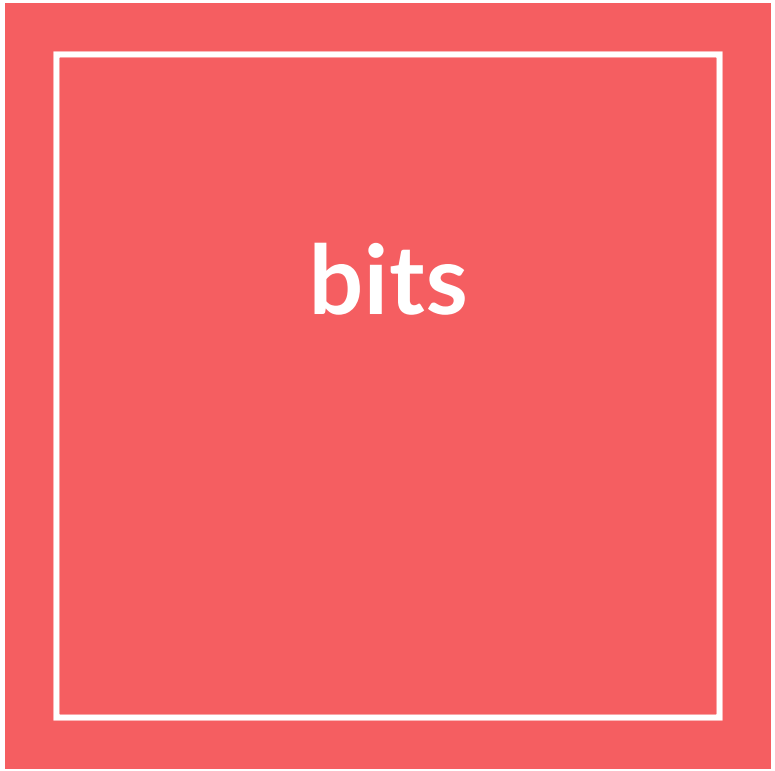
bits, neurônios e microscopia de força atômica

Mardônio França

boitatá lab


bits

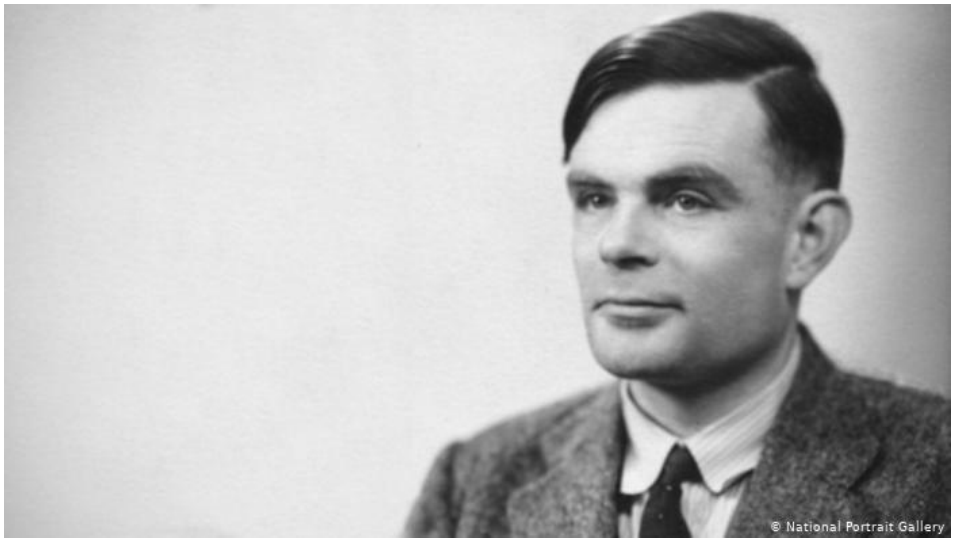
bits | A Computação e a Física

alan turing, o pai da computação

Em junho de 1938, ele obteve seu PhD no Departamento de Matemática de Princeton; sua dissertação, Sistemas de Lógica Baseada em Ordinais, introduziu o conceito de lógica ordinal e a noção de computação relativa, onde as máquinas de Turing são aumentadas com os chamados oráculos, permitindo o estudo de problemas que não podem ser resolvidos por máquinas de Turing.

Entre 1945 e 1947, Turing viveu em Hampton, Londres, enquanto trabalhava no projeto do computador ACE (Automatic Computing Engine) no Laboratório Nacional de Física (NPL - sigla em inglês). Ele apresentou um artigo em 19 de fevereiro de 1946, que foi o primeiro projeto detalhado de um computador capaz de armazenar um programa.

No final de 1947 ele voltou a Cambridge para um ano sabático, durante o qual produziu um trabalho seminal sobre Máquinas Inteligentes que não foi publicado em sua vida.



bits | A Computação e a Física

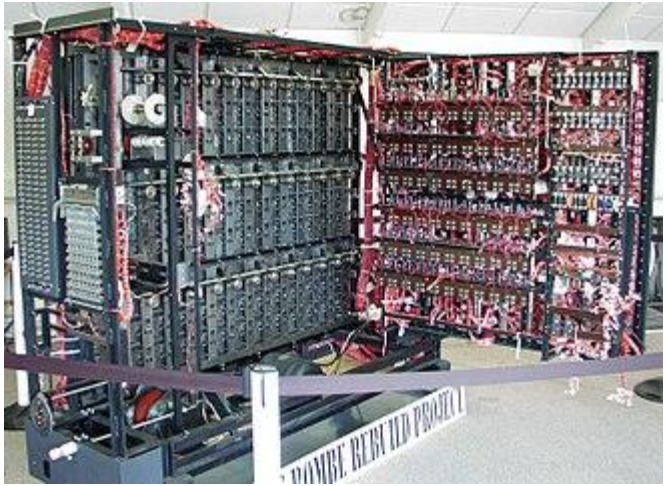
máquinas pensantes

Durante esse tempo, Turing continuou a fazer trabalhos mais abstratos em matemática e em "Computadores e Inteligência" (Mind, outubro de 1950), Turing abordou o problema da inteligência artificial e propôs um experimento que ficou conhecido como teste de Turing, uma tentativa de definir um padrão para uma máquina ser chamada de "inteligente".

A ideia era que se poderia dizer que um computador "pensa" se um interrogador humano não pudesse diferenciá-lo, por meio de conversa, de um ser humano.

No artigo, Turing sugeriu que, em vez de criar um programa para simular a mente de um adulto, seria melhor produzir um mais simples para simular a mente de uma criança e depois submetê-lo a um processo de educação.

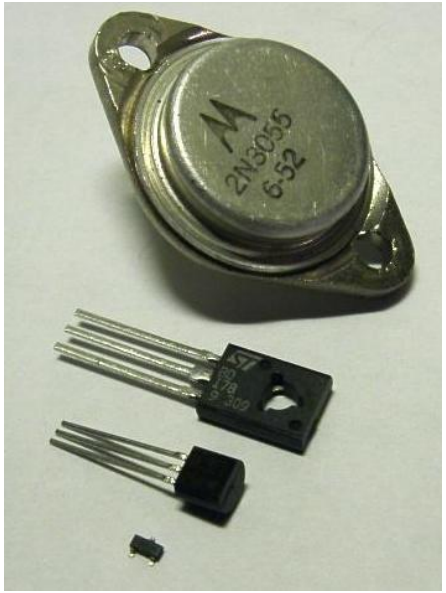
Uma forma invertida do teste de Turing é amplamente usada na Internet; o teste CAPTCHA destina-se a determinar se o usuário é humano ou computador.



Uma réplica completa e funcional de uma bomba eletromecânica no Museu Nacional de Computação em Bletchley Park

bits | O transistor

O transistor é o bloco de construção fundamental dos dispositivos eletrônicos modernos e é onipresente nos sistemas modernos.



Julius Edgar Lilienfeld patenteou um transistor de efeito de campo em 1926, mas não foi possível construir um dispositivo de trabalho naquele momento.

O primeiro dispositivo praticamente implementado foi um transistor de contato pontual inventado em 1947 pelos físicos estadunidenses John Bardeen, Walter Brattain e William Shockley. O transistor revolucionou o campo da eletrônica e abriu caminho para rádios, calculadoras e computadores menores e mais baratos, entre outras coisas.

O transistor está na lista de marcos do IEEE em eletrônica, e Bardeen, Brattain e Shockley dividiram o Prêmio Nobel de Física em 1956 por sua conquista.

neurônios

neurônios | redes neurais artificiais

base biológica | sistema nervoso

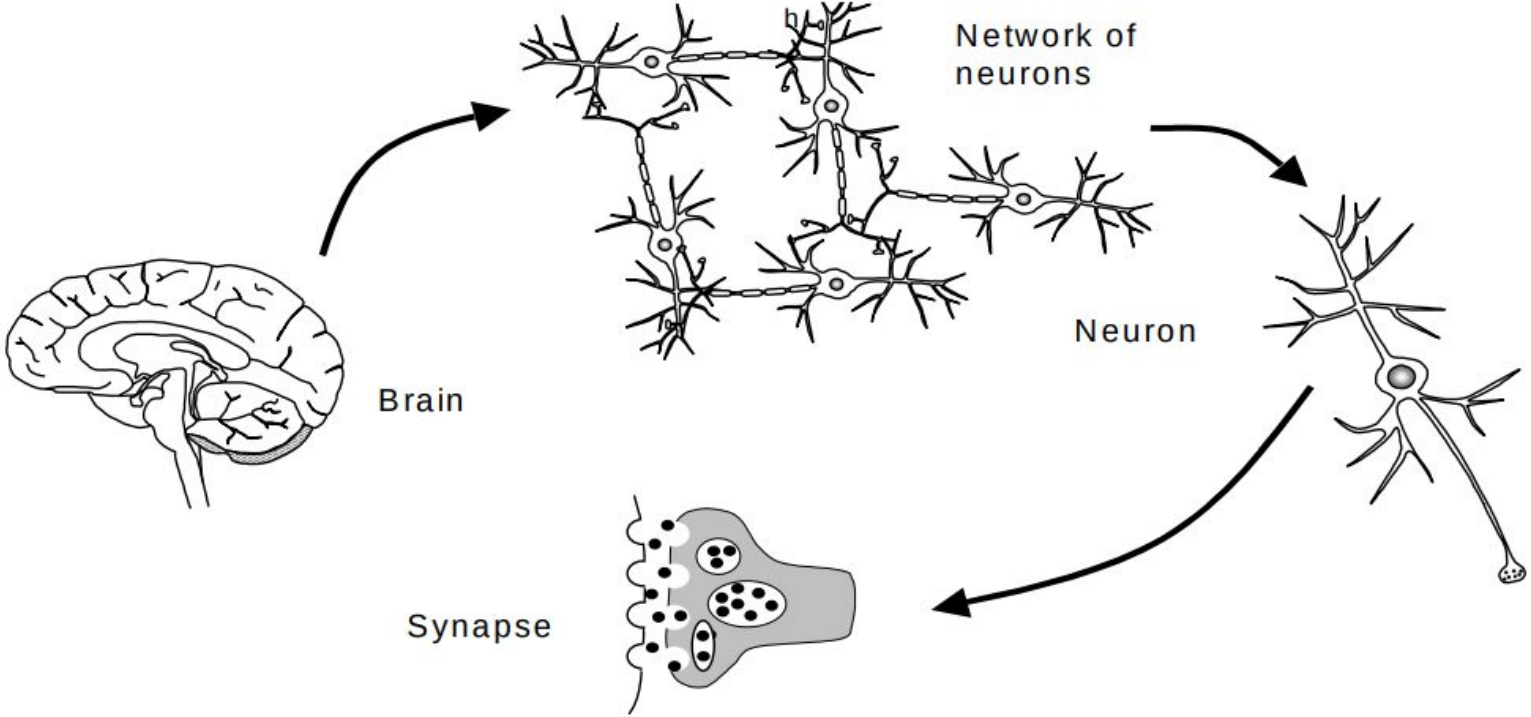
- O sistema nervoso pode ser organizado em diferentes níveis: moléculas, sinapses, neurônios, camadas, mapas e sistemas;

- Os neurônios podem receber e enviar sinais a vários outros neurônios;

- Os neurônios que enviam sinais, chamados de neurônios pré-sinápticos ou “enviadores”, fazem contato com os neurônios receptores ou pós-sinápticos em regiões especializadas denominadas de sinapses.

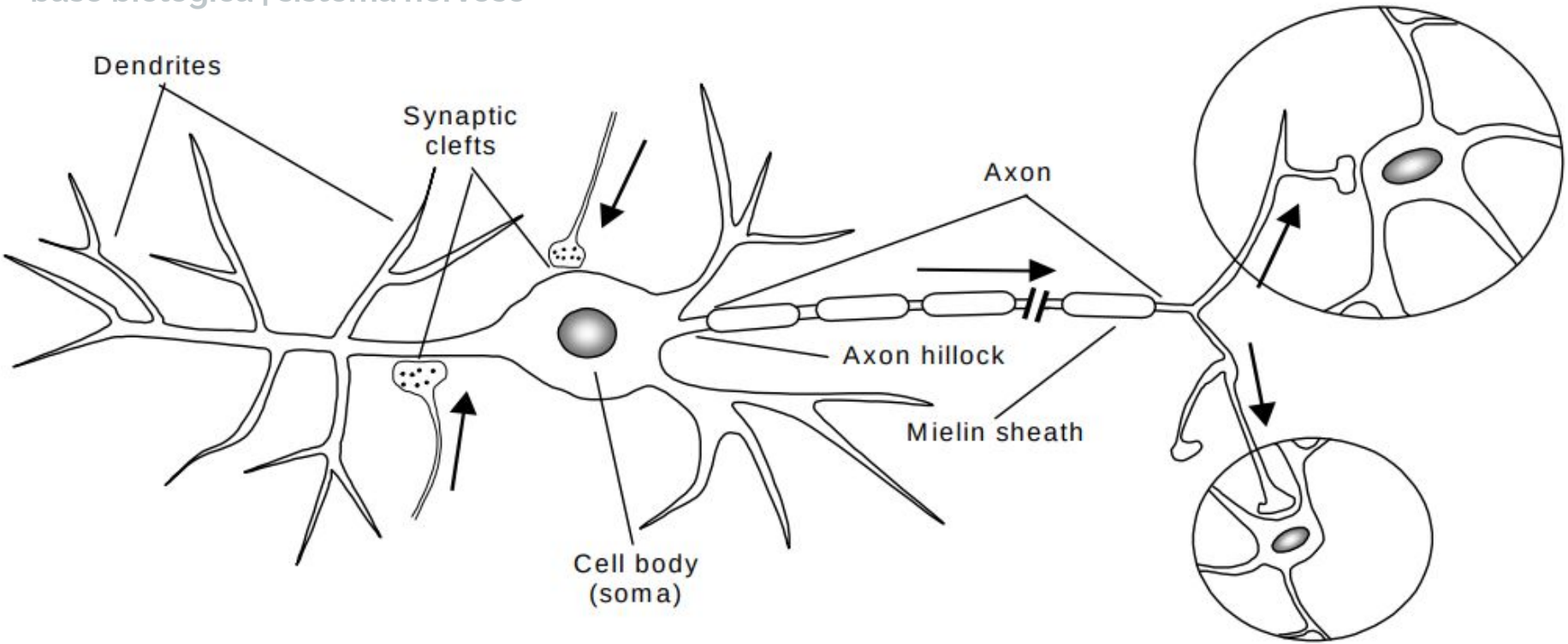
neurônios | redes neurais artificiais

base biológica | sistema nervoso



neurônios | redes neurais artificiais

base biológica | sistema nervoso



neurônios | redes neurais artificiais

- Uma rede neural artificial é um circuito composto por uma grande quantidade de unidades simples de processamento inspiradas no sistema neural.

- As principais partes do neurônio artificial genérico são:

- as sinapses, caracterizadas pelos seus pesos associados;

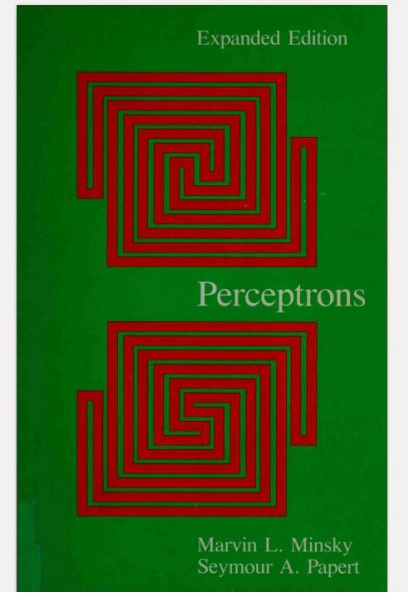
- a junção somadora;

- a barreira de ativação (bias);

- a função de ativação.

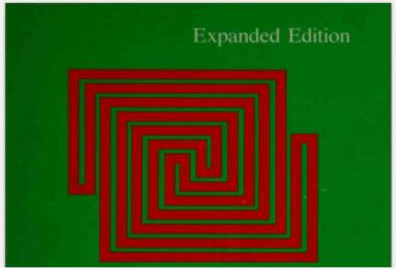
neurônios | redes neurais artificiais

neurônio artificial | referência seminal



neurônios | redes neurais artificiais

neurônio artificial | referência seminal



This book is about perceptrons—the simplest learning machines. However, our deeper purpose is to gain more general insights into the interconnected subjects of parallel computation, pattern recognition, knowledge representation, and learning. It is only because one cannot think productively about such matters without studying specific examples that we focus on theories of perceptrons.

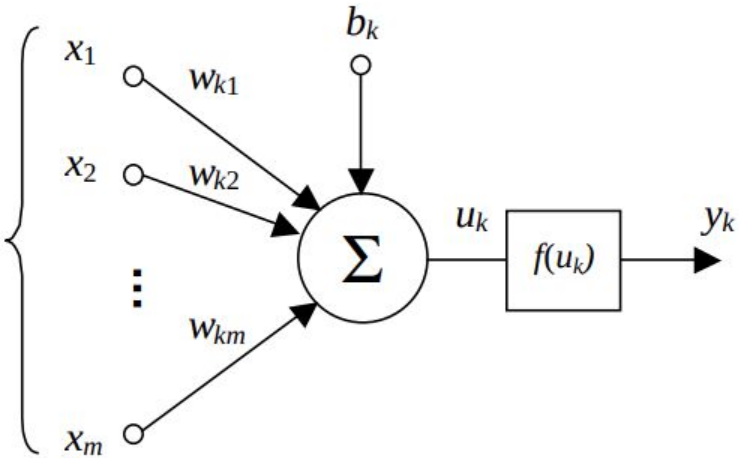
neurônios | redes neurais artificiais

neurônio artificial | referência seminal

In preparing this edition we were tempted to "bring those theories up to date." But when we found that little of significance had changed since 1969, when the book was first published, we concluded that it would be more useful to keep the original text (with its corrections of 1972) and add an epilogue, so that the book could still be read in its original form. One reason why progress has been so slow in this field is that researchers unfamiliar with its history have continued to make many of the same mistakes that others have made before them. Some readers may be shocked to hear it said that little of significance has happened in this field. Have not perceptron-like networks—under the new name *connectionism*—become a major subject of discussion at gatherings of psychologists and computer scientists? Has not there been a "connectionist revolution?" Certainly yes, in that there is a great deal of interest and discussion.

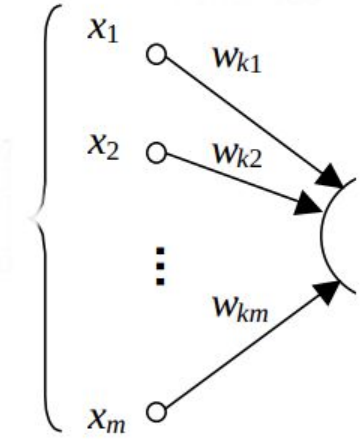
neurônios | redes neurais artificiais

neurônio artificial



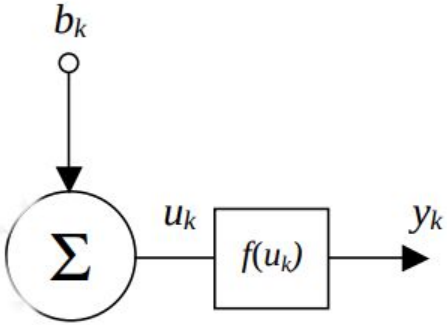
neurônios | redes neurais artificiais

neurônio artificial | sinapses



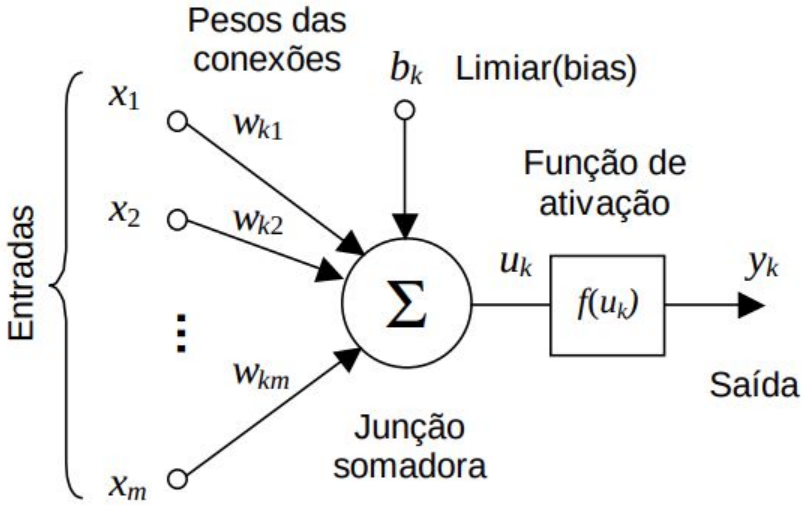
neurônios | redes neurais artificiais

neurônio artificial | bias | função somadora | junção



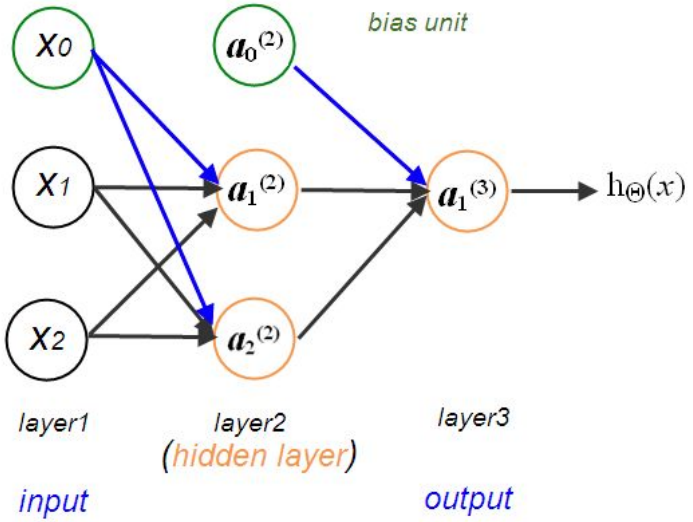
neurônios | redes neurais artificiais

neurônio artificial | arquitetura



neurônios | redes neurais artificiais

neurônio artificial | dinâmica



neurônios | redes neurais artificiais

neurônio artificial | dinâmica

1. $a_i^{(j)}$: "activation" of unit i in layer j
2. $\Theta^{(j)}$: matrix of weights controlling function mapping from layer j to layer $j+1$

Here are the computations represented by the NN picture above:

$$a_0^{(2)} = g(\Theta_{00}^{(1)} x_0 + \Theta_{01}^{(1)} x_1 + \Theta_{02}^{(1)} x_2) = g(\Theta_0^T x) = g(z_0^{(2)})$$

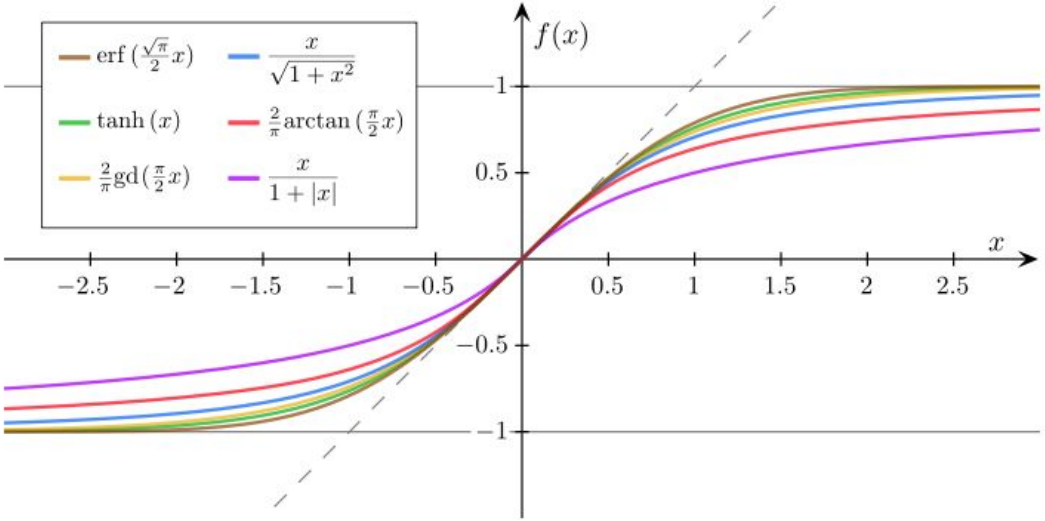
$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2) = g(\Theta_1^T x) = g(z_1^{(2)})$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2) = g(\Theta_2^T x) = g(z_2^{(2)})$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)})$$

neurônios | redes neurais artificiais

neurônio artificial | dinâmica | soluções



neurônios | redes neurais artificiais

neurônio artificial | dinâmica | algoritmo

$$Y = sigm\,(\,sigm\,(W_1 X + B_1)\,W_2 + B2\,)$$

neurônios | redes neurais artificiais

neurônio artificial | dinâmica | algoritmo

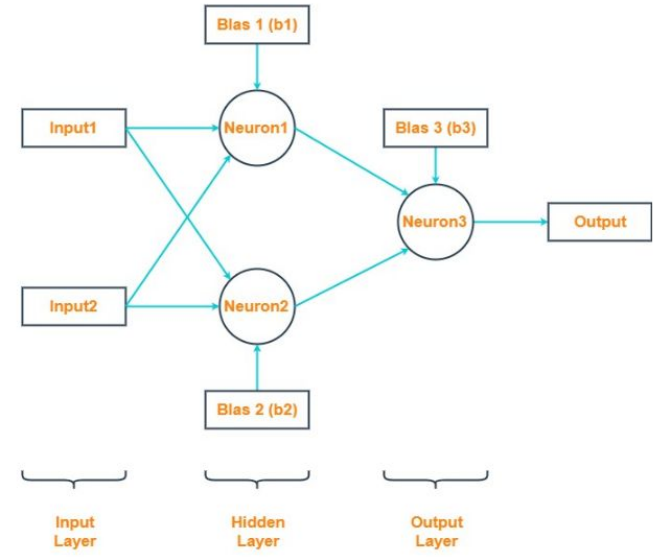
$$Y = sigm\,(\,sigm\,(W_1 X + B_1)\,W_2 + B2\,)$$

Where:

- X is an input value vector, size 2x1 elements
- W1 is a matrix of the coefficient for the first layer, size 2x2 elements
- B1 is a bias for the first layer, a vector with 2x1 elements
- W2 is a vector of the coefficient for the first layer, size 2x1 elements
- B2 is a value of bias for the second layer, size 1x1 element
- Y is an output value, size 1x1 element
- sigm(x) is a sigmoid activation function for neural network

neurônios | redes neurais artificiais

neurônio artificial | dinâmica | algoritmo



neurônios | redes neurais artificiais

neurônio artificial | dinâmica | forward backpropagation

Forward Propagation

If we use matrix notation, the equations of the previous section become:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_0^{(2)} \\ z_1^{(2)} \\ z_2^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} x = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

$$a_0^{(2)} = 1.0$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

neurônios | redes neurais artificiais

neurônio artificial | dinâmica | forward backpropagation

Back Propagation (Gradient computation)

The backpropagation learning algorithm can be divided into two phases: propagation and weight update.
- from wiki - Backpropagatio.

1. **Phase 1: Propagation**
   Each propagation involves the following steps:
   1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.
   2. Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas of all output and hidden neurons.
2. **Phase 2: Weight update**
   For each weight-synapse follow the following steps:
   1. Multiply its output delta and input activation to get the gradient of the weight.
   2. Subtract a ratio (percentage) of the gradient from the weight.

   This ratio (percentage) influences the speed and quality of learning; it is called the learning rate. The greater the ratio, the faster the neuron trains; the lower the ratio, the more accurate the training is. The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.
   Repeat phase 1 and 2 until the performance of the network is satisfactory.

neurônios | redes neurais artificiais

neurônio artificial | dinâmica | forward backpropagation

Also, the derivative of cost function can be written like this:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

We use this value to update weights and we can multiply learning rate before we adjust the weight.

```
self.weights[i] += learning_rate * layer.T.dot(delta)
```

neurônios | redes neurais artificiais

neurônio artificial | problema inicial

- como implementar a função XOR com redes neurais ?

neurônios | redes neurais artificiais

- como implementar a função XOR com redes neurais ?

$$p \oplus q \quad = \quad (p \vee q) \wedge \neg(p \wedge q)$$

X1	X2	Y = X1 xor X2
0	0	0
0	1	1
1	0	1
1	1	0

neurônios | redes neurais artificiais

implementação 1 da função XOR com redes neurais

jupyter base Last Checkpoint: 08/14/2020 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Run Code

In [1]:
```
import numpy as np
```

In [2]:
```
def sigmoid (x):
    return 1/(1 + np.exp(-x))
```

In [3]:
```
def sigmoid_derivative(x):
    return x * (1 - x)
```

In [4]:
```
#Input datasets
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
expected_output = np.array([[0],[1],[1],[0]])
```

In [5]:
```
epochs = 10000
lr = 0.1
inputLayerNeurons, hiddenLayerNeurons, outputLayerNeurons = 2,2,1
```

In [6]:
```
#Random weights and bias initialization
hidden_weights = np.random.uniform(size=(inputLayerNeurons,hiddenLayerNeurons))
hidden_bias =np.random.uniform(size=(1,hiddenLayerNeurons))
output_weights = np.random.uniform(size=(hiddenLayerNeurons,outputLayerNeurons))
output_bias = np.random.uniform(size=(1,outputLayerNeurons))
```

In [7]:
```
print("Initial hidden weights: ",end='')
print(*hidden_weights)
print("Initial hidden biases: ",end='')
print(*hidden_bias)
print("Initial output weights: ",end='')
print(*output_weights)
print("Initial output biases: ",end='')
print(*output_bias)
```

```
Initial hidden weights: [0.16566663 0.58864165] [0.76493029 0.16355254]
Initial hidden biases: [0.07580087 0.62170622]
Initial output weights: [0.81057427] [0.88378203]
Initial output biases: [0.10098036]
```

neurônios | redes neurais artificiais

implementação 1 da função XOR com redes neurais

In [8]:
```
#Training algorithm
```

In [9]:
```
for _ in range(epochs):
    #Forward Propagation
    hidden_layer_activation = np.dot(inputs,hidden_weights)
    hidden_layer_activation += hidden_bias
    hidden_layer_output = sigmoid(hidden_layer_activation)

    output_layer_activation = np.dot(hidden_layer_output,output_weights)
    output_layer_activation += output_bias
    predicted_output = sigmoid(output_layer_activation)

    #Backpropagation
    error = expected_output - predicted_output
    print(error)
    d_predicted_output = error * sigmoid_derivative(predicted_output)

    error_hidden_layer = d_predicted_output.dot(output_weights.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

    #Updating Weights and Biases
    output_weights += hidden_layer_output.T.dot(d_predicted_output) * lr
    output_bias += np.sum(d_predicted_output,axis=0,keepdims=True) * lr
    hidden_weights += inputs.T.dot(d_hidden_layer) * lr
    hidden_bias += np.sum(d_hidden_layer,axis=0,keepdims=True) * lr
```

```
 [ 0.07104841]
 [ 0.07108832]
 [-0.08006252]]
[[-0.0709131 ]
 [ 0.07103615]
 [ 0.07107605]
 [-0.08004744]]
[[-0.07090329]
 [ 0.0710239 ]
 [ 0.07106379]
 [-0.08003237]]
[[-0.07089348]
 [ 0.07101165]
 [ 0.07105153]
 [-0.0800173 ]]
[[-0.07088368]
 [ 0.07099941]
 [ 0.07103928]
 [-0.08000225]]
```

neurônios | redes neurais artificiais

implementação 1 da função XOR com redes neurais

In [10]:
```
print("Final hidden weights: ",end='')
print(*hidden_weights)
print("Final hidden bias: ",end='')
print(*hidden_bias)
print("Final output weights: ",end='')
print(*output_weights)
print("Final output bias: ",end='')
print(*output_bias)

print("\nOutput from neural network after 10,000 epochs: ",end='')
print(*predicted_output)
```

```
Final hidden weights: [3.46729596 5.80123443] [3.46889122 5.81208943]
Final hidden bias: [-5.26076288 -2.24887423]
Final output weights: [-7.44506286] [7.03544267]
Final output bias: [-3.20640143]

Output from neural network after 10,000 epochs: [0.07088368] [0.92900059] [0.92896072] [0.08000225]
```

neurônios | redes neurais artificiais

implementação 2 da função XOR com redes neurais

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
```

In [2]:
```
# Sigmoid Function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

In [3]:
```
# Initialization of the neural network parameters
# Initialized all the weights in the range of between 0 and 1
# Bias values are initialized to 0
def initializeParameters(inputFeatures, neuronsInHiddenLayers, outputFeatures):
    W1 = np.random.randn(neuronsInHiddenLayers, inputFeatures)
    W2 = np.random.randn(outputFeatures, neuronsInHiddenLayers)
    b1 = np.zeros((neuronsInHiddenLayers, 1))
    b2 = np.zeros((outputFeatures, 1))

    parameters = {"W1" : W1, "b1": b1,
                  "W2" : W2, "b2": b2}
    return parameters
```

In [4]:
```
# Forward Propagation
def forwardPropagation(X, Y, parameters):
    m = X.shape[1]
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    b1 = parameters["b1"]
    b2 = parameters["b2"]

    Z1 = np.dot(W1, X) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    cache = (Z1, A1, W1, b1, Z2, A2, W2, b2)
    logprobs = np.multiply(np.log(A2), Y) + np.multiply(np.log(1 - A2), (1 - Y))
    cost = -np.sum(logprobs) / m
    return cost, cache, A2
```

neurônios | redes neurais artificiais

implementação 2 da função XOR com redes neurais

In [5]:
```
# Backward Propagation
def backwardPropagation(X, Y, cache):
    m = X.shape[1]
    (Z1, A1, W1, b1, Z2, A2, W2, b2) = cache

    dZ2 = A2 - Y
    dW2 = np.dot(dZ2, A1.T) / m
    db2 = np.sum(dZ2, axis = 1, keepdims = True)

    dA1 = np.dot(W2.T, dZ2)
    dZ1 = np.multiply(dA1, A1 * (1- A1))
    dW1 = np.dot(dZ1, X.T) / m
    db1 = np.sum(dZ1, axis = 1, keepdims = True) / m

    gradients = {"dZ2": dZ2, "dW2": dW2, "db2": db2,
                 "dZ1": dZ1, "dW1": dW1, "db1": db1}
    return gradients
```

In [6]:
```
# Updating the weights based on the negative gradients
def updateParameters(parameters, gradients, learningRate):
    parameters["W1"] = parameters["W1"] - learningRate * gradients["dW1"]
    parameters["W2"] = parameters["W2"] - learningRate * gradients["dW2"]
    parameters["b1"] = parameters["b1"] - learningRate * gradients["db1"]
    parameters["b2"] = parameters["b2"] - learningRate * gradients["db2"]
    return parameters
```

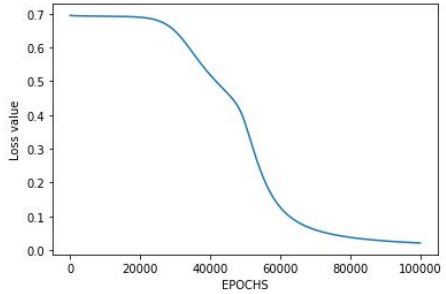
In [7]:
```
# Model to learn the XOR truth table
X = np.array([[0, 0, 1, 1], [0, 1, 0, 1]]) # XOR input
Y = np.array([[0, 1, 1, 0]]) # XOR output
```

In [8]:
```
# Define model parameters
neuronsInHiddenLayers = 2 # number of hidden layer neurons (2)
inputFeatures = X.shape[0] # number of input features (2)
outputFeatures = Y.shape[0] # number of output features (1)
parameters = initializeParameters(inputFeatures, neuronsInHiddenLayers, outputFeatures)
epoch = 100000
learningRate = 0.01
losses = np.zeros((epoch, 1))
```

neurônios | redes neurais artificiais

implementação 2 da função XOR com redes neurais

In [10]:
```
# Evaluating the performance
plt.figure()
plt.plot(losses)
plt.xlabel("EPOCHS")
plt.ylabel("Loss value")
plt.show()
```



In [15]:
```
# Testing
X = np.array([[1, 1, 0, 0], [0, 1, 0, 1]]) # XOR input
cost, cache , A2 = forwardPropagation(X, Y, parameters)
prediction = (A2 > 0.5) * 1.0
```

In [17]:
```
print(cost)
```

```
3.8765000865506587
```

In [18]:
```
print(cache)
```

```
(array([[ -3.74290062, -10.14888679,   2.53547422,  -3.87051196],
       [  2.28389762,  -2.15742733,   6.70695723,   2.26563228]]), array([[2.31372877e-02, 3.91180754e-05, 9.26591
578e-01, 2.04219431e-02],
       [9.07534637e-01, 1.03639204e-01, 9.98779115e-01, 9.05990441e-01]]), array([[-6.27837483, -6.40598617],
       [-4.42305961, -4.44132495]]), array([[2.53547422],
       [6.70695723]]), array([[ 3.86777538, -3.63646137, -4.03780731,  3.87932611]]), array([[0.97952324, 0.025669
14, 0.01733046, 0.97975364]]), array([[-9.72137957,  9.61416436]]), array([[-4.63248543]]))
```

neurônios | redes neurais artificiais

implementação 2 da função XOR com redes neurais

In [15]:
```
# Testing
X = np.array([[1, 1, 0, 0], [0, 1, 0, 1]]) # XOR input
cost, cache , A2 = forwardPropagation(X, Y, parameters)
prediction = (A2 > 0.5) * 1.0
```

In [17]:
```
print(cost)
```

```
3.8765000865506587
```

In [18]:
```
print(cache)
```

```
(array([[ -3.74290062, -10.14888679,   2.53547422,  -3.87051196],
       [  2.28389762,  -2.15742733,   6.70695723,   2.26563228]]), array([[2.31372877e-02, 3.91180754e-05, 9.26591
578e-01, 2.04219431e-02],
       [9.07534637e-01, 1.03639204e-01, 9.98779115e-01, 9.05990441e-01]]), array([[-6.27837483, -6.40598617],
       [-4.42305961, -4.44132495]]), array([[2.53547422],
       [6.70695723]]), array([[ 3.86777538, -3.63646137, -4.03780731,  3.87932611]]), array([[0.97952324, 0.025669
14, 0.01733046, 0.97975364]]), array([[-9.72137957,  9.61416436]]), array([[-4.63248543]]))
```

In [19]:
```
print(A2)
```

```
[[0.97952324 0.02566914 0.01733046 0.97975364]]
```

In [20]:
```
print(prediction)
```

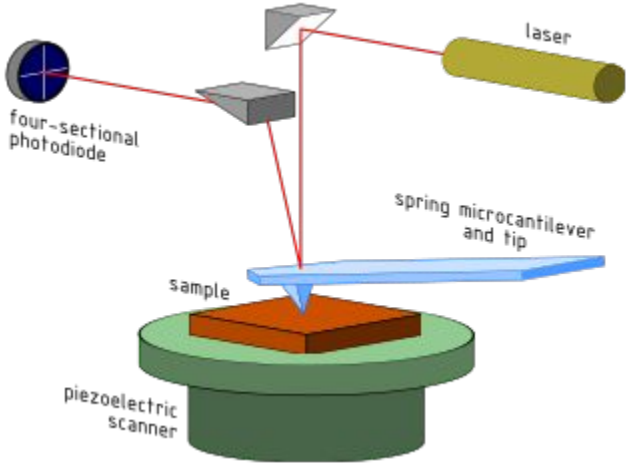
```
[[1. 0. 0. 1.]]
```

microscopia de força atômica

microscopia | microscopia de força atômica

A microscopia de força atômica é uma técnica de análise que consiste na varredura da superfície de uma amostra com uma sonda a fim de obter sua imagem topográfica com resolução atômica, além de mapear certas propriedades mecânicas e físico-químicas dos materiais que as compõe.

Para tal fim, utilizamos o microscópio de força atômica, desenvolvido em 1985 pelo Dr. Gerd Binnig et al. Ao desenvolver o aparelho, Binnig visava medir forças menores que 1μN entre a ponteira (tip) e a superfície da amostra. A técnica se tornou um excelente perfilador topográfico de superfície e medidor de força normal em micro e nanoescala.



AFM : Esquema de funcionamento

microscopia | microscopia de força atômica

A técnica se tornou um excelente perfilador topográfico de superfície e medidor de força normal em micro e nanoescala.

Atualmente as análises são feitas em áreas multidisciplinares como Física, Química, Biologia, Engenharia de Materiais, Eletrônica e Nanotecnologia. Essa variedade é possível, porque a técnica pode ser usada em amostras condutoras ou isolantes, magnéticas ou não magnéticas, secas ou em líquidos.

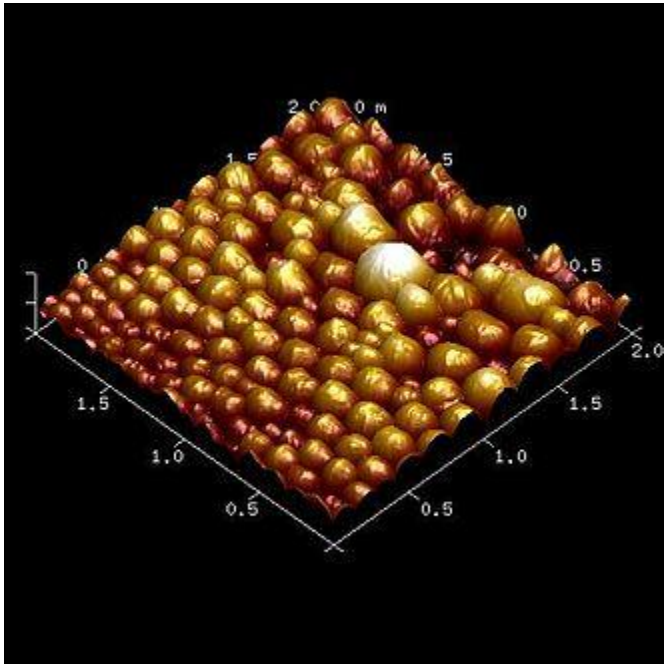


Imagem Topográfica de AFM (3D)

microscopia | microscopia de força atômica

Topografia

Uma das imagens obtidas pela técnica, é a da topografia da amostra. A ponta de prova (tip) sofre repulsão ou atração pela superfície dependendo de sua topografia e essas variações são detectadas pelo movimento do laser incidindo no fotodetector Com isso é possível formar uma imagem topográfica digitalmente e obter diversas informações sobre a amostra, como por exemplo rugosidade e variação de altura.

microscopia | microscopia de força atômica

Fase

Tipo de imagem que aparece ao ser feito o modo Contato Intermitente (Tapping Mode) (ver: microscópio de força atômica), é formada pela diferença de fase que surge quando o cantiléver, que está oscilando em uma frequência específica, passa por regiões da amostra que possuem interações diferentes com o tip ou diferença de altura. Ao comparar a imagem topográfica com a imagem de fase, é possível identificar regiões que possuem maior ou menor adesão que outras por exemplo.

microscopia | microscopia de força atômica

Curva de Força

Imagem presente nas varreduras por Modo Contato (Contact Mode), é formada pela deflexão lateral do cantiléver ao fazer a varredura e é importante, pois nos dá informações sobre medidas de rugosidade do material.

Esquema da variação de força lateral

microscopia | microscopia de força atômica

Força Lateral

O cristal piezoelétrico do eixo z, faz a aproximação do cantiléver à amostra, o tip toca a amostra, realiza uma força específica sobre a mesma (definida pelo usuário) e retrai voltanto a posição original. Esse procedimento gera um gráfico que é a curva de força sobre o material. Com ele temos informações sobre a interação entre o tip e a superfície da amostra.

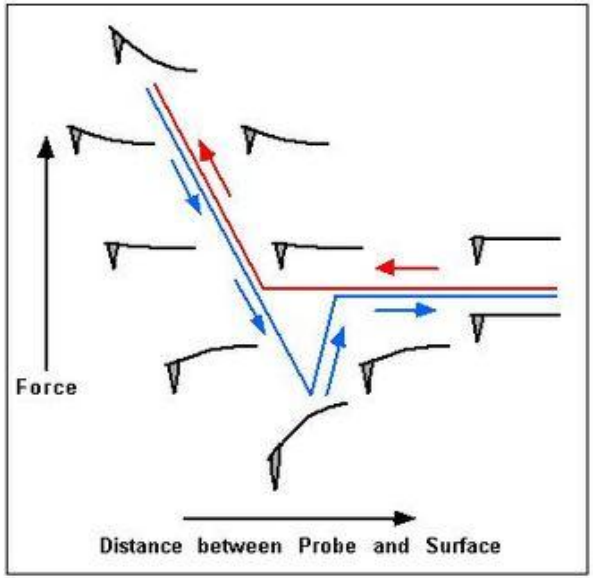


Gráfico de uma curva de força

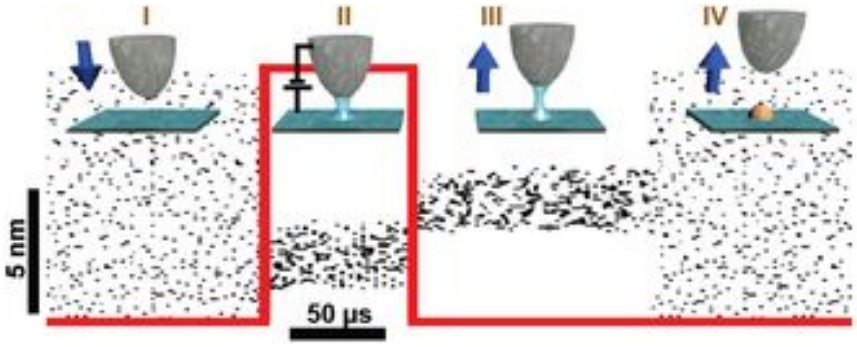
microscopia | microscopia de força atômica

Aplicações | Nanolitografia

Nanolitografia por Oxidação Local (Local oxidation nanolithography)

Muitos métodos para modificações locais de superfície por AFM já foram propostos, porém uma estratégia poderosa é a oxidação anódica local, via crescimento de uma camada de óxido induzida por uma corrente elétrica aplicada entre o tip e a superfície.

Essa técnica possibilita a fabricação de nanofios, nanotransistores, circuitos específicos e uma série de outros dispositivos que podem ser usados em nanoeletrônica. Além disso é possível desenhar moldes que podem ser usados na produção de outros produtos nanotecnológicos

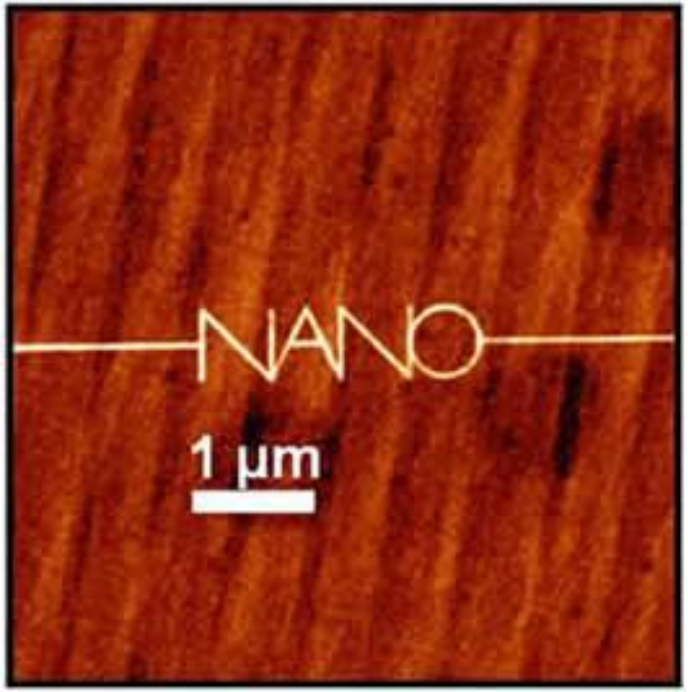


Etapas da nanolitografia

microscopia | microscopia de força atômica

Aplicações | Nanolitografia

Nanolitografia por Oxidação Local (Local oxidation nanolithography)



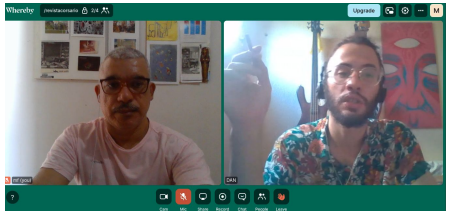
Transistor de nanofio feito por nanolitografia por oxidação local

resultados

resultados iniciais | afm | machine learning

boitatá lab | publicação

Atomic Force Microscopy (AFM) feature extraction and fault diagnosis using Clustering Algorithm based on Minimum Spanning Tree (MST) - xxxiv-encontro-de-fisicos-do-norte-e-nordeste (2020)



Cooperação com Doutorando Daniel Brito (Física UFC)

Cooperação com Lara Hissa Graduanda (Física UFC), Vinicius Sampaio (Computação UECE), Luan Misael Mestre (Física UFC), Ramon R. Chaves Analista de Sistema (Casa Magalhães) e Ludwing Marenco Doutorando (Física UFC)

resultados iniciais | afm | machine learning

boitatá lab | publicação

Atomic Force Microscopy (AFM) feature extraction and fault diagnosis using Clustering Algorithm based on Minimum Spanning Tree (MST)

*Lara D. Hissa[1], Vinicius A. Sampaio[2], Ramon R. Chaves[3], Daniel B. Araújo[1], Ludwing Marenco[1], Mardônio F. J. C. França[3]

[1]Federal University of Ceará (UFC), [2]State University of Ceará (UECE),[3]Casa Magalhães

boitatá
github.com/boitatabrazil

resultados iniciais | afm | machine learning

Introduction

The Atomic Force Microscopy (AFM) is a technique that uses an extremely thin tip to scan the surface of the sample. The main objective is to obtain a nano-scale resolution topographic image. Figure 1 exposes an schematic view of an AFM set up. This technique is employed to investigate the mechanical and physic-chemical properties of samples like adhesiveness and mechanical resistance.

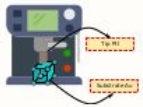


Figure 1: Schematic set up of an Atomic Force Microscope. Using a thin tip, it is possible to construct nano-scale image of a sample using electrical contacts.

The Minimum Spanning Tree (MST) is an optimization algorithm that generates a subnetwork containing all connected vertices constrained to the sum of their weights must be minimum. Figure 2 exposes an schematic view of networks associated with ordering.

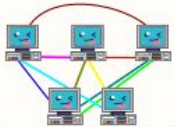


Figure 2: A pictorial representation of the MST. For a complete connected network of computers, the MST is the subnetwork that connects all computers each other by using the less quantity of wire.

Metodology

AFM data acquirement

The data was acquire by measuring current, voltage and force in an specific point where tip contacts the substrate. Figure 3 shows an squematic view of data acquisition.

Time (s)	Voltage (mV)	Current (mA)	Force (mN)
0.5	10	1.9	0.5
1.0	15	-1.9	0.5
1.5	20	1.9	0.5

Figure 3: An example of the AFM data acquirement. Time, voltage, current and force were measured in an specific point.

Correlation matrices ordering via MST

The data was normalized using min-max scaling condition

$$X_{MS} = \frac{X - X_{min}}{X_{max} - X_{min}}, \qquad (1)$$

Results

After that, we computed the correlation distance matrix

$$d(A,B) = \sqrt{2 \cdot [1 - C(A,B)]} \qquad (3)$$

where $C(A,B)$ is the correlation matrix (2) in order to create a complete undirected network. In this network, vertices are associated to each time step of AFM data. Prim's algorithm was used to find the Minimal Spanning Tree. The re-ordering of correlation matrices is made by shuffling each element of each matrix from the highest and lowest correlated vertices [1].

We assume that the contact between tip and substrate is ohmic because of like metal-metal interaction. Figure 6 shows the time evolution of current in ideal and noised measures.

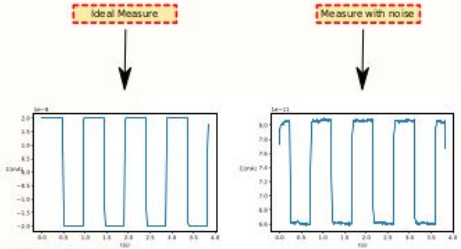


Figure 6: Time evolution of current of ideal and noised AFM measurement.

The ordered MST organized matrices are exposes in Figure 7. In the matrices, reddish colours are associated to negative values of correlation and blueish colors represent positive values. Observe the two well-correlated clusters.

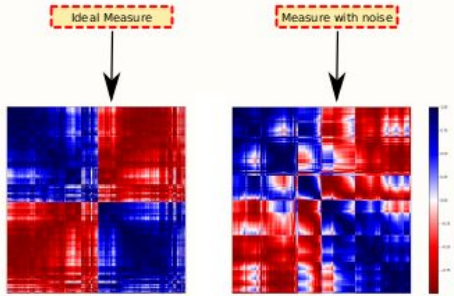


Figure 7: Ordered correlation matrices by MST-Prim algorithm for ideal and noised measures.

Conclusion

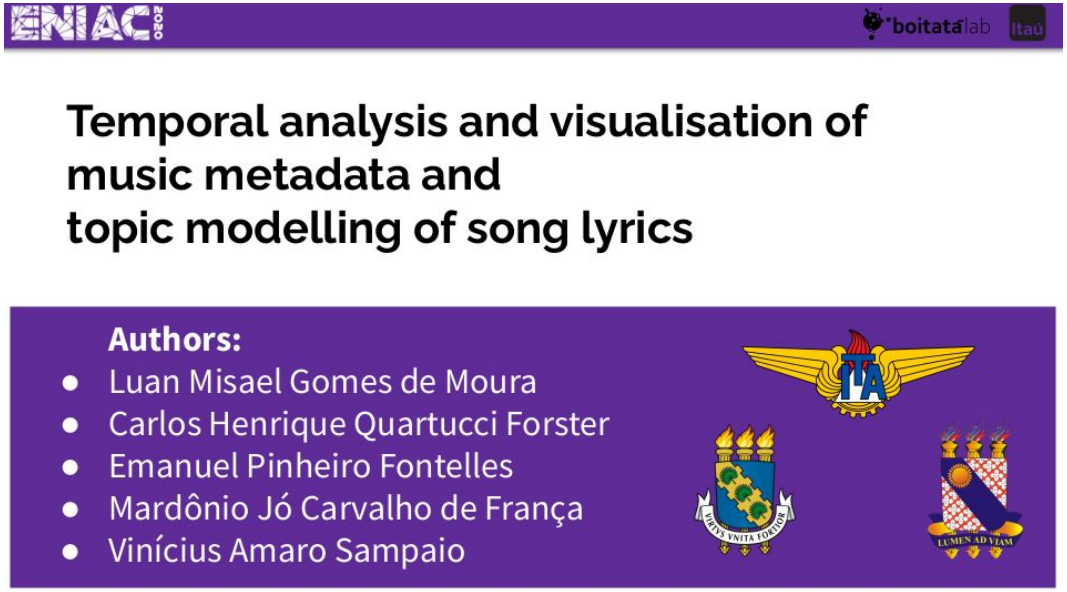
The ordered Prim-MST correlation matrices allow us to identify two well-correlated clusters both, in ideal and noised measures. These clusters correspond to positive and negative states of current among tip and substrate contact.

Reference

[1] Ludwing Marenco, Humberto A. Carmona, Felipe M. Cardoso, José S. Andrade Jr., and Carlos Lenz Cesar. Time evolution of brazilian legislative representatives' behaviour: Polarization as a proxy of political stability, 2019.

resultados iniciais | machine learning

boitatá lab | publicação



conclusão

conclusão | machine learning

Nesse momento o nosso enfoque é implementar as seguintes atividades:

- calibrar os algoritmos para as pontas, lentes e materiais que disponibilizamos no projeto.
- detectar se o material é n-day, qual n?;
- facilitar antecipar possíveis falhas no processo de análise e leitura dos resultados iniciais;
- previamente detectar onde tem mancha na lente, se a ponta se degradou, ou se o material está com problema;

Temos como objetivo de criar filtros/lentes computacionais com ajuda de técnicas de aprendizagem de máquinas para ter um melhor resultado no AFM.

Este trabalho será apresentando em minha monografia como conclusão de curso de Bacharelado de Física. É realizado em parceria com o Grupo Central Analítica - Física / Universidade Federal do Ceará e o Grupo de Estudo em Ciência de Dados Boitatá LAB.

referências

referências

https://www.preparaenem.com/quimica/microscopio-forca-atomica-afm.htm | Accessed 13 April 2021.

https://pt.wikipedia.org/wiki/Alan_Turing | Accessed 13 April 2021.

https://www.turing.org.uk/publications/dnb.html | Accessed 13 April 2021.

https://www.nybooks.com/daily/2014/12/19/poor-imitation-alan-turing/ | Accessed 13 April 2021.

https://en.wikipedia.org/wiki/Transistor | Accessed 13 April 2021.

Riordan, Michael. "Transistor". Encyclopedia Britannica, 26 Mar. 2020, https://www.britannica.com/technology/transistor. | Accessed 13 April 2021.

https://towardsdatascience.com/implementing-the-xor-gate-using-backpropagation-in-neural-networks-c1f255b4f20d | Accessed 13 April 2021.

https://www.analyticsvidhya.com/blog/2020/07/neural-networks-from-scratch-in-python-and-r/ | Accessed 13 April 2021.

https://www.bogotobogo.com/python/python_Neural_Networks_Backpropagation_for_XOR_using_one_hidden_layer.php | Accessed 13 April 2021.

https://svitla.com/blog/how-to-build-a-neural-network-on-tensorflow-for-xor | Accessed 13 April 2021.

COMPUTING MACHINERY AND INTELLIGENCE - A. M. TURING | Mind, Volume LIX, Issue 236, October 1950, Pages 433–460, https://doi.org/10.1093/mind/LIX.236.433

referências

Temporal Analysis and Visualisation of Music - Bracis 2020

A Brief Survey of Deep Learning based methods, against OpenNLP NameFinder for Named Entity Recognition on Portuguese Literary Texts - STIL 2019

Atomic Force Microscopy (AFM) feature extraction and fault diagnosis using Clustering Algorithm based on Minimum Spanning Tree (MST) - xxxiv-encontro-de-fisicos-do-norte-e-nordeste

Thank you !