Perceptron from Scratch Assembly AI

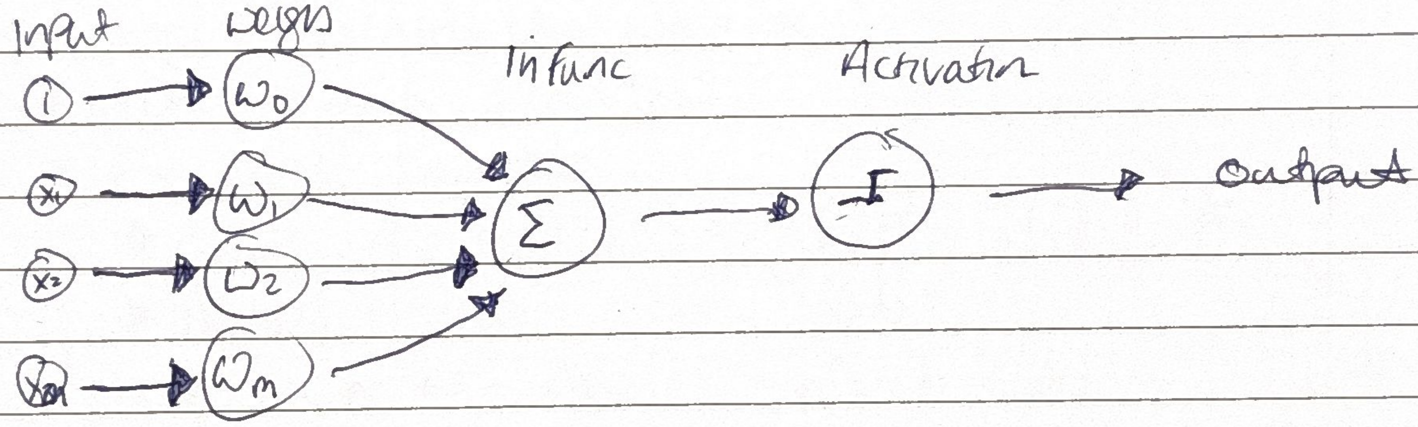
Single perceptron = can only learn lin seperable patterns
→ Multi can learn more complex

= Binary classifier

Seen as a single unit of Arti NNet

it is a single-layer NN w/ unit step function as the activation

unit step = 1 if > 0, else 0



$$f(x) = w^T x + b$$

$$g(z) = \begin{cases} \text{if } z > 0 \\ \text{else } 0 \end{cases}$$

$$\hat{y} = g(f(x)) = g(w^T x + b)$$

now we want to learn & optimize w

Perceptron update rule:
for each train sample x_i

$$w = w + \Delta w$$
$$b = b + \Delta b$$

$\Delta w = a \cdot (y_i - \hat{y}_i) \cdot x_i$

$\Delta b = a \cdot (y_i - \hat{y}_i)$

$(y_i - \hat{y}_i)$ Ka'ach as a switch

$1 - 1 = 0$, $0 - 0 = 0$ $\qquad a = [0, 1]$

$1 - 0 = 1$, $0 - 1 = -1$

→ if wrong then update

Steps

① training (learning the weights)

- init weights (random)
- for each sample
  - ▶ calc $\hat{y} = g(f(x)) = g(w^T x + b)$
  - ▶ update $\Delta w =$ $\qquad \Delta b =$
  - ▶ $w = w + \Delta w$ $\qquad b = b + \Delta b$

Same

② Prediction:

- Calculate $\hat{y} = g(f(x)) = g(w^T + b)$

Code Notes

```
import numpy
class perceptron
Def init(self, learn_rate = 0.01, niters = 1000)
    (things to set w/ self.
        learn rate    ↑
        n-Iters       ↑
        Activ func function global npwhere
        weight = None
        bias   = None

① Def funct(self, x, y)
    Shape of X, samples feats ->

    init Pars
    self.weight = np.zeros(feats) -> rand(
         bias   = 0

    y_ = step   (ensure out is 0, 1 format)

    learn/update                     (epochs)
     loop ~~the~~ range of n_iters
       loop the samples (dataset)
           idx, x_i in enumerate(X)
calc  { lin_pred np.dot($w^T$, x_i) + bias
out   { y_pred = activ func(lin_pred)

update { Δw = a * (y_[idx] - y_pred)
       { weights =+ Δw · w1
         bias    =+ Δw

② Def func(self, X)
     lin_output , ($w^T$ · x) + bias
          ret y_pred = act(lin out)
```

testing flow

```
if __name__ = "__main__":

    #imports
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_spl
    from sklearn import datasets

    Def Accuracy(y_true, y_pred)
        Acc = np.sum(y_true = y_pred)/len(y_tru
        return Acc

               get
    x,y =  make dateset

    X_train, X_test  =  split
    y_train, y_test

                                               iters
    P =  class cls w/ learn rate & epocs
    P.fit(X_train, y_train)
    Pred = P.predict(X_test)

    Print Acc

    Visuals
```