Perceptrons

1. there is a perceptron

At least all perceptrons: $y = sign(X_k + t)$, where $t \in (-\epsilon, \epsilon)$ can correctly classifies this data. Theoretically $t = 0$ is the “best” one because it has the largest margin: $\gamma = \epsilon$.

2. the output perceptron

Use `getData()` to generate data points. Use `preceptron(data).fit()` to train the perceptron.

Here is an example of perceptron: (Fig. 1)

```
D:\Users\endlesstory\Desktop\536\hw3>py PP.py
After 5 steps learning:
 0        1        2        3        4        5        6        7        8        9        10       11       12       13       14
          15       16       17       18       19       b
0.18      0.44     2.33     -0.38    -0.31    0.15     1.30     0.11     -0.43    -1.18    0.73     -0.77    0.77     0.54     1.41
          -1.79    -0.83    -0.65    1.00     7.17     -0.42
```

Fig.1 a perceptron trained with 100 data points.

Compared with the theoretical answer, whose all the weights except w_k are 0, its weights are not that ideal. But compared with $w_k = 7.17$, all the others are relatively small. $\frac{b}{w_k} = -0.06 \in (-1, 1)$.

More generally, we can normalize it to get $w_i \in (-0.22, 0.28) \forall i \in \{1, 2, ..., k - 1\}$. $w_k = 0.86, b = -0.05$. It is not far from the theoretical answer.

3. generate a data set for a given value of □ and run the learning algorithm to completion

Here is the result: (Fig. 2)

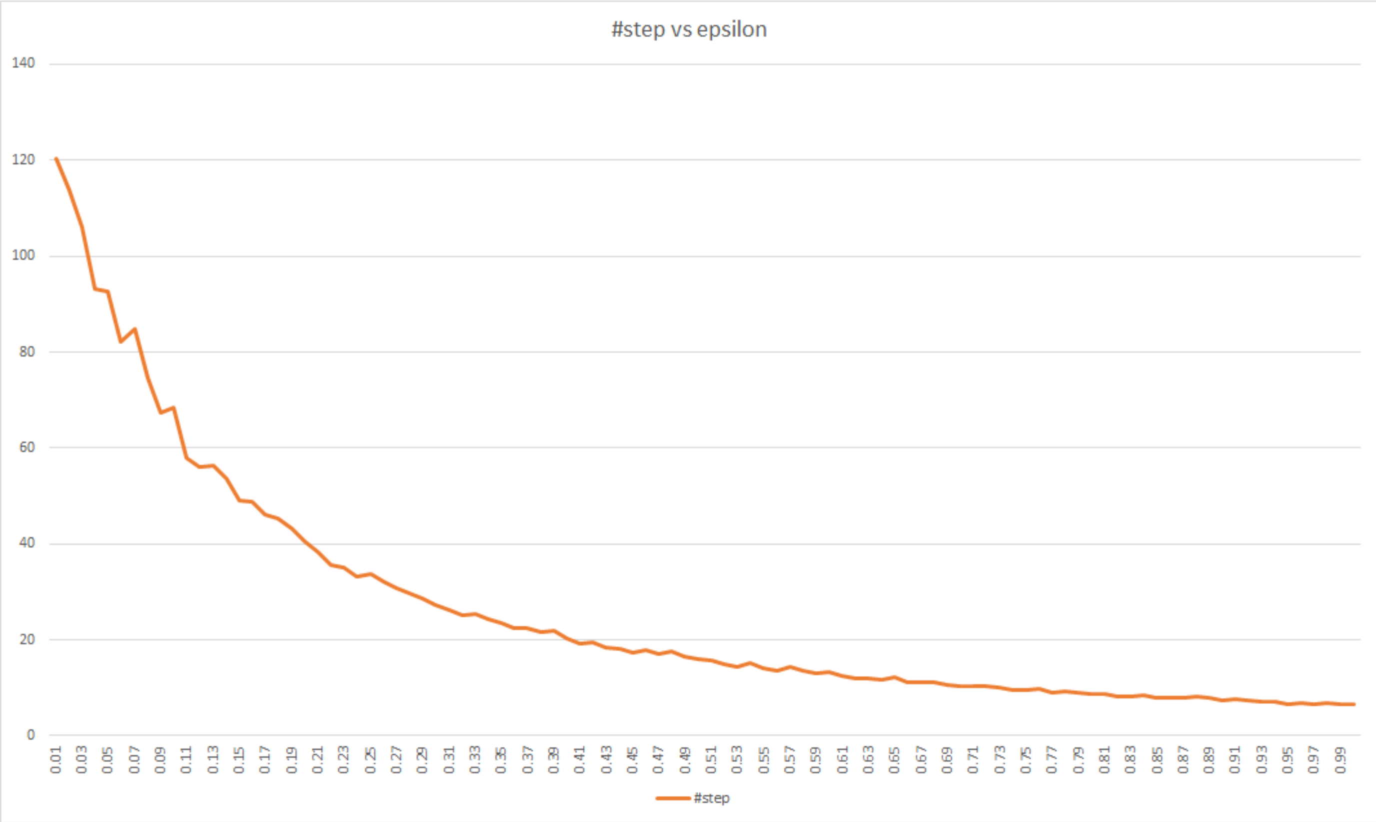

Fig. 3 the number of training steps with different ϵ.

$\#steps \propto \frac{1}{\epsilon^c}$, where $0 < c \leq 2$.

Ideally, $\#steps \propto \frac{1}{\epsilon^2}$. But notice that as ϵ goes up, the range of γ also goes up. Therefore, it is less and less possible that γ is closed to ϵ when ϵ is larger and larger. (We cannot promise the output perceptron is the best perceptron as shown in Q2.) Therefore, the number of steps gets a little bit larger than the ideal one when ϵ is large.

Hence, in general, as the ϵ goes up, the number of steps goes down, while its decreasing rate is also goes down.

4. typically independent of the ambient dimension

Here is the result: (Fig. 4)

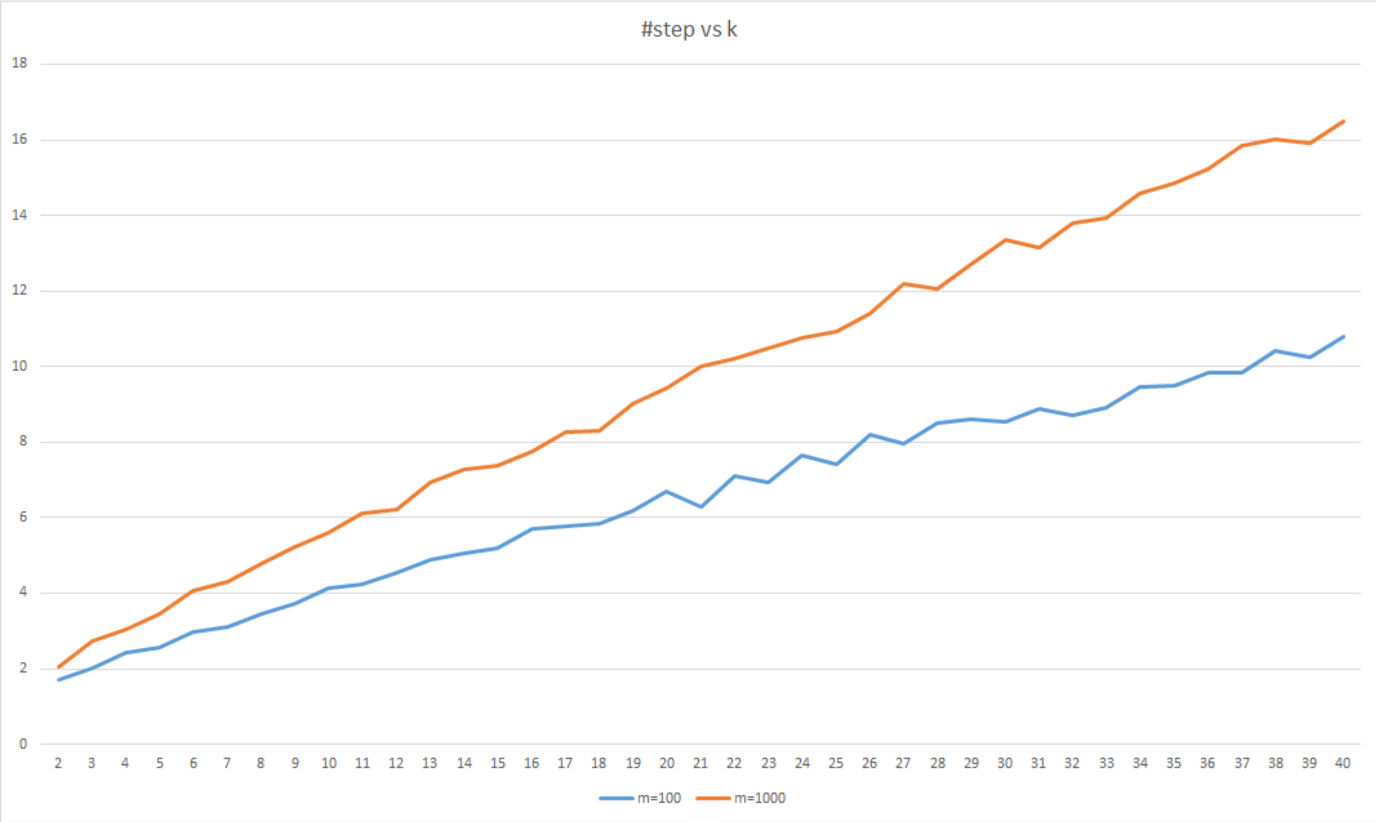

Fig.4 the number of training steps with different k when $m = 100$ and $m = 1000$.

Though the number of steps does increase as the k goes up linearly, it is relatively slow compared with ϵ, where $\#steps = 16.5$ when $\epsilon = 0.49,\ k = 20,\ m = 100$.

Also, it is reasonable to increase beacuse as we increase the dimension, we have to take care of more w_i. Namely, $P(\bigcap_{i=1}^{m-1} \frac{wi}{wm} \in (-\theta, \theta)) < P(\bigcap_{i=1}^{n-1} \frac{wi}{wn} \in (-\theta, \theta))$, where $m > n$, θ is a small number.

5. non-separable data

a. verify

Notice that all data points in the circle whose center is $(0, 0)$, $r = \sqrt{2}$ are negative data points.

Therefore, a sufficient (but not necessary) condition is that there are negative data points and there are positive data points in all 4 quadrant.

```
check(X, Y):
    nX = X[Y == -1]
    pX = X[Y == 1]
    pX1 = pX[pX[:,0]>0 & px[:,1]>0]
    pX2 = pX[pX[:,0]<0 & px[:,1]>0]
```

```
    pX3 = pX[pX[:, 0]<0 & px[:, 1]<0]
    pX4 = pX[pX[:, 0]>0 & px[:, 1]<0]
    if nX.size * pX1.size * pX2.size * pX3.size * pX4.size > 0:
        return True
    else
        return False
```

Since there are `m = 100` data points, it is pretty likely to pass this test.

b. the progression of weight vectors and bias values

Here is the result: (Fig. 5 and 6)

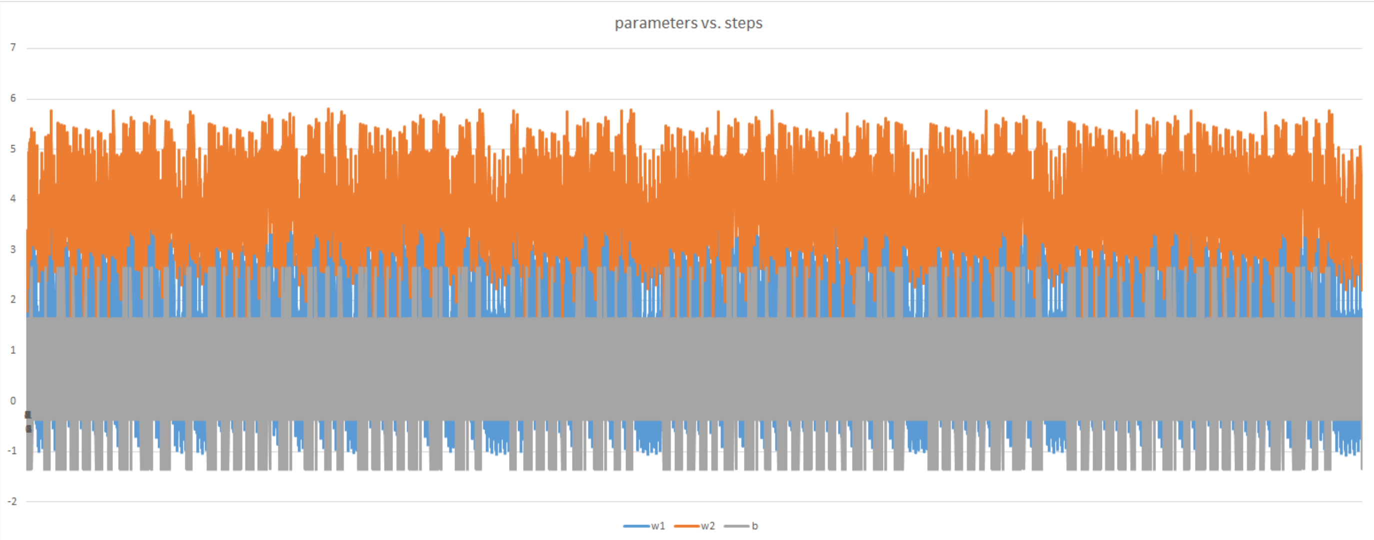

Fig.5 the whole progression while the perceptron is learning a non-separable data set.

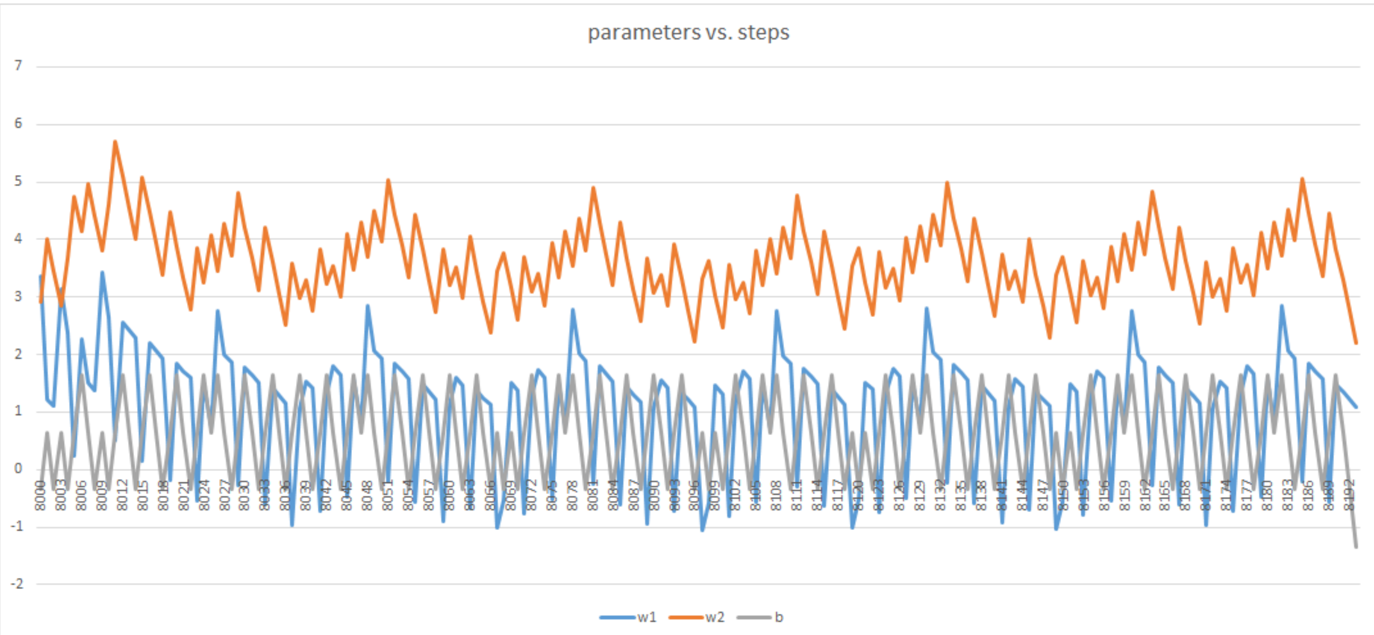

Fig.6 the last part of the progression while the perceptron is learning a non-separable data set.

Compared with separable data (See Fig. 7), the progression always keeps “shaking” in a small range.

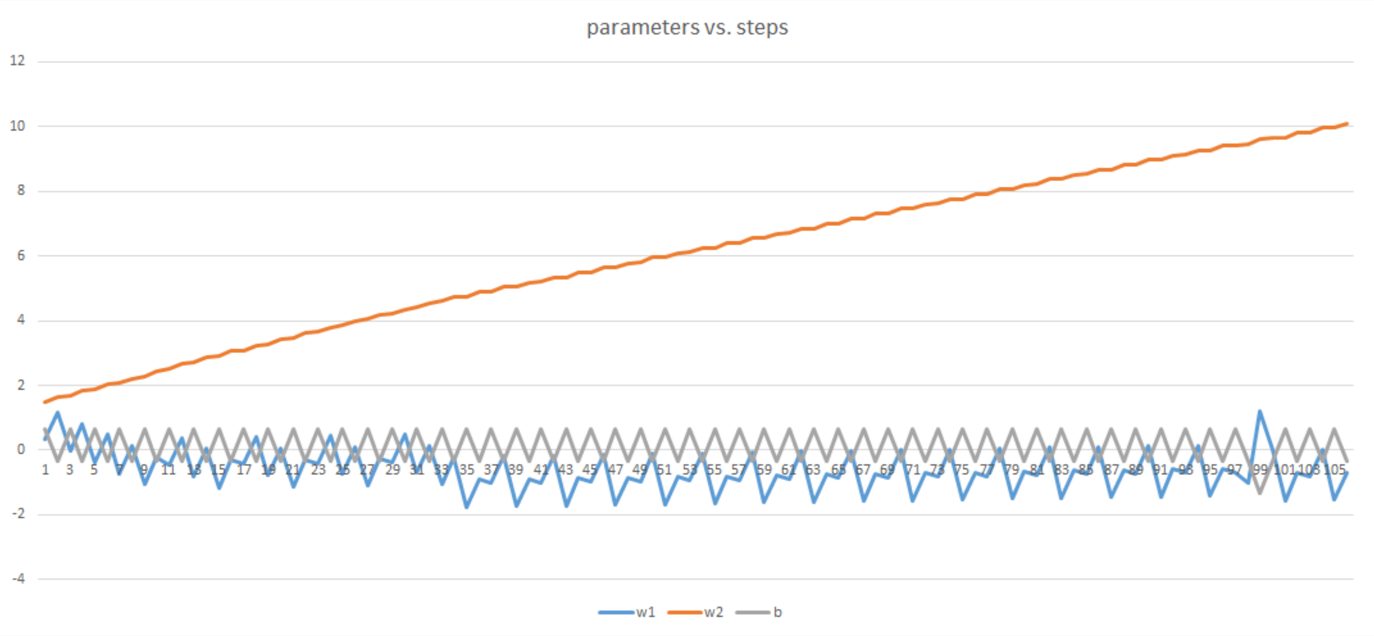

Fig. 7 the whole progression while the perceptron is learning a separable data set.

Therefore, we can consider use a patient function to halt this algorithm:

```
fit(patientStep, patientThreshold):
    step = 1
    learningFlag = True
    prevPara = (w, b)
    while learningFlag:
        learningFalg = False
        if step // patientStep == 0:
            dist = getDist((w, b), prevPara)
            if dist < getThreshold(patientThreshold, step):
                return False
            else:
                prevPara = (w, b)
        for i in range(m):
            if forward(X[i]) != Y[i]:
                step = step + 1
                learningFlag = True
                w = w + Y[i] * X[i]
                b = b + Y[i]
                break
```

```
getDist(temp, prev):
    return sum(square(temp - prev))
getThreshold(patientT, step):
    return someFunction(patientT, step)
```

`getDist()` is used to get the difference between temporary parameters and `patientStep` -step-previous parameters. If they are similar, we might suspect that the data could be non-separable. `getThreshold()` is a slowly increasing function with `step` , and therefore we can avoid the uncertainness of setting `patientThreshold` . Because if it is non-separable, the parameter will keep “shaking” in a small range, and `getThreshold()` will eventually get larger than that range.