Supervised Learning Perceptron

Natthariya Laopracha

Computer Science of Mahasarakham University

Supervised Learning VS Unsupervised Learning

Supervised Learning

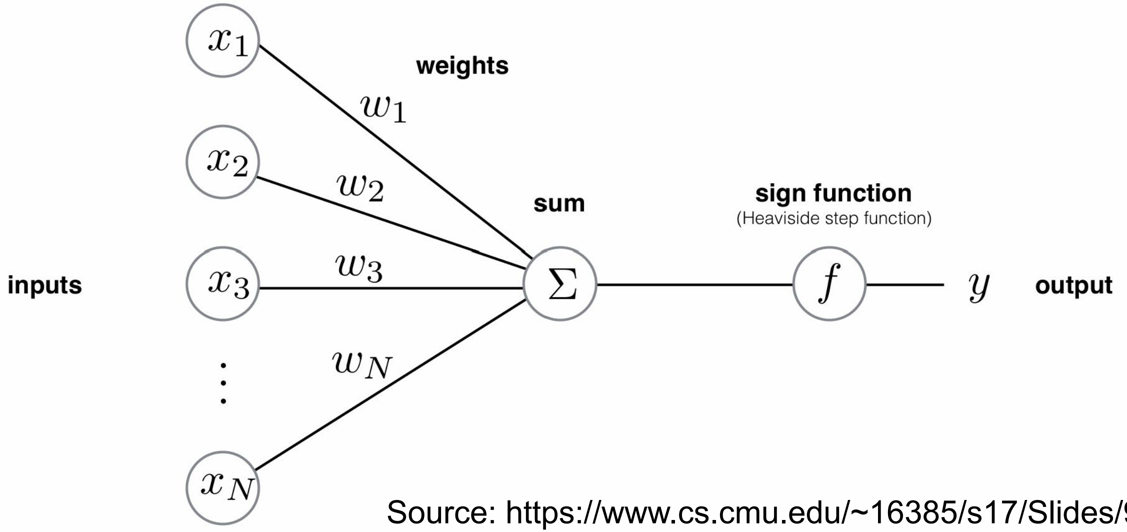
-2.14	-1.62	0
-3.33	-0.44	0
-1.05	-3.85	0
0.38	0.95	0
-0.05	-1.95	0
-3.20	-0.22	0
-2.26	0.01	0
-1.41	-0.33	0
-1.20	-0.71	0
-1.69	0.80	0
-1.52	-1.14	0
3.88	0.65	1
0.73	2.97	1
0.83	3.94	1
1.59	1.25	1
3.92	3.48	1
3.87	2.91	1
1.14	3.91	1
1.73	2.80	1
2.95	1.84	1
2.61	2.92	1

Unsupervised Learning

-2.14	-1.62
-3.33	-0.44
-1.05	-3.85
0.38	0.95
-0.05	-1.95
-3.20	-0.22
-2.26	0.01
-1.41	-0.33
-1.20	-0.71
-1.69	0.80
-1.52	-1.14
3.88	0.65
0.73	2.97
0.83	3.94
1.59	1.25
3.92	3.48
3.87	2.91
1.14	3.91
1.73	2.80

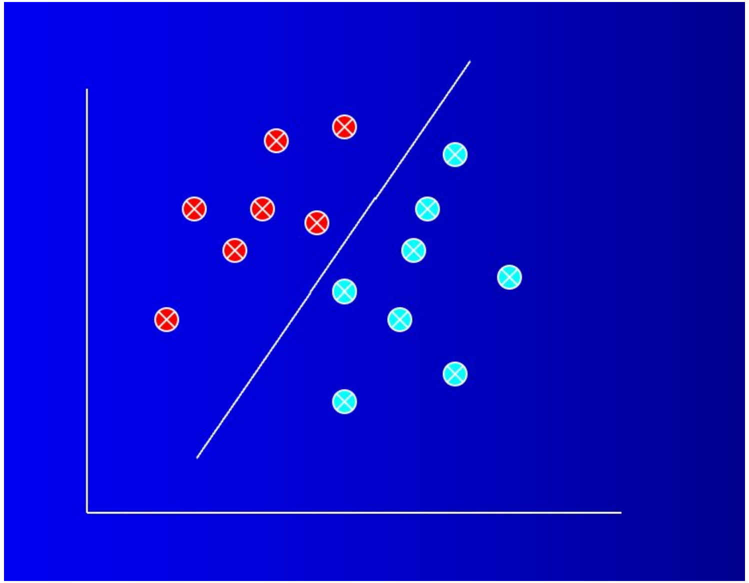
Perceptron

- A single artificial neuron that computes its weighted input and uses a threshold activation function.
- It effectively separates the input space into two categories by hyperplane.
- The Perceptron is used for binary classification.

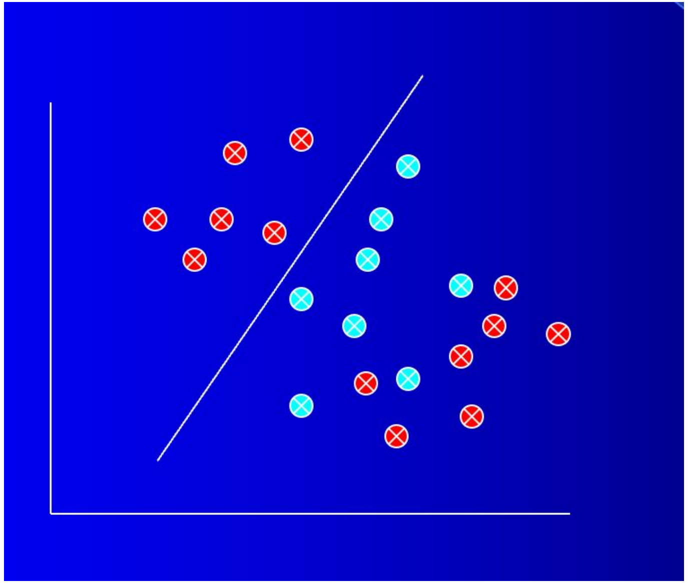


Source: https://www.cs.cmu.edu/~16385/s17/Slides/9.1_Perceptron.pdf

Perceptron



Linear Separability



Limited Functionality of Hyperplane

Source: CS 270 - Perceptron

Perceptron of steps computation

1. Preparation data

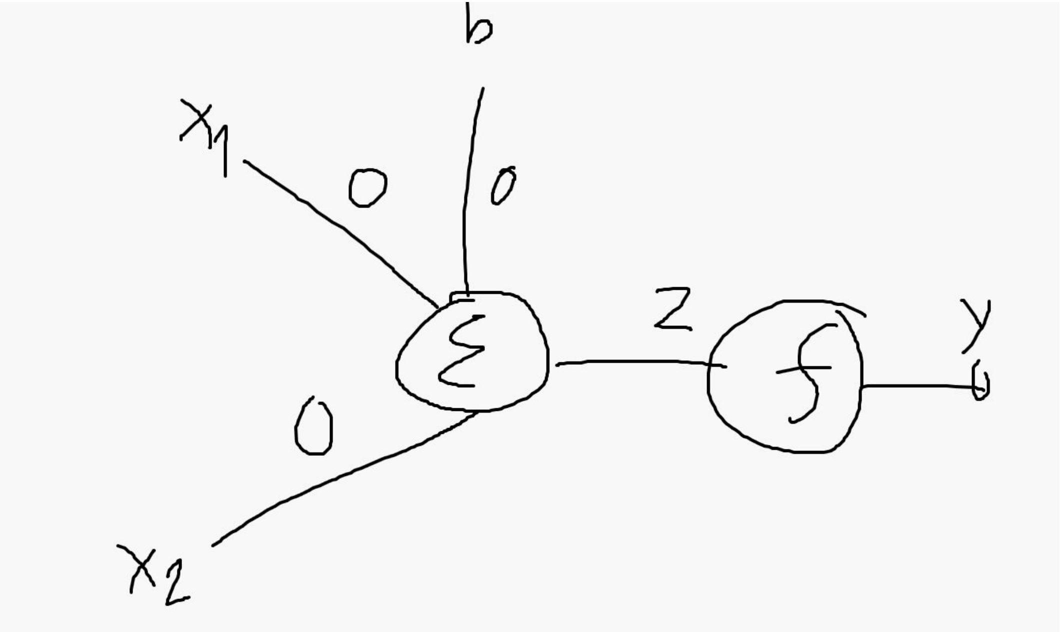
x1	x2	yt(target)
1	1	1
0	0	0
0	1	0
1	0	0

Perceptron of steps computation

2. Random weight and assign bias

w1=0, w2=0, b=1*0

Epoch 1:

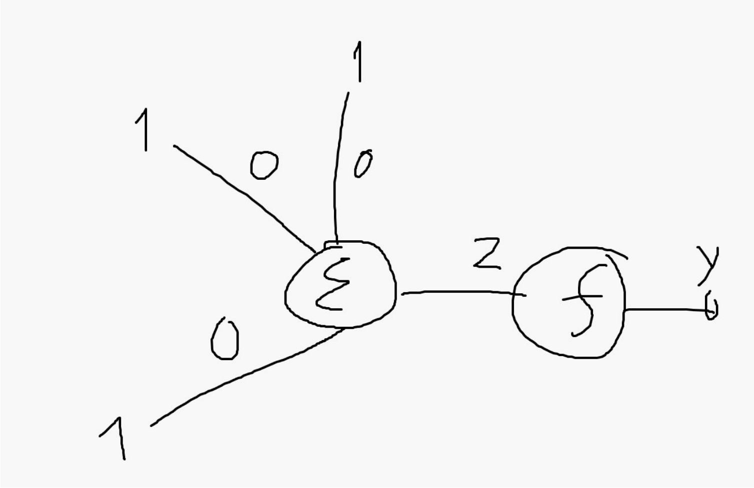


Perceptron of steps computation

3. Compute summation $z=b+x1w1+x2w2+...+xnwn$

$z=1*0+(1*0)+(1*0)$

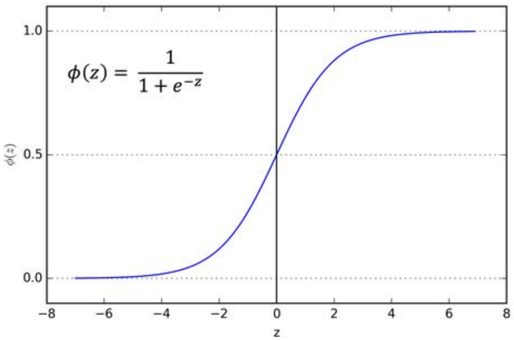
$z= 0$



x1	x2	yt(target)
1	1	1
0	0	0
0	1	0
1	0	0

Perceptron of steps computation

3. Use activation function example activation function
   3.1 Sigmoid function activation



Perceptron of steps computation

3. Use activation function example activation function

$$y = \frac{1}{1 + e^{-z}}$$

$$y = \frac{1}{1 + e^{-0}}$$

y=0.5

Perceptron of steps computation

4. Compute error (e)

   e=yt-y

   e=1-0.5

   e=0.5

5. Update weight

   newW=e*xi+oldw

   neww1=0.5*1+0=0.5

   neww2=0.5*1+0=0.5

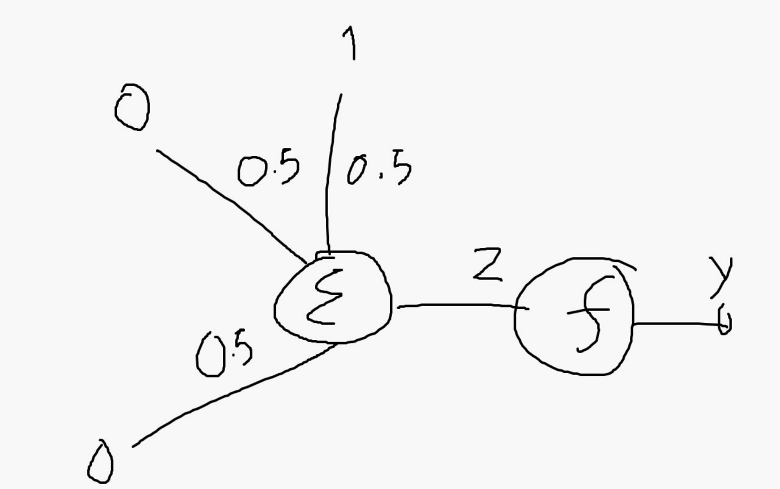
   newbias=0.5*1+0=0.5

6. Feed data input row 2 for learning in perceptron and go in step 3.

Stop when the condition is false.

Perceptron of steps computation

Epoch 2:



x1	x2	yt(target)
1	1	1
0	0	0
0	1	0
1	0	0

Using model of perceptron

y=w1*x1+w2x2+...+wnxn+bias
output=f(y)

Example

Input data

x1	x2	b	yt(target)
1	1	1	1
0	0	1	0
0	1	1	0
1	0	1	0

Work

Compute 10 epochs from data example

Code

```
import numpy as np
import matplotlib.pyplot as plt
data=np.array([[1,1,1],[0,0,0],[0,1,0],[1,0,0]])
```

```
X_train=data[0:2,0:2]
X_test=data[3:4,0:2]
y_train=data[0:2,2]
y_test=data[3:4,2]
print(X_test)
print(y_test)
```

Code

```
plt.scatter(X_train[y_train==0, 0], X_train[y_train==0, 1], label='class 0', marker='o')
plt.scatter(X_train[y_train==1, 0], X_train[y_train==1, 1], label='class 1', marker='s')
plt.title('Training set')
plt.xlabel('feature 1')
plt.ylabel('feature 2')
plt.xlim([-3, 3])
plt.ylim([-3, 3])
plt.legend()
plt.show()
```

Code

```python
class Perceptron():
    def __init__(self, num_features):
        self.num_features = num_features
        self.weights = np.zeros((num_features, 1), dtype=np.float32)
        self.bias = np.zeros(1, dtype=np.float32)

    def forward(self, x):
        linear = np.dot(x, self.weights) + self.bias # comp. net input
       # print('  Weights: %s\n' % linear)
        predictions = np.where(linear > 0., 1, 0)
        return predictions

    def backward(self, x, y):
        predictions = self.forward(x)
        errors = y - predictions
        return errors

    def train(self, x, y, epochs):
        for e in range(epochs):

            for i in range(y.shape[0]):
                errors = self.backward(x[i].reshape(1, self.num_features), y[i]).reshape(-1)
                self.weights += (errors * x[i]).reshape(self.num_features, 1)
                self.bias += errors

    def evaluate(self, x, y):
        predictions = self.forward(x).reshape(-1)
        accuracy = np.sum(predictions == y) / y.shape[0]
        return accuracy
```

Code

```
ppn = Perceptron(num_features=2)
ppn.train(X_train, y_train, epochs=10)

print('Model parameters:\n\n')
print('  Weights: %s\n' % ppn.weights)
print('  Bias: %s\n' % ppn.bias)
```

Code

```
train_acc = ppn.evaluate(X_train, y_train)
print('Train set accuracy: %.2f%%' % (train_acc*100))
```

```
test_acc = ppn.evaluate(X_test, y_test)
print('Test set accuracy: %.2f%%' % (test_acc*100))
```