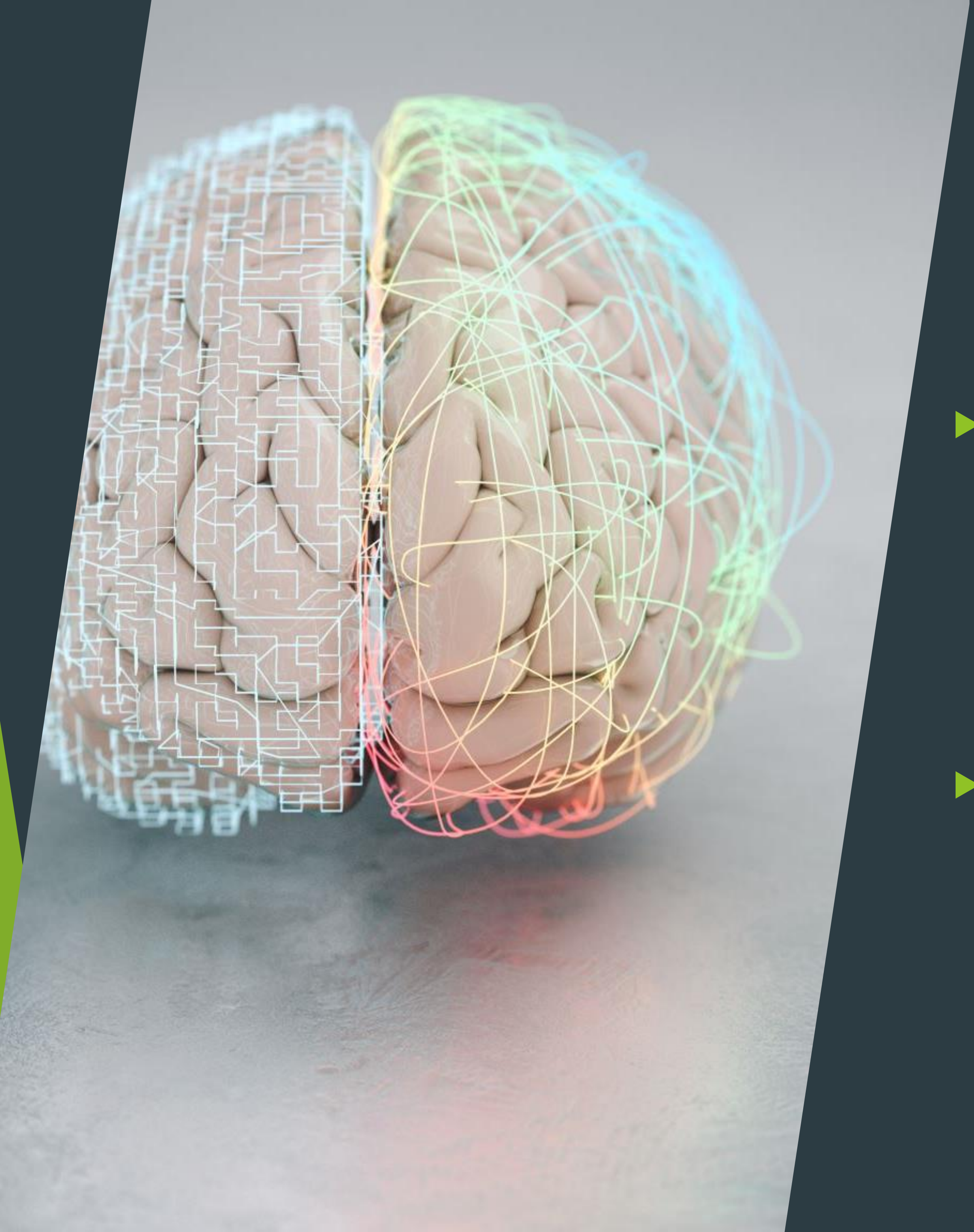
# Artificial intelligence

Heiderich Valentin

2023

# Content

- The Theory
  - The Brain
  - Neural Networks in programming
- Social Aspects and Politics
  - AI Act



# The Brain

- Neurons:

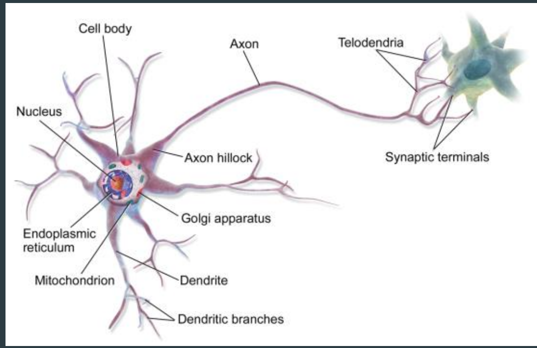
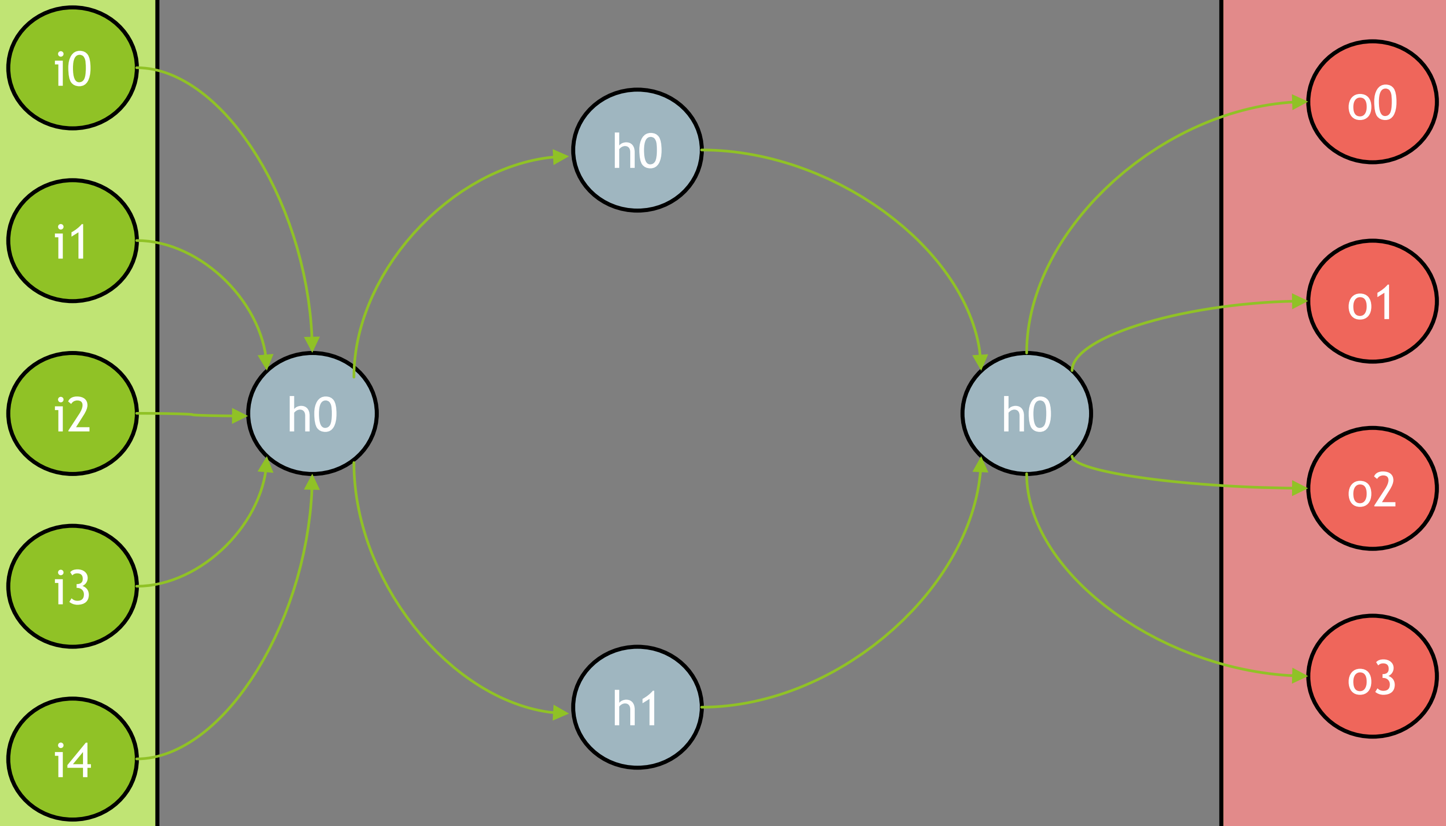



Image URL

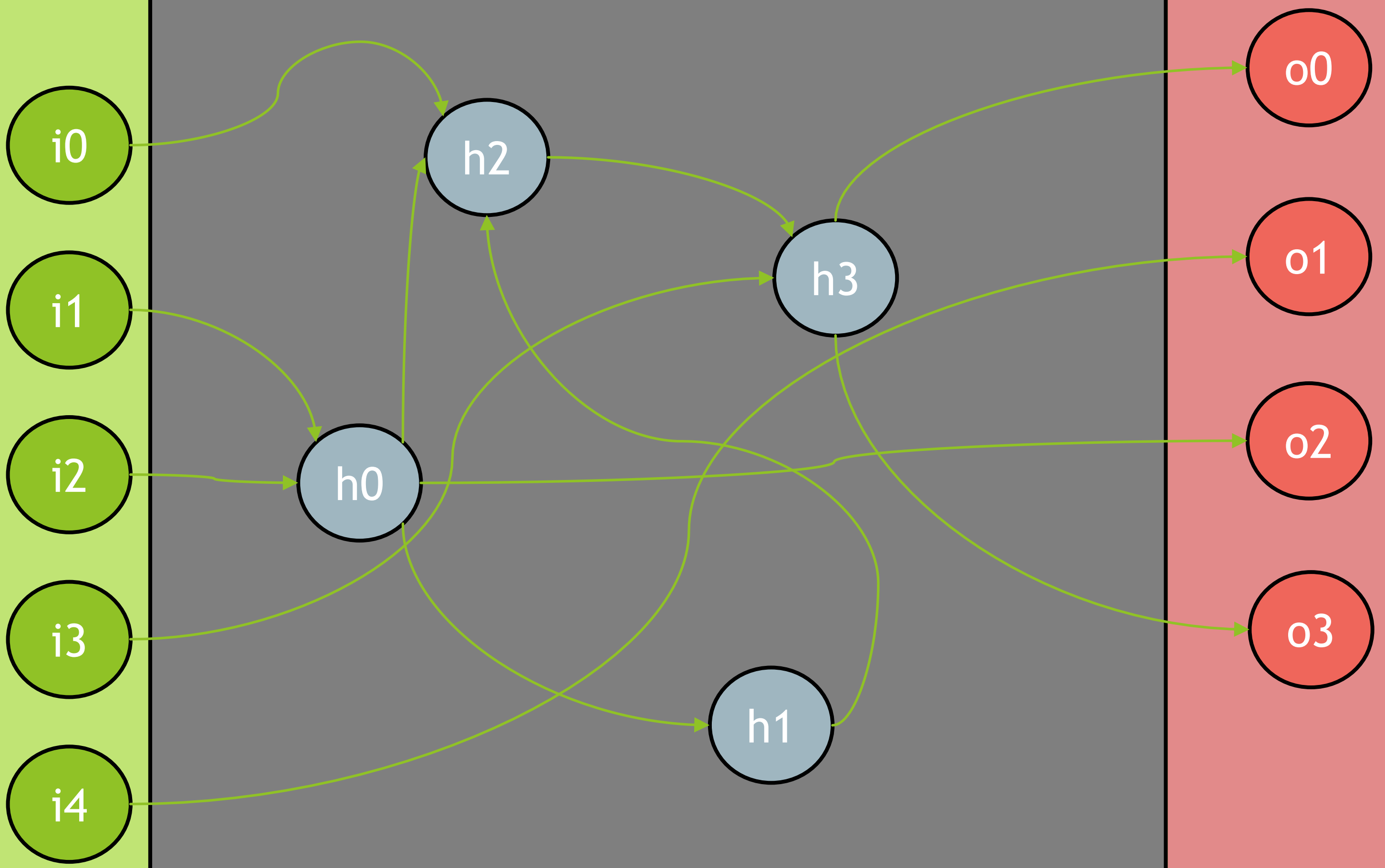
- Neural Networks:

Image URL

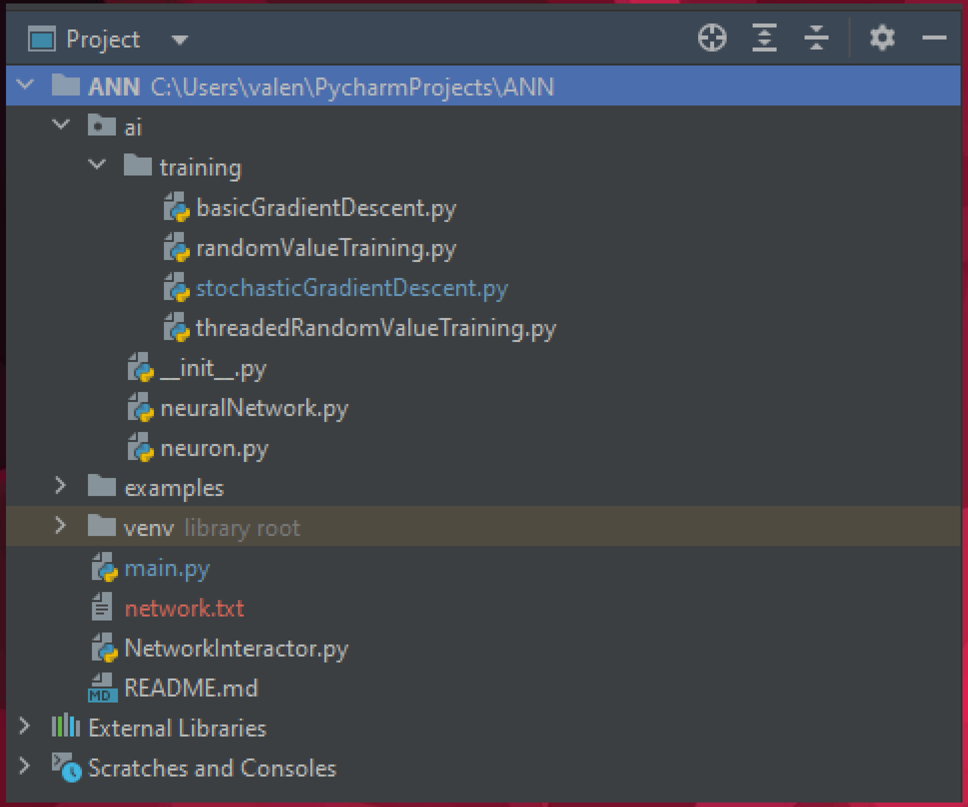
# Neural Networks in programming



# Neural Networks in programming




https://www.youtube.com/watch?v=N3tRFayqVtk



Project
ANN C:\Users\valen\PycharmProjects\ANN
ai
training
basicGradientDescent.py
randomValueTraining.py
stochasticGradientDescent.py
threadedRandomValueTraining.py
__init__.py
neuralNetwork.py
neuron.py
examples
venv library root
main.py
network.txt
NetworkInteractor.py
README.md
External Libraries
Scratches and Consoles

```python
import random
import math


1 usage  Valentin *
def sigmoid(output):                              # expanded activation function
    try: return 1 / (1 + math.exp(-output))
    except OverflowError: return 0


10 usages  vh64g *
class neuron:
    vh64g *
    def __init__(self):
        self.weights = None  #all weights connected to the specific neuron
        self.costGradientWeights = [] # for gd training algorithms
        self.costGradientBias = None # for gd training algorithms
        self.bias = 1 # bias added to summed value
        self.output = None # init output

    5 usages (5 dynamic)   new *
    def randomize(self, input_count): # create random weights and a random bias, input count must equal len(cn)
        new_weights = [] # init nw var
        for i in range(input_count): new_weights.append(random.uniform(-10, 10)) # create a random weight (type: float) for each connection (between -10 and 10)
        self.bias = random.uniform(-10, 10) # create random bias (between -10 and 10)
        self.weights = new_weights # push weights

    11 usages (11 dynamic)   new *
    def calc(self, inputs): # calculating the output weight based on some input list len(inputs) must equal len(weights)!
        self.output = 0 # clearing output var
        for i in range(len(inputs)): self.output += inputs[i] * self.weights[i] # sum products of each input value multiplied with the corresponding weight
        self.output += self.bias # adding bias value
        self.output = sigmoid(self.output) # using a sigmoid func on the output for clamping
        return self.output # return output
```

```python
class artificialNeuralNetwork:
    vh64g *
    def __init__(self, input_layer, output_layer, hidden_layers=None):

        self.input_layer = input_layer  # structure of input layer: [neuron1, neuron2, ...]
        self.hidden_layers = hidden_layers  # structure of hidden layers: [[layer1_neurons], [layer2: neurons, ...]
        self.output_layer = output_layer  # structure of output layer: [neuron1, neuron2, ...]

        if hidden_layers is None: self.hidden_layers = [] # avoid null pointer exceptions

        self.out = [] # init output value

        self.randomize()

    6 usages (5 dynamic)  vh64g +1 *
    def randomize(self):
        for layer in self.hidden_layers: # call the randomize function of each neuron in each hidden layer and the output layer, passing the len(inputs)
            if self.hidden_layers.index(layer) == 0:
                for neuron in self.hidden_layers[self.hidden_layers.index(layer)]:
                    neuron.randomize(len(self.input_layer))
            else:
                for neuron in self.hidden_layers[self.hidden_layers.index(layer)]:
                    neuron.randomize(len(self.hidden_layers[self.hidden_layers.index(layer) - 1]))
        for neuron in self.output_layer:
            neuron.randomize(len(self.hidden_layers[-1]))

    11 usages (11 dynamic)  vh64g *
    def calc(self, inputs): # calculating the output of the entire neural network
        self.input_layer = inputs # getting inputs
        self.out = [] # init output: type: list
        for layer in self.hidden_layers: # loop through each hidden layer
            for neuron in self.hidden_layers[self.hidden_layers.index(layer)]: # loop through each neuron in hidden layer
                if self.hidden_layers.index(layer) == 0: x = neuron.calc(self.input_layer) # calculating each neuron output of h0 based on input values
                else: x = neuron.calc([neuron.output for neuron in self.hidden_layers[self.hidden_layers.index(layer) - 1]])
                # calculating the other neurons, using the output values of the last layer
        for neuron in self.output_layer:
            middle_outs = [neuron.output for neuron in self.hidden_layers[-1]] # calculating the output for the output layer
            self.out.append(abs(neuron.calc(middle_outs))) # push output
        return self.out # return output
```

# AI Act

Svenja Hahn MEP

Some text ‘bout the AI Act

https://fdphamburg.de/sites/default/files/styles/uv_full_content_large_16_9/public/2023-02/Svenja%20Hahn%20EP.jpg?itok=elfRan2t

https://www.europarl.europa.eu/meps/de/197444/SVENJA_HAHN/home