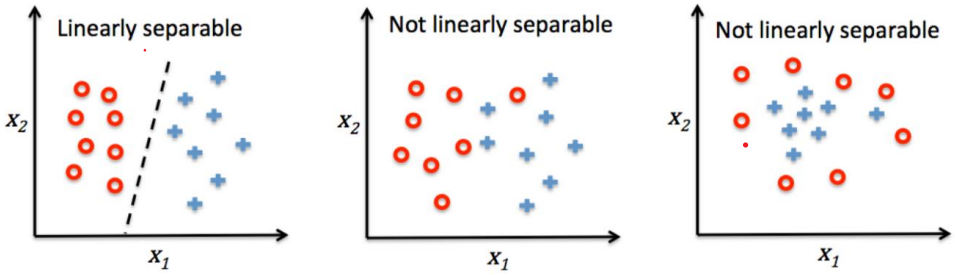
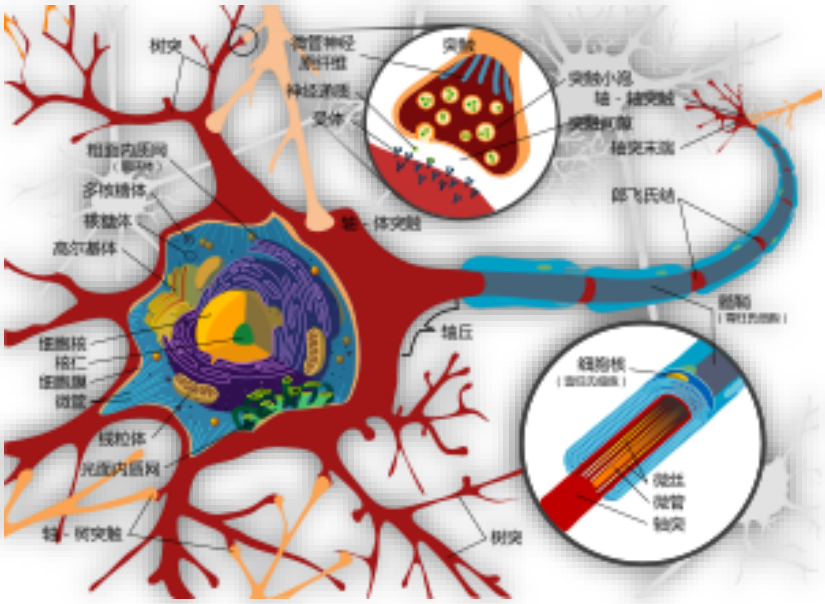
線性分類-感知器(Perceptron)

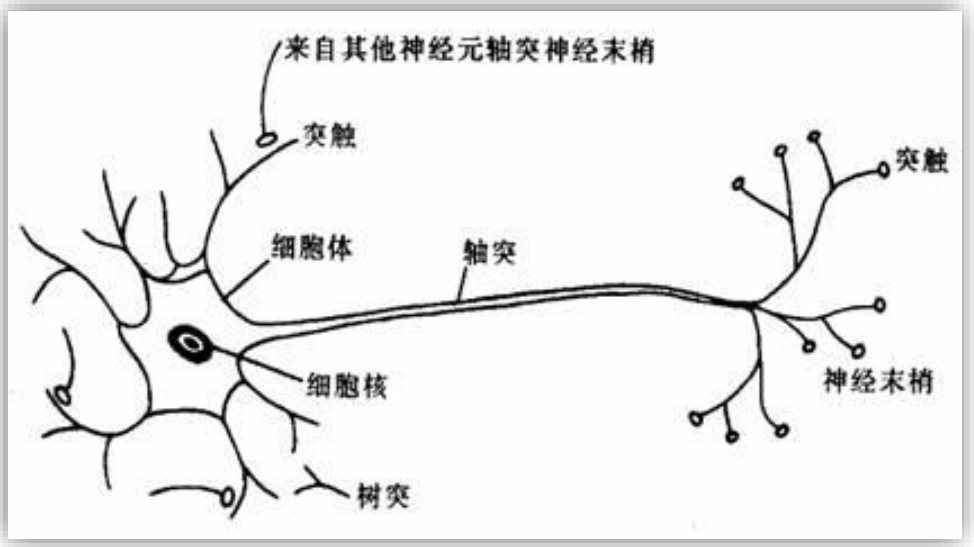
一、探究動機

機器學習(machine learning)或人工智慧(artificial intelligence)的技術目前確實應用在各個領域，在機器學習領域最早被開發出來的學習演算法是感知器(Perceptron)，也稱為感知器學習演算法(Perceptron Learning Algorithm，簡稱 PLA)。感知器演算法只有在資料是**線性可分(linearly separable)**的形況下才能正確分類(演算法才會停止)，換句話說，當資料可以使用直線、平面或是超平面切開的資料稱為線性可分割的。以 2 維資料來說，就是可以在平面上找一條線去完全切出這兩群，3 維資料的話就是可以在空間中找一個平面可以完全切割兩群，不會誤判。下圖中，最左圖即是 2 維情況下線性可分割的情況，其他則不是。



感知器的發展來自於模擬人類的神經系統的學習想法，並經過數學模型的建構後，讓機器可以模擬人類學習的最簡單的系統。下圖是人類神經系統的示意圖，其中突觸是神經接收訊息的地方(input)，若 input 的訊號總和強度大於等於某一個值(threshold)，這個神經元就會透過軸突(axon)發送訊息到右邊的突觸給下一個神經元進行反應。舉例來說，當手拿一杯比較熱的水，手的神經接受到水的溫度超過可以承受的程度，大腦就可以下達放下杯子的命令。





資料科學家**把突觸當成感測器，接受外界的輸入(input)**，而這些**輸入經過加權後的值，再與「可以承受的程度」的某一個定值(假設為b)進行比較後進行反應**。因此，可以將數學模式定義如下：

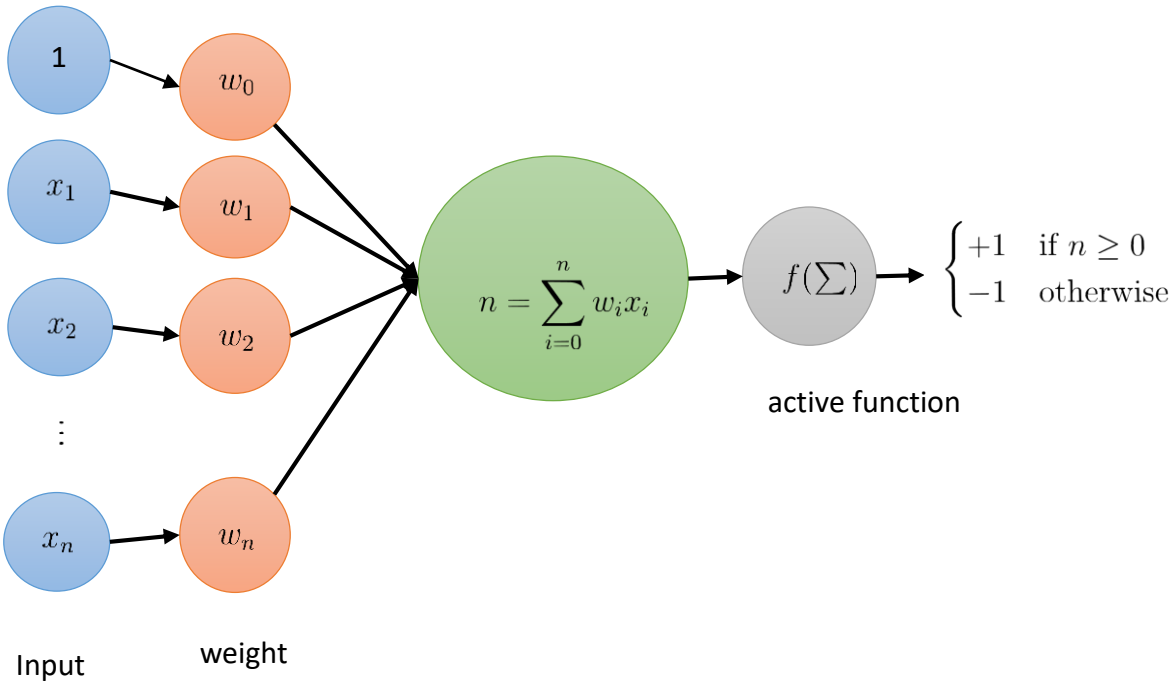
$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \mathbf{w}^T \cdot \mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b$$

或

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}, \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \text{其中} w_0 = b$$

當$n = 2$時，就是直線方程式$w_1 x_1 + w_2 x_2 + b = 0$。有趣的是這個模型並成了數學內積的運算，並將運算後的結果透過函數映射值來進行反應或是決策。以二元分類器(*binary classifier*)為例，把矩陣上的輸入$\mathbf{x}$(實數值向量)經過加權計算，再透過函數(直線方程式)$f(n)$，就可以得到兩類的結果，其中

$$f(n) = \begin{cases} +1 & \text{if } n \geq 0 \\ -1 & \text{otherwise} \end{cases}。$$



對學生來說，「機器能夠學習」或是「電腦能夠學習」是一件很酷的事，但卻通常認為機器學習或是人工智慧好像是很遙遠的事。其實，背後牽涉到許多數學理論和電腦計算的概念，原來簡單的情況下，高中數學也可以被運用在上面。因此，本計畫將運用高中數學在簡單情況下的機器學習方法 2 維，且是 2 類資料感知器的學習行為進行瞭解，藉由 C++程式語言進行實作，希望能讓各位對於機器學習能更有概念。

二、感知器的學習演算法

基本上要讓電腦能夠工作就是需要有演算法(algorithm)，感知器學習演算法在 2 類資料，通常設定為+1 和-1 兩類，其**學習行為**是：

Step 1：以隨機方式初始化權重w。

Step 2：將**所有資料**，即二維平面上所有的選定的點（通常是又稱為**訓練樣本**），利用 Step 1 中的直線來進行類別判定，若是正確判定，則不進行更新，然若有對訓練資料點**誤判(真實標籤不符的資料)，則對直線進行權重更新**。其實，對於**權重更新通常就是讓直線進行旋轉和平移**，且往**對的方向去進行**。依此類推，直到找到可以切割兩類資料的直線則停止執行。

Step 3:將測試樣本利用學習到的直線來進行判定後計算辨識正確率，已瞭解學習到的直線在測試樣本的辨識效果。

由上述演算法可知：**一開始先隨便給一條直線，並將訓練樣本點一個一個送給直線判定，有需要則進行直線的旋轉和平移，以找到切割線**。

Guarantee of PLA

PLA **演算法停止**必須滿足「**訓練集**所有樣本都是線性可分的（*linear separable*）」，也就是說**平面上必須至少存在一條線的，可以將資料「一刀兩斷」**。

$$w_j^{(i)} = w_j^{(i-1)} + \Delta w_j$$

$$b^{(i)} = b^{(i-1)} + \Delta b^{(i-1)}$$

，其中i表示疊代次數，j表示維度

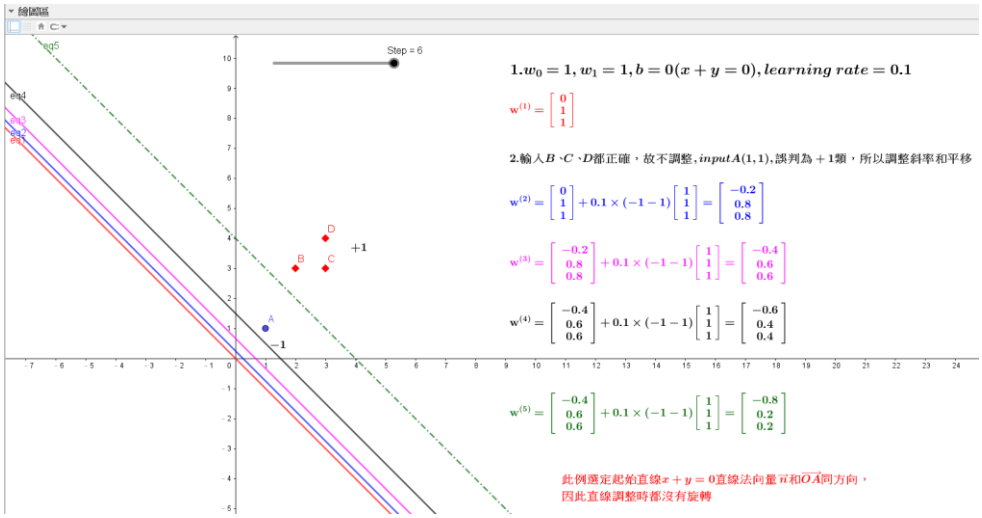
又

$$\Delta w_j^{(i-1)} = \zeta(y - \hat{y}) x_j^{(i-1)}$$

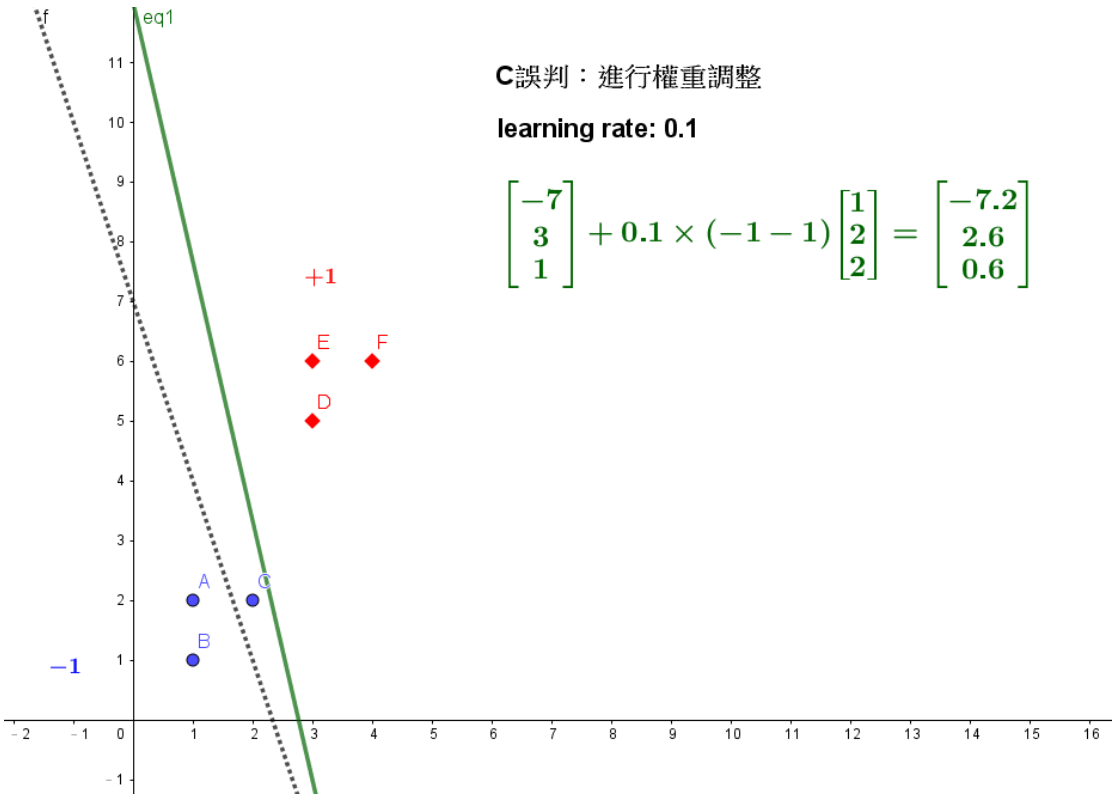
$$\Delta b^{(i-1)} = \zeta(y - \hat{y})$$

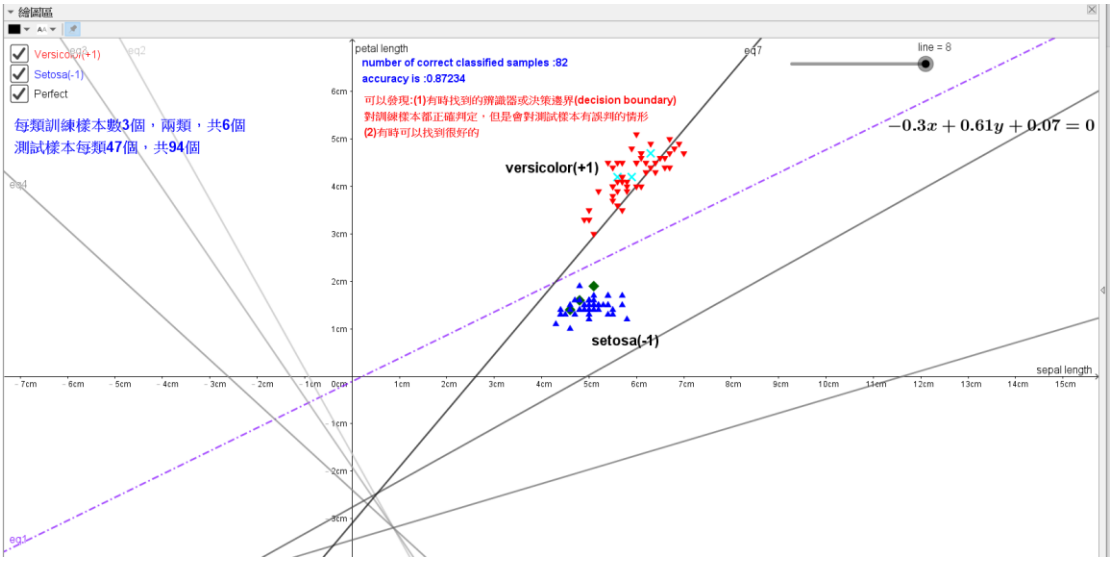
，其中y表示訓練樣本的真實類別(true label or target)，$\hat{y}$表示輸入直線的判定之類別(classified label or prediction)，ζ表示學習率(*learning rate*) ，$x_j^{(i-1)}$表示誤判訓練樣本的座標值。

【案例-只平移】



【案例-轉向又平移】


f
eq1
C誤判：進行權重調整
learning rate: 0.1
$\begin{bmatrix}-7\\3\\1\end{bmatrix}+0.1\times(-1-1)\begin{bmatrix}1\\2\\2\end{bmatrix}=\begin{bmatrix}-7.2\\2.6\\0.6\end{bmatrix}$
+1
E
F
D
A
C
B
−1


繪圖區
Versicolor(+1)
Setosa(-1)
Perfect
每類訓練樣本數3個，兩類，共6個
測試樣本每類47個，共94個
petal length
number of correct classified samples :82
accuracy is :0.87234
可以發現:(1)有時找到的辨識器或決策邊界(decision boundary)
對訓練樣本都正確判定，但是會對測試樣本有誤判的情形
(2)有時可以找到很好的
line = 8
$-0.3x+0.61y+0.07=0$
versicolor(+1)
setosa(-1)
sepal length

程式參考架構

```
#define DIM 2    //dimensionality 維度數
#define n_train 6    //訓練樣本數
#define n_test 94 //測試樣本數
#define TIMES 100 //設定最多執行的次數

class perceptron {
  public:
    perceptron();
    ~perceptron();
    double w[DIM]; //權值
    double b; //bias  偏移值
    void learning(double p[n_train][DIM], double t[n_train]);//訓練樣本和他們的label
    double classifing(double *p); //p 指向某個測試樣本，傳回判定類別，這裡設定 double 是為了方便

    void initval(); //隨機設定直線 w, b 初始值
    int hardlim(double val); //根據 w_1*(x^i_1)+w_2*(x^i_2)+b 值，判斷+1 或-1
    double lr; //learning rate:  可以設定為 0.1
    double error; //錯誤:  初始值可以設定為 0.9，當有誤判時 error += （t[i]-hardlim(output)）;
};

double acc(double *target ,double *predict);//計算測試樣本的正確率, target 指標指向測試樣本正確的 label, predict 指標指向測試樣本送入直線判定後的類別，看多少相同的就是正確判定，並計算正確率。例如：return 88/94;

//training samples
double X[n_train][DIM];
double label[n_train];
//test samples
double Y[n_test][DIM];
double target[n_test];
double pred[n_test];

int main(void){
```

```
return 0;
}
```

【提示】

訓練過程

```
int times=0;
while (fabs(Perceptron.error)> 0.001 && times <TIMES) { //沒有錯誤或是設定次數到了
        times++;
        Perceptron.learning(X, label);
}
```