Câu 1:

* Ý nghĩa hàm lỗi (error function)

Nếu về một perceptron trong bối cảnh học có giám sát. Hãy tưởng tượng rằng chúng ta có một số nhãn cho đầu ra y\_j của perceptron j cần phải là: hãy gọi đây là đầu ra mong muốn của chúng ta d\_j . Đầu ra mong muốn d\_j này là sự thật cơ bản, đầu ra mục tiêu mà chúng ta biết perceptron nên được tạo ra. Nói cách khác, đầu ra mong muốn này giám sát perceptron của chúng ta theo trọng lượng mà nó nên học để tạo ra một đầu ra nhất định.

Nếu chúng ta biết đầu ra mong muốn là gì, chúng ta có thể sử dụng một số phép toán để xác định một số hàm cho phép chúng ta so sánh đầu ra mong muốn d\_j với *đầu ra thực tế y\_j* mà perceptron tạo ra. Nhưng chúng tôi cũng muốn hàm này phụ thuộc vào trọng lượng vì trọng lượng là những gì chúng tôi muốn cập nhật khi huấn luyện perceptron. Hãy gọi hàm này là một hàm *lỗi* vì nếu đầu ra thực tế không bằng đầu ra mong muốn, thì giá trị của hàm lỗi là khác 0 và chúng ta có thể nói rằng perceptron đã mắc lỗi!

Hàm này là hàm lỗi và mục tiêu là giảm thiểu nó bằng cách cập nhật lặp đi lặp lại các trọng số.

* Mục đích của Thuật toán học Perceptron:

Cũng giống như các thuật toán của machine learning, PLA cũng đi tìm biên phân chia bằng cách tối thiểu một hàm mất mát. Việc xây dựng hàm mất mát của PLA phải thỏa mãn hàm số phải khả vi (nhằm mục đích có thể tính được đạo hàm bậc nhất) và từ đó sử dụng gradient descent để tìm nghiệm global minimum.

Câu 2:

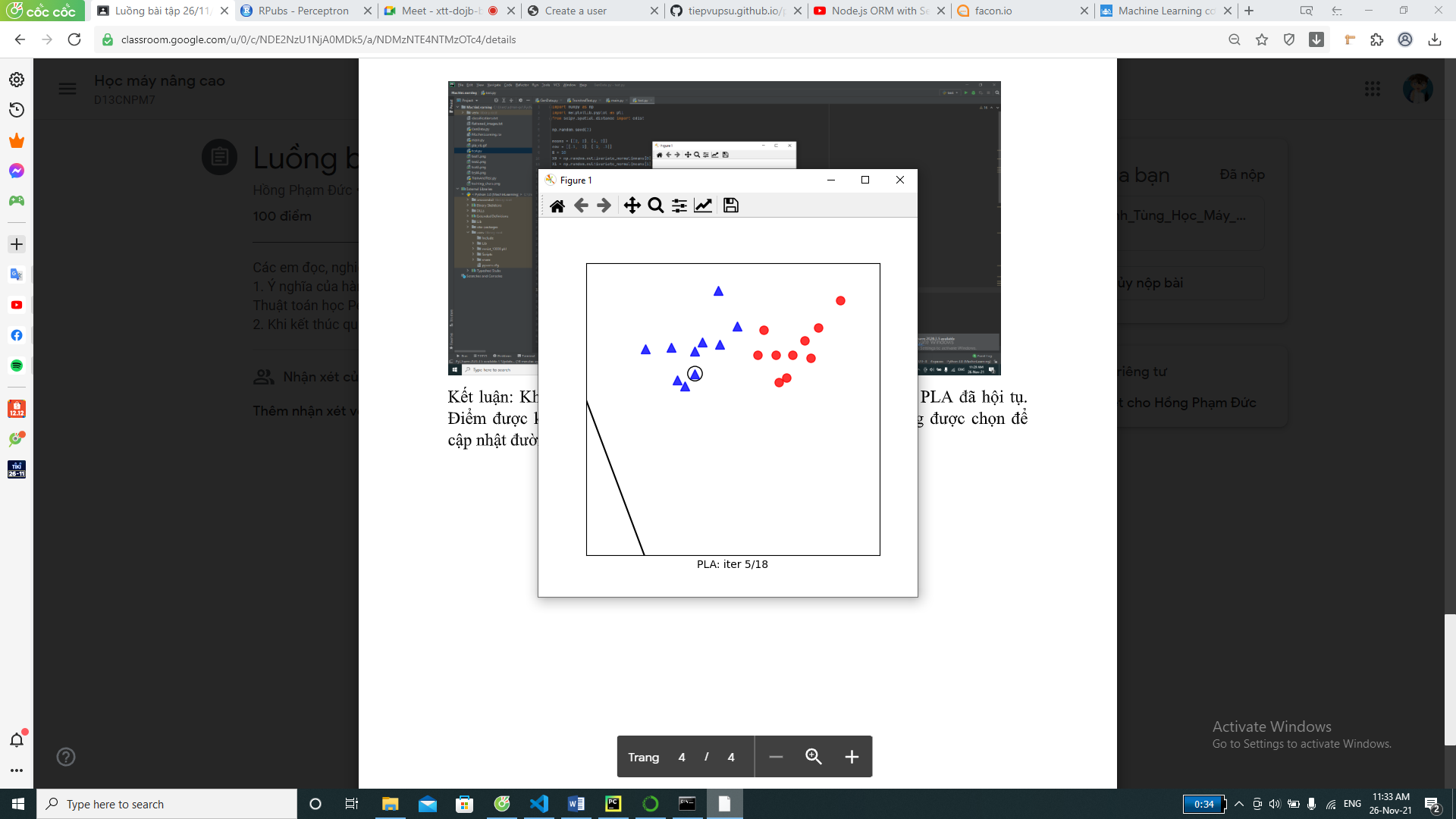
Tóm tắt PLA

1. Chọn ngẫu nhiên một vector hệ số ww với các phần tử gần 0.
2. Duyệt ngẫu nhiên qua từng điểm dữ liệu xixi:
3. Kiểm tra xem có bao nhiêu điểm bị misclassifed. Nếu không còn điểm nào, dừng thuật toán. Nếu còn, quay lại bước 2.

VD:

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.spatial.distance import cdist  
  
np.random.seed(2)  
  
means = [[2, 2], [4, 2]]  
cov = [[.3, .2], [.2, .3]]  
N = 10  
X0 = np.random.multivariate\_normal(means[0], cov, N).T  
X1 = np.random.multivariate\_normal(means[1], cov, N).T  
  
X = np.concatenate((X0, X1), axis=1)  
y = np.concatenate((np.ones((1, N)), -1 \* np.ones((1, N))), axis=1)  
# Xbar  
X = np.concatenate((np.ones((1, 2 \* N)), X), axis=0)  
  
  
def h(w, x):  
 return np.sign(np.dot(w.T, x))  
  
  
def has\_converged(X, y, w):  
 return np.array\_equal(h(w, X), y) # True if h(w, X) == y else False  
  
  
def perceptron(X, y, w\_init):  
 w = [w\_init]  
 N = X.shape[1]  
 mis\_points = []  
 while True:  
 # mix data  
 mix\_id = np.random.permutation(N)  
 for i in range(N):  
 xi = X[:, mix\_id[i]].reshape(3, 1)  
 yi = y[0, mix\_id[i]]  
 if h(w[-1], xi)[0] != yi:  
 mis\_points.append(mix\_id[i])  
 w\_new = w[-1] + yi \* xi  
  
 w.append(w\_new)  
  
 if has\_converged(X, y, w[-1]):  
 break  
 return (w, mis\_points)  
  
  
d = X.shape[0]  
w\_init = np.random.randn(d, 1)  
(w, m) = perceptron(X, y, w\_init)  
print(m)  
  
  
# print(w)  
# print(len(w))  
  
def draw\_line(w):  
 w0, w1, w2 = w[0], w[1], w[2]  
 if w2 != 0:  
 x11, x12 = -100, 100  
 return plt.plot([x11, x12], [-(w1 \* x11 + w0) / w2, -(w1 \* x12 + w0) / w2], 'k')  
 else:  
 x10 = -w0 / w1  
 return plt.plot([x10, x10], [-100, 100], 'k')  
  
  
## Visualization  
import matplotlib.animation as animation  
from matplotlib.animation import FuncAnimation  
  
  
def viz\_alg\_1d\_2(w):  
 it = len(w)  
 fig, ax = plt.subplots(figsize=(5, 5))  
  
 def update(i):  
 ani = plt.cla()  
 # points  
 ani = plt.plot(X0[0, :], X0[1, :], 'b^', markersize=8, alpha=.8)  
 ani = plt.plot(X1[0, :], X1[1, :], 'ro', markersize=8, alpha=.8)  
 ani = plt.axis([0, 6, -2, 4])  
 i2 = i if i < it else it - 1  
 ani = draw\_line(w[i2])  
 if i < it - 1:  
 # draw one misclassified point  
 circle = plt.Circle((X[1, m[i]], X[2, m[i]]), 0.15, color='k', fill=False)  
 ax.add\_artist(circle)  
 # hide axis  
 cur\_axes = plt.gca()  
 cur\_axes.axes.get\_xaxis().set\_ticks([])  
 cur\_axes.axes.get\_yaxis().set\_ticks([])  
  
 label = 'PLA: iter %d/%d' % (i2, it - 1)  
 ax.set\_xlabel(label)  
 return ani, ax  
  
 anim = FuncAnimation(fig, update, frames=np.arange(0, it + 2), interval=1000)  
 # save  
 anim.save('pla\_vis.gif', dpi=100, writer='imagemagick')  
 plt.show()  
  
  
viz\_alg\_1d\_2(w)

Kết quả:



Kết luận: Khi kết thúc quá trình học Perceptron, cập nhật 18 lần, PLA đã hội tụ. Điểm được khoanh tròn màu đen là điểm misclassified tương ứng được chọn để cập nhật đường boundary