AI BIOLOGICAL COMPUTATION

Computer Homework 1 - TLU and General Neurons

By Mohamad Hosein Faramarzi
Bachelor's student at Sharif university of technology

fall semester-1402

Sharif University of Technology
Electrical Engineering Department

Course Proffesor Dr.Sepideh Hajipour

Contents

1 Q1-General Neuron

1.1 Q1_A

We do this using the code you can see in the bellow that calls main function:

```
%% Sample Matlab code
initial_W1=1;
initial_W2=1;
initial_T=1;
Qbeta=1;
Z_plot(initial_W1,initial_W2,initial_T,Qbeta)
```

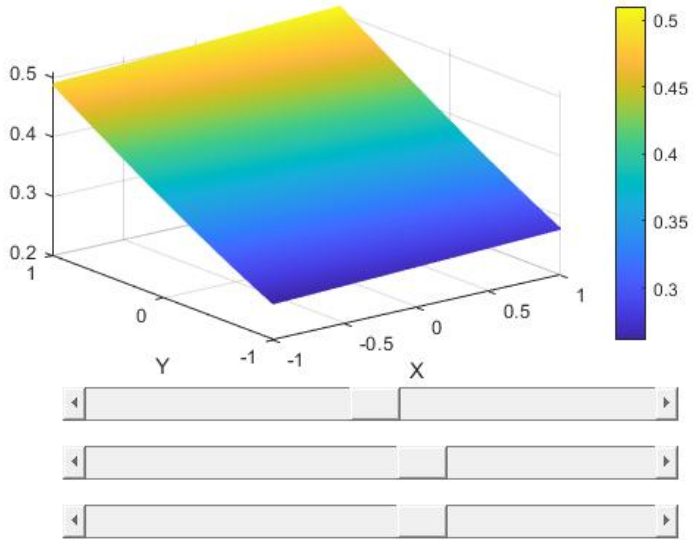
and then definint the two functions to make slider and update the f_act based on the values of sliders:

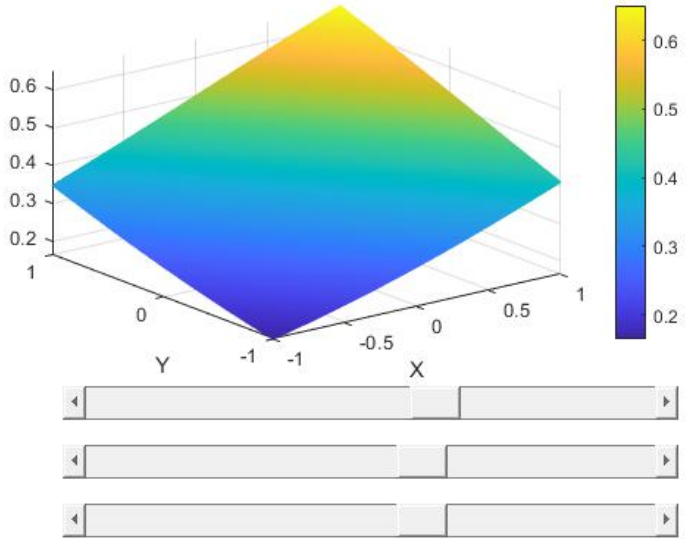
```
%% Sample Matlab code
function update_plot (src, event,new_w1, new_w2, new_T)
    global p % Access the global variable p
    dx = 0.01;
    dy = 0.01;
    beta = 0.5;

    x = -1:dx:1;
    y = -1:dy:1;
    [X,Y] = meshgrid(x,y);

    % Get the values of sliders for w1, w2, and T
    w1 = get(new_w1, 'Value');
    w2 = get(new_w2, 'Value');
    T = get(new_T, 'Value');
```

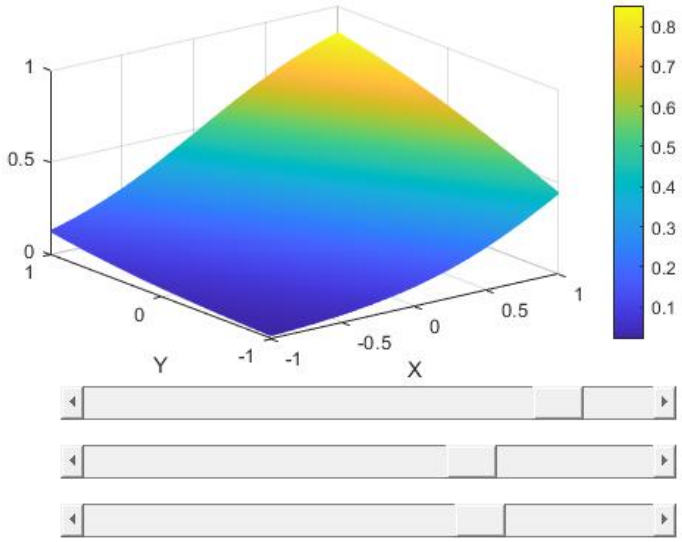
```
%% Sample Matlab code
    % Calculate the activation function
    f_act = 1./(1 + exp(-1*beta.*(w1*X+w2*Y-T)));

    % Update the plot with the new activation function values
    p.ZData = f_act;

    % Print the current values of w1, w2, and T
    fprintf("w1= %d   ,   w2 = %d   ,   T = %d \n", w1, w2, T);
end

function Z_plot(init_w1, init_w2, init_T, beta)
    global p
    dx = 0.01;
    dy = 0.01;

    x = -1:dx:1;
    y = -1:dy:1;
    [X,Y] = meshgrid(x,y);

    % Calculate the initial activation function values
    f_act = 1./(1 + exp(-1*beta.*(init_w1*X+init_w2*Y-init_T)));

    f = figure;
    ax = axes('Parent',f,'position',[0.13 0.39  0.77 0.54]);

    % Create sliders for w1, w2, and T
    s_w1 = uicontrol('Parent',f,'Style','slider','Position',[81,110,419,23],
       ...
                  'value',init_w1, 'min',-5, 'max',5);
    s_w2 = uicontrol('Parent',f,'Style','slider','Position',[81,70,419,23],
       ...
                  'value',init_w2, 'min',-5, 'max',5);

    s_T = uicontrol('Parent',f,'Style','slider','Position',[81,30,419,23],
       ...
                  'value',init_T, 'min',-5, 'max',5);

    % Create a 3D plot of the activation function
    p = mesh(ax, X, Y, f_act);
    xlabel('X');
    ylabel('Y');
    colorbar;

    % Set callbacks for sliders to update the plot
    s_w1.Callback = {@update_plot, s_w1, s_w2, s_T};
    s_w2.Callback = {@update_plot, s_w1, s_w2, s_T};
    s_T.Callback = {@update_plot, s_w1, s_w2, s_T};
end
```

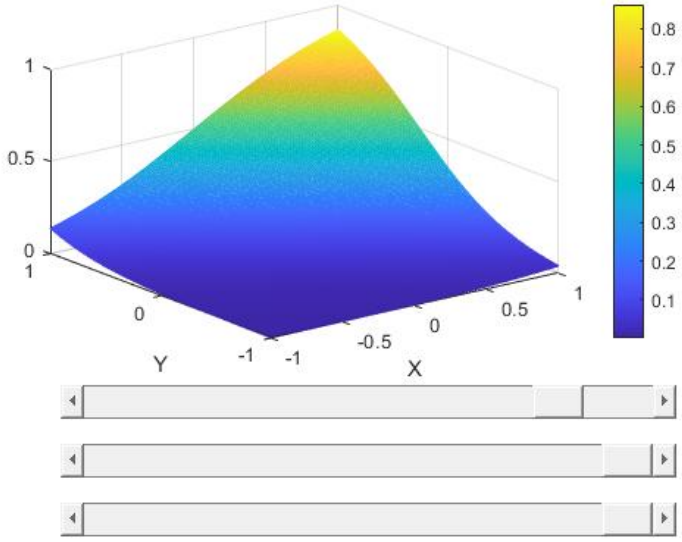
The code defines two functions for visualizing a Threshold Logic Unit (TLU) activation function in a 3D plot. The first function, update_plot, is a callback function that gets called whenever a slider

is moved. It takes the values of three sliders (representing weights w1 and w2, and threshold T) as input. It calculates the activation function using the logistic sigmoid function and updates the plot with the new values. The second function, Z_plot, sets up the initial parameters and creates the figure, sliders, and 3D plot. It also establishes the callbacks for the sliders to ensure that the plot is updated dynamically when sliders are adjusted. The global variable p is used to store the plot, allowing it to be accessed and updated by both functions. Overall, this code provides an interactive tool for visualizing and adjusting the activation function of a TLU.
So we have this plot as result.we see the output function(output of activation function i.e. Z for 4 different values with changing slider) :






1
0.5
0
1
0
-1
Y
-1
-0.5
0
0.5
1
X
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

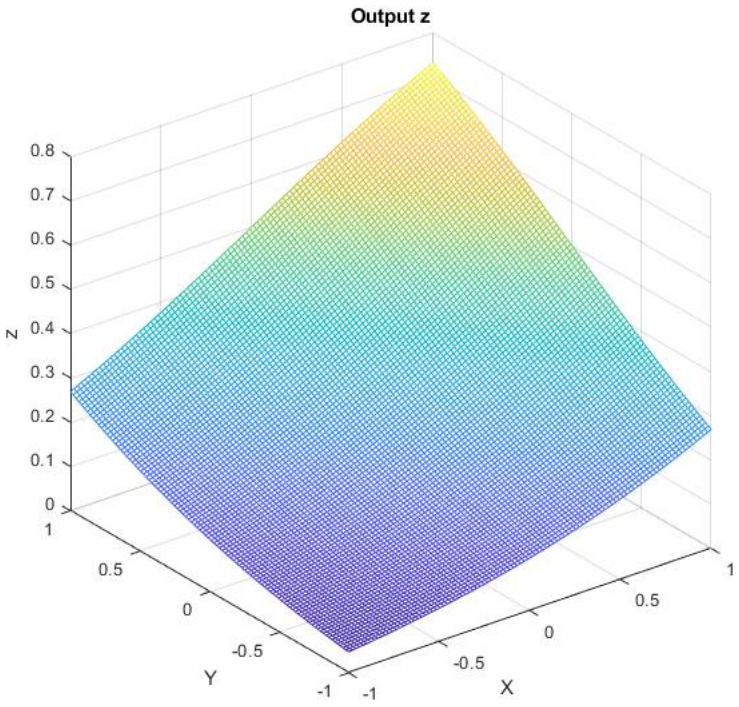

1
0.5
0
1
0
-1
Y
-1
-0.5
0
0.5
1
X
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

1.2 Q1_B

In this part we don't need slider and the form of plotting and updates that we defined in previous section and we just plot with constant values with this simple code:

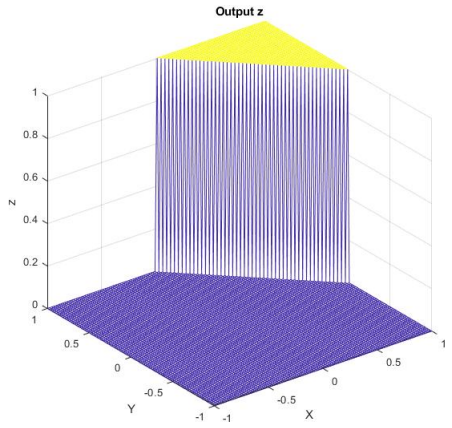
```
%% Sample Matlab code
fig = figure('Position', [100, 100, 800, 600]);

% Create axes for the plot
ax = axes('Parent', fig, 'Position', [0.3, 0.1, 0.6, 0.8]);
W1=1;
W2=1;
T=1;

    [X, Y] = meshgrid(linspace(-1, 1, 100));

    % Calculate z using the provided activation function
    b = 1; % You can adjust this value if needed
    fact = 1./(exp(-b*(W1.*X + W2.*Y - T)) + 1);

    % Plot z
    cla(ax); % Clear previous plot
    mesh(ax, X, Y, fact);
    xlabel(ax, 'X');
    ylabel(ax, 'Y');
    zlabel(ax, 'z');
    title(ax, 'Output z');
```

So the result is :



1.3 Q1-C

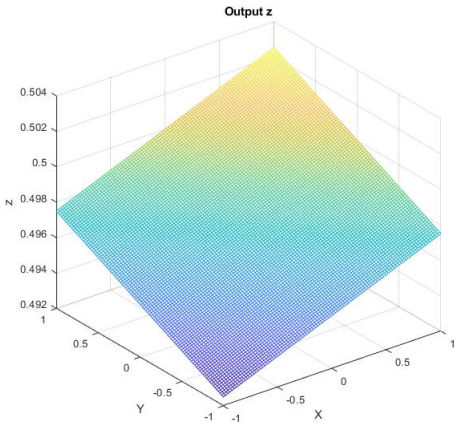
If we define β with a high value like 1000 we have actually implemented Or logical operation which shows that the output is equal to zero only if both X and Y have a really small values that you can see this in the plot in the bellow:



In this case, a high beta value will cause the sigmoid function to have a steeper transition from the lower to upper bounds, making it behave more like a step function, particularly for inputs that are near the threshold T.
This behavior is somewhat similar to the Heaviside step function, which is a classic step function often used in binary classification problems. The Heaviside step function outputs 1 if the input is greater than or equal to 0, and 0 otherwise. With a high beta value, sigmoid function becomes more step-like for inputs near T, resembling the behavior of the Heaviside step function.

1.4 Q1-D

We now put 0.01 in beta value and we can see this result :



A low value of beta will result in a much smoother transition from the lower to upper bounds. This means that the function will behave more like a gradual slope, and the output will vary more smoothly with changes in the inputs X and Y.

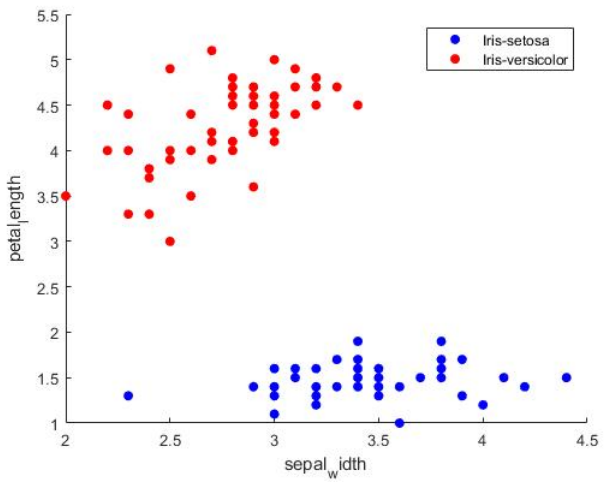
2 Q2-iris data set

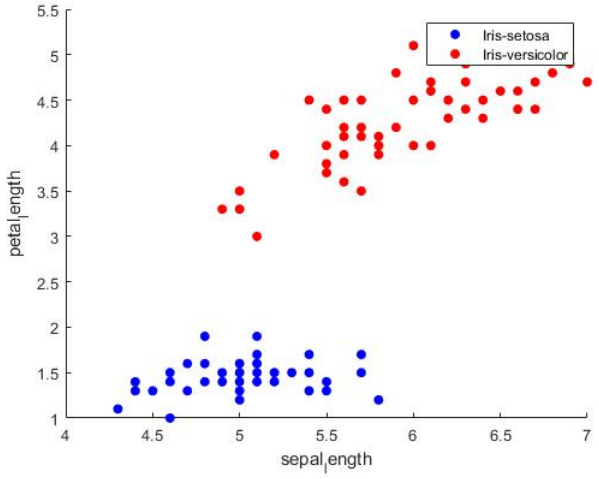
2.1 Q2-A

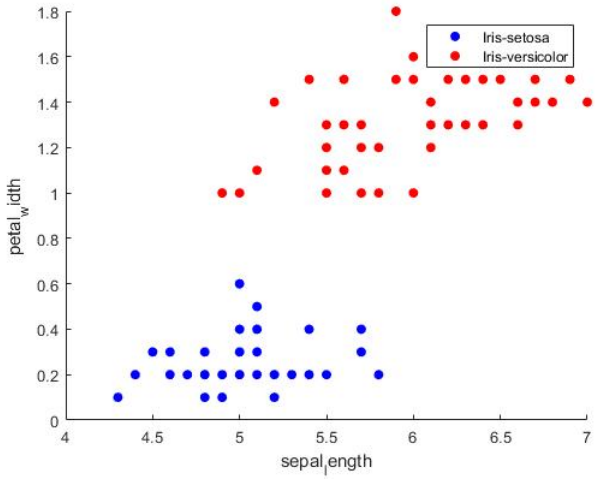
In this section we have to separate first 2 classes from main data set.First 2 classes are Iris-setosa and Iris-versicolor.After that we have to make a for in 4 features that these classes have and at the end plot 6 2D charts that shows distribution of 2 classes.This section should be done based on the code in the bellow:

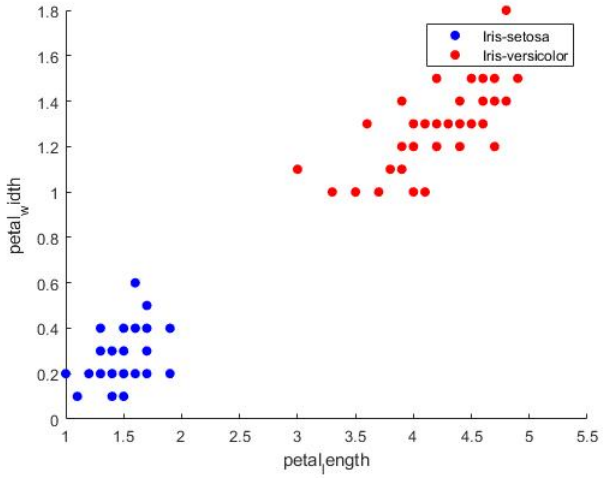
```
% Step 1: Load the Data
data = readtable('iris.csv'); % Assuming the file is in the same directory as your script
% Add column names
data.Properties.VariableNames = {'sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'iris_type'};
% Step 2: Separate Classes (Iris-setosa and Iris-versicolor)
setosa = data(strcmp(data.iris_type, 'Iris-setosa'), :);
versicolor = data(strcmp(data.iris_type, 'Iris-versicolor'), :);

% Step 3: Plot Features
features = {'sepal_length', 'sepal_width', 'petal_length', 'petal_width'};

% Create scatter plots for various feature pairs
for i = 1:length(features)
    for j = i+1:length(features)
        figure;
        scatter(setosa.(features{i}), setosa.(features{j}), 'b', 'filled', 'DisplayName', 'Iris-setosa');
        hold on;
        scatter(versicolor.(features{i}), versicolor.(features{j}), 'r', 'filled', 'DisplayName', 'Iris-versicolor');
        xlabel(features{i});
        ylabel(features{j});
        legend();
        hold off; end end
% Step 4: Analyze Linearity
```

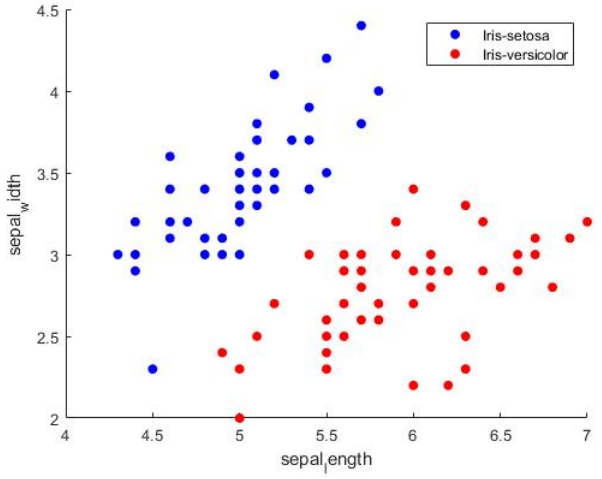
The result of this simulation is this plots:

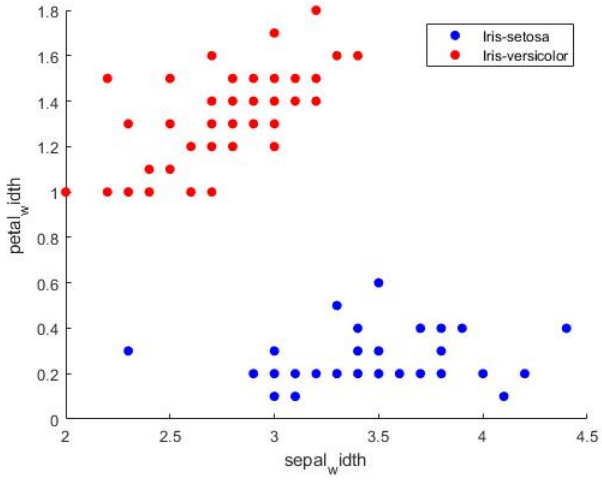



Iris-setosa
Iris-versicolor
petal_length
sepal_length
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
4
4.5
5
5.5
6
6.5
7

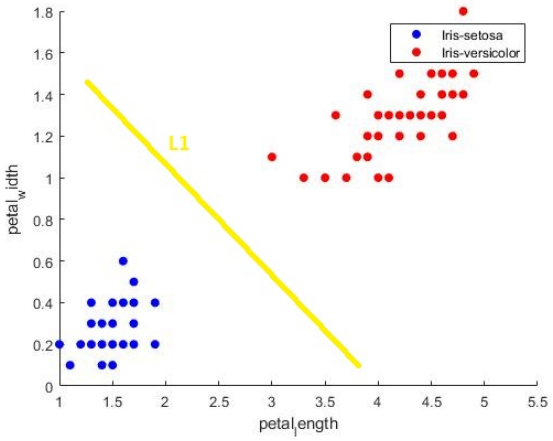

Iris-setosa
Iris-versicolor
petal_width
sepal_length
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
4
4.5
5
5.5
6
6.5
7


Iris-setosa
Iris-versicolor
petal_width
petal_length
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5



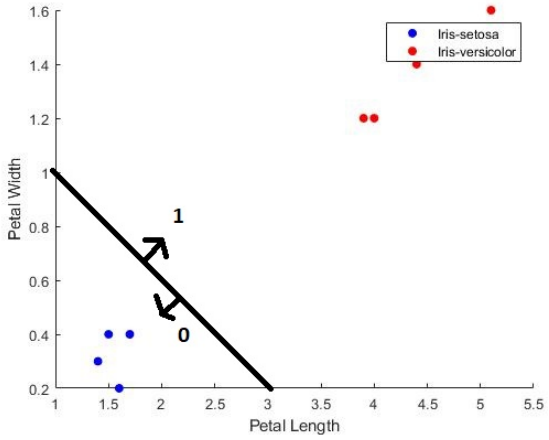


Based on the plots we made we can see that most of them can easily be classified with single line with different Slope and intercept.Some of plots show difficult classification like the plot of sepal width-sepal length but the plot of petal width-petal length is more separable so we choose this plot for other parts.It can be theoretically shown in the figure bellow:



2.2 Q2-b

With **rng and randperm** functions we can generate 5 random data from complete data and then plot them.We also show the linear classifier we have to implement by TLU:



Based on this line the equation of single TLU is shown in the bellow which shows a 2 input single output TLU with $w_1 = 3$ and $W_2 = 1$ and $\theta = 3$:

$$\begin{cases} 3x + y > 3 & out = 1 \\ o.w & out = 0 \end{cases}$$

2.3 Q2-C

In batch learning, the entire training dataset is used to update the model's parameters in one go. The model processes all the training data, computes the gradients of the loss function with respect to the parameters, and then updates the parameters accordingly.In online learning, the model is updated incrementally, one sample or a small batch of samples at a time. The model learns from incoming data in real-time and continuously adapts to changes so we conclude that main difference in this two way are the order and process of updating.So we use this algorithm to implement batch learning and online learning for our data.
ONLINE:

```
%% Sample Matlab code
while online_error ~= 0 && online_iteration < End_iteration
    online_error = 0;
    online_iteration = online_iteration + 1;

    for i = 1:size(Data_attached, 1)
        x = Data_attached(i, 1:2);
        target_label = Data_attached(i, 3);
        target_value = strcmp(target_label{1, 1}, 'Iris-versicolor');
        activation = x{1, 1} * online_weights(1) + x{1, 2} * online_weights
           (2) - online_threshold;

        if activation >= 0
            predicted_value = 1;
        else
            predicted_value = 0;
        end
        W1_variation_online(online_iteration) = online_weights(1);
        W2_variation_online(online_iteration) = online_weights(2);
        onlineThresholdChanges(online_iteration) = online_threshold;
```

```
%% Sample Matlab code
        % Updating Process
        if(target_value ~= predicted_value)
            online_error = target_value - predicted_value;
            online_threshold = online_threshold - (online_learning_rate *
               online_error);
            online_weights = online_weights + online_learning_rate * (
               online_error) * [x{1, 1}, x{1, 2}];
        end end end
```

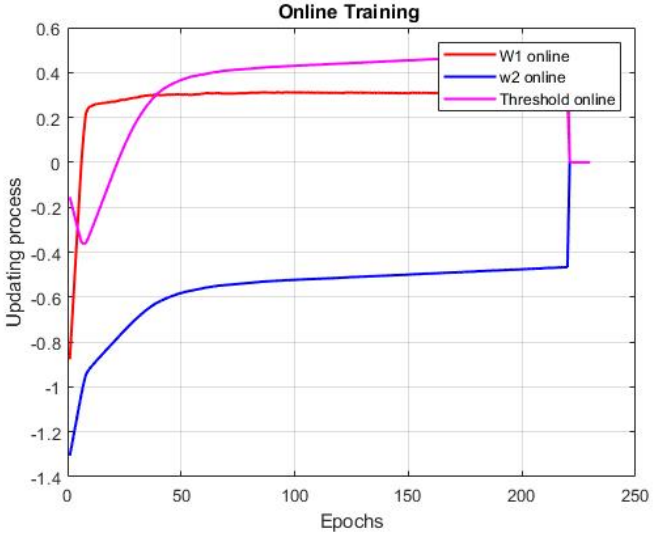
BATCH:

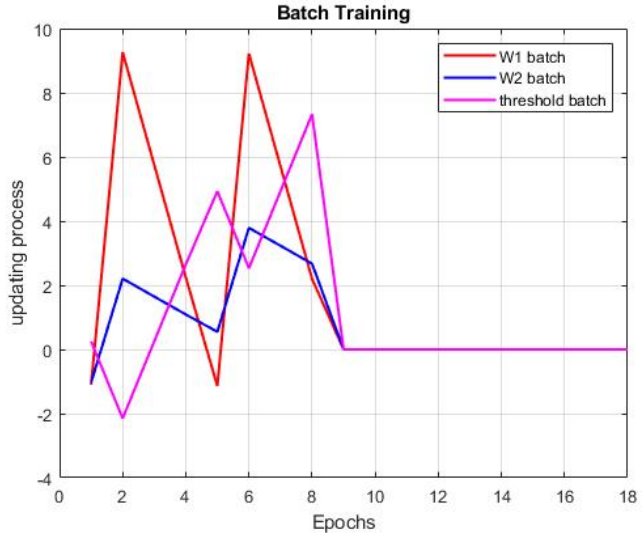
```
%% Sample Matlab code
while batch_error ~= 0 && batch_iteration < End_iteration
    batch_error = 0;
    batch_iteration = batch_iteration + 1;

    % Store updates for weights and threshold
    delta_weights = [0, 0];
    delta_threshold = 0;

    for i = 1:size(Data_attached, 1)
        x = Data_attached(i, 1:2);
        target_label = Data_attached(i, 3);
        target_value = strcmp(target_label{1, 1}, 'Iris-versicolor');
        activation = x{1, 1} * batch_weights(1) + x{1, 2} * batch_weights(2)
            - batch_threshold;

        % Updating Process
        if activation >= 0
            predicted_value = 1;
        else
            predicted_value = 0;
        end

        if target_value ~= predicted_value
            batch_error = target_value - predicted_value;
            delta_weights = delta_weights + batch_learning_rate * (
               batch_error) * [x{1, 1}, x{1, 2}];
            delta_threshold = delta_threshold - (batch_learning_rate *
               batch_error);
        end
    end

    W1_variation_batch(batch_iteration) = batch_weights(1);
    W2_variation_batch(batch_iteration) = batch_weights(2);
    Threshold_variation_batch(batch_iteration) = batch_threshold;

    % Batch update for weights and threshold
    batch_weights = batch_weights + delta_weights;
    batch_threshold = batch_threshold + delta_threshold;
end
```

The first code implements an online learning algorithm for a binary classification task. It iteratively updates model parameters after processing individual data points. The loop continues until the error is zero or a maximum number of iterations is reached. For each data point, features and target labels are extracted, and the activation value is computed. Depending on the activation value, the model predicts either class 1 or class 0. The weights and threshold are updated based on prediction errors. The second code implements a batch learning algorithm. It updates model parameters after processing the entire dataset. Similar to the online learning code, the loop continues until the error is zero or a maximum number of iterations is reached. Features and target labels are extracted for each data point, and the activation value is computed. The model predicts class 1 or class 0 based on the activation value, and the weights and threshold are updated accordingly.

2.4 Q2-D

In this section we plot the process of updating 3 parameters.Two of them are weights (W1 and W2) and also threshold.These plots can clearly show the difference between batch and online learning (training).In online training we have more continues changing process for parameters cause after each error computation for every input we do update but in batch learning we can easier see the steps of changing and its more discrete. Note that the number of epochs in this 2 way of learning aren't same because the start from different initial values and also the process of updating are different so one of them can reach to proper classifier with less epochs.

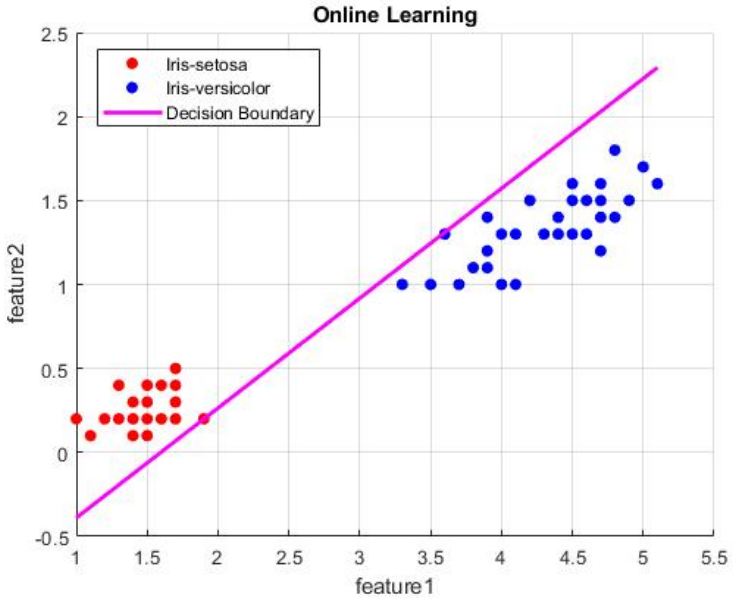


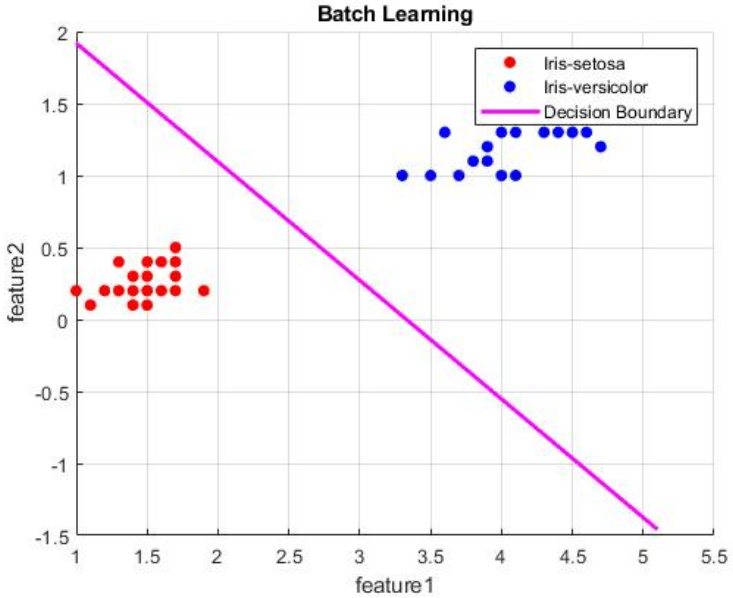


Online Learning Updates weights after each data point.Batch Learning Updates weights after processing the entire dataset.

2.5 Q2-E

Now we have to plot the whole data in a 2D plot along with decision boundary created by each learning process (online or Batch).There is 2 different linear classifier in different directions so it is logical that batch learning ended with fewer iterations.Another thing we can understand about this 2 plot is that both of them finally did the classification in a good way.





In this linear classifiers at first we have a bunch of data with 2 chased feature that specify the data is plotted based on which features and shows the axes of out plot.Then we have linear TLU which give this 2 features values as inputs and after multiplying them at weights and compare the summation with defined threshold choose that the data belongs to class 1 or class 2.Conceptually, weights in a TLU represent the importance or significance of each input. They determine the contribution of each input value towards the final decision.The threshold in a TLU sets a boundary that the weighted sum of inputs must cross in order for the TLU to produce an output.It acts as a decision boundary or

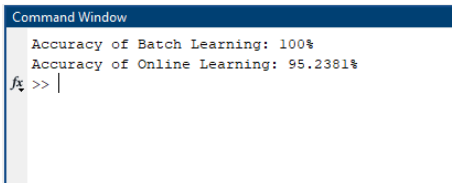
activation level. If the weighted sum exceeds the threshold, the TLU produces an output; otherwise, it remains inactive.The threshold essentially defines the level of activation required for the TLU to "fire" and produce a positive output.Together, the weights and threshold work in conjunction to determine the activation level of the TLU based on the weighted sum of inputs. This activation level is then compared to the threshold to determine the final output of the TLU (either active or inactive). Adjusting these parameters during training allows the TLU to learn and make accurate predictions or classifications based on the input data.

2.6 Q2-F

Now with other 20 percent of data we test the classifier we trained and compute the accuracy.After checking the test data with our classifier and predict their class we come back to their actual classes and regarding this data we compute accuracy :

```
%% Sample Matlab code
%accuracy computing
Batch_accuracy = (num_correct_predictions_batch / num_test_samples) * 100;
Online_accuracy = (num_correct_predictions_online / num_test_samples) * 100;

% Printing accuracy
disp(['Accuracy of Batch Learning: ', num2str(Batch_accuracy), '%']);
disp(['Accuracy of Online Learning: ', num2str(Online_accuracy), '%']);
```

We can see the final result in the figure of command window in Matlab :

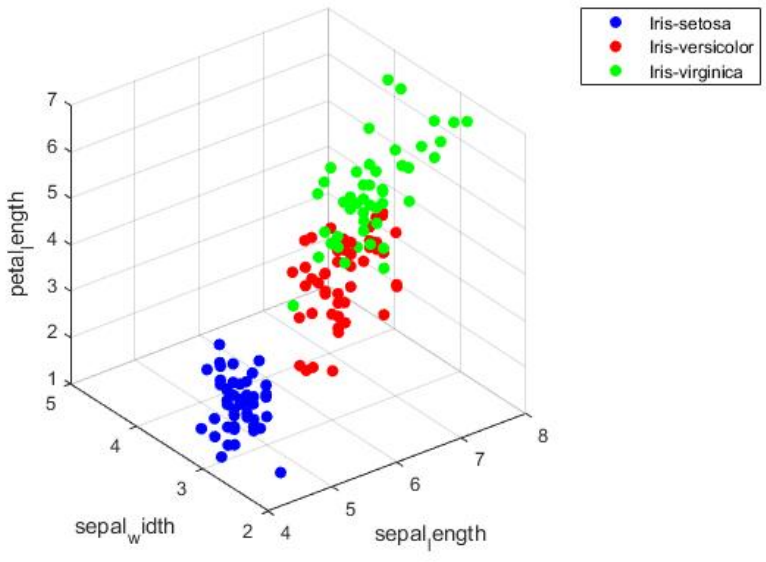


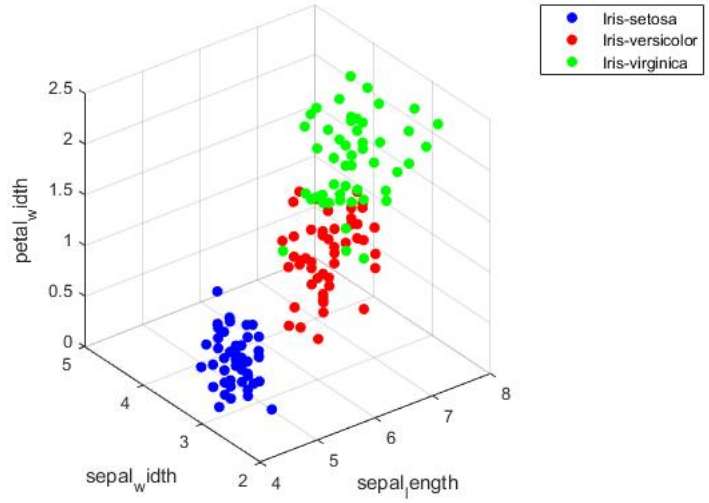
2.7 Q2-G

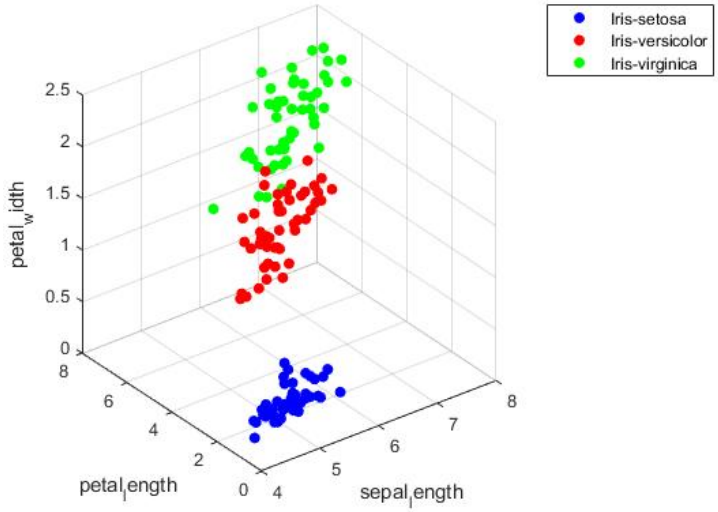
The algorithm of this part is alike with part A with the difference that in this part we choose 3 feature in each plot and after that plot all 3 types :

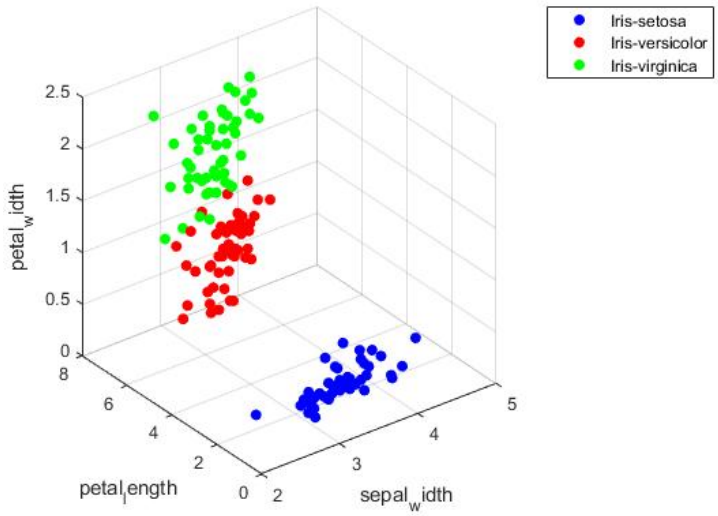
```
%% Sample Matlab code
% Step 1: Load the Data
data = readtable('iris.csv'); % Assuming the file is in the same directory
    as your script

% Add column names
data.Properties.VariableNames = {'sepal_length', 'sepal_width', '
    petal_length', 'petal_width', 'iris_type'};

% Step 2: Separate Classes
setosa = data(strcmp(data.iris_type, 'Iris-setosa'), :);
versicolor = data(strcmp(data.iris_type, 'Iris-versicolor'), :);
virginica = data(strcmp(data.iris_type, 'Iris-virginica'), :);
```

```
%% Sample Matlab code
% Step 3: Plot Features in 3D
features = {'sepal_length', 'sepal_width', 'petal_length', 'petal_width'};

for i = 1:length(features)
    for j = i+1:length(features)
        for k = j+1:length(features)
            figure;
            scatter3(setosa.(features{i}), setosa.(features{j}), setosa.(
               features{k}), 'b', 'filled', 'DisplayName', 'Iris-setosa');
            hold on;
            scatter3(versicolor.(features{i}), versicolor.(features{j}),
               versicolor.(features{k}), 'r', 'filled', 'DisplayName', 'Iris
               -versicolor');
            scatter3(virginica.(features{i}), virginica.(features{j}),
               virginica.(features{k}), 'g', 'filled', 'DisplayName', 'Iris-
               virginica');
            xlabel(features{i});
            ylabel(features{j});
            zlabel(features{k});
            legend();
            hold off;
        end
    end
end
```




Iris-setosa
Iris-versicolor
Iris-virginica
petal_width
sepal_width
sepal_length


Iris-setosa
Iris-versicolor
Iris-virginica
petal_width
petal_length
sepal_length


Iris-setosa
Iris-versicolor
Iris-virginica
petal_width
petal_length
sepal_width

Now we have 3 classes that we should separate them so with a single TLU that just determine a simple hyperplane in 3D plane we cannot do the classification but with a set of hyper planes and Confluence of them that is a result of combination of TLUs we can do this classification based on the plots we made. To perform a 3-class classification in a 3D plane using Threshold Logic Units (TLUs), employ the one-versus-all (OvA) strategy. Begin by preparing a labeled dataset with three distinct classes. Select three features as dimensions for the 3D space. Set up three TLUs, one for each class. Then, for each TLU, treat one class as "positive" and the combination of the other two as "negative" during training. Train the TLUs to distinguish between the positive class and the combined negatives. When classifying a new data point, pass it through all three TLUs. The TLU with the highest activation level, indicating proximity to the decision boundary, will predict the class. In case of a tie, implement a tie-breaking strategy. This approach allows you to effectively use a combination of TLUs for 3-class classification in a 3D plane.