Artificial Neural Network

Jony Sugianto
jony@evolvemachinelearners.com
0812-13086659
github.com/jonysugianto

Neural Network Application

https://www.youtube.com/watch?v=hPKJBXkyTKM

Neural Network Application

https://www.youtube.com/watch?v=-96BEoXJMs0

Biological Neuron

https://www.youtube.com/watch?v=NsBaPtemAjs

Dopamin and the brain

https://www.youtube.com/watch?v=lIJW9XLDE-I

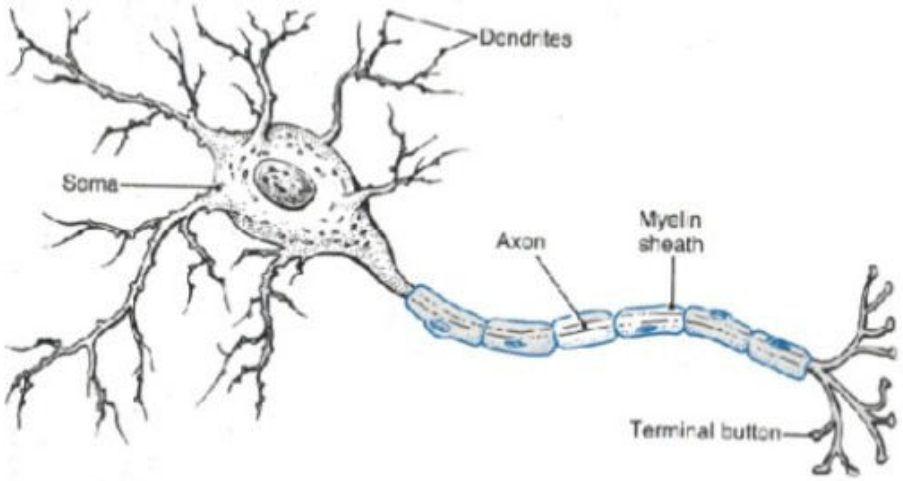
From Biology to Computer Model

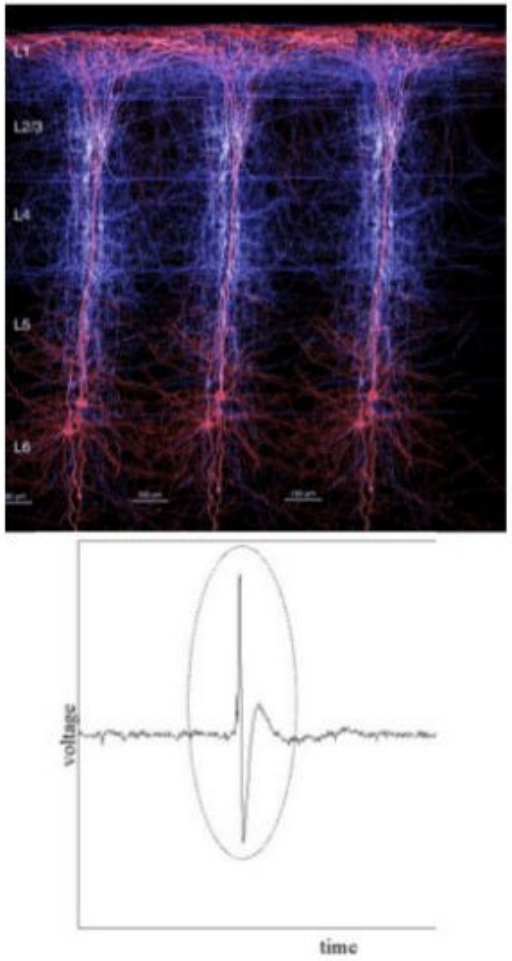
EVOLVE
MACHINE LEARNERS

https://www.youtube.com/watch?v=gcK_5x2KsLA

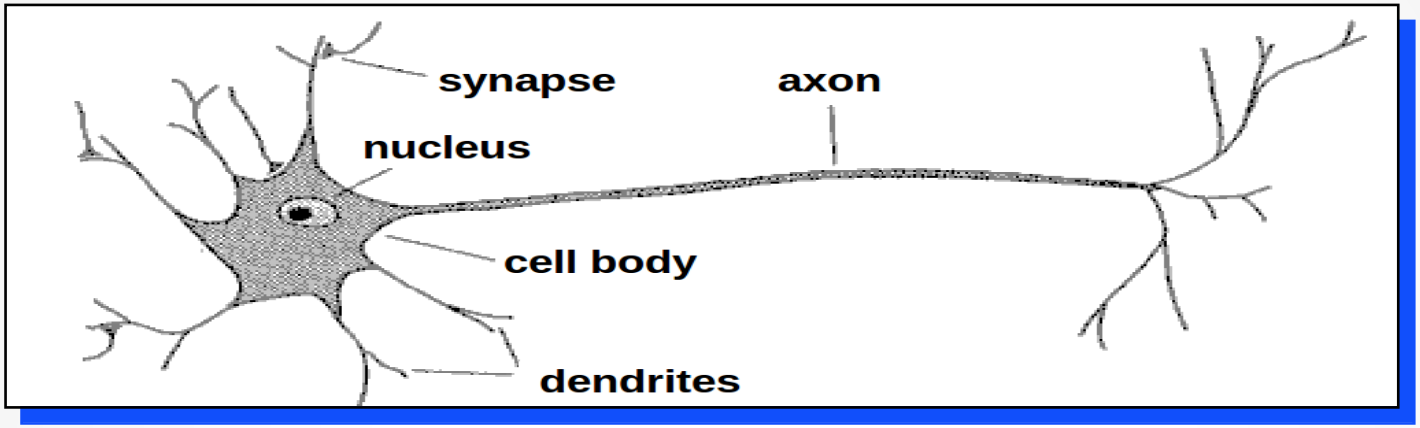
How does your brain work?

- 100,000,000,000 neurons
- 10,000 dendritic inputs per neuron
- 1 electrical output





Biological neuron



- A neuron has
  - A branching input (dendrites)
  - A branching output (the axon)
- The information circulates from the dendrites to the axon via the cell body
- Axon connects to dendrites via synapses
  - Synapses vary in strength
  - Synapses may be excitatory or inhibitory

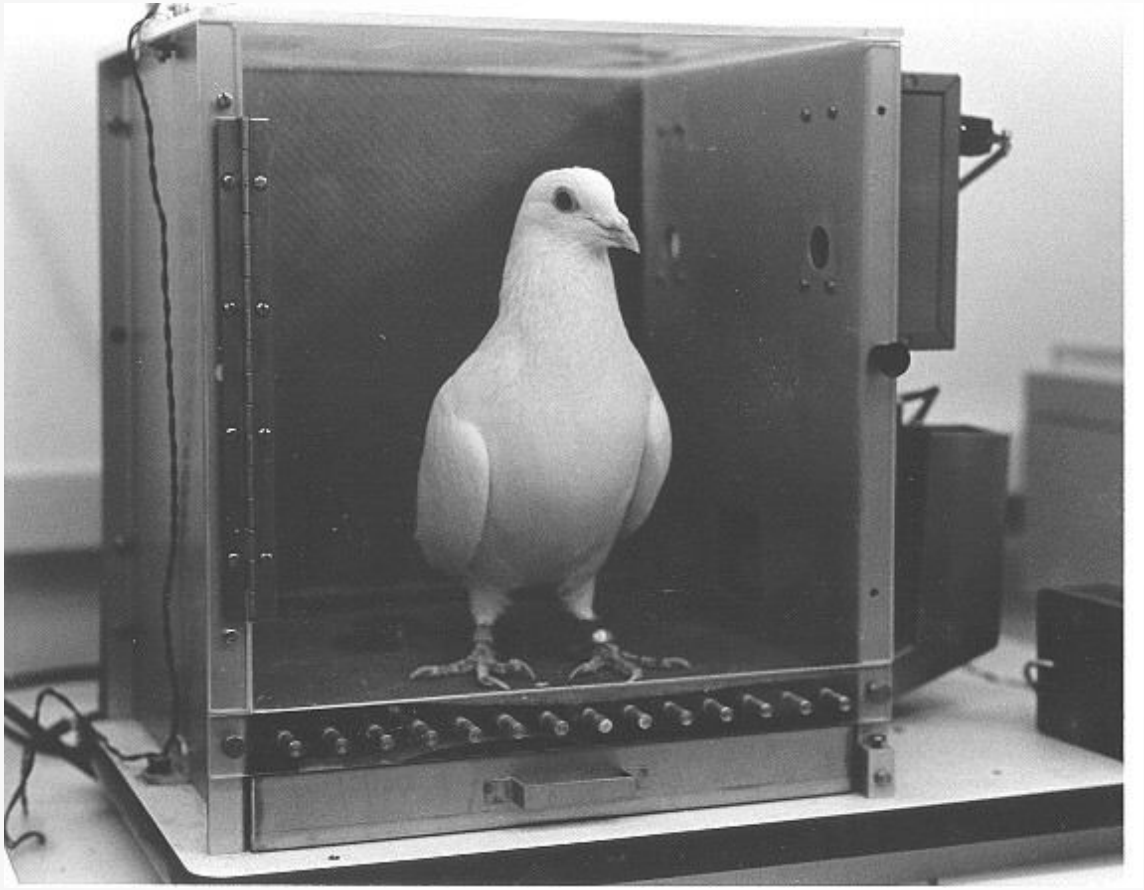
Properties of the brain

- It can learn, reorganize itself from experience
- It adapts to the environment
- It is robust and fault tolerant

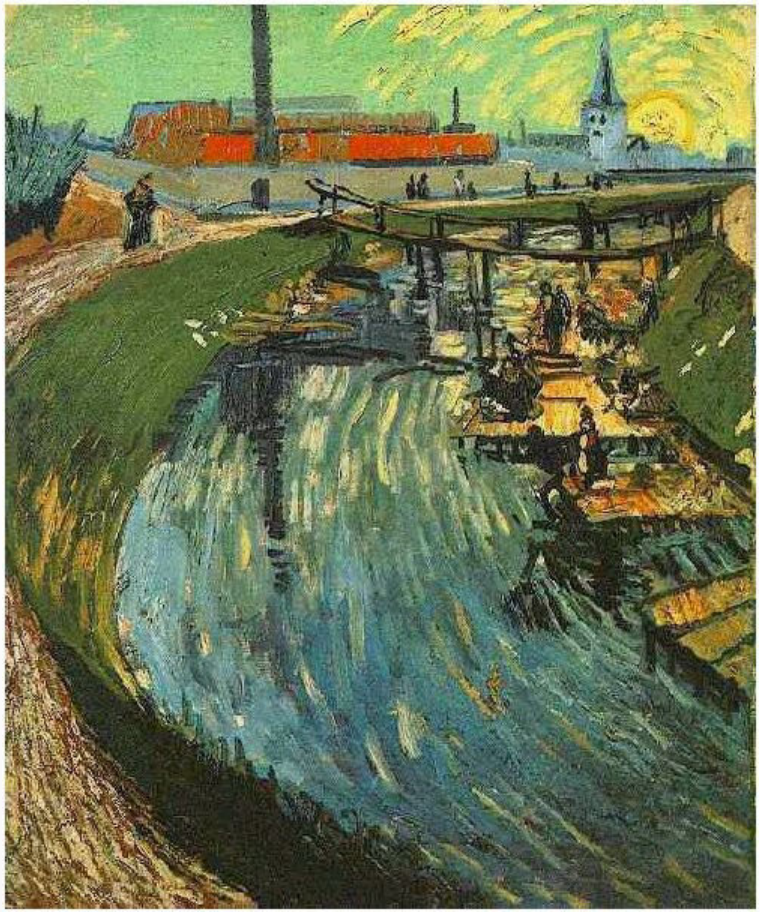
Biological Neural Nets

- Pigeons as art experts (Watanabe *et al.* 1995)
-
-
  - Experiment:
    - Pigeon in Skinner box
    - Present paintings of two different artists (e.g. Chagall / Van Gogh)
    - Reward for pecking when presented a particular artist (e.g. Van Gogh)

Pigeons as art experts



Pigeons as art experts



Pigeons as art experts

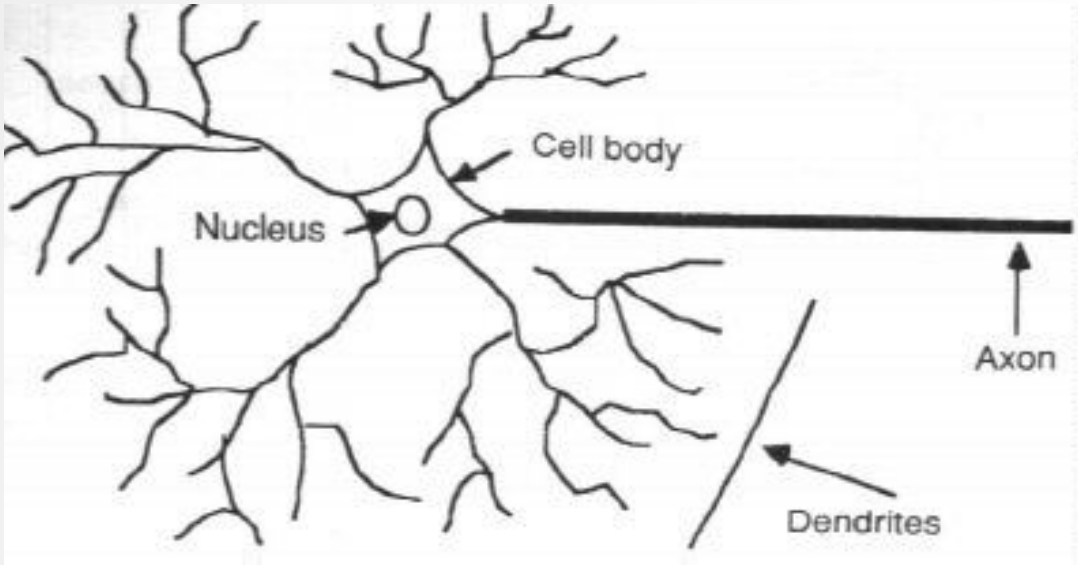
Pigeons as art experts

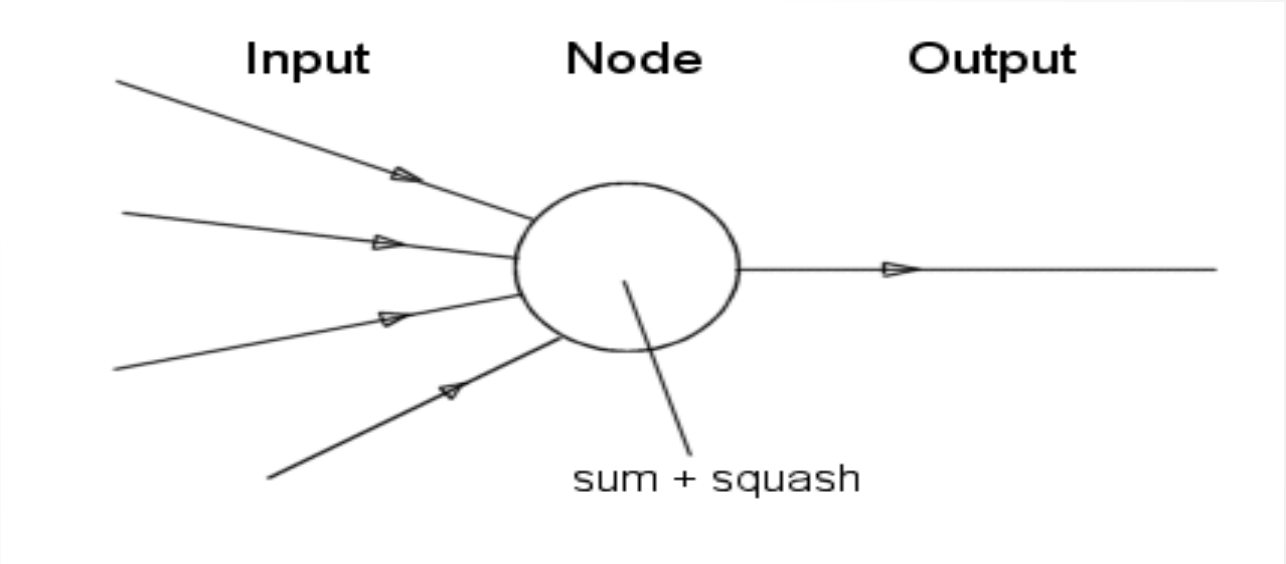
- Pigeons were able to discriminate between Van Gogh and Chagall with 95% accuracy (when presented with pictures they had been trained on)
- Discrimination still 85% successful for previously unseen paintings of the artists
- Pigeons do not simply memorise the pictures They can extract and recognise patterns (the ‘style’) They generalise from the already seen to make predictions
- this is what neural networks (biological and artificial) are good at (unlike conventional computer)

What are Neural Networks?

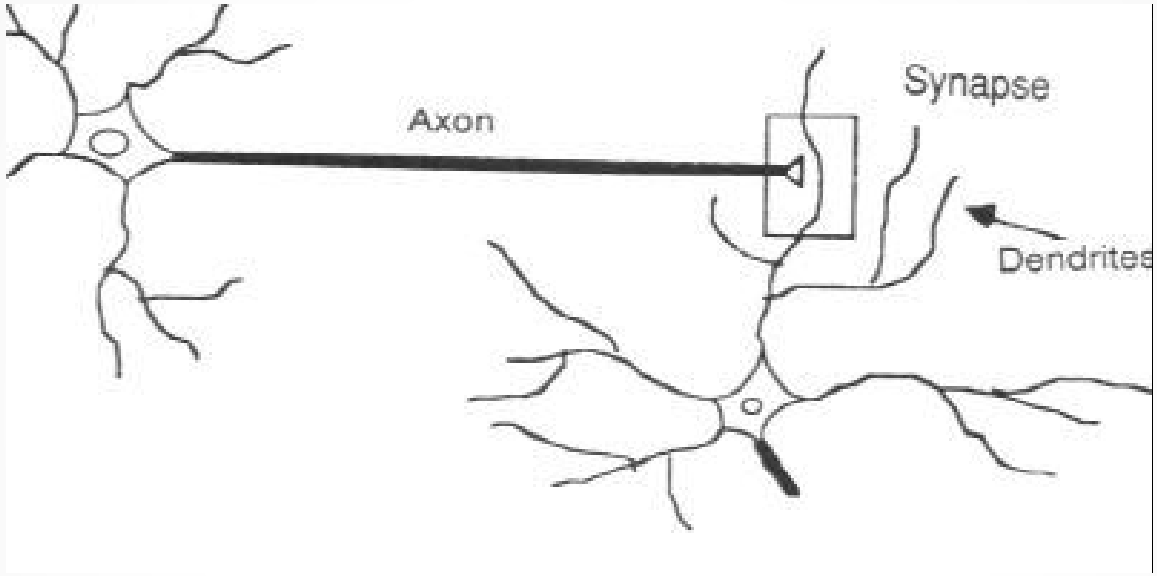
- Models of the brain and nervous system
- Highly parallel
  - Process information much more like the brain than a serial computer
- Learning
  - Very simple principles
  - Very complex behaviours
- Applications
  - As powerful problem solvers
  - As biological models

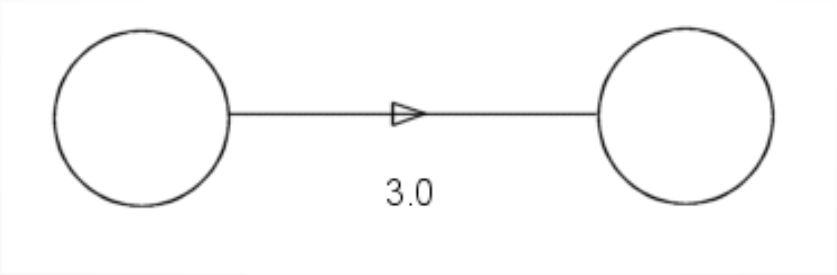
Bilogical Neurone vs. Artificial Neuron





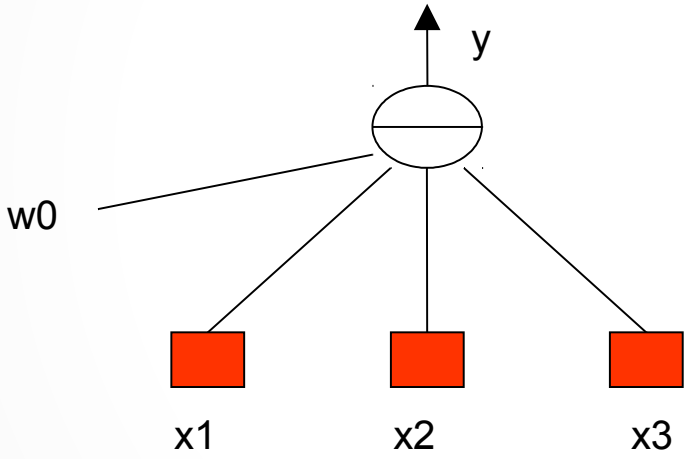
Synapse vs. weight



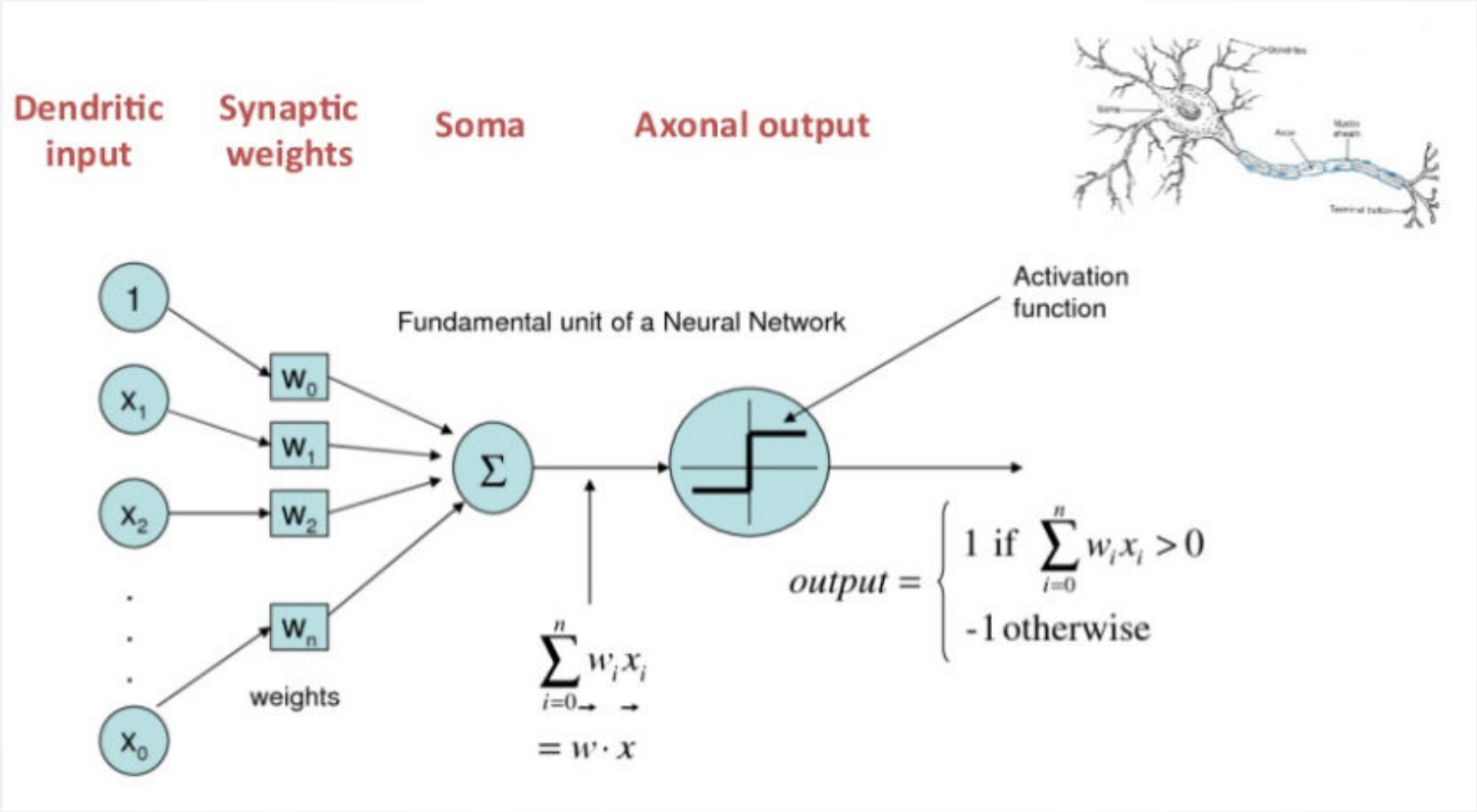


What is an artificial neuron ?

- Definition : Non linear, parameterized function with restricted output range

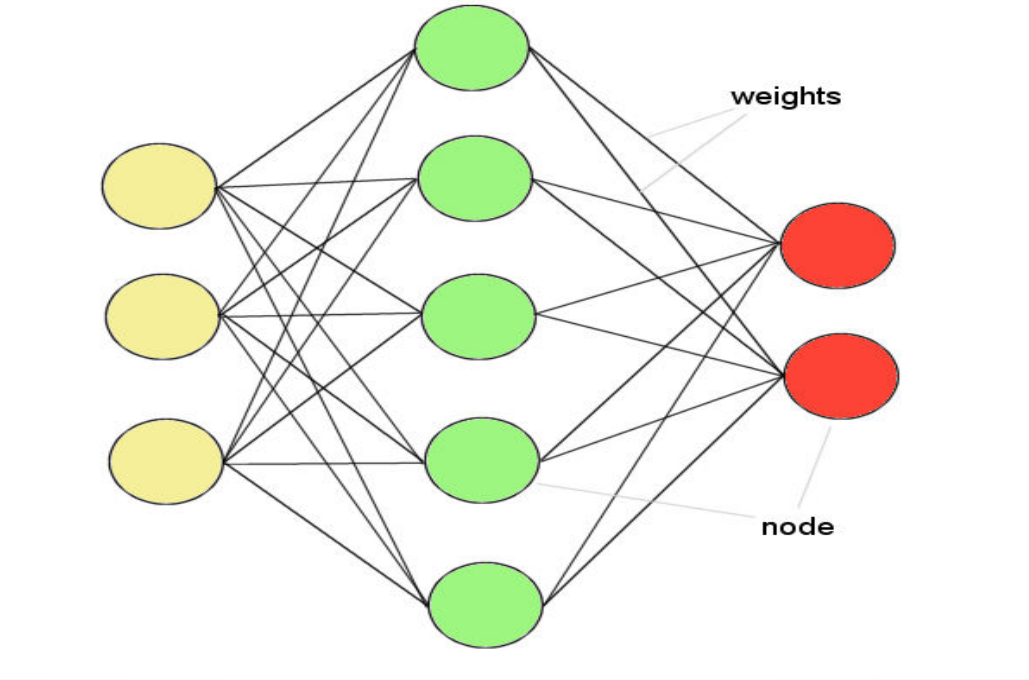


One Simple Abstraction



ANNs – The basics

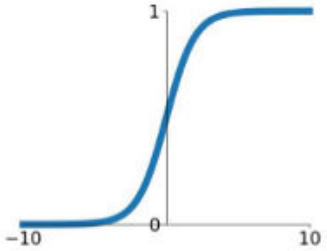
- ANNs incorporate the two fundamental components of biological neural nets:
  1. Neurones (nodes)
  2. Synapses (weights)



Activation Functions

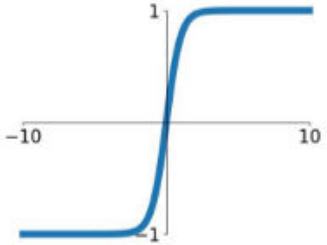
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



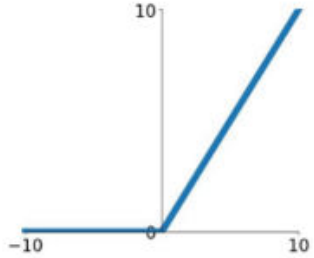
tanh

$$\tanh(x)$$



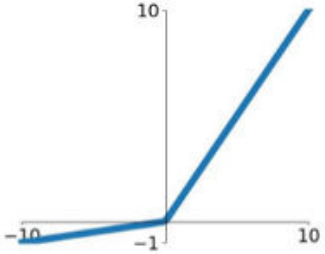
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

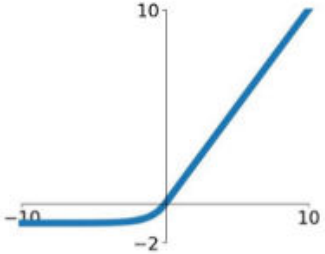


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

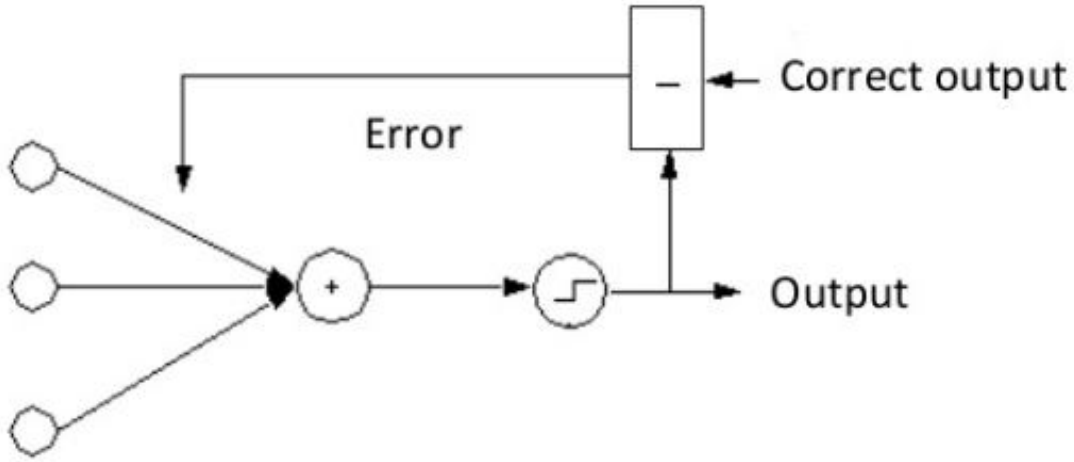
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



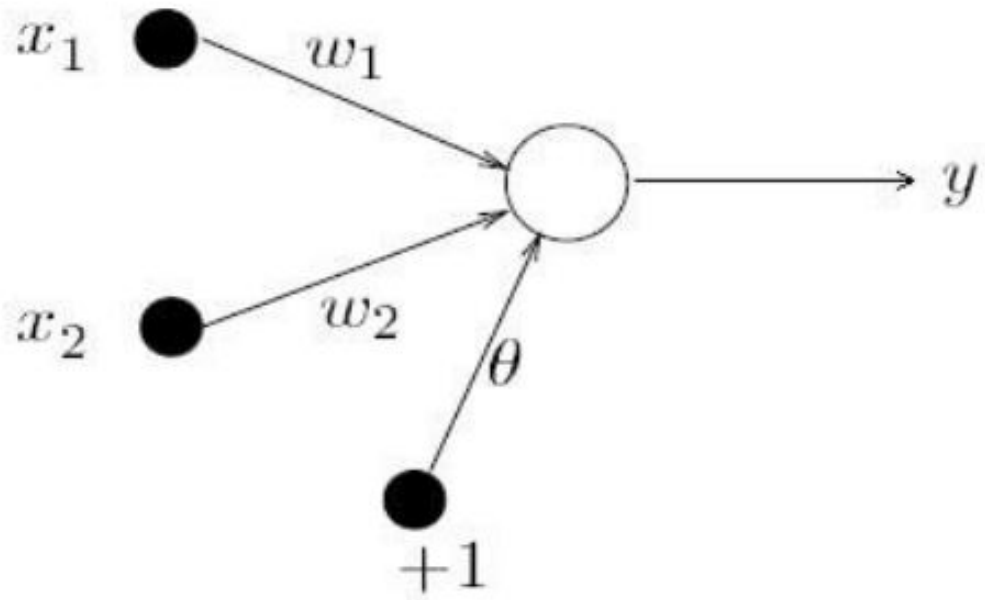
How to learn the weights?

- If we know what output should look like, can compute error and update weights to minimize it
  - Optimization problem, typically use gradient descent



Simple Perceptron

- The perceptron is a single layer feed-forward neural network.



Simple Perceptron

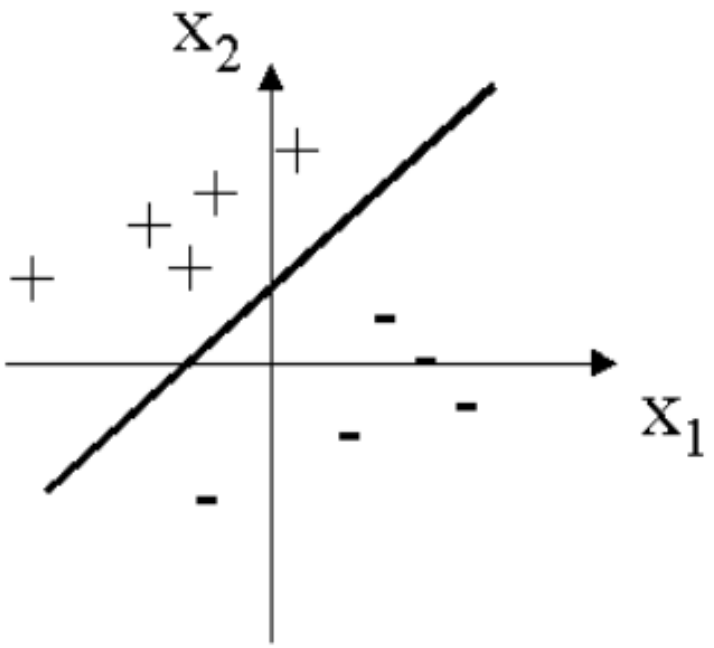
- Simplest output function

$$y = \operatorname{sgn}\left(\sum_{i=1}^{2} w_i x_i + \theta\right)$$

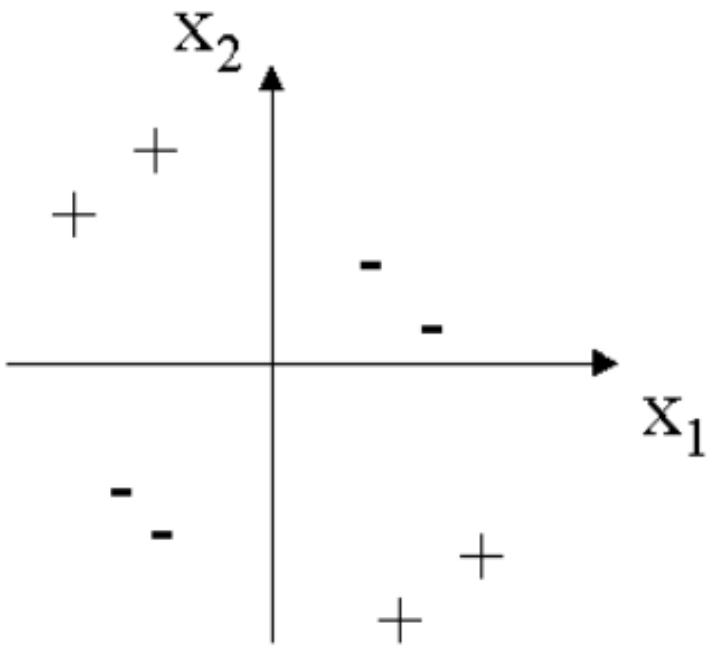
$$\operatorname{sgn}(s) = \begin{cases} 1 & \text{if } s > 0 \\ -1 & \text{otherwise.} \end{cases}$$

- Used to classify patterns said to be linearly separable

Linearly separable



Linearly Separable



Not Linearly Separable

$$w_1 x_1 + w_2 x_2 + \theta = 0$$

Perceptron Learning Algorithm

- We want to train the perceptron to classify inputs correctly
- Accomplished by adjusting the connecting weights and the bias
- Can only properly handle linearly separable sets

Perceptron Learning Algorithm

- We have a "training set" which is a set of input vectors used to train the perceptron.
- During training both w_i and θ *(bias)* are modified for convenience, let $w_0 = \theta$ and $x_0 = 1$
- *Let,* η*,* the *learning rate*, be a small positive number (small steps lessen the possibility of destroying correct classifications)
- Initialise w_i to some values

Perceptron Learning Algorithm

$$\text{Desired output} \qquad d(n) = \begin{cases} +1 & \text{if } x(n) \in \text{set } A \\ -1 & \text{if } x(n) \in \text{set } B \end{cases}$$

1. Select random sample from training set as input
2. If classification is correct, do nothing
3. If classification is incorrect, modify the weight vector *w* using

$$w_i = w_i + \eta d(n) x_i(n)$$

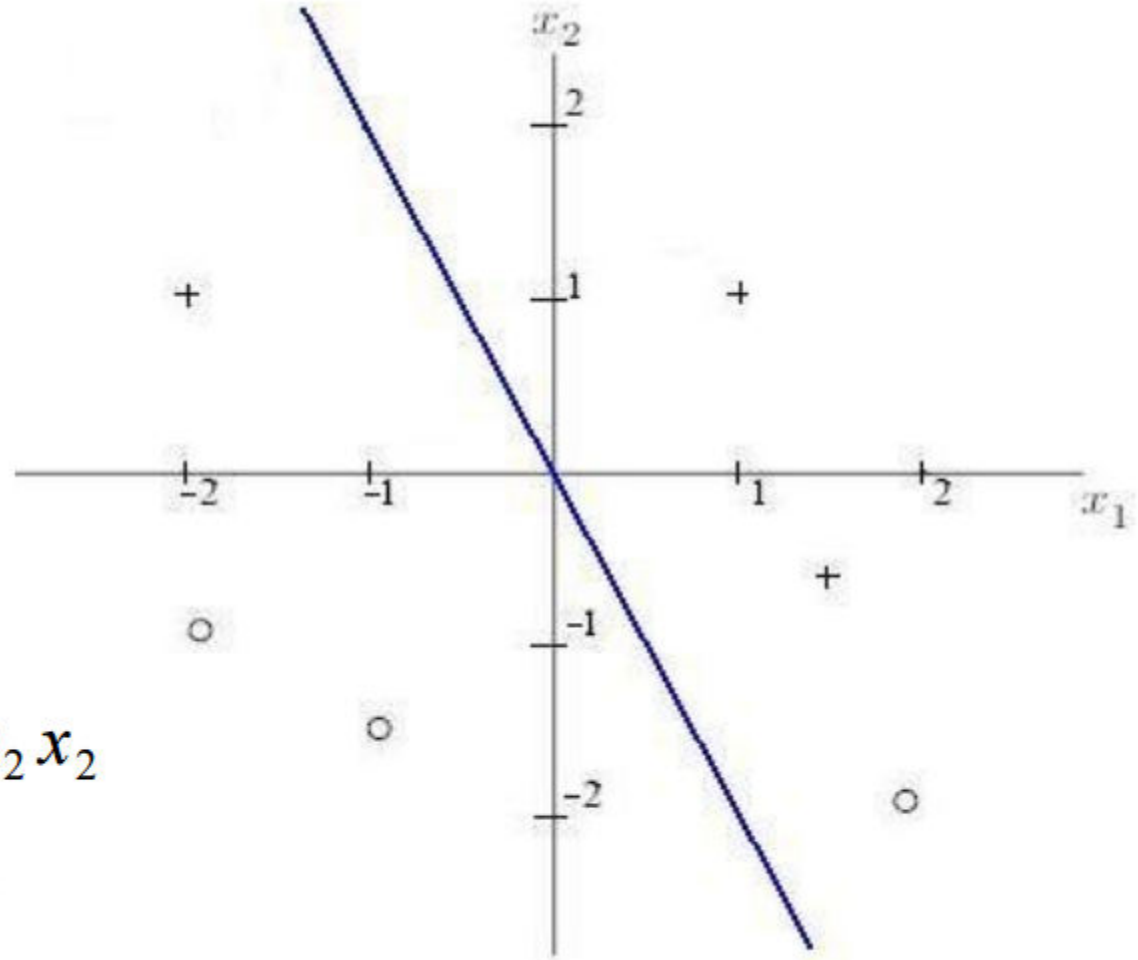
Repeat this procedure until the entire training set is classified correctly

Learning Example

Initial Values:

$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$



$$0 = w_0 + w_1 x_1 + w_2 x_2$$
$$= 0 + x_1 + 0.5x_2$$
$$\Rightarrow \quad x_2 = -2x_1$$

Learning Example

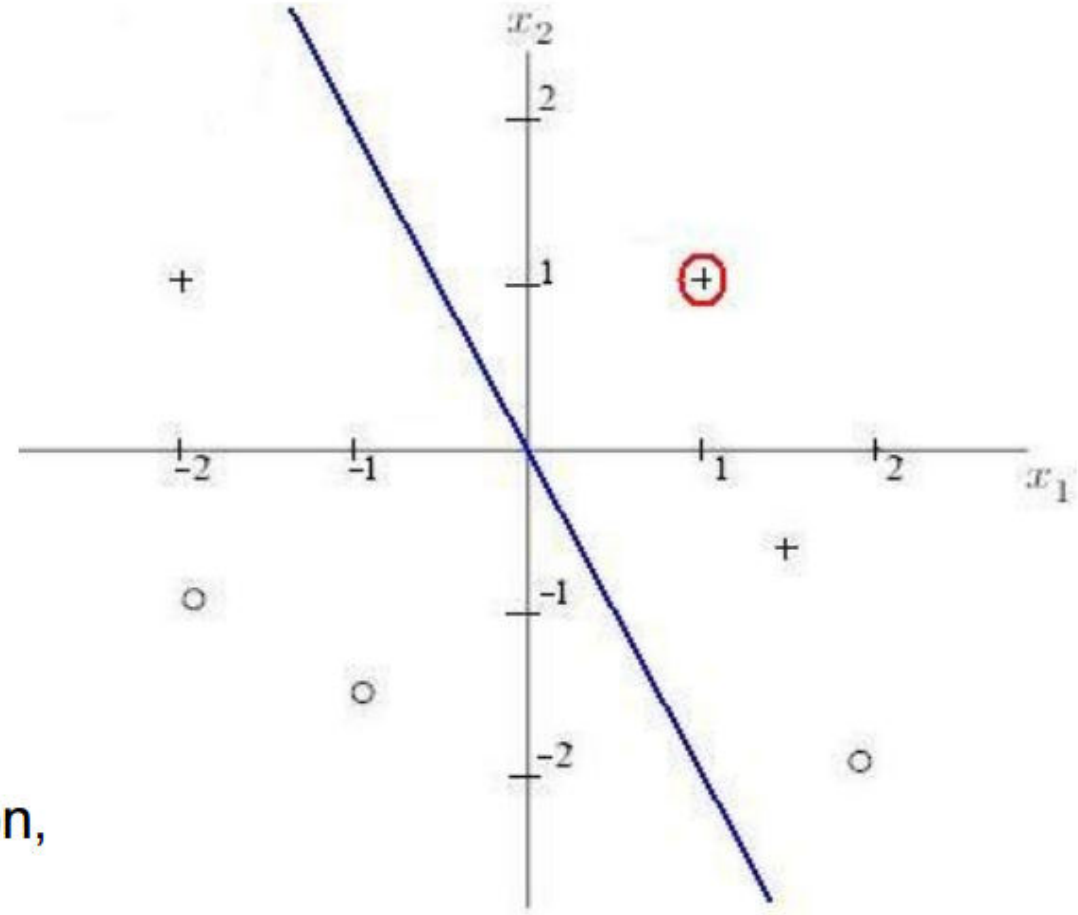
$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$x_1 = 1, x_2 = 1$

$w^T x > 0$

Correct classification,
no action



Learning Example

$\eta = 0.2$

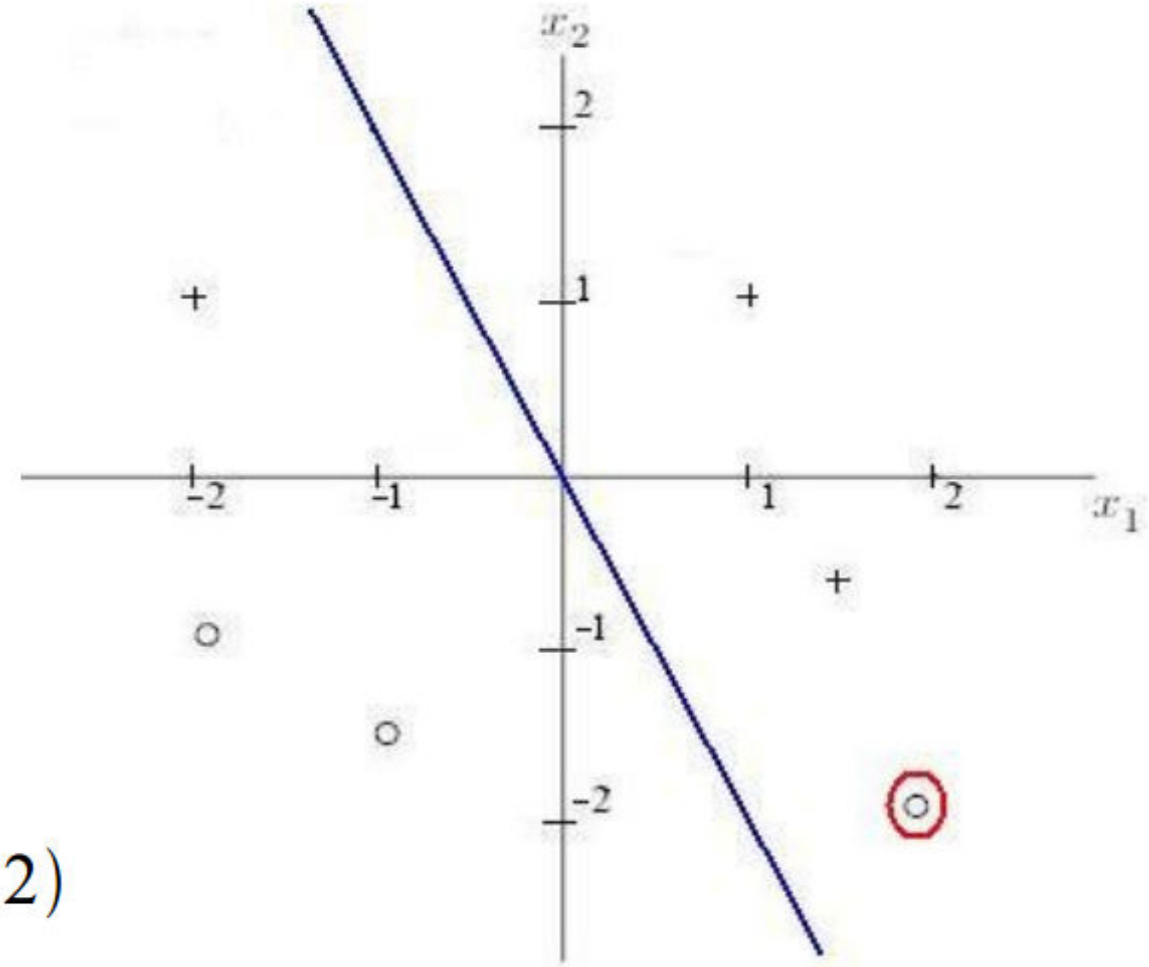
$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$x_1 = 2, x_2 = -2$

$$w_0 = w_0 - 0.2 * 1$$

$$w_1 = w_1 - 0.2 * 2$$

$$w_2 = w_2 - 0.2 * (-2)$$



Learning Example

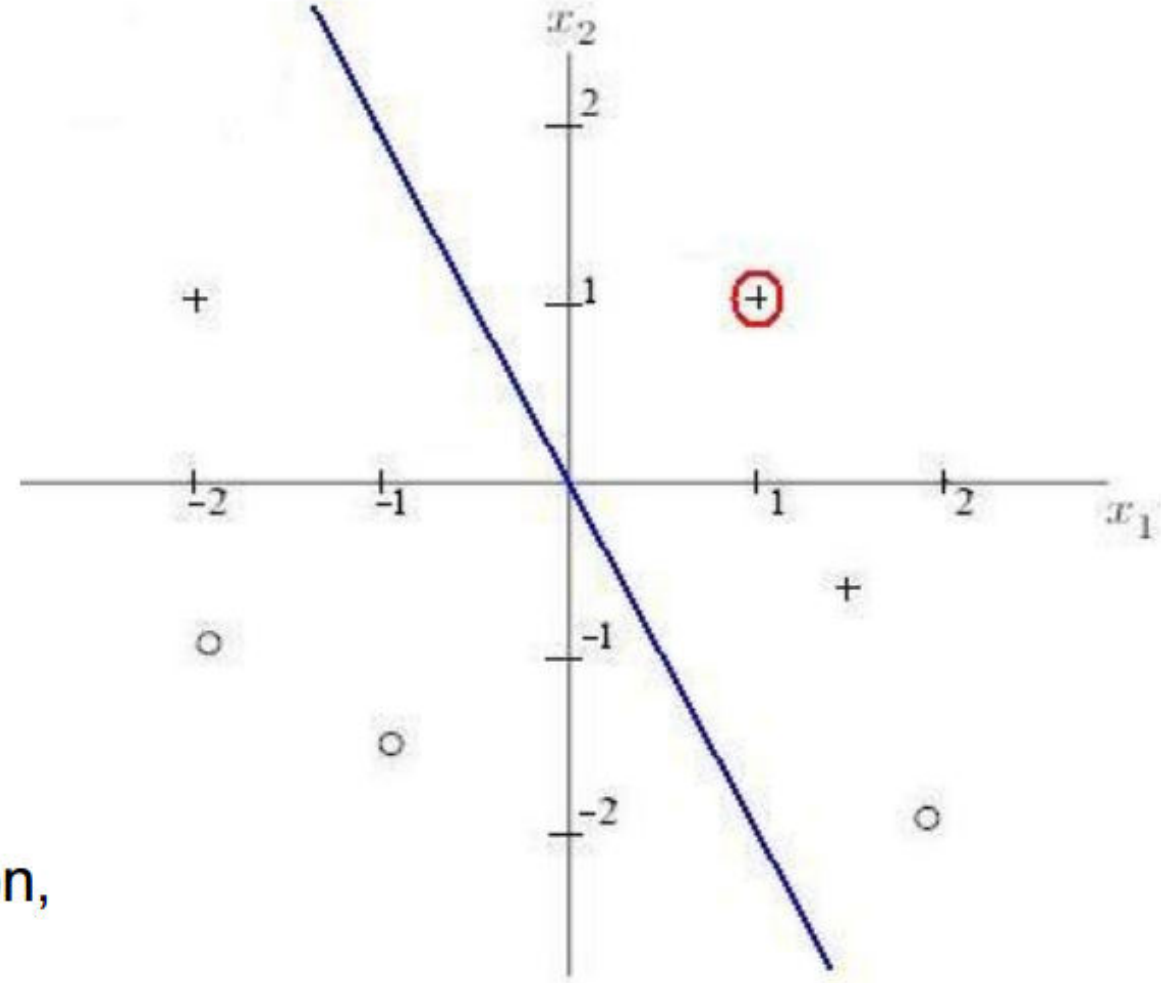
$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$x_1 = 1, x_2 = 1$

$w^T x > 0$

Correct classification,
no action



Learning Example

$\eta = 0.2$

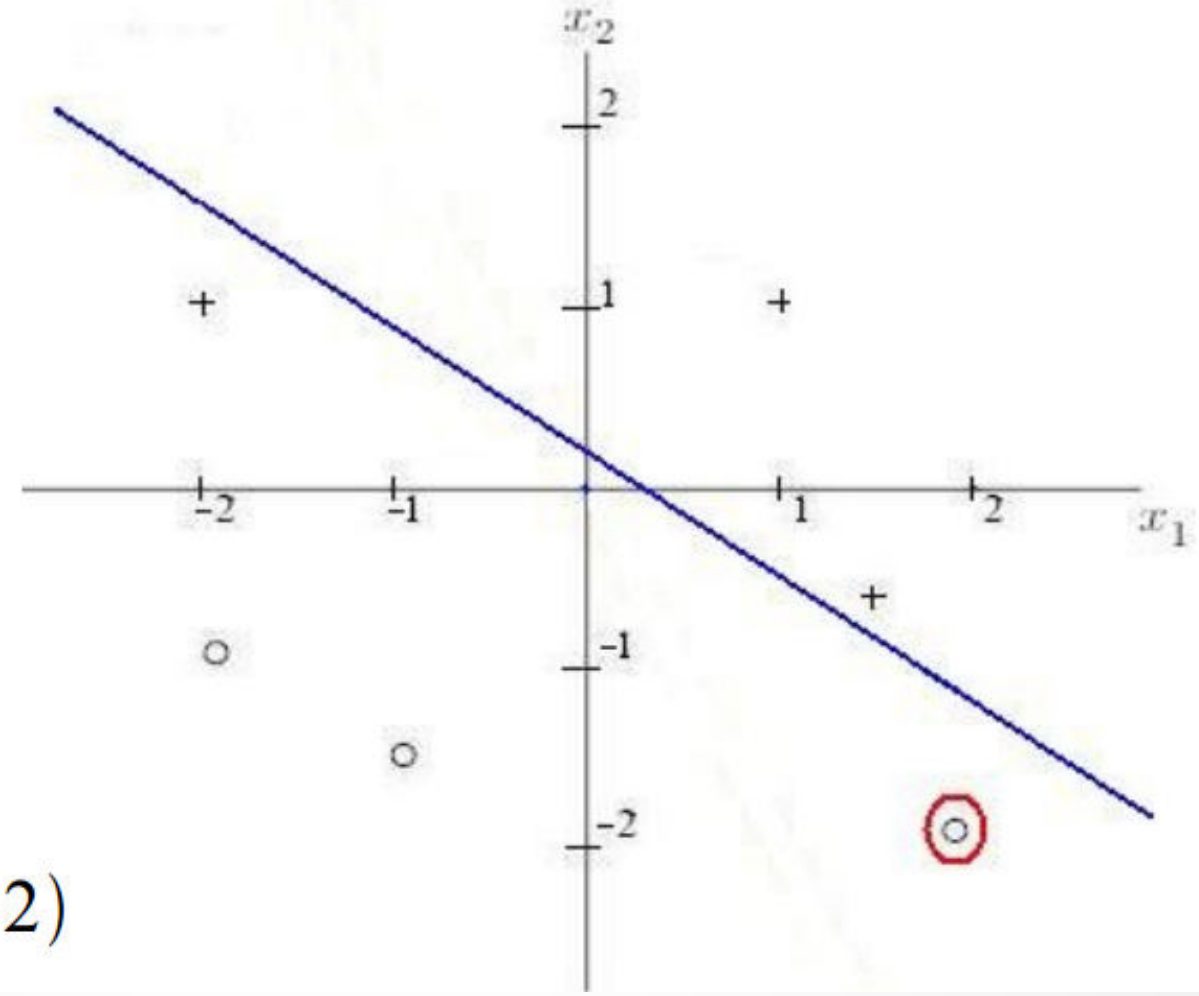
$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$x_1 = 2, x_2 = -2$

$$w_0 = w_0 - 0.2 * 1$$

$$w_1 = w_1 - 0.2 * 2$$

$$w_2 = w_2 - 0.2 * (-2)$$



Learning Example

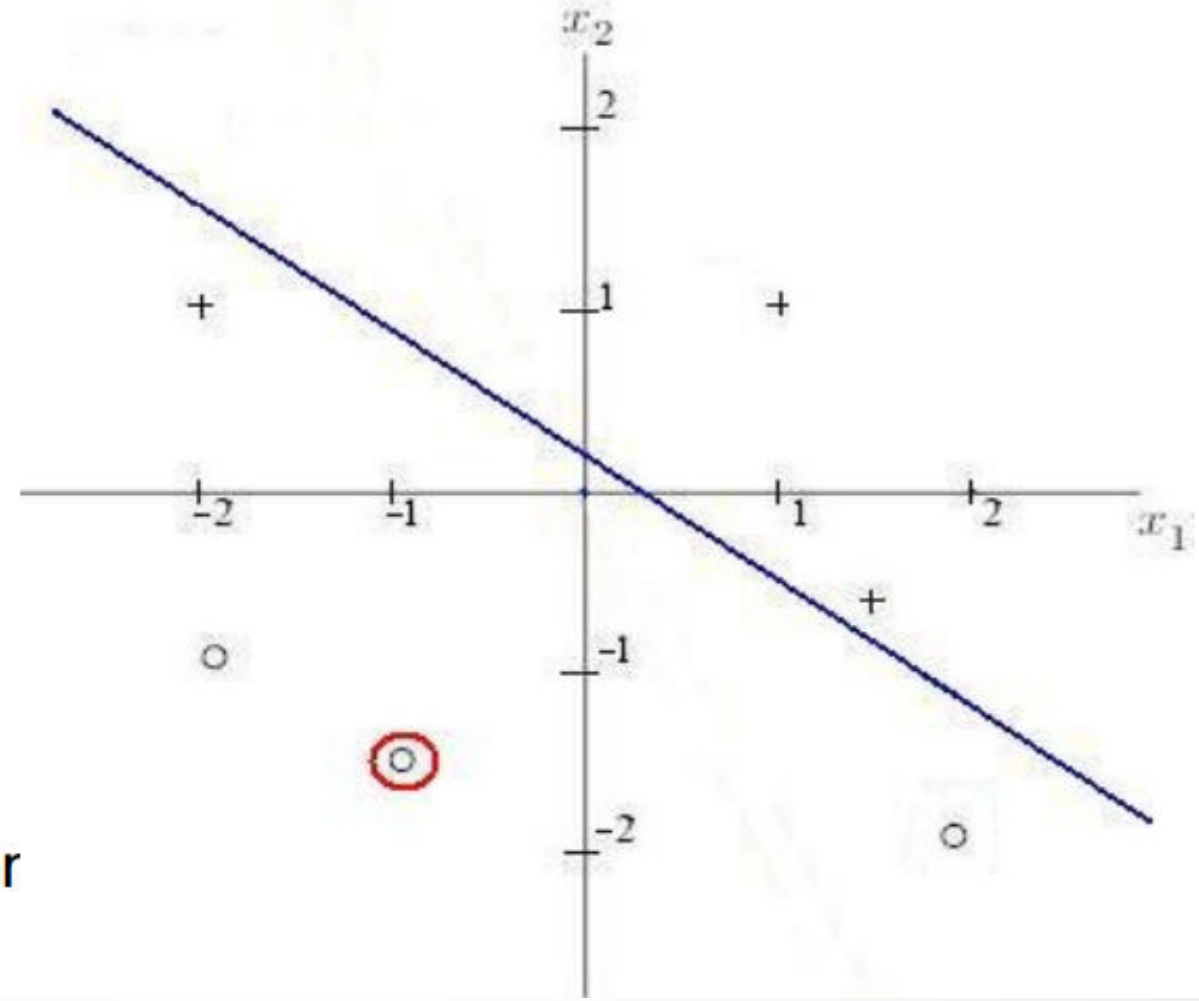
$\eta = 0.2$

$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$x_1 = -1, x_2 = -1.5$

$w^T x < 0$

Correct classificatior
no action



Learning Example

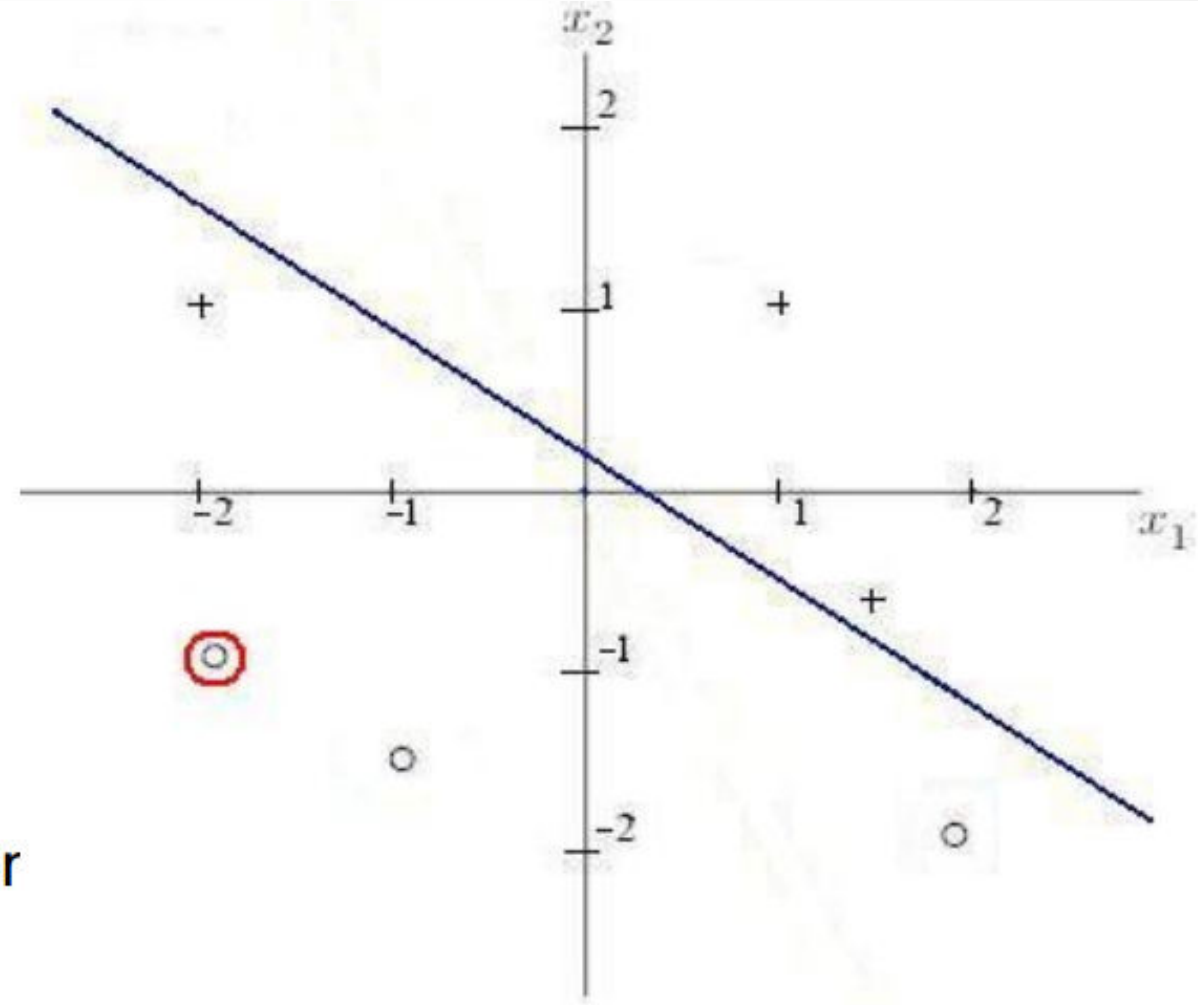
$\eta = 0.2$

$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$x_1 = -2, x_2 = -1$

$w^T x < 0$

Correct classificatior
no action



Learning Example

$\eta = 0.2$

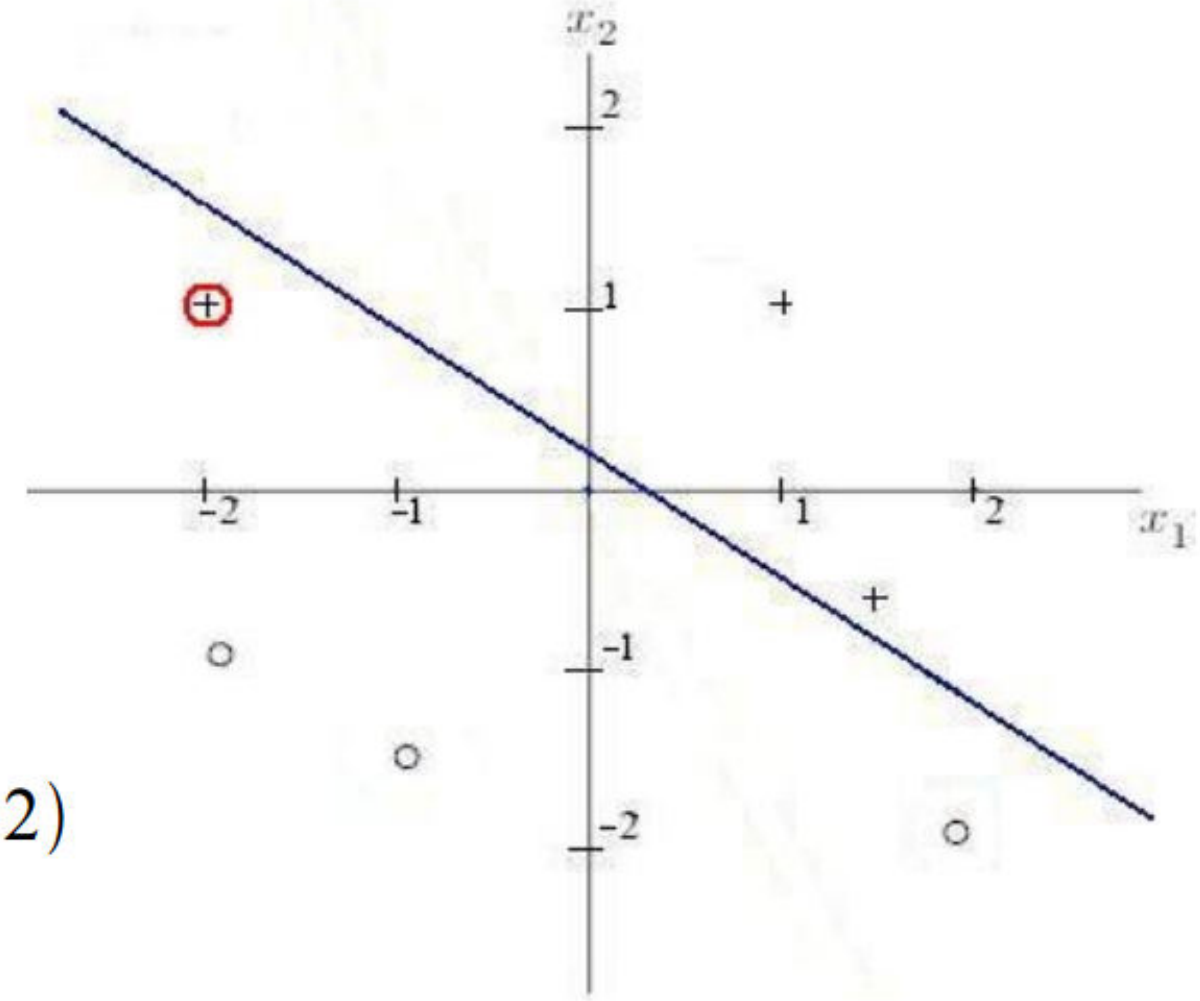
$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$x_1 = -2, x_2 = 1$

$$w_0 = w_0 + 0.2 * 1$$

$$w_1 = w_1 + 0.2 * (-2)$$

$$w_2 = w_2 + 0.2 * 1$$



Learning Example

$\eta = 0.2$

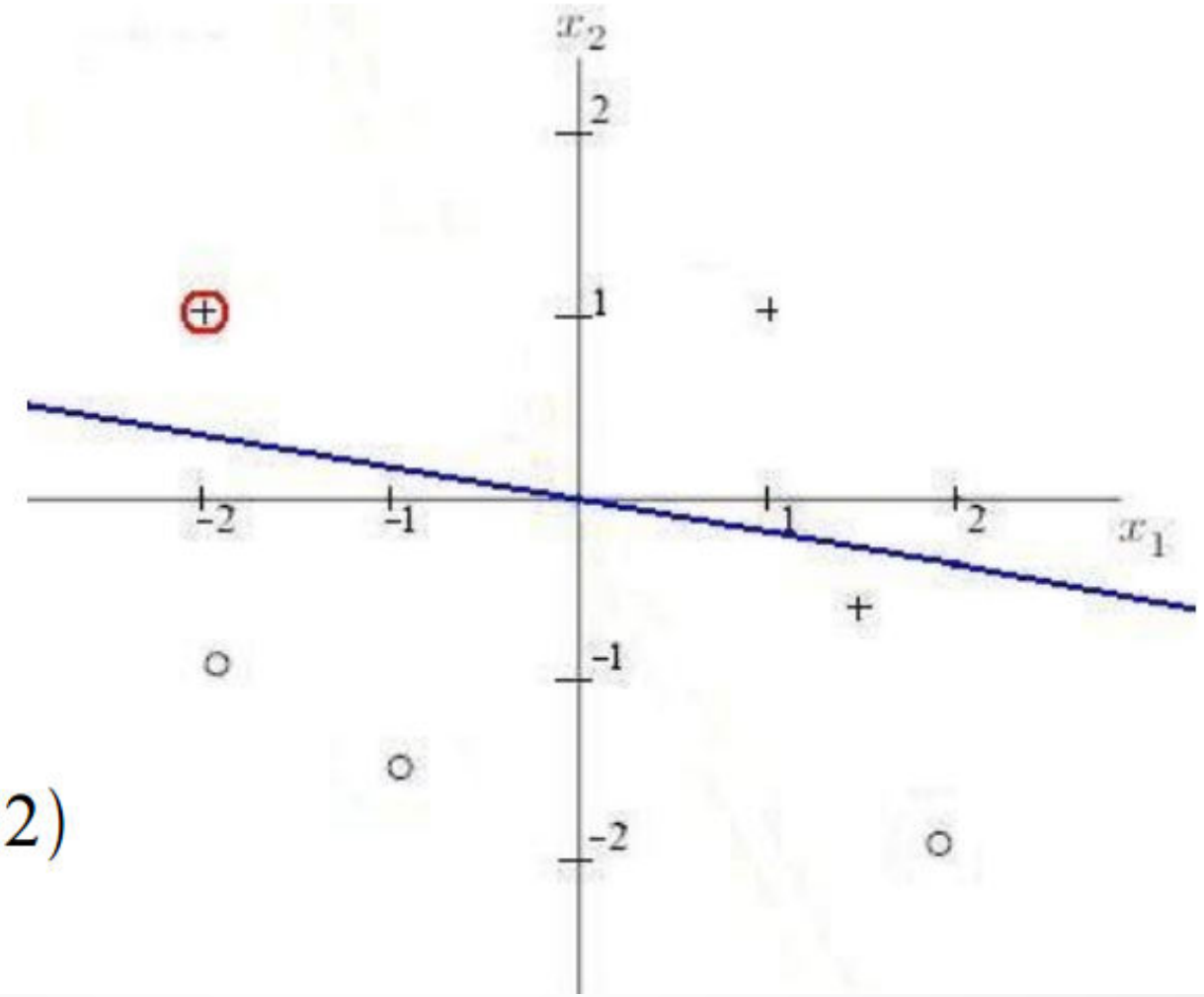
$$w = \begin{pmatrix} 0 \\ 0.2 \\ 1.1 \end{pmatrix}$$

$x_1 = -2, x_2 = 1$

$$w_0 = w_0 + 0.2 * 1$$

$$w_1 = w_1 + 0.2 * (-2)$$

$$w_2 = w_2 + 0.2 * 1$$



Learning Example

$\eta = 0.2$

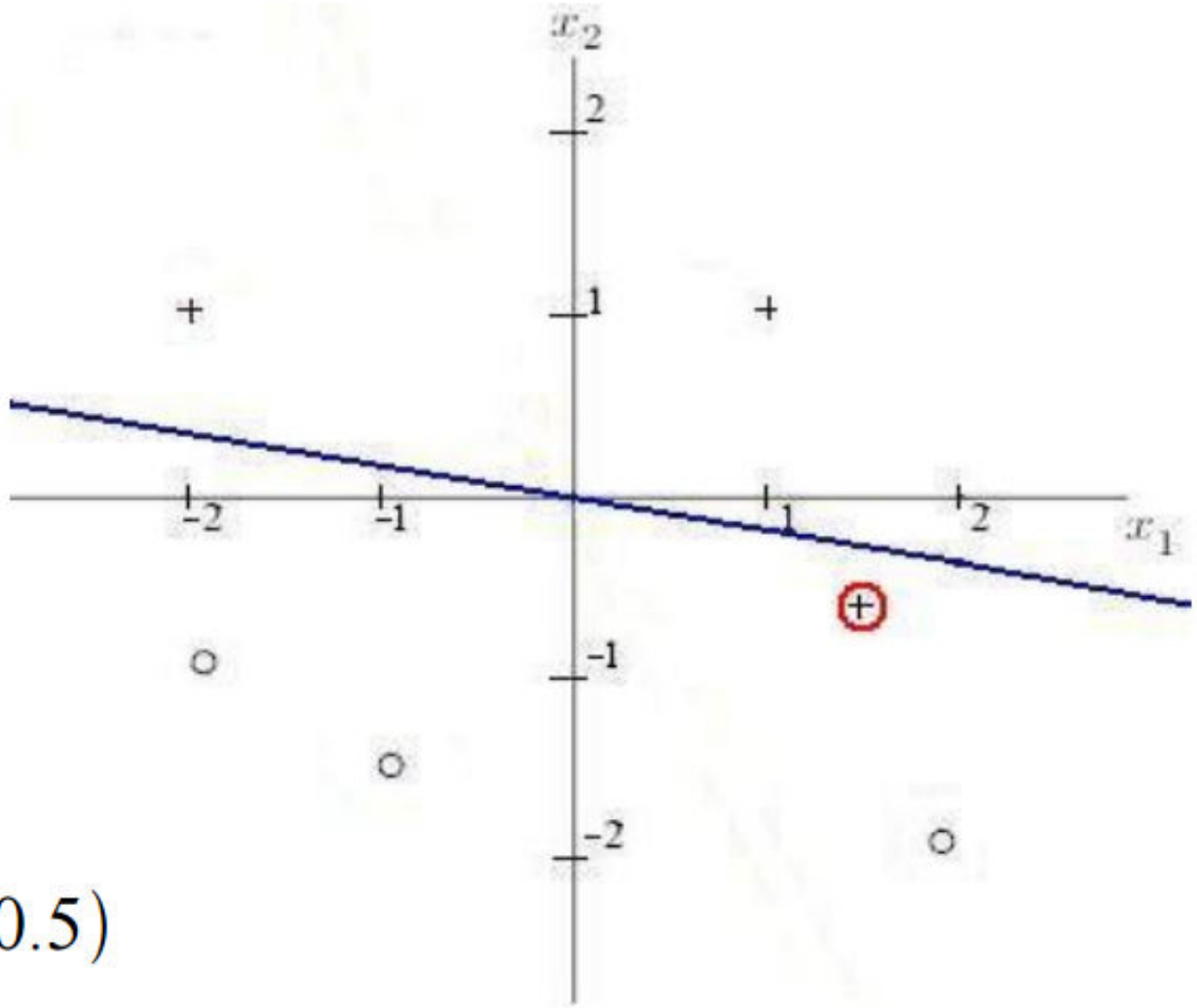
$$w = \begin{pmatrix} 0 \\ 0.2 \\ 1.1 \end{pmatrix}$$

$x_1 = 1.5, x_2 = -0.5$

$$w_0 = w_0 + 0.2 * 1$$

$$w_1 = w_1 + 0.2 * 1.5$$

$$w_2 = w_2 + 0.2 * (-0.5)$$



Learning Example

$\eta = 0.2$

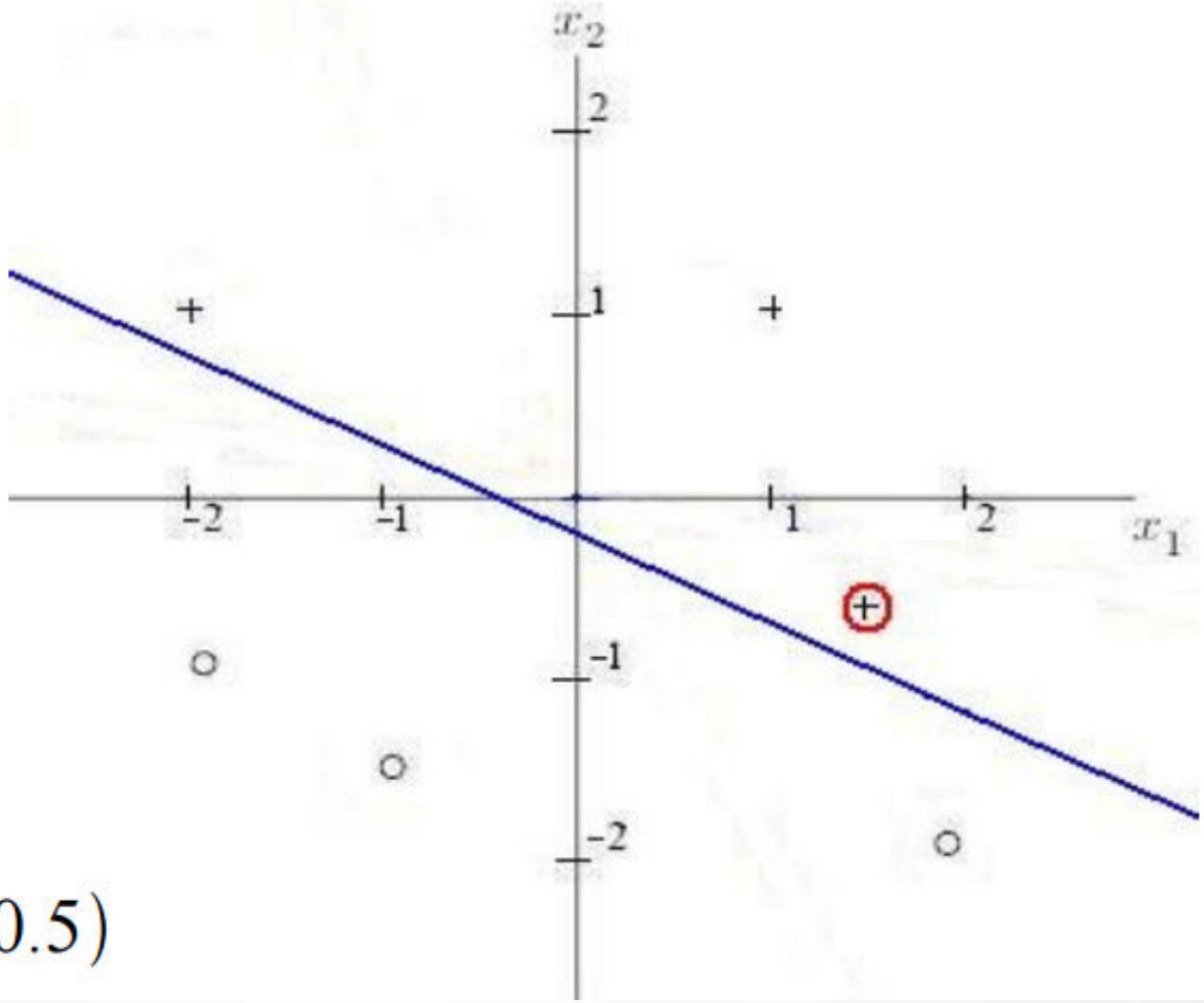
$$w = \begin{pmatrix} 0.2 \\ 0.5 \\ 1 \end{pmatrix}$$

$x_1 = 1.5, x_2 = -0.5$

$$w_0 = w_0 + 0.2 * 1$$

$$w_1 = w_1 + 0.2 * 1.5$$

$$w_2 = w_2 + 0.2 * (-0.5)$$



Implementation Single Node Neuron

EVOLVE
MACHINE LEARNERS

```
w=[0,0]
b=[0]

def create_perceptron():
    w[0] = random.random()
    w[1] = random.random()
    b[0] = random.random()
```

Implementation Single Node Neuron

EVOLVE
MACHINE LEARNERS

```
def activate(x):
    sum= (w[0] * x[0]) + (w[1] * x[1]) + b[0]
    if sum > 0 :
        return 1
    else:
        return -1
```

Implementation Single Node Neuron

EVOLVE
MACHINE LEARNERS

```
def compute_error(target,predicted):
    error = target - predicted
    return error
```

Implementation Single Node Neuron

EVOLVE
MACHINE LEARNERS

```
def learn(error, learning_rate, x):
    w[0] = w[0] + learning_rate*error*x[0]
    w[1] = w[1] + learning_rate*error*x[1]
    b[0] = b[0] + learning_rate*error*1
```

Implementation Single Node Neuron

EVOLVE
MACHINE LEARNERS

```
def one_step_learn(x_sample, y_sample, learning_rate):
    predicted = activate(x_sample)
    error = compute_error(y_sample, predicted)
    #print error
    learn(error, learning_rate, x_sample)
```

Implementation Single Node Neuron

EVOLVE
MACHINE LEARNERS

```
def train(x_dataset, y_dataset, learning_rate, n_iteration):
    for i in range (n_iteration):
        rn = random.randint(0, len(x_dataset)-1)
        x_sample = x_dataset[rn]
        y_sample = y_dataset[rn]
        one_step_learn(x_sample, y_sample, learning_rate)
#         if i %100:
#             print ('w old', w)
#             print ('b old', b)
```

Implementation Single Node Neuron

EVOLVE
MACHINE LEARNERS

```
if __name__ == "__main__":
    x_dataset = [[0,0],
                 [0,1],
                 [1,0],
                 [1,1]]

    y_dataset = [-1,1,1,1]
    #y_dataset = [-1,-1,-1,1]
    print('activation before learning')

    for x in x_dataset:
        print(activate(x))

    learning_rate = 0.01
    create_perceptron()
    train(x_dataset, y_dataset, learning_rate, 500)

    print('activation after learning')
    for x in x_dataset:
        print(activate(x))
```

Perceptron with numpy

Input Data

```
X = np.array([
    [-2, 4],
    [4, 1],
    [1, 6],
    [2, 4],
    [6, 2]
])
```

Include Bias into Input Data

```
X = np.array([
    [-2, 4, 1],
    [4, 1, 1],
    [1, 6, 1],
    [2, 4, 1],
    [6, 2, 1]
])
```

Perceptron with numpy

EVOLVE
MACHINE LEARNERS

```
import numpy as np
from matplotlib import pyplot as plt
```

Input Data

```
X = np.array([
    [-2, 4],
    [4, 1],
    [1, 6],
    [2, 4],
    [6, 2]
])
```

Include Bias into Input Data

```
X = np.array([
    [-2, 4, 1],
    [4, 1, 1],
    [1, 6, 1],
    [2, 4, 1],
    [6, 2, 1]
])
```

Output Data

```
y = np.array([-1,-1,1,1,1])
```

Perceptron with numpy

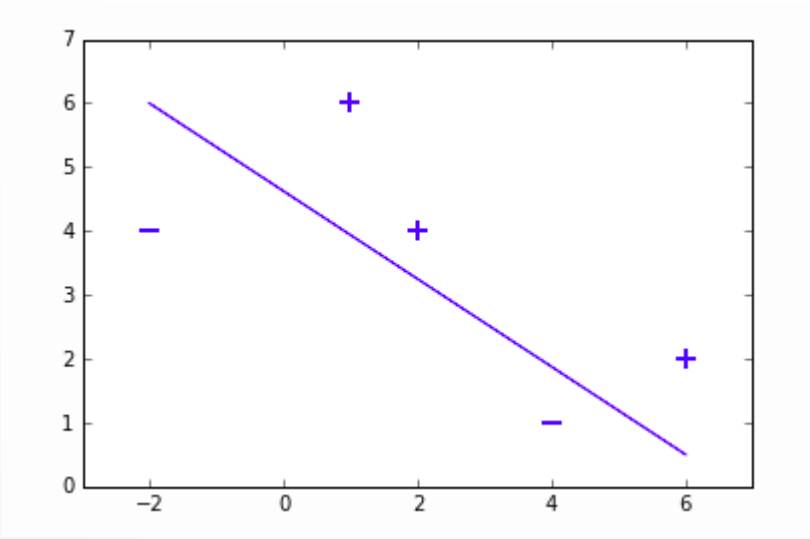

```
def plot(X):
    for d, sample in enumerate(X):
        # Plot the negative samples
        if d < 2:
            plt.scatter(sample[0], sample[1], s=120, marker='_', linewidths=2)
        # Plot the positive samples
        else:
            plt.scatter(sample[0], sample[1], s=120, marker='+', linewidths=2)

    # Print a possible hyperplane, that is seperating the two classes.
    plt.plot([-2,6],[6,0.5])
    plt.show()
```

Perceptron with numpy

```
def plot(X):
    for d, sample in enumerate(X):
        # Plot the negative samples
        if d < 2:
            plt.scatter(sample[0], sample[1], s=120, marker='_', linewidths=2)
        # Plot the positive samples
        else:
            plt.scatter(sample[0], sample[1], s=120, marker='+', linewidths=2)

    # Print a possible hyperplane, that is seperating the two classes.
    plt.plot([-2,6],[6,0.5])
    plt.show()
```



Perceptron with numpy

EVOLVE
MACHINE LEARNERS

```
def perceptron_sgd(X, Y):
    w = np.zeros(len(X[0]))
    eta = 1
    epochs = 10

    for epoch in range(epochs):
        for i, x in enumerate(X):
            if (np.dot(X[i], w)*Y[i]) <= 0:
                w = w + eta*X[i]*Y[i]
    return w
```

Perceptron with numpy


```
def perceptron_sgd_plot(X, Y):

    w = np.zeros(len(X[0]))
    eta = 1
    n = 30
    errors = []

    for t in range(n):
        total_error = 0
        for i, x in enumerate(X):
            if (np.dot(X[i], w) * Y[i]) <= 0:
                total_error += (np.dot(X[i], w) * Y[i])
                w = w + eta * X[i] * Y[i]
        errors.append(total_error * -1)

    plt.plot(errors)
    plt.xlabel('Epoch')
    plt.ylabel('Total Loss')

    return w
```