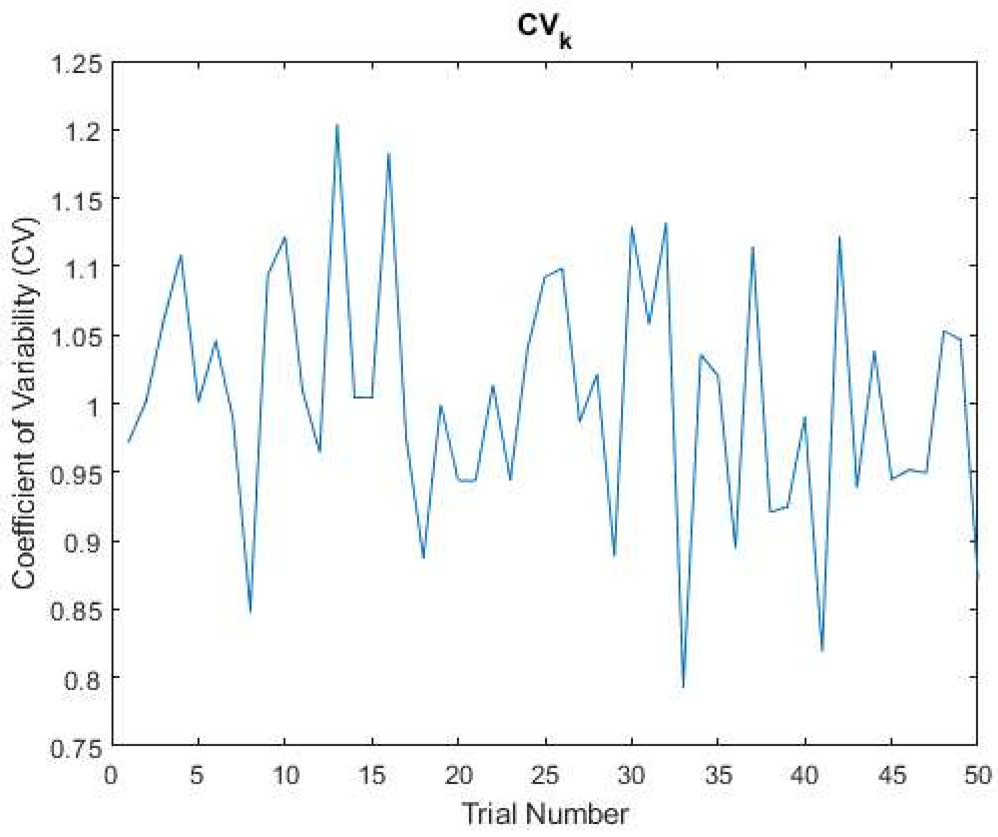
```
T = 5; % s
N = 50; % spikes
lambda = 10; % spikes/s
ISI = zeros((2*lambda*T),N);
S = zeros((2*lambda*T), N);
for i = 1:N
    u = rand((2*lambda*T),1);
    ISI(:, i) = -log(u) / lambda;
    S(:, i) = cumsum(ISI(:, i));
end
S(S>T)=NaN;

% 2. Raster plot
fig1 = figure(1);
plot(S, 1:N, '.k')
xlabel('Time (s)')
ylabel('Trial Number')
title('Raster Plot')

% 3. Firing rate
f = sum(~isnan(S), 'all') / (N*T);

% 4. PSTH
fTi = [];
Ti = 0.1:0.2:T;
for i = 0:0.2:(T-0.2)
    fTi(end+1) = sum((S>=i & S<(i+0.2)), 'all');
end
fTi = fTi ./ (N*0.2);
fig2 = figure(2);
bar(Ti, fTi, 'hist')
xlim([0 T])
xlabel('Time (s)')
ylabel('Firing Rate (spikes/s)')
title('Peri-Stimulus Time Histogram (PSTH) Plot')

% 5. CV
CVk = zeros(1, N);
for k = 1:N
```

```
    CVk(k) = std(ISI(:, k)) / mean(ISI(:, k));
end
k = 1:N;
fig3 = figure(3);
plot(k, CVk)
xlabel('Trial Number')
ylabel('Coefficient of Variability (CV)')
title('CV_{k}')
CV = mean(CVk);

% 6. Fano factor
spike_count = zeros(1, N);
for k = 1:N
    spike_count(k) = sum(~isnan(S(:, k)))';
end
FF = var(spike_count) / mean(spike_count);

f
CV
FF
```

```
f =

    9.8600


CV =

    1.0038


FF =

    0.9010
```

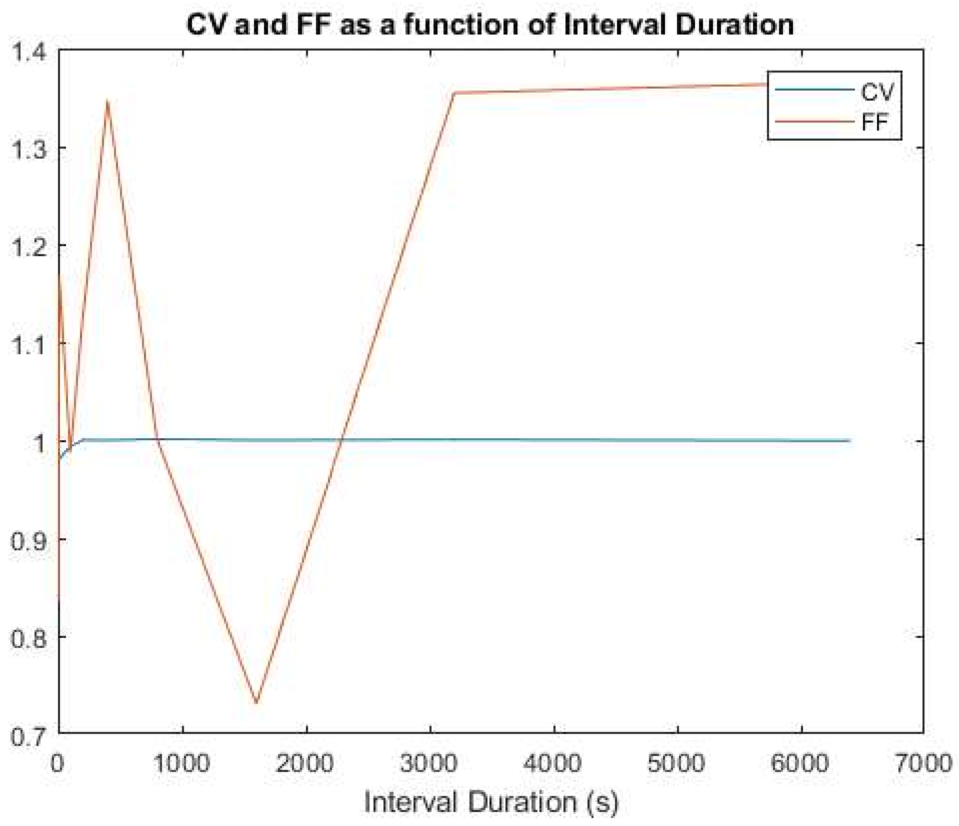

Raster Plot
Trial Number
0
5
10
15
20
25
30
35
40
45
50
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Time (s)


Peri-Stimulus Time Histogram (PSTH) Plot
Firing Rate (spikes/s)
0
2
4
6
8
10
12
14
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Time (s)



7. Repeat 5 & 6

```
durations = [5, 10, 100, 200, 400, 800, 1600, 3200, 6400];
CV_vector = zeros(size(durations));
FF_vector = zeros(size(durations));

for d = 1:length(durations)
    % 1. Spike trains
    T = durations(d); % s
    N = 50; % spikes
    lambda = 10; % spikes/s
    ISI = zeros((2*lambda*T),N);
    S = zeros((2*lambda*T), N);
    for i = 1:N
        u = rand((2*lambda*T),1);
        ISI(:, i) = -log(u) / lambda;
        S(:, i) = cumsum(ISI(:, i));
    end
    S(S>T)=NaN;

    % 3. Firing rate
    f = sum(~isnan(S), 'all') / (N*T);

    % 5. CV
    CVk = zeros(1, N);
    for k = 1:N
        CVk(k) = std(ISI(:, k)) / mean(ISI(:, k));
    end
    k = 1:N;
    CV = mean(CVk);
    CV_vector(d) = CV;
    % 6. Fano factor
    spike_count = zeros(1, N);
    for k = 1:N
        spike_count(k) = sum(~isnan(S(:, k)))';
    end
    FF = var(spike_count) / mean(spike_count);
    FF_vector(d) = FF;

end
fig4 = figure(4);
plot(durations, CV_vector)
hold on
plot(durations, FF_vector)
xlabel('Interval Duration (s)')
legend('CV', 'FF')
title('CV and FF as a function of Interval Duration')
hold off
```



Published with MATLAB® R2022b

Part B: Perceptron learning

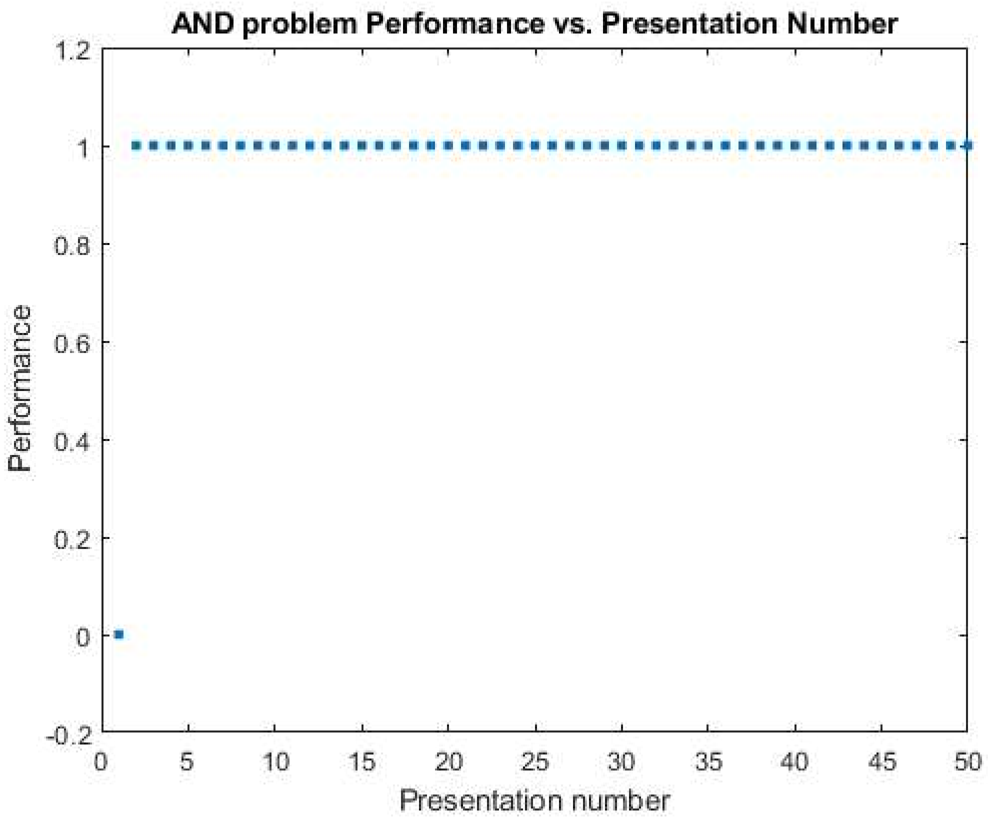
Contents

1. AND problem

```
clear

N = 2; % input units
eta = 1; % learning rate
w = zeros(1, N+1); % initial synaptic weights
inputs = [1 -1 1 -1; 1 1 -1 -1; -1 -1 -1 -1]; % input patterns (with x3=-1)
presentations = 50;
performance = zeros(1, presentations);

for i = 1:presentations
    x = inputs(:, (randi(size(inputs, 2)))); % select an input pattern
    y = sign(dot(w, x)); % compute output

    if x(1)==1 && x(2)==1
        yt_and = 1;
    else
        yt_and = -1;
    end

    dw1 = eta*(yt_and-y)*x(1);
    dw2 = eta*(yt_and-y)*x(2);
    dw3 = eta*(yt_and-y)*x(3);

    w(1) = w(1) + dw1;
    w(2) = w(2) + dw2;
    w(3) = w(3) + dw3;

    if y == yt_and
        performance(i) = 1;
    else
        performance(i) = 0;
    end
end

% plot performance vs presentation number
fig1 = figure(1);
plot(1:presentations, performance, '.', 'MarkerSize', 10);
xlabel('Presentation number');
ylabel('Performance');
ylim([-0.2 1.2]);
title('AND problem Performance vs. Presentation Number')
```

```
%test
for i = 1:size(inputs, 2)
    x = inputs(:, i);
    y = sign(dot(w, x));
    fprintf('Input pattern: [%d %d %d], Output: %d\n', x(1), x(2), x(3), y);
end
```

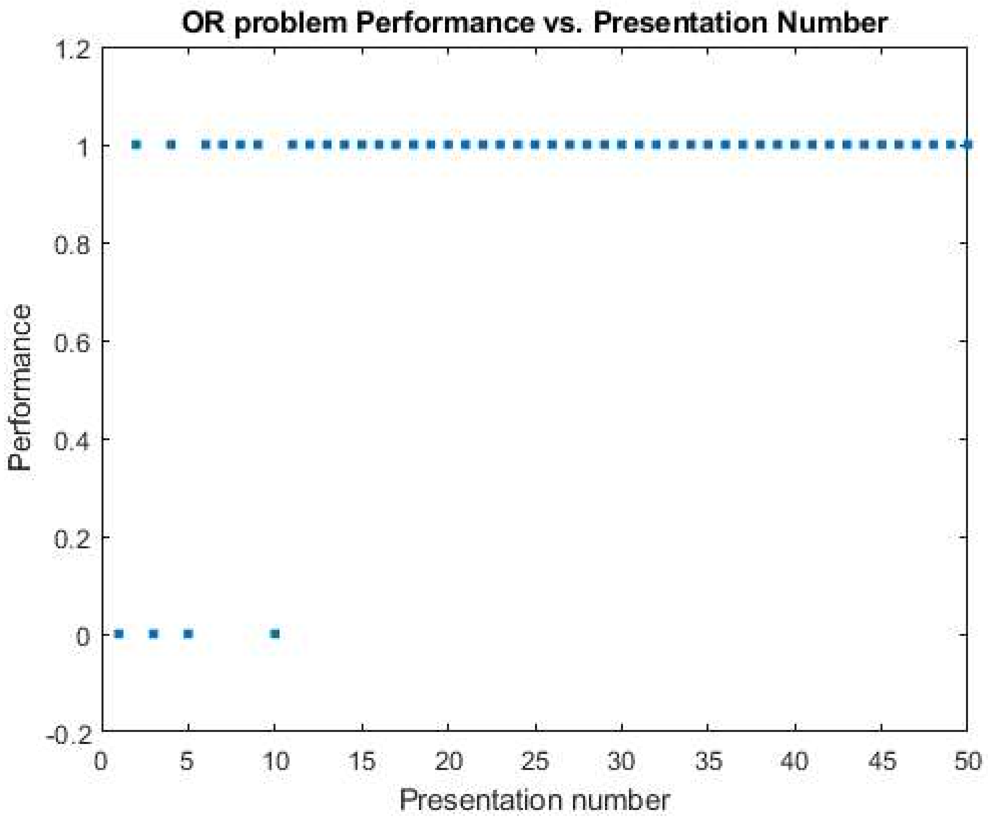
```
Input pattern: [1 1 -1], Output: 1
Input pattern: [-1 1 -1], Output: -1
Input pattern: [1 -1 -1], Output: -1
Input pattern: [-1 -1 -1], Output: -1
```



OR problem

```
clear

N = 2; % input units
eta = 1; % learning rate
w = zeros(1, N+1); % initial synaptic weights
inputs = [1 -1 1 -1; 1 1 -1 -1; -1 -1 -1 -1]; % input patterns (with x3=-1)
presentations = 50;
performance = zeros(1, presentations);

for i = 1:presentations
    x = inputs(:, (randi(size(inputs, 2)))); % select an input pattern
    y = sign(dot(w, x)); % compute output

    if x(1)==1 || x(2)==1
        yt_or = 1;
    else
        yt_or = -1;
    end

    dw1 = eta*(yt_or-y)*x(1);
    dw2 = eta*(yt_or-y)*x(2);
    dw3 = eta*(yt_or-y)*x(3);

    w(1) = w(1) + dw1;
    w(2) = w(2) + dw2;
    w(3) = w(3) + dw3;

    if y == yt_or
        performance(i) = 1;
    else
        performance(i) = 0;
    end
end

% plot performance vs presentation number
fig2 = figure(2);
plot(1:presentations, performance, '.', 'MarkerSize', 10);
xlabel('Presentation number');
ylabel('Performance');
ylim([-0.2 1.2]);
title('OR problem Performance vs. Presentation Number')

%test
for i = 1:size(inputs, 2)
    x = inputs(:, i);
    y = sign(dot(w, x));
    fprintf('Input pattern: [%d %d %d], Output: %d\n', x(1), x(2), x(3), y);
end
```

```
Input pattern: [1 1 -1], Output: 1
Input pattern: [-1 1 -1], Output: 1
Input pattern: [1 -1 -1], Output: 1
Input pattern: [-1 -1 -1], Output: -1
```

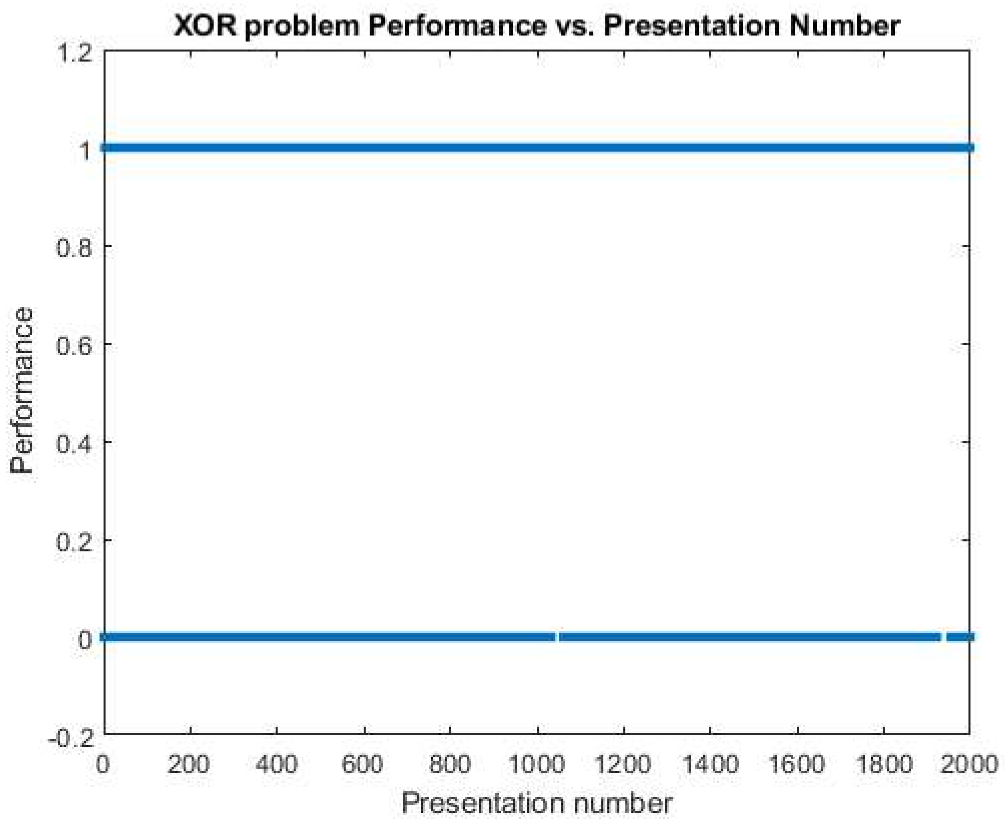


XOR problem

```
clear

N = 2; % input units
eta = 1; % learning rate
w = zeros(1, N+1); % initial synaptic weights
inputs = [1 -1 1 -1; 1 1 -1 -1; -1 -1 -1 -1]; % input patterns (with x3=-1)
presentations = 2000;
performance = zeros(1, presentations);

for i = 1:presentations
    x = inputs(:, (randi(size(inputs, 2)))); % select an input pattern
    y = sign(dot(w, x)); % compute output

    if (x(1)==-1 && x(2)==1) || (x(1)==1 && x(2)==-1)
        yt_xor = 1;
    else
        yt_xor = -1;
    end

    dw1 = eta*(yt_xor-y)*x(1);
    dw2 = eta*(yt_xor-y)*x(2);
    dw3 = eta*(yt_xor-y)*x(3);

    w(1) = w(1) + dw1;
    w(2) = w(2) + dw2;
    w(3) = w(3) + dw3;

    % compute performance
    if y == yt_xor
```

```
        performance(i) = 1;
    else
        performance(i) = 0;
    end
end
% plot performance vs presentation number
fig3 = figure(3);
plot(1:presentations, performance, '.', 'MarkerSize', 10);
xlabel('Presentation number');
ylabel('Performance');
ylim([-0.2 1.2]);
title('XOR problem Performance vs. Presentation Number')

%test
for i = 1:size(inputs, 2)
    x = inputs(:, i);
    y = sign(dot(w, x));
    fprintf('Input pattern: [%d %d %d], XOR Output: %d\n', x(1), x(2), x(3), y);
end
```

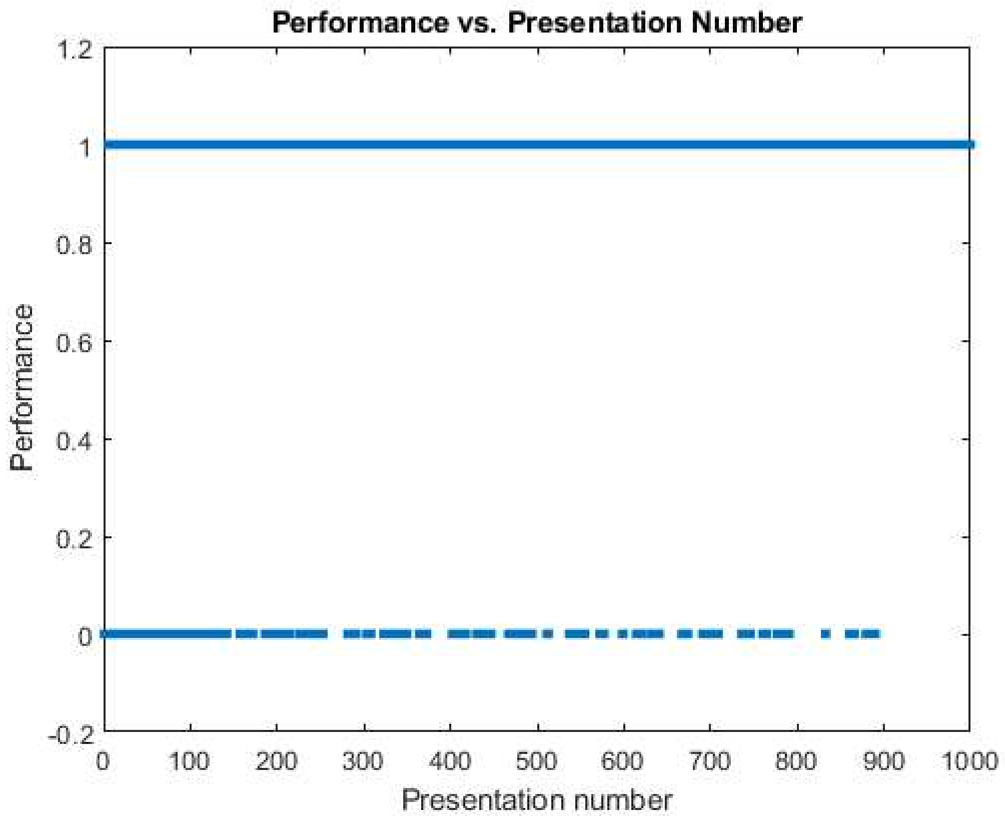
```
Input pattern: [1 1 -1], XOR Output: -1
Input pattern: [-1 1 -1], XOR Output: -1
Input pattern: [1 -1 -1], XOR Output: -1
Input pattern: [-1 -1 -1], XOR Output: 1
```



2. Random patterns

```
clear
M = 40;
```

```
N = 50;
inputs = randi(2,N,M) - 1;
inputs = inputs -1;
eta = 1; % learning rate
w = zeros(N, 1); % initial synaptic weights
classes = [-ones(1,M/2) ones(1,M/2)];
classes = classes(randperm(length(classes)));
presentations = 1000;
performance = zeros(1, presentations);

for i = 1:presentations
    index = randi(size(inputs, 2));
    x = inputs(:,index);
    yt = classes(index);
    y = sign(dot(w, x)); % compute output

    if y == yt
        performance(i) = 1;
    else
        performance(i) = 0;
    end
    for j = 1:N
        dw = eta * (yt - y) * x(j);
        w(j) = w(j) + dw;
    end
end

fig4 = figure(4);
plot(1:presentations, performance, '.', 'MarkerSize', 10);
xlabel('Presentation number');
ylabel('Performance');
ylim([-0.2 1.2]);
title('Performance vs. Presentation Number')
```

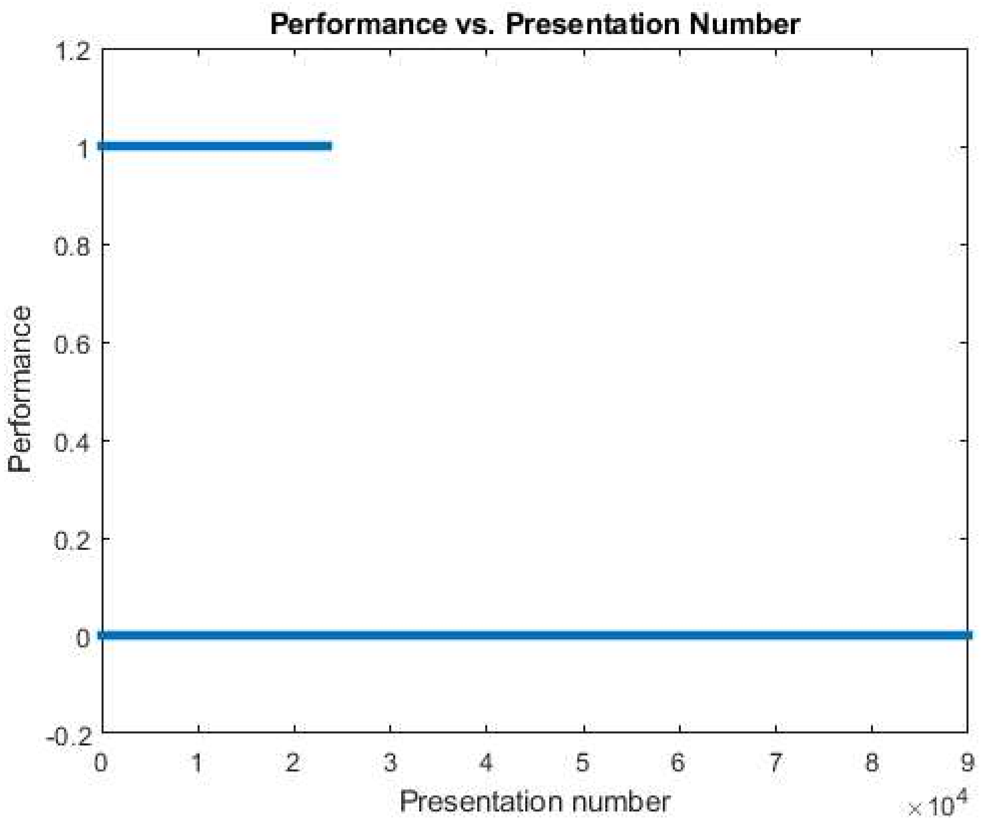


2d.

```
clear
M = 90;
N = 50;
inputs = randi(2,N,M) - 1;
inputs = inputs -1;
eta = 1; % learning rate
w = zeros(N, 1); % initial synaptic weights
classes = [-ones(1,M/2) ones(1,M/2)];
classes = classes(randperm(length(classes)));
presentations = 1000*M;
performance = zeros(1, presentations);

for i = 1:presentations
    index = randi(size(inputs, 2));
    x = inputs(:,index);
    yt = classes(index);
    y = sign(dot(w, x)); % compute output

    if y == yt
        performance(i) = 1;
    else
        performance(i) = 0;
    end
    for j = 1:N
        dw = eta * (yt - y) * x(j);
        w(j) = w(j) + dw;
    end

    if i>200 && all(performance(i-199 : i) == 1)
```

```
        disp(['n_{conv} =', num2str(i)]);
        break
    end
end

fig4 = figure(4);
plot(1:presentations, performance, '.', 'MarkerSize', 10);
xlabel('Presentation number');
ylabel('Performance');
ylim([-0.2 1.2]);
title('Performance vs. Presentation Number')
```

```
n_{conv} =23457
```



Published with MATLAB® R2022b