Neural Networks

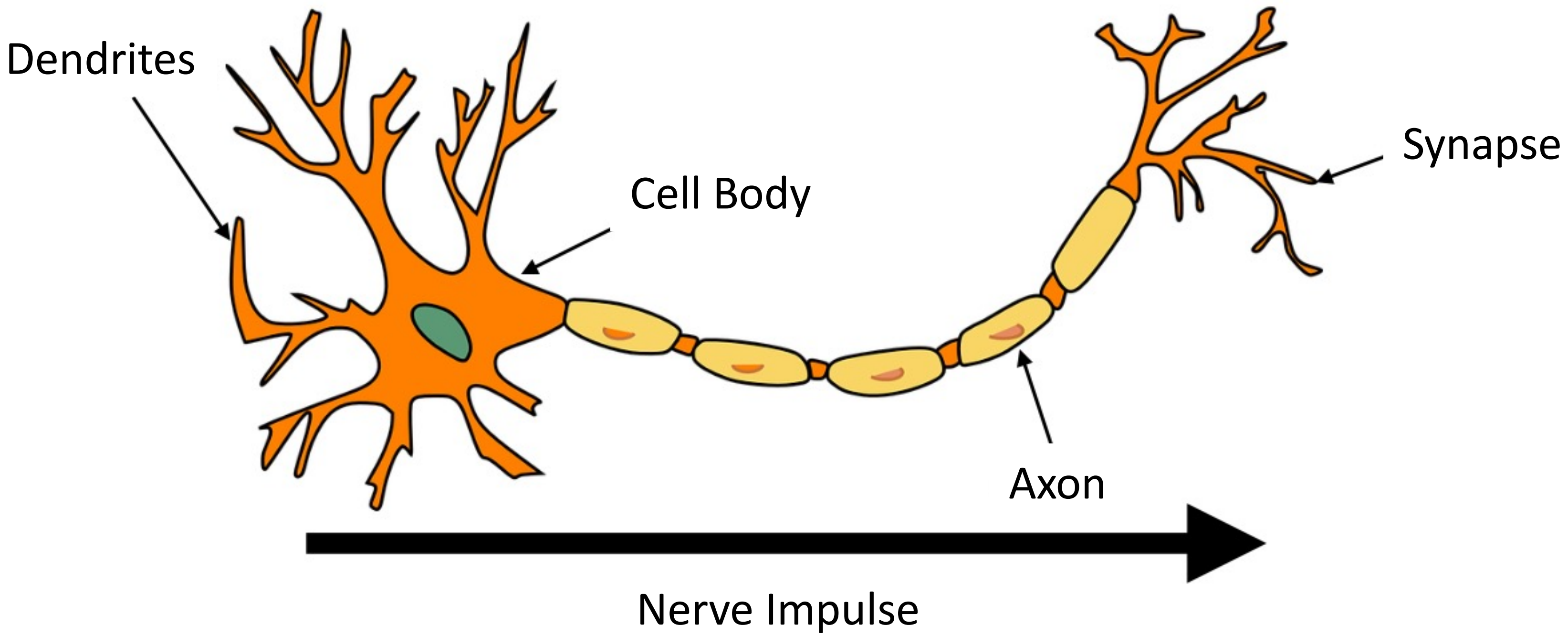
Deep Reinforcement Learning

Alberto Sardinha
sardinha@inf.puc-rio.br

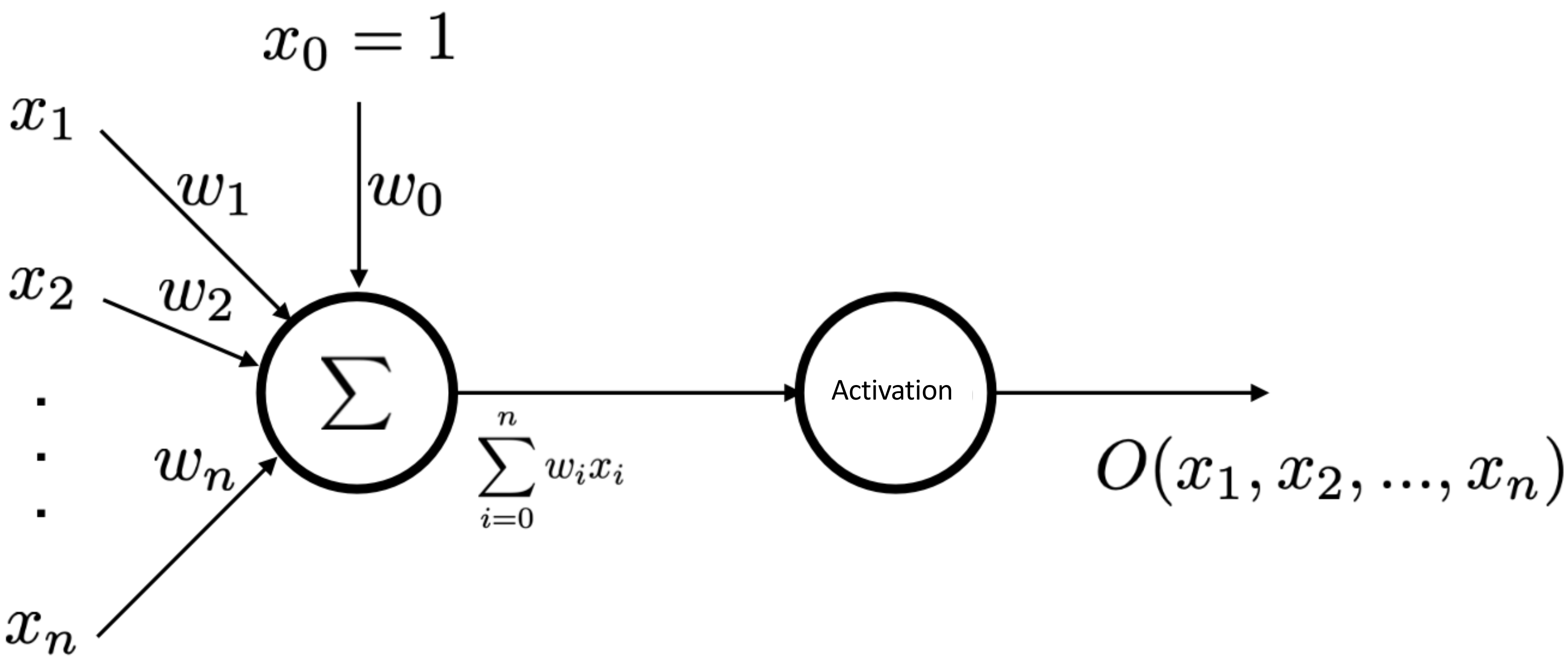
Outline

- **Motivation**
- Perceptron
- Multilayer Perceptron

Biological Neuron



Artificial Neuron

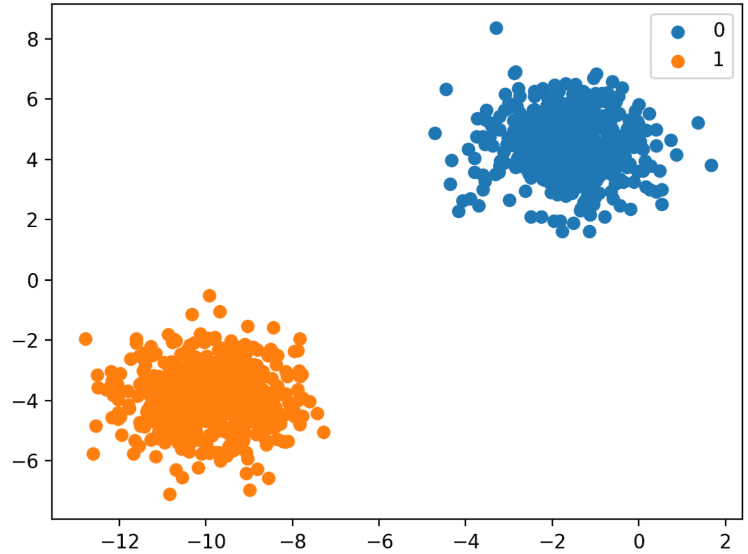


Outline

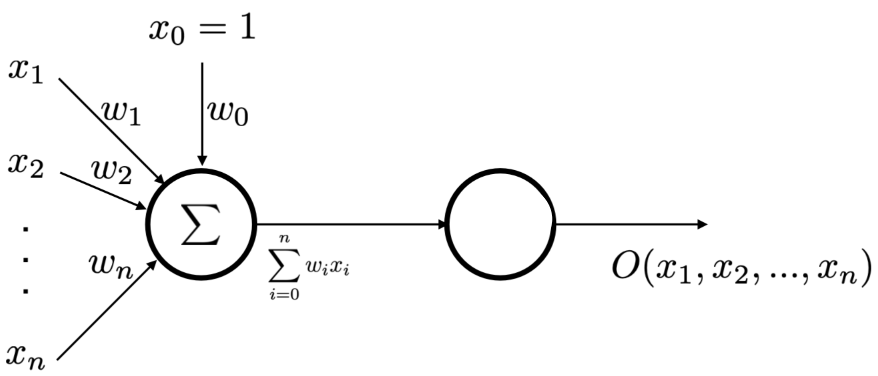
- Motivation
- **Perceptron**
- Multilayer Perceptron

Perceptron

- One of the first Artificial Neural Networks
- Created by Frank Rosenblatt in 1958
- Composed of a single artificial neuron
- Used for binary classification



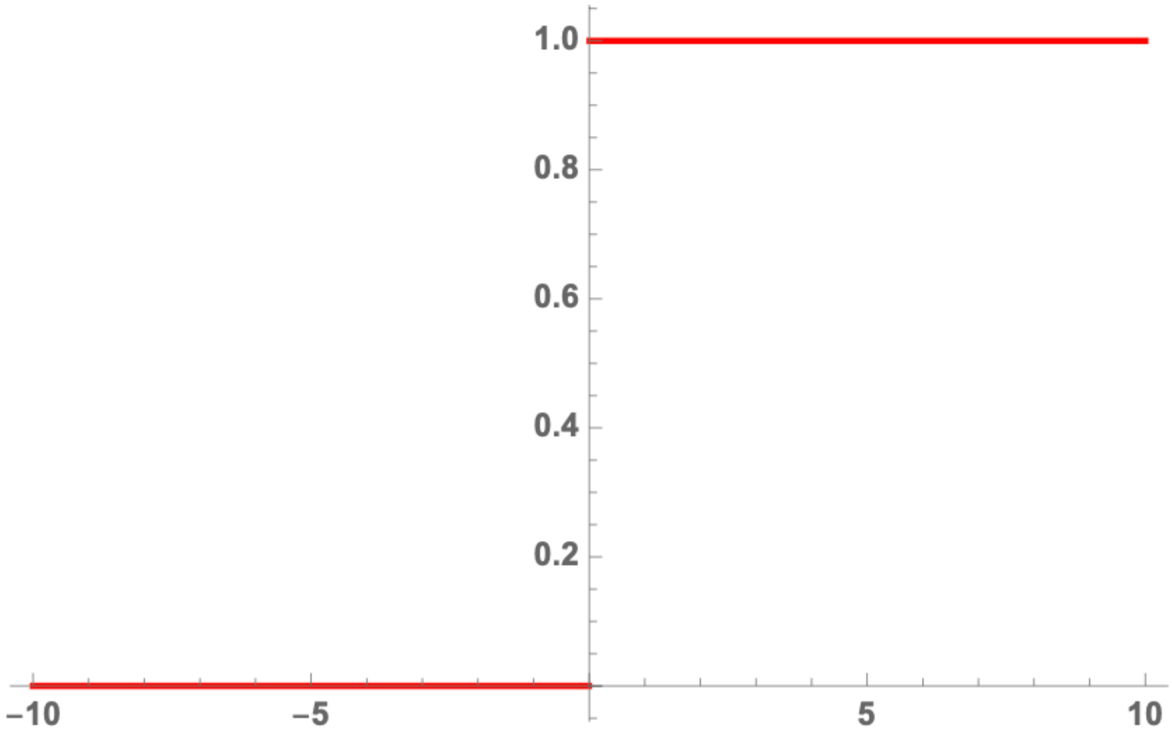
Perceptron



- $\boldsymbol{x_1, x_2, \ldots, x_n}$ **are real numbers** (e.g., pixels - 0 (black) to 255 (white))
- $\boldsymbol{x_i}$ is multiplied by $\boldsymbol{w_i}$
- **Bias**: a fixed input ($\boldsymbol{x_0} = \mathbf{1}$)
- **Weighted sum:** $\sum_{i=0}^{n} w_i x_i$ or $w^T x$
- **Activation function:**

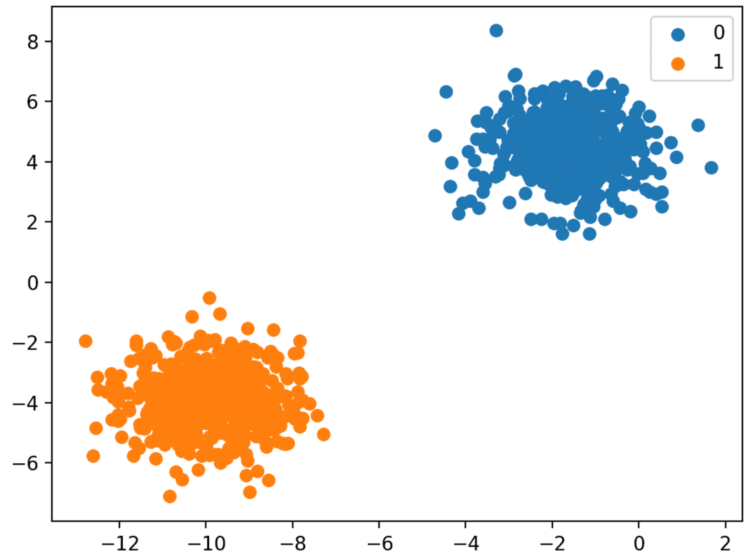
$$O(x) = \begin{cases} 1 & w^T x > 0 \\ 0 & w^T x \le 0 \end{cases}$$

Perceptron



Perceptron

- **Learning process**
  - **Training set with N examples**
  - Each **training example** has:
    - $x^d = < x_1^d, x_2^d, \ldots, x_n^d >$
    - $y^d = 0$ or 1 (class)
  - Find the values of $\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_n}$ that can **correctly classify** each training example



Perceptron

- How do we find the values of $\boldsymbol{w_1}, \boldsymbol{w_2}, \dots, \boldsymbol{w_n}$?
  - Use the following **learning rule:**

$$w_i = w_i + \eta\underbrace{(y^d - \hat{y}^d)}_{\text{Prediction error}}x_i^d$$

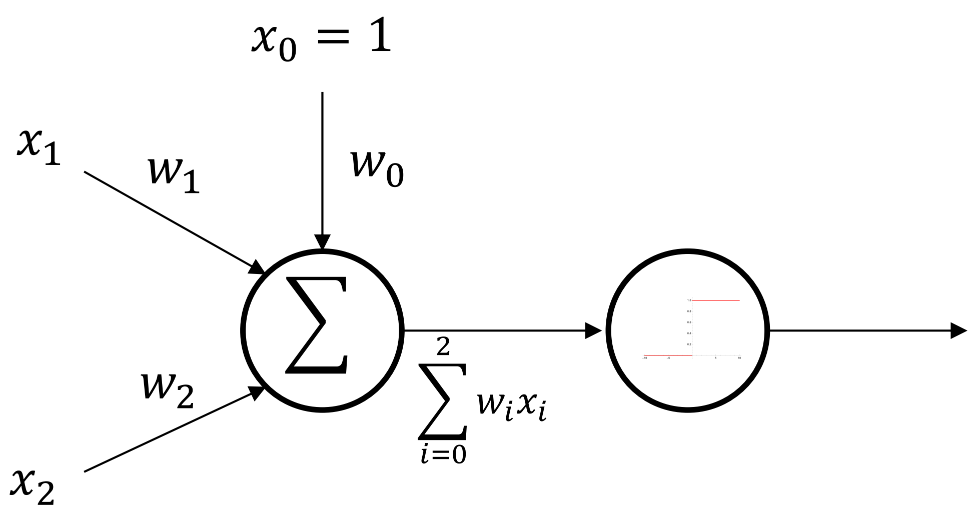
Perceptron

- **Example of the training set**

Training examples	x_1^d	x_2^d	y^d
d = 1	0	0	0
d = 2	0	1	0
d = 3	1	0	0
d = 4	1	1	1

Perceptron

- Let's use this perceptron



- Let's initialize the weights: $w_0 = w_1 = w_2 = 0$
- Let's use $\eta = 0.01$

Perceptron

- Applying the Perceptron learning rule:

  - **First training example**: $x_1^1 = 0$, $x_2^1 = 0$, and $y^1 = 0$

$$\hat{y}^1 = O\left(w_0 x_0 + \sum_{i=1}^{2} w_i x_i^1\right) = O(0\times1 + 0\times0 + 0\times0) = 0$$

$$w_0 = w_0 + \eta(y^1 - \hat{y}^1)x_0 = 0 + 0.01(0 - 0)1 = 0$$
$$w_1 = w_1 + \eta(y^1 - \hat{y}^1)x_1^1 = 0 + 0.01(0 - 0)0 = 0$$
$$w_2 = w_2 + \eta(y^1 - \hat{y}^1)x_2^1 = 0 + 0.01(0 - 0)0 = 0$$

Perceptron

- Applying the Perceptron learning rule:

  - **Second training example**: $x_1^2 = 0$, $x_2^2 = 1$, and $y^2 = 0$

$$\hat{y}^2 = O\left(w_0 x_0 + \sum_{i=1}^{2} w_i x_i^2\right) = O(0\times1 + 0\times0 + 0\times1) = 0$$

$$w_0 = w_0 + \eta(y^2 - \hat{y}^2)x_0 = 0 + 0.01(0 - 0)1 = 0$$
$$w_1 = w_1 + \eta(y^2 - \hat{y}^2)x_1^2 = 0 + 0.01(0 - 0)0 = 0$$
$$w_2 = w_2 + \eta(y^2 - \hat{y}^2)x_2^2 = 0 + 0.01(0 - 0)1 = 0$$

Perceptron

- Applying the Perceptron learning rule:

  - **Third training example**: $x_1^3 = 1$, $x_2^3 = 0$, and $y^3 = 0$

$$\hat{y}^3 = O\left(w_0 x_0 + \sum_{i=1}^{2} w_i x_i^3\right) = O(0\times1 + 0\times1 + 0\times0) = 0$$

$$w_0 = w_0 + \eta(y^3 - \hat{y}^3)x_0 = 0 + 0.01(0 - 0)1 = 0$$
$$w_1 = w_1 + \eta(y^3 - \hat{y}^3)x_1^3 = 0 + 0.01(0 - 0)1 = 0$$
$$w_2 = w_2 + \eta(y^3 - \hat{y}^3)x_2^3 = 0 + 0.01(0 - 0)0 = 0$$

Perceptron

- Applying the Perceptron learning rule:
  - **Fourth training example**: $x_1^4 = 1$, $x_2^4 = 1$, and $y^4 = 1$

$$\hat{y}^4 = O\left(w_0 x_0 + \sum_{i=1}^{2} w_i x_i^4\right) = O(0\times1 + 0\times1 + 0\times1) = 0$$

$$w_0 = w_0 + \eta(y^4 - \hat{y}^4)x_0 = 0 + 0.01(1 - 0)1 = 0.01$$
$$w_1 = w_1 + \eta(y^4 - \hat{y}^4)x_1^4 = 0 + 0.01(1 - 0)1 = 0.01$$
$$w_2 = w_2 + \eta(y^4 - \hat{y}^4)x_2^4 = 0 + 0.01(1 - 0)1 = 0.01$$

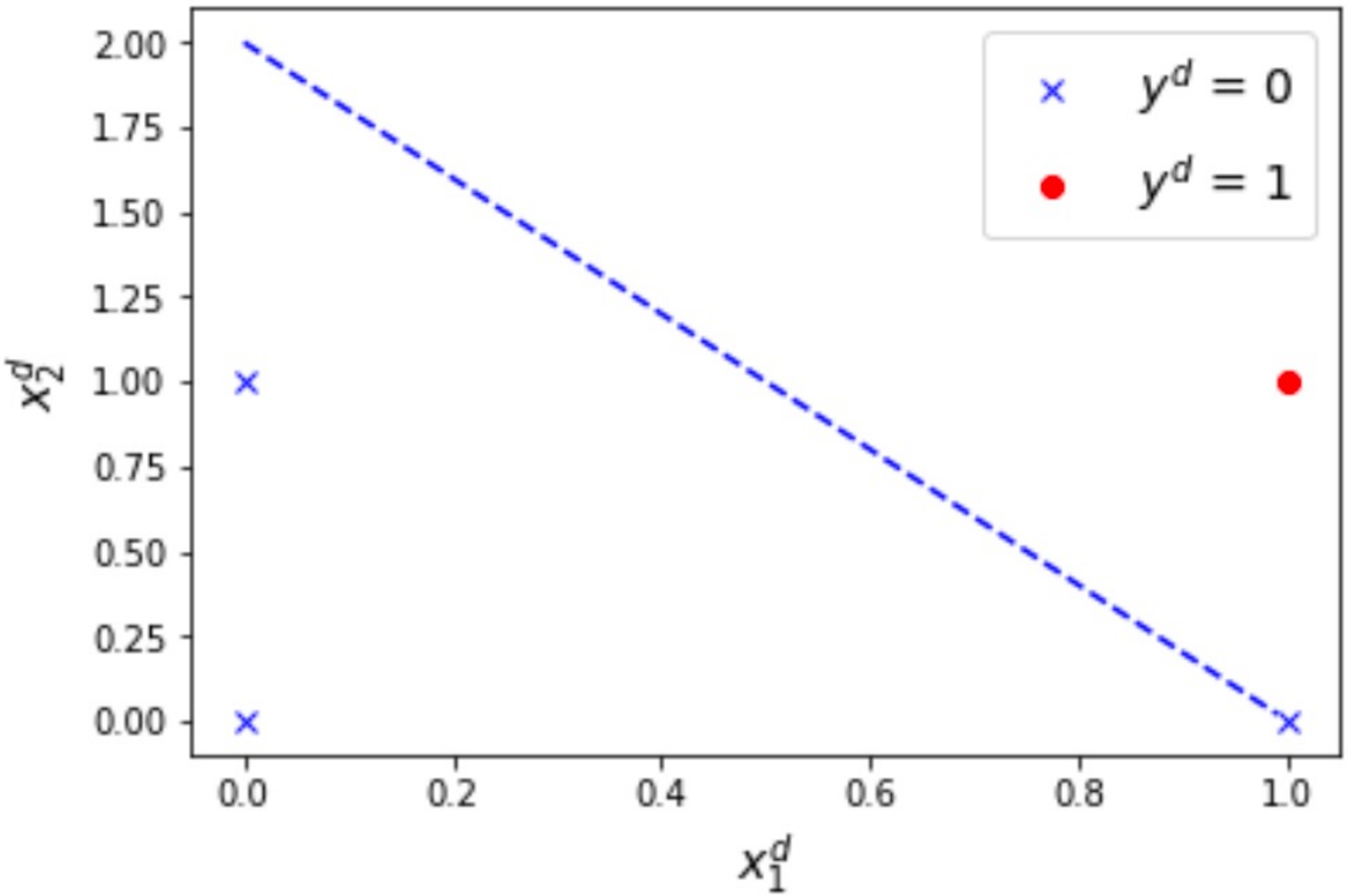
Perceptron

Continue the learning process until Perceptron can correctly classify all the training examples

Perceptron

- **Learned weight values:**
  - $w_0 = -0.02$
  - $w_1 = 0.02$
  - $w_2 = 0.01$
- **Decision boundary:**
  - $w_0x_0 + w_1x_1 + w_2x_2 = 0$
  - $-0.02 + 0.02x_1 + 0.01x_2 = 0$
  - $x_2 = -\frac{0.02x_1 - 0.02}{0.01}$

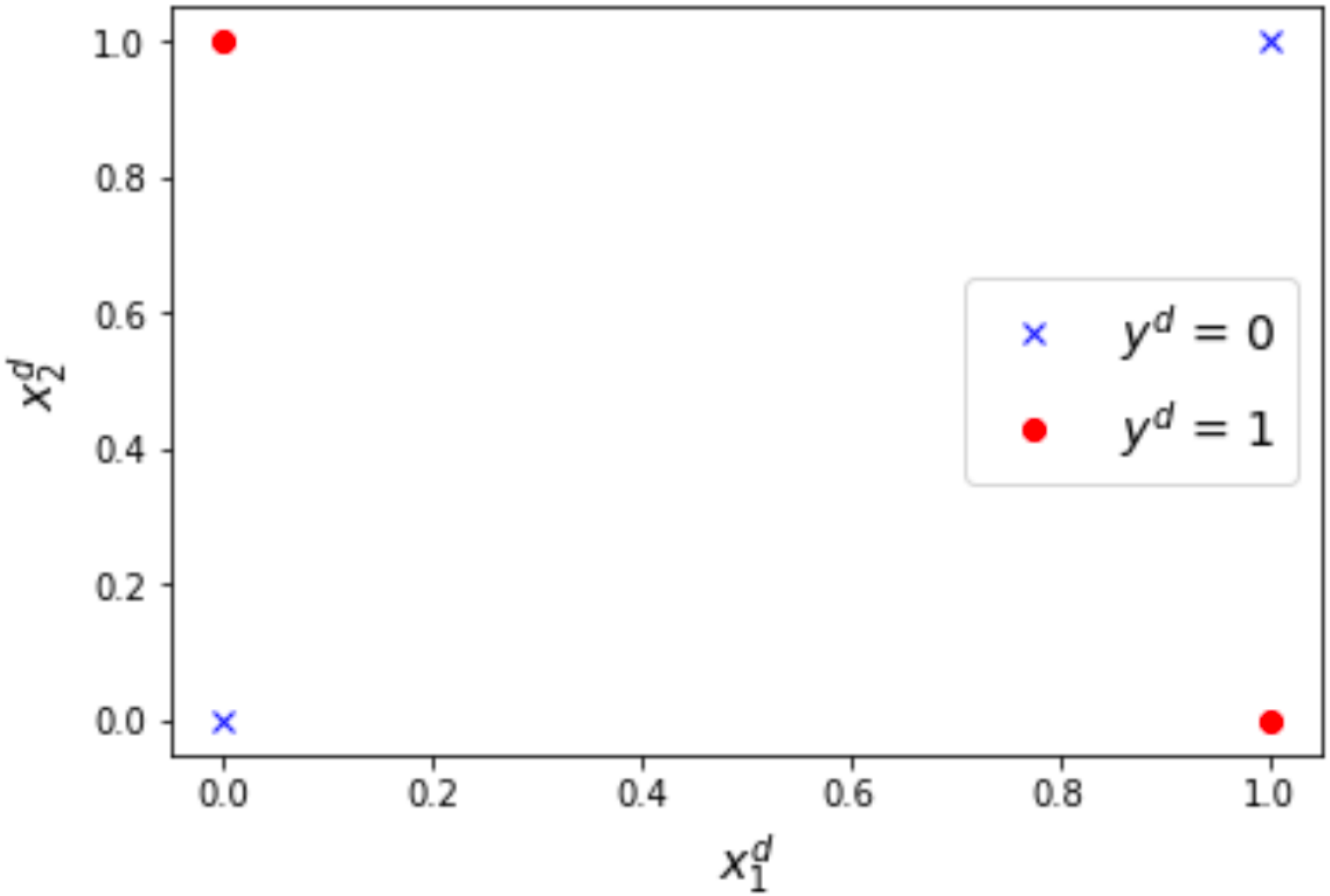
Perceptron



2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0
x_1^d
x_2^d
$y^d = 0$
$y^d = 1$

Perceptron

- Can Perceptron learn this function?



Perceptron

```python
import numpy as np
import matplotlib.pyplot as plt

# initialization
num_imputs = 2
weights = np.zeros(num_imputs + 1)
learning_rate = 0.01
max_iterations = 1000
```

```python
# Perceptron's output
def perceptron_output(example):
    weighted_sum = np.dot(example, np.transpose(weights[1:])) + weights[0]
    if weighted_sum > 0:
        return 1
    else:
        return 0
```

Perceptron

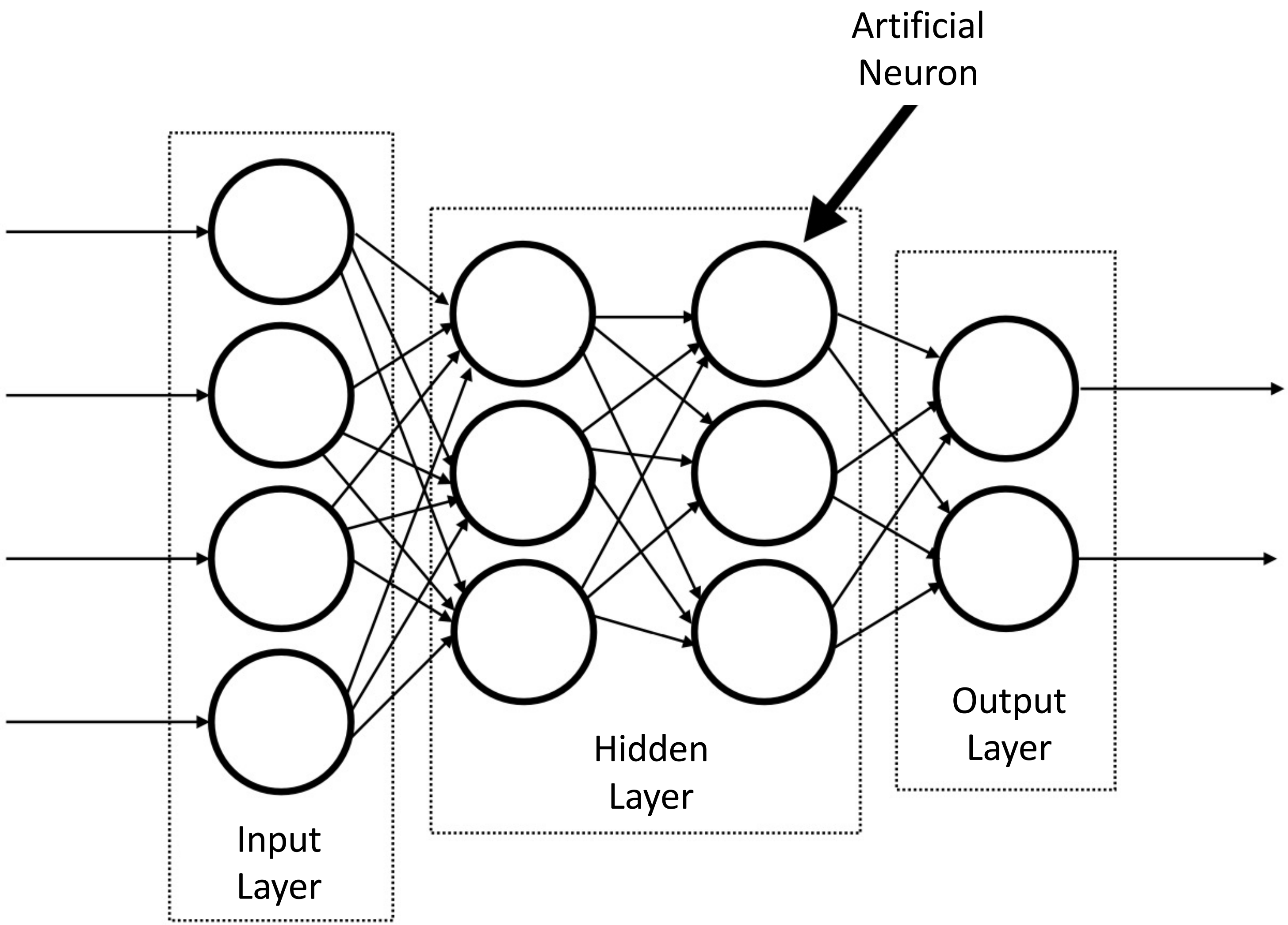
```python
# train the Perceptron
def train(training_examples, outputs):
    i=0
    error = 0
    while True:
        for example, output in zip(training_examples, outputs):
            predicted_output = perceptron_output(example)
            error += (output - predicted_output)**2
            weights[1:] += learning_rate * (output - predicted_output) * example
            weights[0] += learning_rate * (output - predicted_output)
            i += 1
            if i >= max_iterations:
                print("reached mximum number of iterations")
                error = 0
                break
        if error == 0:
            print("Number of iterations to train perceptron = ", i)
            print("Weights = ", weights)
            break
        error = 0
```

```python
#training examples
training_examples = np.array([[0,0],[0,1],[1,0],[1,1]])
outputs = np.array([[0],[0],[0],[1]])
train(training_examples, outputs)
```

Outline

- Motivation
- Perceptron
- **Multilayer Perceptron**

Multilayer Perceptron



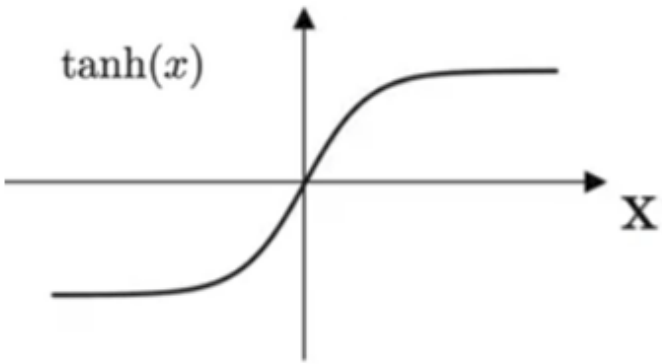
Multilayer Perceptron

- We can **learn complex and non-linear functions** with MLP
  - Classification problems
  - Regression problems
  - Probability distributions
  - Etc.
- An MLP is considered a "**universal approximator**"

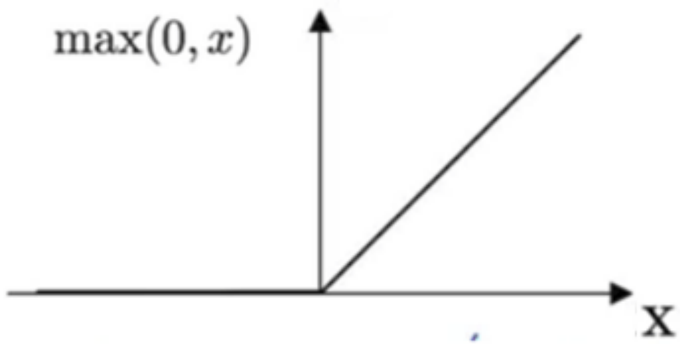
Multilayer Perceptron

- MLPs can be implemented with different activation functions

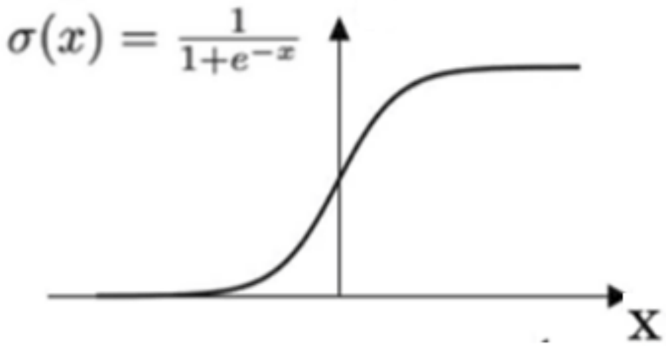
Hyper Tangent Function



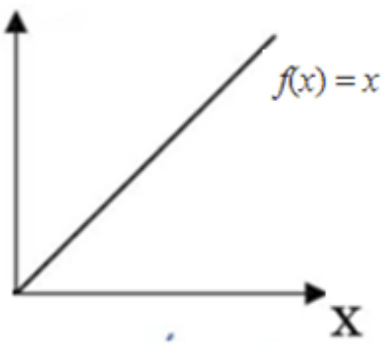
ReLU Function



Sigmoid Function



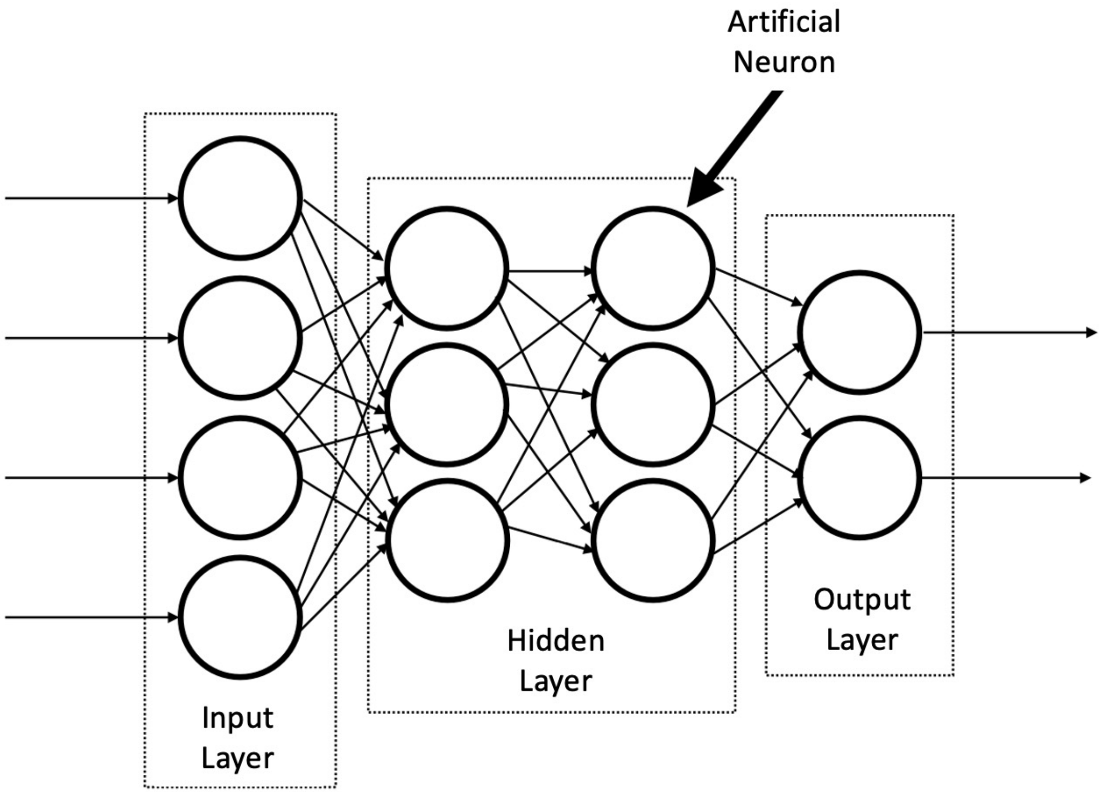
Identity Function



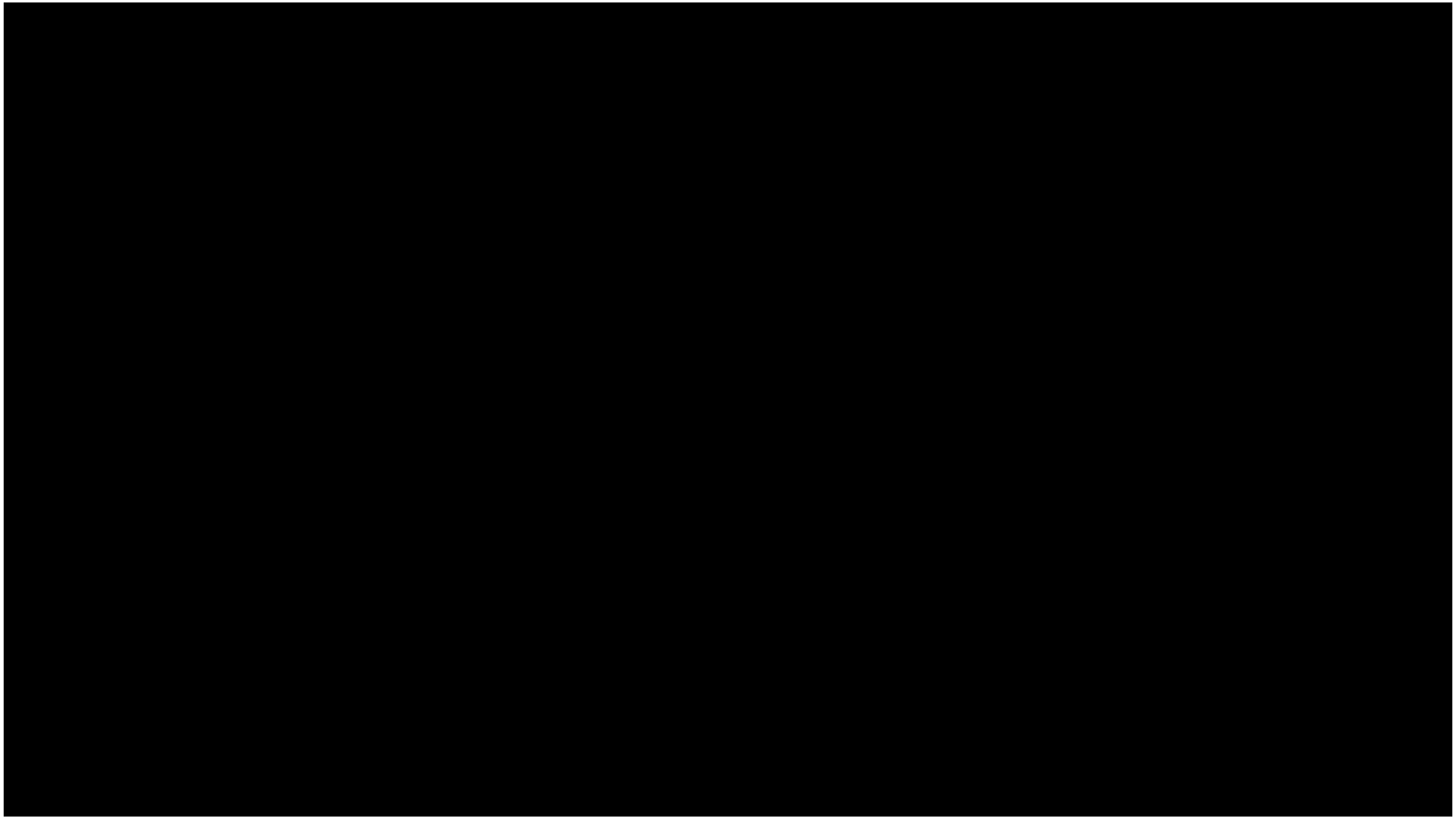
Multilayer Perceptron

- Can I use the Perceptron learning rule?

$$w_i = w_i + \eta(y^d - \hat{y}^d)x_i^d$$



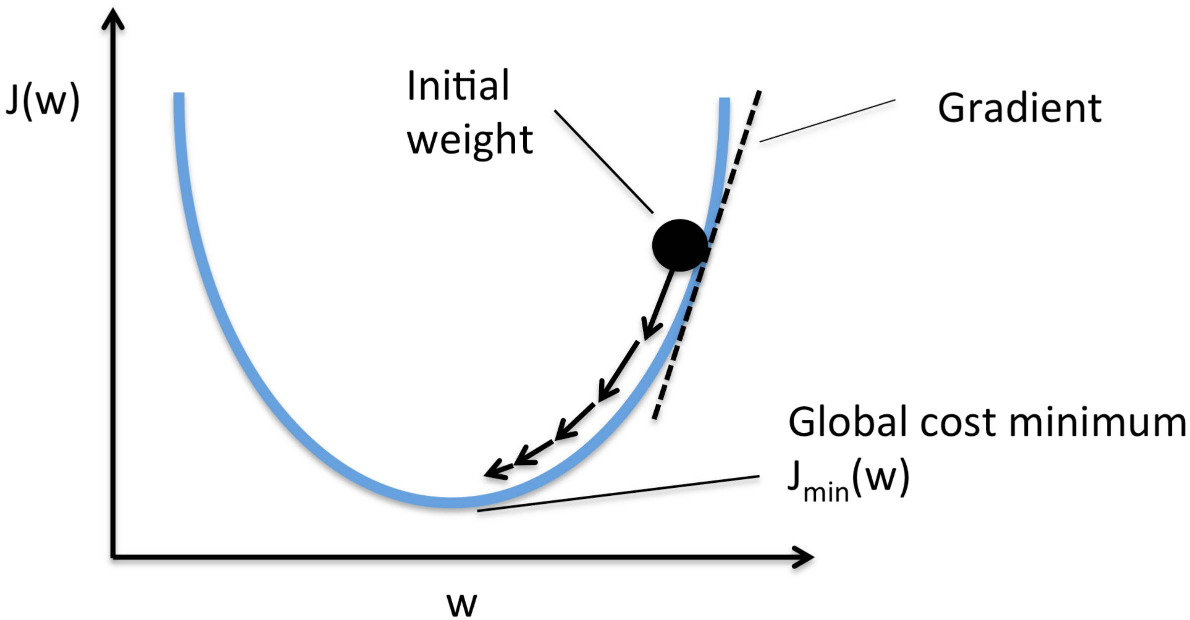
Multilayer Perceptron

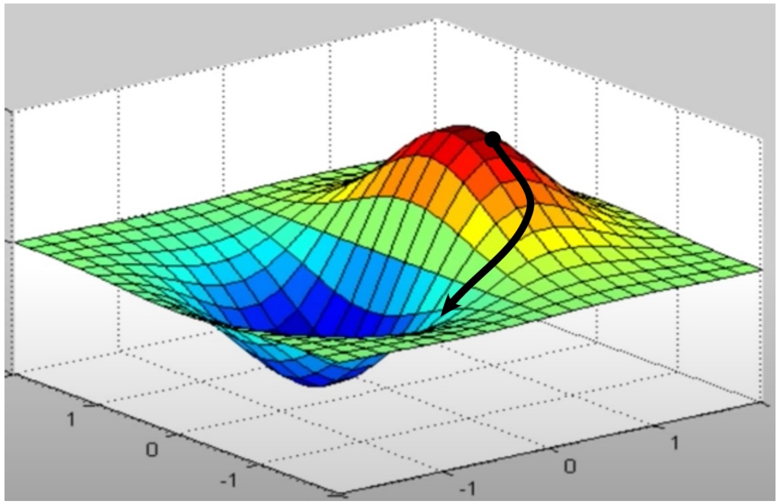


https://youtu.be/sZAlS3_dnk0

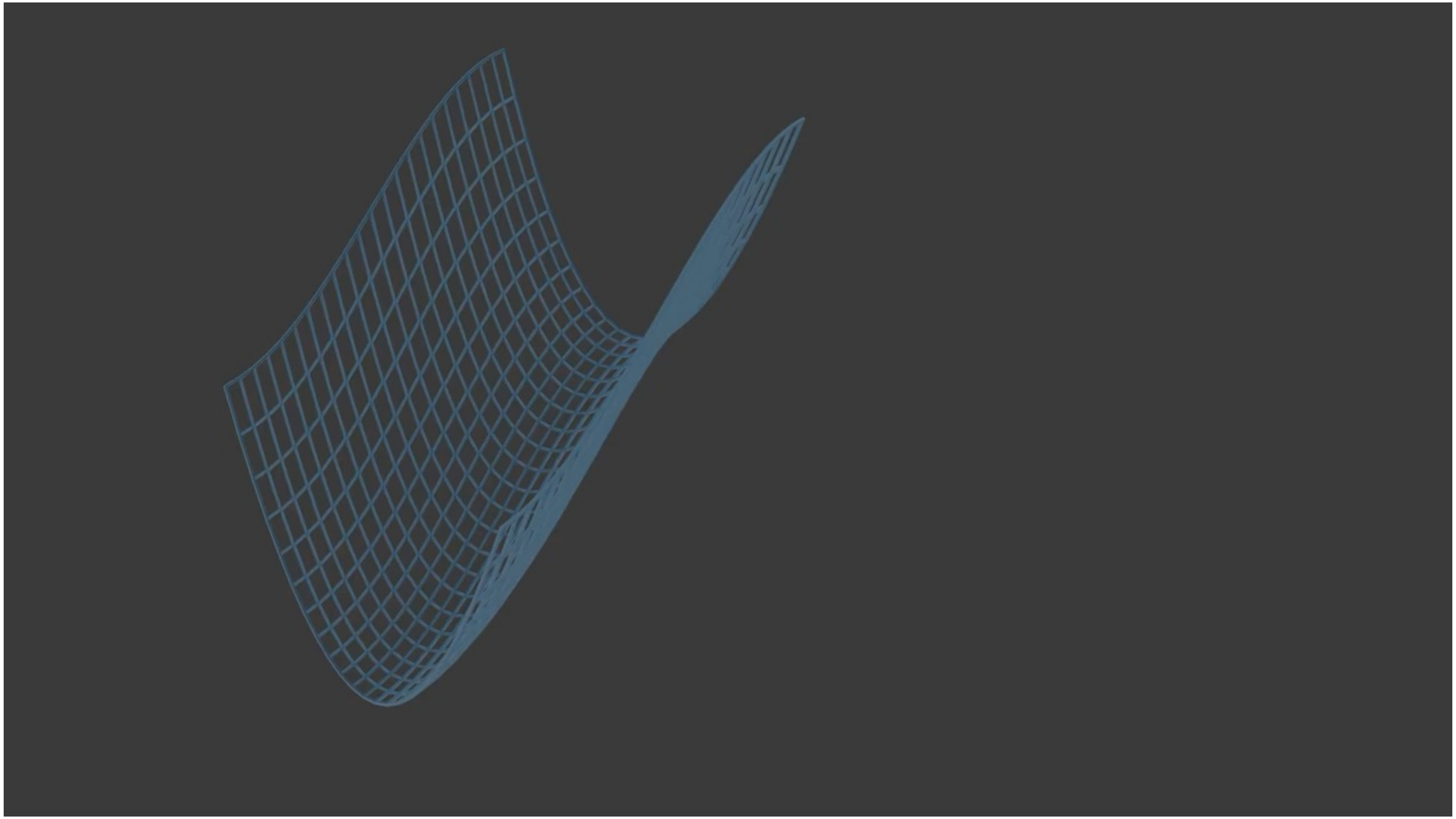
Multilayer Perceptron

- What is gradient descent?





Multilayer Perceptron



https://www.youtube.com/watch?v=qg4PchTECck

Multilayer Perceptron

- Let us assume the following loss function (least mean square):

$$J = \frac{1}{2}\sum_{k}(t_k - o_k)^2$$

- Use gradient descent

$$w_u = w_u + \Delta w_u$$

$$\Delta\, w_u = -\eta \nabla J(w_u)$$

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.

η is the learning rate (e.g., .05). n_{in} is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and $.05$).
- Until the termination condition is met, Do
  - For each $\langle \vec{x}, \vec{t} \rangle$ in $training\_examples$, Do

    Propagate the input forward through the network:

    1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

    Propagate the errors backward through the network:

    2. For each network output unit k, calculate its error term δ_k

    $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

    3. For each hidden unit h, calculate its error term δ_h

    $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{T4.4}$$

    4. Update each network weight w_{ji}

    $$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

    where

    $$\Delta w_{ji} = \eta\, \delta_j\, x_{ji} \tag{T4.5}$$

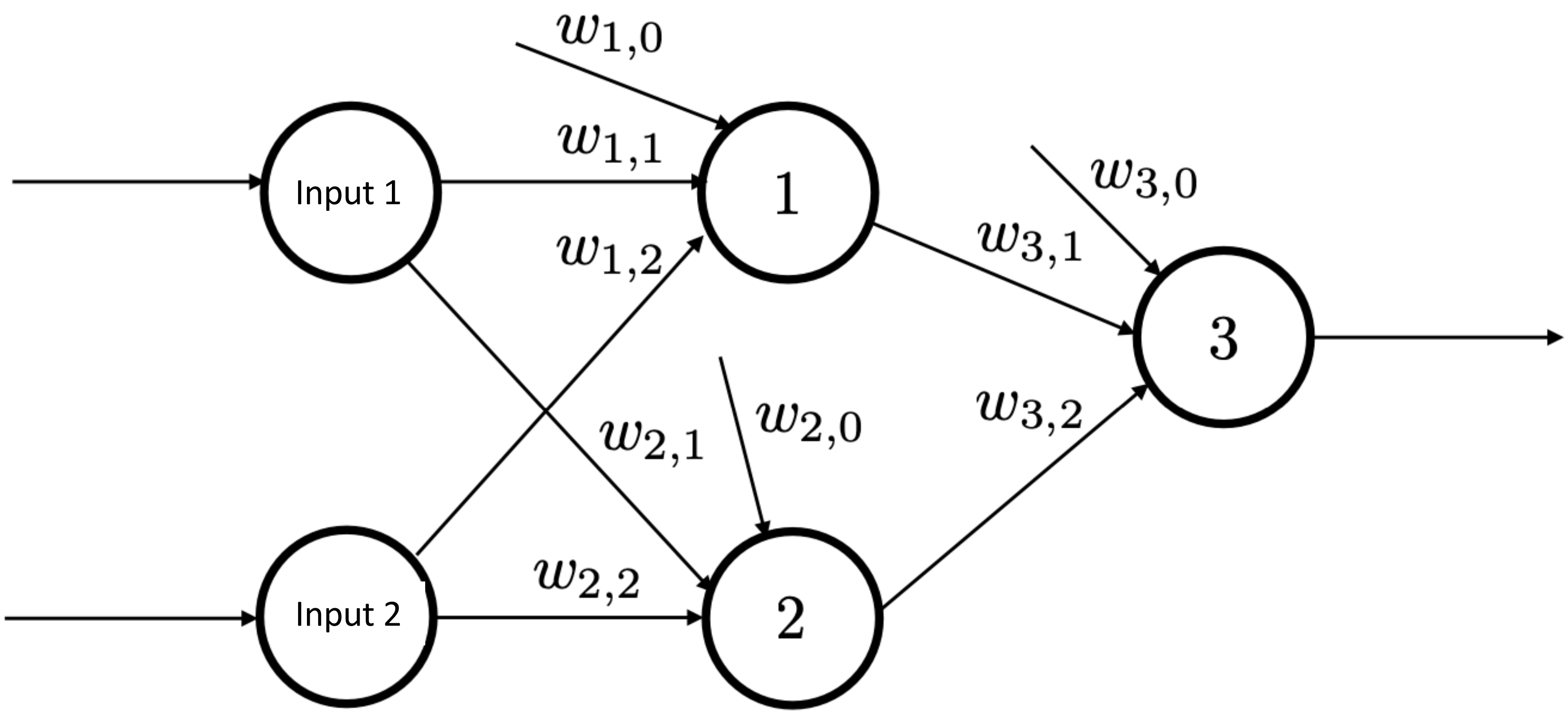
Multilayer Perceptron

- Example of training set

Training examples	x_1^d	x_2^d	y^d
d = 1	0	0	0
d = 2	0	1	1
d = 3	1	0	1
d = 4	1	1	0

Multilayer Perceptron

- Let’s use this neural network



Multilayer Perceptron

```
import numpy as np

# auxiliary functions

def logistic(x):
   return 1/(1 + np.exp(-x))

def logistic_derivative(x):
   return x * (1 - x)

# training examples

inputs = np.array([[0,0],[0,1],[1,0],[1,1]])

outputs = np.array([[0],[1],[1],[0]])

inputs_with_bias = np.concatenate((np.ones((inputs.shape[0],1)),inputs), axis=1)
```

```
# neural network hyperparameters

num_units_input = 2
num_units_hidden = 2
num_units_output = 1

# weights of unit 1 (unit 1 of hidden layer)
weights1 = np.random.uniform(size=(num_units_input+1,1))

# weights of unit 2 (unit 2 of hidden layer)
weights2 = np.random.uniform(size=(num_units_input+1,1))

# weights of unit 3 (unit of output layer)
weights3 = np.random.uniform(size=(num_units_hidden+1,1))

iterations = 100000
learning_rate = 0.1
```

Multilayer Perceptron

```python
#Backpropagation

for _ in range(iterations):

    #Forward pass

    activation_u1 = np.dot(inputs_with_bias,weights1)
    output_u1 = logistic(activation_u1)

    activation_u2 = np.dot(inputs_with_bias,weights2)
    output_u2 = logistic(activation_u2)

    inputs_u3 = np.concatenate((output_u1,output_u2),axis=1)
    inputs_u3_with_bias = np.concatenate((np.ones((inputs_u3.shape[0],1)), inputs_u3),axis=1)
    activation_u3 = np.dot(inputs_u3_with_bias,weights3)
    output_u3 = logistic(activation_u3)

    #Propagate error backwards

    error_term_u3 = logistic_derivative(output_u3) * (outputs - output_u3)

    error_term_u2 = logistic_derivative(output_u2) * (weights3[2] * error_term_u3)

    error_term_u1 = logistic_derivative(output_u1) * (weights3[1] * error_term_u3)

    #Update weights

    delta_weights3 = learning_rate * np.dot(error_term_u3.T,inputs_u3_with_bias)
    weights3 += delta_weights3.T

    delta_weights2 = learning_rate * np.dot(error_term_u2.T,inputs_with_bias)
    weights2 += delta_weights2.T

    delta_weights1 = learning_rate * np.dot(error_term_u1.T,inputs_with_bias)
    weights1 += delta_weights1.T
```

Multilayer Perceptron

```
print("Weights of unit 1: ")
print(*weights1)
print("Weights of unit 2: ")
print(*weights2)
print("Weights of unit 3: ")
print(*weights3)
print("\nOutput after 100,000 iterations: ")
print(*output_u3)
```

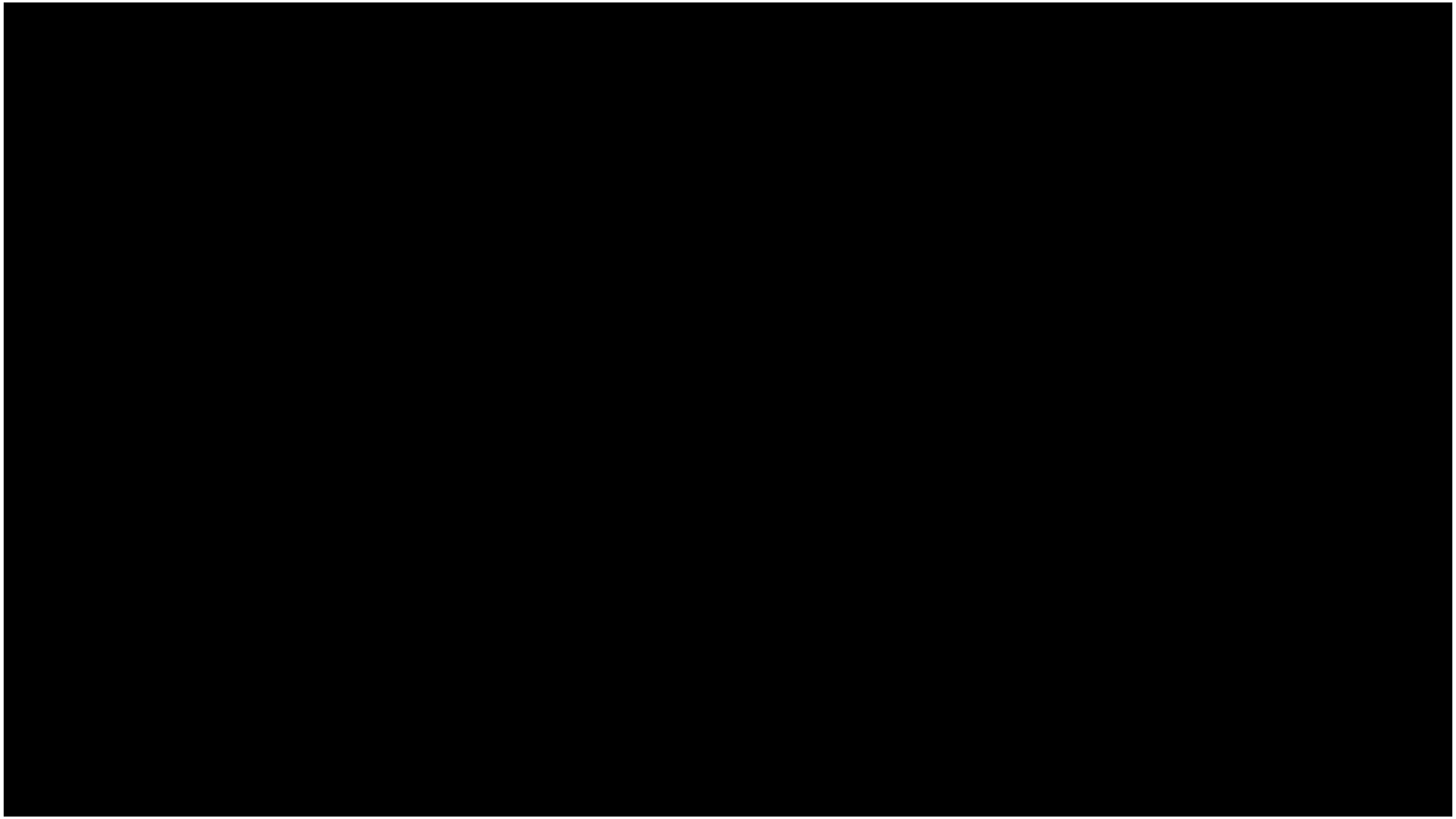
```
Weights of unit 1:
[-7.39251513] [4.82258129] [4.81665517]
Weights of unit 2:
[-3.02502763] [6.74793027] [6.72318093]
Weights of unit 3:
[-4.78993008] [-11.00507] [10.29431688]

Output after 100,000 iterations:
[0.01312546] [0.98876488] [0.98878312] [0.01156051]
```

What is Deep Learning?

- Multilayer Perceptron with many layers
- Convolutional Neural Network (CNN)
- Recurrent neural network
- Generative AI
- etc

CNNs



https://youtu.be/YRhxdVk_sIs

Thank You

sardinha@inf.puc-rio.br