THE UNIVERSITY OF CALGARY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ENEL 525 Machine Learning for Engineers

LAB 1 Pattern Classification

23rd September (Lab B02) and 26th September (Lab B01) 2024

Objective: Classify patterns using perceptron learning

To do:

I. Given a set of input vectors and their respective classes, $\{(p_1, t_1), (p_2, t_2), \cdots, (p_N, t_N)\}$, apply the perceptron learning rule to classify these input vectors and save your program as a Python file. The perceptron learning algorithm can be implemented in Python as follows:

Initialization

1. Create a new Python (**.py**) or Jupyter Notebook (**.ipynb**) file in your favourite editor/IDE (e.g., VS Code) and read the first input vector $\boldsymbol{p}_1$ **.**

2. Initialize the weight vector and bias i.e.

$$\boldsymbol{W}(0) = [0\ 0] \text{ and } \boldsymbol{b}(0) = 0$$

3. Compute the initial output by applying the weights and bias to the first training vector $\boldsymbol{p}_1$ i.e.

$$\boldsymbol{a}(0) = hardlim(\boldsymbol{W}(0) * \boldsymbol{p}_1 + \boldsymbol{b}(0))$$

The hard limit transfer function can be realized as:

$$\text{hardlim(n)} = \begin{cases} 1 \text{ if n} \geqslant 0 \\ 0 \text{ otherwise} \end{cases}$$

4. Calculate the initial error $\boldsymbol{e}(0) = \boldsymbol{t_1} - \boldsymbol{a}(0)$

5. Update the weight vector and bias by applying the results of 2 to 4 following the equations:

$$\boldsymbol{W}(1) = \boldsymbol{W}(0) + \boldsymbol{e}(0)\, \boldsymbol{p}_1^T$$

$$\boldsymbol{b}(1) = \boldsymbol{b}(0) + \boldsymbol{e}(0)$$

6. Repeat steps 3 to 5 using $\boldsymbol{p}_2, \ldots, \boldsymbol{p}_N$. Convert the corresponding errors $\boldsymbol{e}(0)$, $\boldsymbol{e}(1)$, $\boldsymbol{e}(2)$, $\boldsymbol{e}(3)$ into a vector ***flag*** in such a manner that if $\boldsymbol{e}(\text{i}) \neq 0$, flag(i)=1, otherwise flag(i)=0.

Iterations

7. Use loops to perform the iterative update. At each loop, apply the input vectors $\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots$ and their corresponding outputs $\boldsymbol{t}_1, \boldsymbol{t}_2, \cdots$ sequentially to update the weight and bias. The termination criterion of the loop should be such that the weight vector and bias are updated until all the input

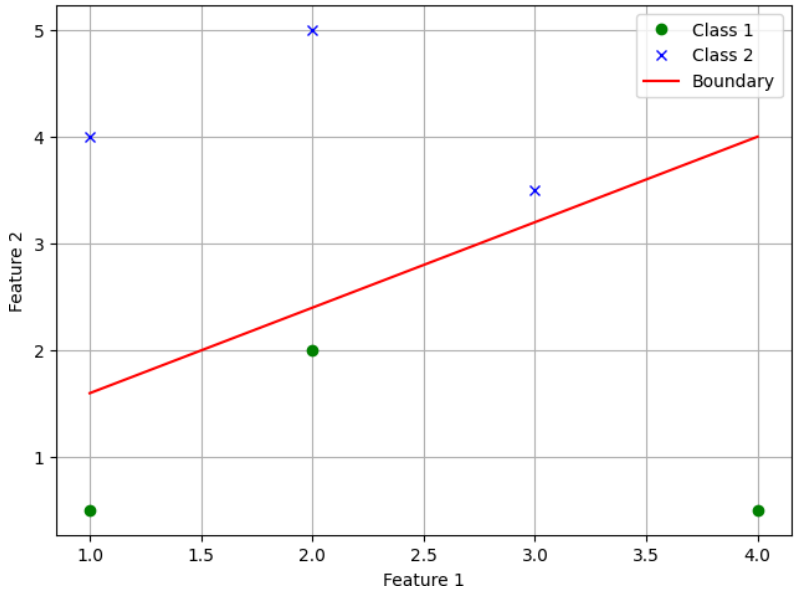
vectors are correctly classified (i.e., flag(i)=0, $1 \le i \le N$). The pseudo codes for iteration are as follows:

```
while  (at least one flag(i) ≠ 0)
{
  initialize flag(i) = 1, for 1 ≤ i ≤ N;
  for k = 1: N
      {
       compute a(k), e(k);
       if  e(k) = 0
           flag(k) = 0;
      end
      update W, b;
      }
   end
}
```

II. Plot the decision boundary.

III. During the lab, show the TA your code and its output correctly classifying the provided points (see additional information). The TA will provide some new points to classify; demonstrate your code on these new sets of points. You will then be asked some questions related to the lab and the perceptron learning rule in general which will be marked.

Example output plot:



Additional Information

Class 1 points	Class 2 points
[1 4], [2 5], [3 3.5]	[1 0.5], [2 2], [4 0.5]

As a matrix:

$$p = \begin{bmatrix} 1 & 2 & 3 & 1 & 2 & 4 \\ 4 & 5 & 3.5 & 0.5 & 2 & 0.5 \end{bmatrix}$$

In Python:

```
p = numpy.array([[1, 2, 3, 1, 2, 4], [4, 5, 3.5, 0.5, 2, 0.5]])
t = np.array([1, 1, 1, 0, 0, 0])
```

Task:
Plot the decision boundary and the points in the same figure.

Points on the decision boundary p(x, y) satisfy:
W(1)*x + W(2)*y + b = 0

So, calculate "y" over a range of x = [-1, 6] and plot.

Side notes

You can use matplotlib's pyplot and numpy libraries for easier implementation:

```
import matplotlib.pyplot as plt
import numpy as np
```

Accessing or assigning to a single column:

For example, changing column 1 of p: p[:, 1]=[1, 4]

To plot a single point, say (1,2):

matplotlib.pyplot.plot(1, 2, '*') ← the '*' makes the single point appear as a star.
Can use other symbols, such as 'o'.

To plot a line, given vectors of x and y coordinates:

matplotlib.pyplot.plot(x, y)

The consecutive plot() commands are drawn onto the same canvas until matplotlib.pyplot.show() command is reached:

```
# Plot class 1 samples
matplotlib.pyplot.plot(…)
# Plot class 2 samples
matplotlib.pyplot.plot(…)
# Plot boundary
matplotlib.pyplot.plot(…)
matplotlib.pyplot.show()
```