CHALMERS UNIVERSITY OF TECHNOLOGY

FFR135 Artificial Neural Network

Example_Sheet_2

Yankun Xu (940630-0237)

(yankun@student.chalmers.se)

2017-10-16

1. **One-dimensional Kohonen network**:

In this Kohonen network, we map two-dimensional input patterns into one-dimensional output space with 100 neurons. In each updating, we draw an input pattern $\underline{\xi}$ from P generated input patterns, then we need find the winning neuron i_0 to make $|\underline{\xi} - \underline{w}_{i_0}| \leqslant |\underline{\xi} - \underline{w}_i|$ for all i, which means i_0 is the closest one to the input pattern. Next, we use $\underline{w}_i \leftarrow \underline{w}_i + \delta\underline{w}_i$ to update weights, and $\delta\underline{w}_i = \eta\Lambda(i, i_0)(\underline{\xi} - \underline{w}_i)$, where $\Lambda(i, i_0) = exp(\frac{-|r_i - r_{i_0}|^2}{2\sigma^2})$ is neighboring function which equals 1 for $i = i_0$ and decays for otherwise.

1a:

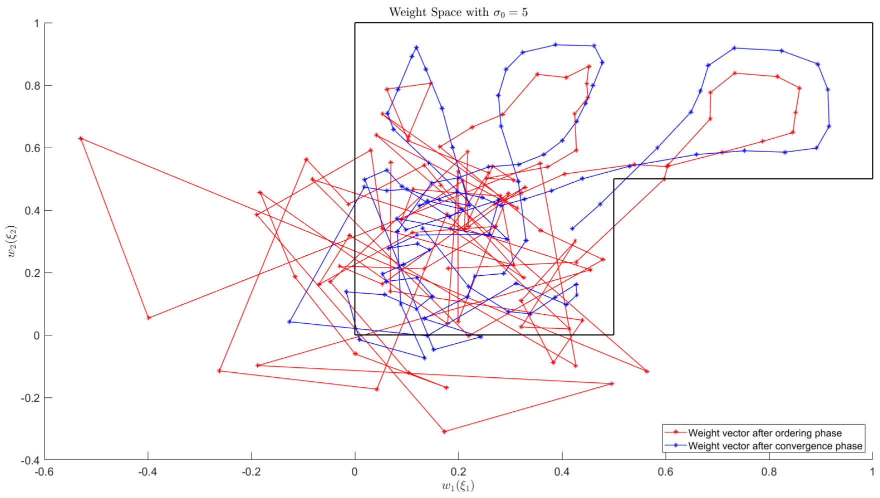


Figure 1: Weight vector space with $\sigma_0 = 100$

In Figure 1, the weight space is not able to learn the properties of input patterns which is two-dimensional uniform distribution between 0 and 1 (except ξ_1 and ξ_2 are both larger than 0.5).
In neighbouring function, the parameter Gaussian width σ decides how far away weights from winning neuron would be updated. In Kohonen network, ordering phase is used to make topological ordering of weight vectors, in this case, $\sigma_0 = 100$ is used in ordering phase initially and decreases during the learning procedure. A larger Gaussian width means we update too many neurons far away from winning neuron, although there is a decreased updating rule for σ, it only converges to 5 finally after 3000 iterations. We can image that we almost always drag too many neurons towards the feed input pattern in ordering phase, thus it is hardly to converge after ordering phase. We can see that the red curve has many components outside input space and its shape looks randomly. In convergence phase, we use quite small constant for σ_{conv} and η_{conv}, and much more iterations in order to adjust topographical map precisely corresponding to input space. Unfortunately, we have a bad result after ordering phase, and convergence phase runs based on this bad result. Therefore, we don't get a good result finally even if the blue curve looks better than red curve.

1b:

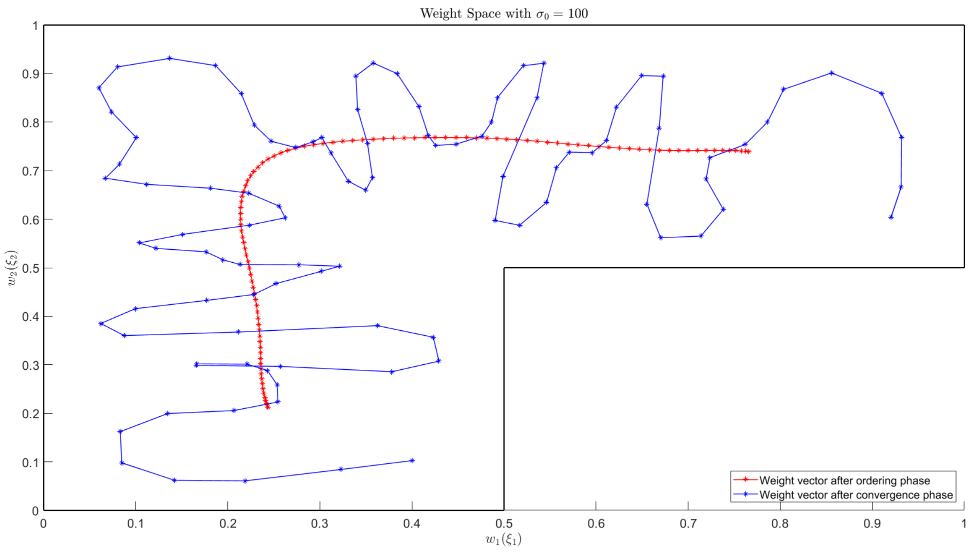


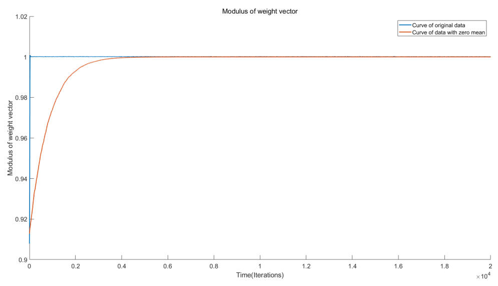
Figure 2: Weight vector space with $\sigma_0 = 5$

Compared to quite larger σ_0 in 1a, $\sigma_0 = 5$ is used in ordering phase initially in this case, which means we update winning neuron and neurons very closed to winning neuron, because 5 is small, which is final converged value in 1a, and it converges to 0.25 after 3000 iterations. After learning, weight vectors are successfully mapped from uniform distribution between -1 and 1 into input space, and its one-dimensional shape also looks like input space roughly(a broken line), which is shown as red curve in Figure 2.

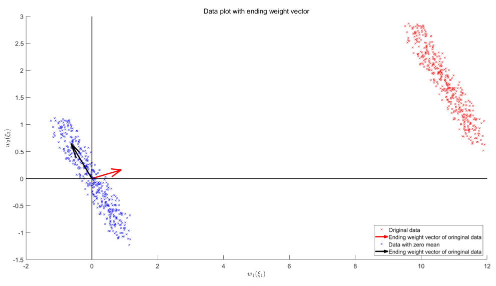
In convergence phase, we set the same parameters as 1a used, this could make topographical map represent the input space much more precisely, because we always only update winning neuron and closed neuron in small learning step. Luckily, we have a good result after ordering phase this time, due to the property of uniform distribution in input space, we find that after convergence phase the output space is able to fill the input space good enough in one-dimension, and coincidentally the shape of this line looks like *Peano Curve*, which is shown as blue curve in Figure 2.

2. Unsupervised Oja's learning with one linear unit:

Oja's rule can be considered as a improvement for simple hebbian learning rule because it could prevent the updating weight vector from blowing up. Oja's rule adjust weight vector as $\underline{w} \leftarrow \underline{w} + \delta \underline{w}$, where $\delta \underline{w} = \eta\zeta(\underline{\xi} - \zeta \underline{w})$.



(a) $|\underline{w}|$ over time



(b) Data plot with ending weight vector

Figure 3: Result for dataset without and with zero mean

We train our data without and with zero mean separately. In Figure 2a and 2b, we can see that weight vectors converge to 1 in both two cases, but the direction of weight vector and the time to converge are different. There are three properties when learning rules converges to $\underline{w}^*$: (a) $|\underline{w}^*| = 1$, (b) $\underline{w}^*$ is eigenvector of $\mathbb{C}'$ with maximal eigenvalue (where $\mathbb{C}' =< \underline{\xi}^T, \underline{\xi} >$), (c) $\underline{w}^*$ maximise $< \zeta^2 >$.(Lecture notes, p151)
Let's consider data with zero mean firstly, in this case, $< \xi >= 0$ have $< \zeta >= 0$, Oja's rule maximises $< \zeta^2 >$ and finds the maximal principal direction, which is eigenvector of $\mathbb{C}'$ with maximal eigenvalue, shown as black vector in Figure 3b. In the case of data without zero mean ($< \xi > \neq 0$), however, Oja's rule still maximises $< \zeta^2 >$, as we know $\zeta = \underline{w}^T \underline{\xi}$, so this ending vector $\underline{w}^*$ in the direction of the data cluster rather than the direction of maximal principal component. Furthermore, due to the weight decays, the modulus of $\underline{w}^*$ equals to 1 in both cases. As to the time to converge, data without zero mean converges faster, when we look at the Oja's rule, in each step we drag weight vector towords the feed input pattern, and the direction of vector is closer to the eigenvector of $\mathbb{C}'$ with maximal eigenvalue, the modulus of vector is closer to 1. All of the data without zero mean cluster in the top left of the original point, all feed pattern drag weight vector towards this area, but in the case of data with zero mean, input patters are surrounding the original point, it must be waste of time to learning.
In Figure 3b, it is obvious to use data with zero mean to find maximal principal component direction of input data. $\mathbb{C}'$ of this data is $[137, -130; -130, 142]$, and the maximal principal component direction is [-0.6987;0.7154] for maximal eigenvalue 270. In practical, we have $[-0.9865; -0.1641]$ in case of original data and $[-0.6963; 0.7177]$ in case of data with zero mean, we can see that result of experiment using data with zero mean is almost the same as theoretical value. By the way, If we look at the $\mathbb{C}'$ of original data, $[-0.9865; -0.1641]$ follows the direction of eigenvector with maximal eigenvalue, but it is not in the direction of maximal principal component.

3. Unsupervised simple competitive learning combined with supervised simple perceptron:

In this task, we combine unsupervised simple competitive learning and supervised learning together to perform classification task, unsupervised simple competitive learning can help us cluster the data and then use simple perceptron method to make classification. In simple competitive learning, for each update, we find winning neuron i_0 such that $g_{i_0}(\underline{x}) \geqslant g_i(\underline{x})$, we update $\underline{w}_{i_0} \leftarrow \underline{w}_{i_0} + \delta\underline{w}_{i_0}$, where $\delta\underline{w}_{i_0} = \eta(\underline{x} - \underline{w}_{i_0})$. By the way, in this task, I use stochastic gradient descent(SGD) during the supervised learning.

3a:

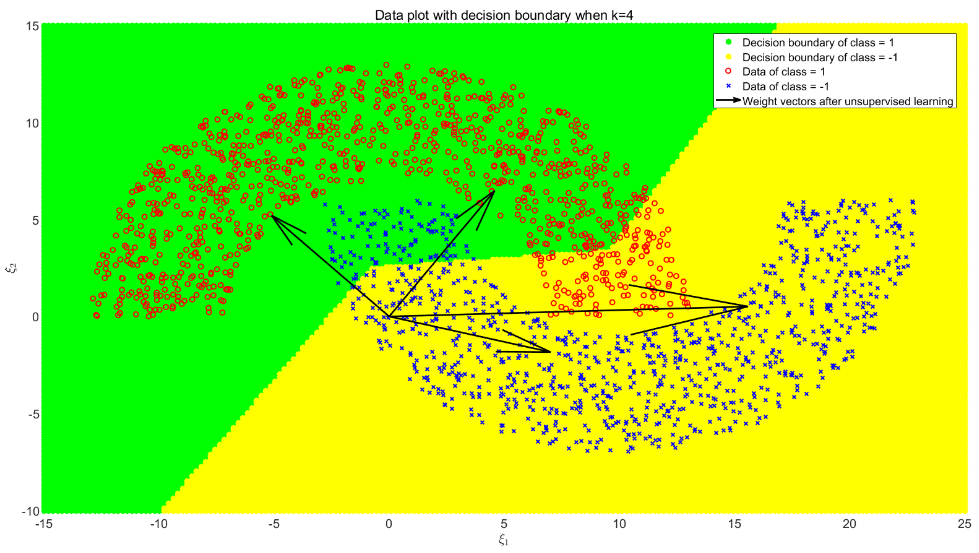


Figure 4: Data plot with decision boundary when k=4

When $k = 4$, the average classification error among 20 runs is 0.148, it means this network performs good but not perfectly. When we look at Figure 4, we can see that the data in the centre part of figure is hard to classify, and this network also poorly classify these area data.

During the unsupervised simple competitive learning, network is clustering the data into k clusters, in this case k=4. In Figure 4, the black vectors point to each cluster after unsupervised learning, this network is blind to the properties of these 4 cluster data, if we want to make classification task, we need add a simple supervised learning to tell the network which cluster is class 1 or -1. According to the black vectors shown in figure, we know that 4 clusters should not be enough to represent all data, what part the decision boundary classify wrongly is also the part black vectors miss to cluster. If you investigate the shape of decision boundary, it looks like a combination of three linear lines.

3b:

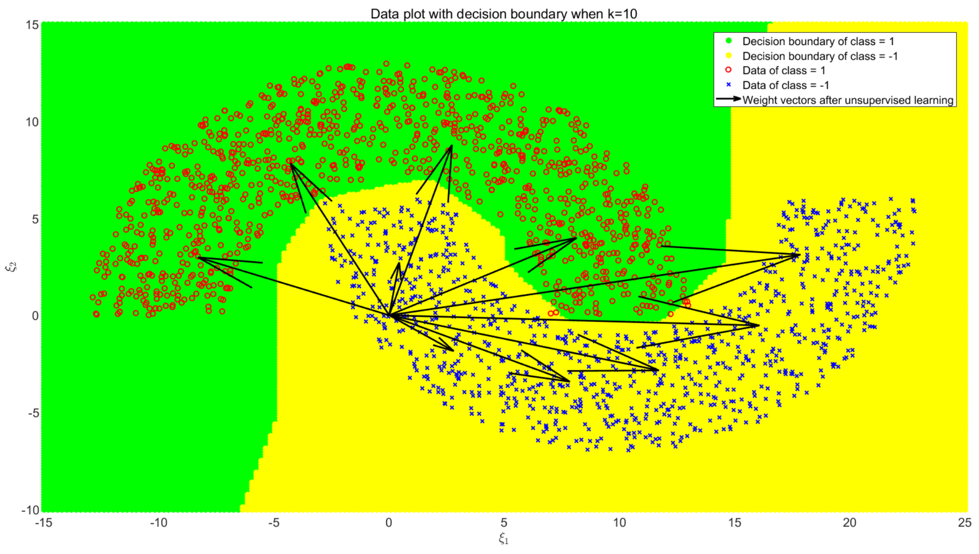


Figure 5: Data plot with decision boundary when k=10

In case of $k = 10$, the average classification error among 20 runs is 0.007, it means this network perform perfectly. Compared to 3a, the decision boundary of this network looks very nice and makes classification of data much better. When we look at black vectors, the network is clustering the data into 10 clusters, especially in the centre part of the figure, the data in that area, which is ignored to be clustered in 3a, is also clustered, it is important for next step using simple supervised preceptron to make classification.

If you investigate the shape of decision boundary, it looks like a combination of ten linear lines, plus the investigation in 3a, it should not be a coincidence. According to my knowledge, I think it seems be similar with the task3 in example_sheet1, the layer with k neurons in this case is similar with the hidden layer in the supervised learning with one hidden layer, more neurons in hidden layer you have, better the network will perform.

3c:

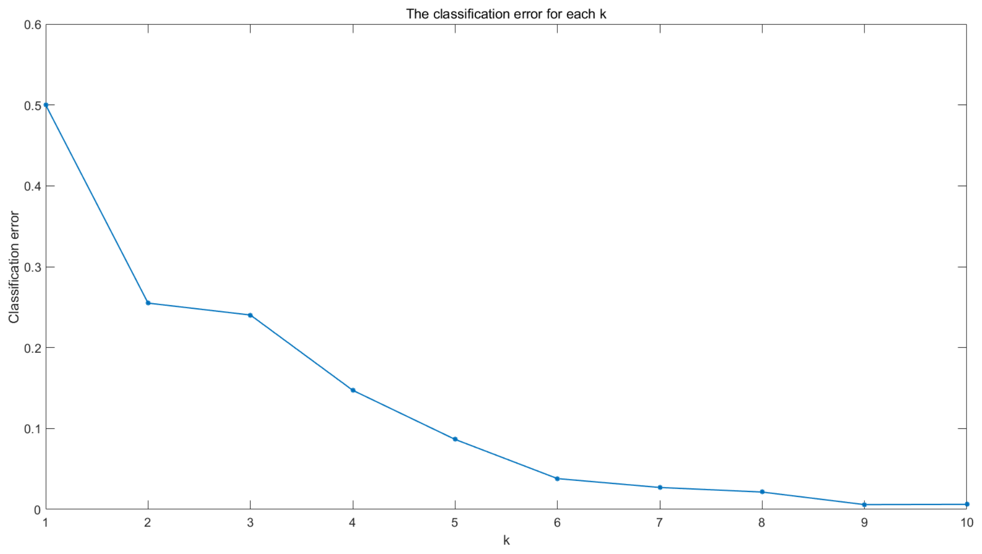


Figure 6: Classification error for different k

Figure 6 is the average classification error among 20 runs for each number of Gaussian neurons (from 1 to 10). We can see that, when $k = 1$ the netwrok does not work. Along with increasing of k, classification error decreases significantly. It means if we could cluster the data into more clusters during the unsupervised simple competitive learning, we would achieve much better performance. When $k \geqslant 5$, the network works good enough because the classification error will be less than 0.1, and when $k \geqslant 9$, classification error is converged, thus the network reach its optimal performance.

Appendix: Codes for tasks in Matlab

Task1:

```
clear;clc;clf;close all

% generate data
N_big = 1500;
N = 1000;
x1 = rand(N_big,1);
x2 = rand(N_big,1);
index = find(x1 > 0.5 & x2 <= 0.5);
x1(index) = [];
x2(index) = [];
data = [x1(1:N,:),x2(1:N,:)];

% initialization
T_order = 1e3;
sigma_0 = 100; % = 100 for 3a, = 5 for 3b
lr_0 = 0.1;
tau_sigma = 300;

T_conv = 2e4;
sigma_conv = 0.9;
lr_conv = 0.01;

neurons = 100;
weights = 2*rand(2,neurons) - 1; % 2*100

% order phase
for t = 1:T_order
    sigma = sigma_0 * exp(-t/tau_sigma);
    lr = lr_0 * exp(-t/tau_sigma);

    mu = randi([1,N]);
    pattern = data(mu,:)'; % 2*1
    dist = zeros(1,neurons);
    for i = 1:neurons
        dist(i) = norm(pattern - weights(:,i));
    end
    win_index = find(dist == (min(dist)));

    delta_weights = zeros(size(weights));
    for j = 1:neurons
        delta_weights(:,j) = lr * exp(-0.5 * abs(j - win_index)^2/ sigma^2)...
            .* (pattern - weights(:,j));
    end
    weights = weights + delta_weights;
end
```

```
figure(1); hold on
plot(weights(1,:),weights(2,:),'r-*','LineWidth',1)

% convergence phase
% weights = 2*rand(2,neurons) - 1;
for t = 1:T_conv
    sigma = sigma_conv;
    lr = lr_conv;

    mu = randi([1,1000]);
    pattern = data(mu,:)'; % 2*1
    dist = zeros(1,neurons);
    for i = 1:neurons
        dist(i) = norm(pattern - weights(:,i));
    end
    win_index = find(dist == (min(dist)));

    delta_weights = zeros(size(weights));
    for j = 1:neurons
        delta_weights(:,j) = lr * exp(-0.5 * abs(j - win_index)^2/ sigma^2)...
            .* (pattern - weights(:,j));
    end
    weights = weights + delta_weights;
end

figure(1);
plot(weights(1,:),weights(2,:),'b-*','LineWidth',1)
plot([0,0],[0,1],'k','LineWidth',1.5);plot([0.5,0.5],[0,0.5],'k','LineWidth',1.5);
plot([1,1],[0.5,1],'k','LineWidth',1.5);plot([0,0.5],[0,0],'k','LineWidth',1.5);
plot([0.5,1],[0.5,0.5],'k','LineWidth',1.5);plot([0,1],[1,1],'k','LineWidth',1.5);
title('Weight Space with $\sigma_{0}=100$','Interpreter','latex','FontSize',18)
xlabel('$w_1(\xi_1)$','Interpreter','latex','FontSize',14)
ylabel('$w_2(\xi_2)$','Interpreter','latex','FontSize',14)
legend('Weight vector after ordering phase','Weight vector after convergence phase','Lo
set(gca,'FontSize',15)
```

Task2:

```
clear;clc;clf;close all

data = importdata('data_ex2_task2_2017.txt');

lr = 0.001;
time = 2e4;
weights = 2*rand(2,1) - 1;
weights_modulus = zeros(time,1);

for t = 1:time
    index = randi([1,size(data,1)]);
    pattern = data(index,:)'; % 2*1
    zeta = pattern' * weights; % 1*2 * 2*1
    delta_weights = lr * zeta .* (pattern - zeta .* weights);
    weights = weights + delta_weights;
    weights_modulus(t) = norm(weights);
end


figure(1); hold on
plot(1:time,weights_modulus,'Linewidth',2)

figure(2); hold on
plot(data(:,1),data(:,2),'rx','Linewidth',1)
quiver(0,0,weights(1),weights(2),'r','Linewidth',3,'MaxHeadSize',2)

% data with zero mean
data_zero_mean = [data(:,1) - mean(data(:,1)) , data(:,2) - mean(data(:,2))];
weights = 2*rand(2,1) - 1;
for t = 1:time
    index = randi([1,size(data_zero_mean,1)]);
    pattern = data_zero_mean(index,:)'; % 2*1
    zeta = pattern' * weights; % 1*2 * 2*1
    delta_weights = lr * zeta .* (pattern - zeta .* weights);
    weights = weights + delta_weights;
    weights_modulus(t) = norm(weights);
end

figure(1);
plot(1:time,weights_modulus,'Linewidth',2)
xlabel('Time(Iterations)')
ylabel('Modulus of weight vector')
legend('Curve of original data','Curve of data with zero mean')
title('Modulus of weight vector','FontSize',18)
set(gca,'FontSize',15)

figure(2); hold on
```

```
plot(data_zero_mean(:,1),data_zero_mean(:,2),'bx','Linewidth',1)
quiver(0,0,weights(1),weights(2),'k','Linewidth',3,'MaxHeadSize',2)
plot([-2 12],[0 0],'k','Linewidth',1.5);
plot([0 0],[-2 3],'k','Linewidth',1.5);
xlabel('$w_1(\xi_1)$','Interpreter','latex','FontSize',14)
ylabel('$w_2(\xi_2)$','Interpreter','latex','FontSize',14)
title('Data plot with ending weight vector','FontSize',18)
legend('Original data','Ending weight vector of oringinal data',...
    'Data with zero mean','Ending weight vector of oringinal data',...
    'Location','southeast')
set(gca,'FontSize',15)
xlim([-2,12])
ylim([-1.5,3])
```

Task3:

```
clear;clc;clf;close all

data = importdata('data_ex2_task3_2017.txt');
runs = 20;
k=10; % k=4 for 3a , k=10 for 3b , comment out for 3c

% class_10 = zeros(10,1); % delete '%' when run 3c
% for k=1:10 % delete '%' when run 3c
class_error_runs = zeros(runs,1);
weights_unsup_runs = zeros(2*runs,k);
weights_sup_runs = zeros(k,runs);
bias_runs = zeros(runs,1);
for r = 1:runs
    % unsupervised learning parameters
    lr_uns = 0.02;
    time = 1e5;
    weights_unsupervised = 2*rand(2,k) - 1; % 2*k

    % supervised learning parameters
    lr_s = 0.1;
    beta = 0.5;
    steps = 3e3;
    weights_supervised = 2*rand(k,1) - 1; % k*1
    bias = 2*rand(1,1)-1;

    % unsupervised learning
    for t = 1:time
        index_pattern = randi([1 size(data,1)]);
        feed_pattern = data(index_pattern,2:3)'; % 2*1
        activation = zeros(k,1);

        for j = 1:k
            activation(j) = exp(- norm(feed_pattern - weights_unsupervised(:,j))^2 /2 )
        end

        activation = activation ./ sum(activation);
        index_win_unit = find(activation == max(activation));
        delta_win_weights = lr_uns * (feed_pattern - weights_unsupervised(:,index_win_u
        weights_unsupervised(:,index_win_unit) = weights_unsupervised(:,index_win_unit)
    end
    weights_unsup_runs(2*(r-1)+1:r*2,:) = weights_unsupervised;

    % supervised simple perceptron learning

    for s = 1:steps
        % feed pattern
        index_SGD = randi([1 size(data,1)]);
        feed_pattern = data(index_SGD,2:3)';
        feed_target = data(index_SGD,1);
        perceptron_neurons = zeros(k,1);

        for j = 1:k
            perceptron_neurons(j) = exp(- norm(feed_pattern - weights_unsupervised(:,j)
        end

        perceptron_neurons = perceptron_neurons ./ sum(perceptron_neurons);
        b = weights_supervised' * perceptron_neurons - bias;
        output = tanh(beta * b);

        % BP update
        weights_supervised = weights_supervised + lr_s*beta*(1-output^2)*...
            (feed_target - output) .* perceptron_neurons;
        bias = bias - lr_s*beta*(1-output^2)*(feed_target - output);
    end
    weights_sup_runs(:,r) = weights_supervised;
    bias_runs(r) = bias;

    num_error = 0;
    for i = 1:size(data,1)
        pattern = data(i,2:3)'; %2*1
        target = data(i,1);
        output_unsup = zeros(k,1);
        for j = 1:k
            output_unsup(j) = exp(- norm(pattern - weights_unsupervised(:,j))^2 / 2 );
        end
        output_unsup = output_unsup ./ sum(output_unsup); % 4*1
        output = tanh(beta*(weights_supervised' * output_unsup - bias));
        if sign(output) ~= target
            num_error = num_error + 1;
        end
    end
    class_error_runs(r) = num_error / size(data,1) ;
end
%     class_10(k) = mean(class_error_runs);  % delete '%' when run 3c
% end  % delete '%' when run 3c

%% decision boundary
best_index = find(class_error_runs == min(class_error_runs));
best_index = best_index(1);
best_unsup_weights = weights_unsup_runs((best_index-1)*2+1:2*best_index,:);
best_sup_weights = weights_sup_runs(:,best_index);
best_bias = bias_runs(best_index);

num_points = 200;
[X,Y] = meshgrid(linspace(-15,25,num_points),linspace(-10,15,num_points));
```

```
decision1 = [];
decision2 = [];
for x = 1:num_points
    for y = 1:num_points
        input_pattern = [X(x,y);Y(x,y)]; % 2*1
        output_unsup = zeros(k,1);
        for j = 1:k
            output_unsup(j) = exp(- norm(input_pattern - best_unsup_weights(:,j))^2 / 2
        end
        output_unsup = output_unsup ./ sum(output_unsup); % 4*1
        output = tanh(beta*(best_sup_weights' * output_unsup - bias));
        if output >= 0
            decision1 = cat(1,decision1,[X(x,y) Y(x,y)]);
        else
            decision2 = cat(1,decision2,[X(x,y) Y(x,y)]);
        end
    end
end

%% plot 3a and 3b
class1 = data(data(:,1) == 1, 2:3);
class2 = data(data(:,1) == -1, 2:3);
figure(1); hold on
plot(decision1(:,1),decision1(:,2),'g*','Linewidth',5)
plot(decision2(:,1),decision2(:,2),'y*','Linewidth',5)
plot(class1(:,1), class1(:,2), 'ro','Linewidth',1.5);
plot(class2(:,1), class2(:,2), 'bx','Linewidth',1.5);
for i = 1:k
quiver(0,0,weights_unsupervised(1,i),weights_unsupervised(2,i),'k','Linewidth',2,'MaxHe
end
title('Data plot with decision boundary when k=10','FontSize',18)
xlabel('$\xi_1$','Interpreter','latex','FontSize',14)
ylabel('$\xi_2$','Interpreter','latex','FontSize',14)
legend('Decision boundary of class = 1','Decision boundary of class = -1',...
    'Data of class = 1','Data of class = -1','Location','northeast')
set(gca,'FontSize',15)

%% plot 3c
% please set 3c code in the first part and run
% then run this part independantly
% figure;
% plot(1:10,class_10,'-*','Linewidth',1.5)
% xlabel('k','FontSize',14)
% ylabel('Classification error','FontSize',14)
% title('The classification error for each k','FontSize',18)
% set(gca,'FontSize',15)
% ylim([0,0.6])
% xlim([1,10])
```