Understanding AI: From Perceptrons to Neural Networks

Coen

Table of contents

Chapter 1

index

Welcome to this comprehensive guide on artificial intelligence, focusing on the journey from perceptrons to neural networks. This book is designed to provide you with a solid foundation in AI concepts, starting from the basic building blocks and progressing to more complex neural network architectures.

1.1 Book Structure

This book is organized into several main sections:

1. **AI Overview**: A broad introduction to artificial intelligence and its key concepts
2. **Perceptron Fundamentals**: Understanding the basic building block of neural networks
3. **Neural Networks**: Exploring more complex architectures and their applications
4. **Next Steps**: Practical guidance for further learning and development

Each chapter builds upon the previous ones, providing a structured learning path through the material.

1.2 Prerequisites

While this book is designed to be accessible to beginners, basic knowledge of mathematics and programming concepts will be helpful. Python examples are used throughout the book to demonstrate practical implementations.

1.3 How to Use This Book

We recommend reading the chapters in order, as concepts build upon each other. Code examples are provided to help reinforce the theoretical concepts with practical implementations.

> **Note**
>
> This book is part of a workshop series on AI and machine learning, with a focus on practical understanding and implementation.

Chapter 2

AI Overview

This video gives a 10 minute overview of the AI Field: How do terms like ***AI***, ***Machine Learning***, ***Deep Learning*** and ***Generative AI*** relate to each other?

You can watch the video "AI, Machine Learning, Deep Learning and Generative AI Explained" at: https://www.youtube.com/watch?v=qYNweeDHiyU

Part I

Perceptron Fundamentals

Chapter 3

Perceptron

3.1 The idea behind a Perceptron comes from a neuron in a brain

In the next example a Perceptron is introduced. A Perceptron is a way of creating an artificial 'simulation' of the way a human brain works. In 1943 Warren S. McCulloch and Walter Pitts expressed the idea in an article 'A Logical Calculus of the Ideas Immanent in Nervous Activity'.

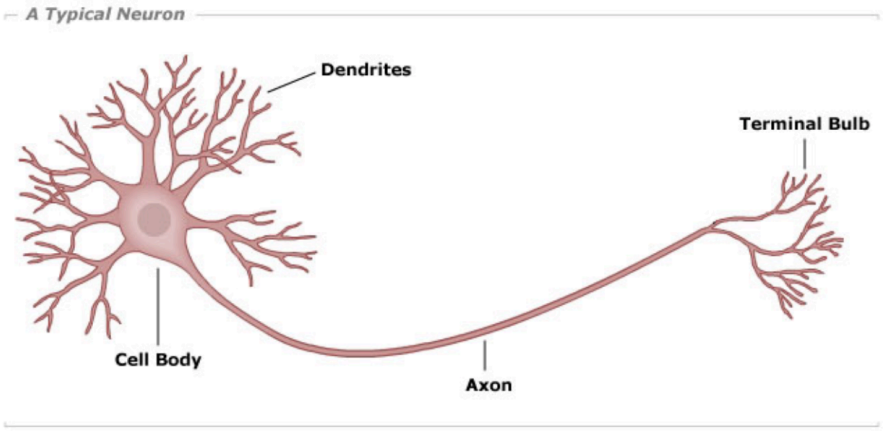


Figure 3.1: A typical neuron

3.2 Implementing this in a machine

The idea behind a Perceptron is that it has a number of `inputs`, each input having its own `weight`, furthermore it has a `bias`. Every input channel accepts a numerical value, and the weight determines the importance of that input.

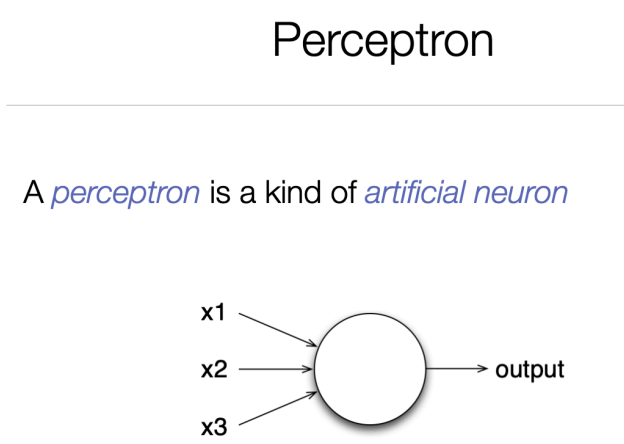


Figure 3.2: Perceptron diagram

This example has 3 inputs: the `feeds` into the inputs are called `x1`, `x2` and `x3`. The corresponding weights are called `w1`, `w2` and `w3`. To calculate the output of the Perceptron first the `'weighted sum'` is calculated, then the bias is added to that sum:

```
z := w1*x1 + w2*x2 + w3*x3 + bias
```

After that it is **`activated`** (or: **`the activation function`** is calculated), for which in this example we take the formula: - if value of z is above 0, output will be 1 - otherwise output will be 0

In code the 'z' can be calculated by:

```
| x1 x2 x3 w1 w2 w3 bias |
"When a neuron is `fired` that means there are `input` values.
The input values are called x1 until x3 here:"
x1 := 0.35 .
x2 := 1.2 .
x3 := 0.54 .
```

```
"The weights w1 to w3 are properties of the neuron, as is the bias:"
w1 := 0.234 .
w2 := 0.32 .
w3 := 0.58 .
bias := 5 .
z := (w1 * x1) + (w2 * x2) + (w3 * x3) + bias.
```

After which the **activation** gives:

```
z > 0
    ifTrue: [1]
    ifFalse: [0]
```

Well, that was an example with 3 inputs. Maybe play around a bit with the input values (x1, x2, x3) to get output 0. (inspect the playgrounds again after changing the values)

Now we will take a look at some more generic code that is being executed in this system: (why is this more generic?)

```
| inputs_x weights_w bias |
inputs_x :=  #(0.35  1.2  0.54).
weights_w := #(0.234 0.32 0.58).
bias := 5.
z := (inputs_x
    with: weights_w
    collect: [ :x :w | x * w ])
        sum
            + bias
```

Answer: This is more generic because you can add a fourth (and fifth, and…) value to `inputs_x` and `weights_w`: the code will still work, as long as the number of input values is equal to the number of weights.

The code used by your machine (your laptop running GToolkit with this workshop) is like this:

{{gtMethod:Neuron»feed:|expanded=true}}

For a Neuron the so-called activationFunction is a 'step' function: If `z>0` the output of the Perceptron will be `1`, otherwise `0`, which you can see coded in method 'eval:' in class 'StepAF':

{{gtMethod:StepAF»eval:|expanded=true}}

The '^' means that the method will 'return' that value.

Chapter 4

applying-the-perceptron

Title: Applying the Perceptron: AND/OR #Applying the Perceptron: AND/OR

Now what can we use that for? Well, look at the code below: it creates a Perceptron with weights `w1 := 1`, `w2 := 1`, put together as an array: `#( 1 1 )` and `bias := -1.5`.

```
"Perceptron with weights 1, 1 with bias -1.5"
perceptron := Neuron new
        step;
        weights: #(1 1);
        bias: -1.5.
```

You can look at the tabs to see multiple ways of looking at the Perceptron. Not all tabs will mean something to you at this moment. Next step is to see what the `output` will be at given `inputs`:

```
perceptron fire: #(1 1)
```

You can change the values in the array. Try all of these: #(0 0), #(0 1), #(1 0), #(1 1) and you will find that this perceptron has **ONLY output 1 if both inputs are 1** . That means it behaves as an **AND** logical gate. Looking at the weights and bias, do you understand that? If you look at the weights and bias that seems to be right: rewrite the formula for `z` for 2 inputs, filling in the known numbers, gives: `z := 1*x1 + 1*x2 - 1.5` If you calculate the `z` for all 4 input combinations you see that if either `x1` or `x2` (or both) equals `0` than the sum of them will be less then `1.5`, which means the perceptron will output `0`. Only when `x1` and `x2` both have value `1` will the value of `z` be greater than '0', so only then will the perceptron output `1`. Well, programmers like to test their code automatically. Here you see a method that checks all these 4 cases: first a perceptron is created, then 'assert:equals:' is called 4 times with the 4

inputs and corresponding outputs. Let's look at the first assert: `self assert: (perceptron feed: #(0 0)) equals: 0.` In this line the perceptron is fed two zero's: if the result is zero nothing special happens (inspecting shows a green thingy after the method name and shows a 'FiredNeuronShot'), but if something else comes out the program will warn you: the inspected result is 'nil' and after the method name you see an orange thingy . You can change the values to try that if you want. {{gtExample:WorkshopAIPerceptronGT»testANDGate}} First a variable `p` is created (by putting it between vertical bars), then it gets assigned a perceptron that acts as an AND gate, by calling the method we saw before. Then a message `'assert:equals:'` is sent (to self) to check that if you `FEED` the perceptron the input `0` and `0` , it will output `0` (that's the first test, figure out whether you agree to all 4 tests). Just like it is possible to configure a perceptron as an AND gate it is also possible to configure one as an OR gate (output is 1 if at least one of the inputs is 1), a NOR gate (only 1 if the inputs are 0) or a NOT gate (input 0 gives output 1 and input 1 gives output 0). Which of the following is which, and can you complete the code for the missing one?

```
perceptron := Neuron new
        step;
        weights: #(-1 -1);
        bias: 0.5 .
```

```
perceptron := Neuron new
        step;
        weights: #( -1 );
        bias: 0.5 .
```

```
perceptron := ??? "can you complete this code for the missing third?"
```

Chapter 5

Decision-making

Title: Decision making: Metal Concert #Decision making: Metal Concert

A perceptron can be used in decision making. Suppose there is a Metal concert and you may want to go, but it depends on 3 parameters/inputs: - x1: Is the weather good? - x2: Do you have company to go with you? - x3: Can you get there easy using public transport? Using 0 for false, 1 for true. When you prioritize going together you give a heigh weight to the company, w2 := 6 for example:

```
perceptron := Neuron new step;
    weights: #( 3 5 2 );
    bias: -4.5
```

Good weather, no company, difficult by public transport: would I go?

```
perceptron fire: #(1 0 0)
```

Bad weather, good company, difficult by public transport: would I go?

```
perceptron fire: #(0 1 0)
```

Good weather, no company, easy by public transport?

```
perceptron fire: #(1 0 1)
```

Chapter 6

The-learning-Perceptron

Title: The Learning Perceptron #The Learning Perceptron

The Perceptron can ***learn*** when we train it: ***Training*** means we give input to the perceptron **and** we also tell it what the outputvalue is are that we want for those input values. Let's look at an example: - First we create a perceptron, 2 inputs with weights -1 and -1, and a bias of 2.
- For input values `1` and `1`, written as: `#( 1 1 )` this Perceptron will give an output value of 0 initially. Check this by `inspecting` the next 2 scripts:

```
perceptron := Neuron new step.
perceptron weights: #( -1 -1 ).
perceptron bias: 2.
^ perceptron
```

Set the input values to `#( 1 1 )` respectively, and see what the output value is.

```
^ perceptron fire: #( 1 1 )
```

Training: Now we are going to tell that for inputs `#( 1 1 )` the desired output is 1. The next script is one trainings step:

```
perceptron train: #( 1 1 ) desiredOutput: 1
```

If you look at the We call this ***labeled data***: We have inputs (***data***) and tell the machine what the output should be for for those inputs (***the desired output*** or ***label***).

```
10 timesRepeat: [ perceptron train: #( 0 1 ) desiredOutput: 0 ].
^ perceptron
```

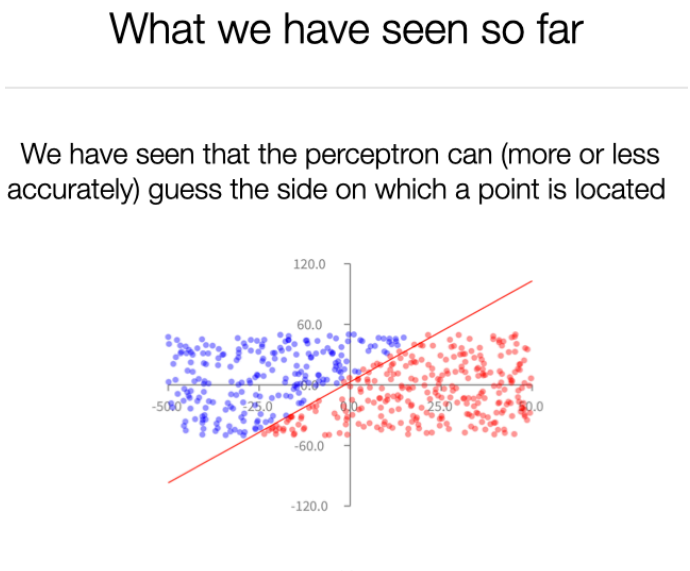
```
"After training the input #(0 1) will give output 0"
^ perceptron fire: #( 0 1 )
"If you are curious as to what method 'train:desiredOutput:' does:
```

```
click at the little triangle behind 'desiredOutput:' in the third script.
(the triangle does only appear AFTER you executed the first script)"
```

Chapter 7

A problem with the Perceptron

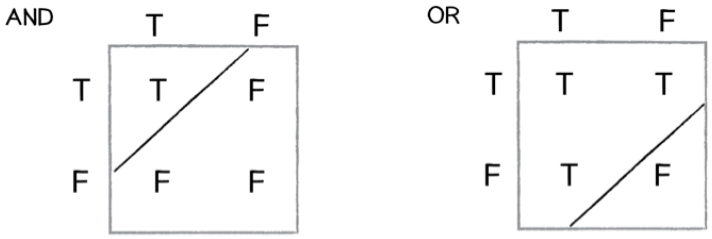
Title: A problem with the Perceptron # A problem with the Perceptron



- So the Perceptron was able to learn to classify on which side of a straight line dots were, only by training it with a lot of points and telling them which side of the line they should be. Looking back at a different example: a Perceptron as a AND and OR port:

What we have seen so far

We can easily make our perceptron to represent the AND, OR logical operations

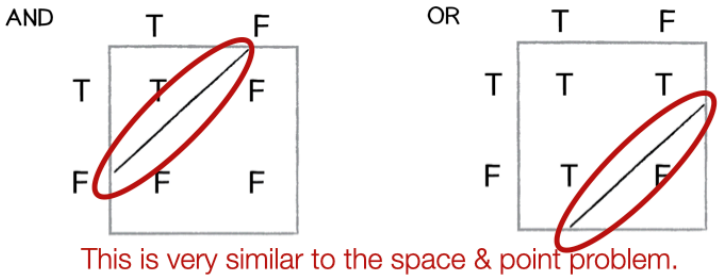


38

- Wait!? This picture also shows a straight line!

What we have seen so far

We can easily make our perceptron to represent the AND, OR logical operations

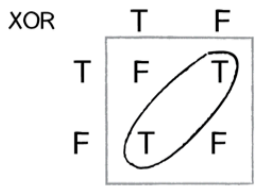


This is very similar to the space & point problem.
It is all about having a line as a limit

39

- Instead of 2 colors we see 2 different values (T and F).

Limitation of a perceptron



With the XOR operation, you cannot have one unique line that limit the range of true and false

- 40

Only 1 Perceptron can not act as an XOR-Gate. For that we need to create a network of Perceptrons. As you may imagine a lot of 'contexts' are more complex than linear.

Part II

Neural Networks

Chapter 8

What is a Neural Network?

Title: Neural Network #Neural Network

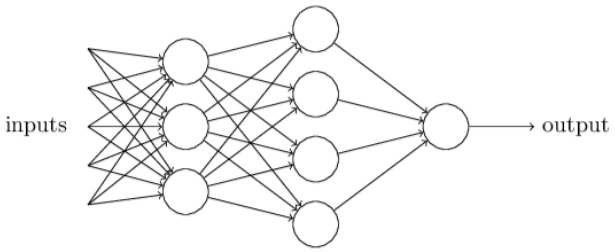
Chapter 9

What is a Neural Network?

Neural Networks, also called ***Artificial Neural Networks (ANNs)*** are computational systems modeled after the biological neural networks found in human and animal brains. Typically, an ANN is composed of multiple layers of neurons, with each neuron in one layer being connected to every neuron in the following layer. This type of structure is often referred to as a ***multilayer perceptron*** or ***MLP***. A neuron represents a single unit that can be either active (firing) or inactive, depending on the activity of neurons in the previous layer, the strength (or weight) of the connections, and a bias factor. -

Network of neurons

A network has the following structure



41

Back to a case that was impossible with a Perceptron Example network and training it on getting this behavior: Inputs 0 and 0 must give output 0. Inputs 0 and 1 must give output 1. Inputs 1 and 0 must give output 1. Inputs 1 and 1 must give output 0. In the following video, you'll see an explanation of the layered structure of neural networks and how neurons are activated, illustrated through a classification problem. The video provides an intuitive rationale for why neural networks are often made up of multiple layers. The layers between the input and output layers are referred to as hidden layers. Toward the end, the video also clarifies the difference between a sigmoid neuron and a rectified linear unit (ReLU) neuron.

```
network := NNetwork new.
network configure: 2 hidden: 3 nbOfOutputs: 1.

20000 timesRepeat: [
    network train: #(0 0) desiredOutputs: #(0).
    network train: #(0 1) desiredOutputs: #(1).
    network train: #(1 0) desiredOutputs: #(1).
    network train: #(1 1) desiredOutputs: #(0).
].
^ network
```

Check if it does what it should by inspecting the result of:

```
network feed: #(0 0)
network feed: #(0 1)
network feed: #(1 0)
network feed: #(1 1)
```

Chapter 10

ANN inexplained a video

We all know how challenging it can be to read someone else's handwriting. Yet, despite variations in handwriting style, we are generally quite successful at recognizing letters or digits. For example, the human brain is remarkably adept at identifying different representations of the number "5."

But what is a neural network? | Deep learning chapter 1

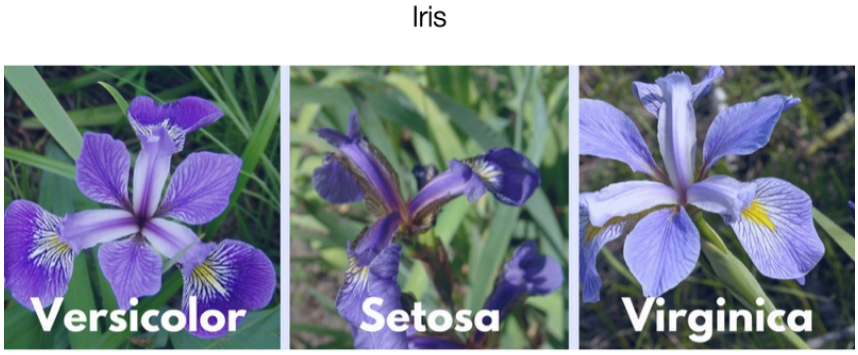
Are the results correct?

Chapter 11

Step 1 - Read data

Title: Classifying with the Iris dataset #Classifying with the Iris dataset

A famous example: The Iris dataset case



- 44

Using a *machine* to classify what type of iris we have data from. Dataset came from here. Some more info.

Chapter 12

Step 1 - Read data

```
- In the 'iris.csv' file there is data about 150 flowers of three types.
- ```Pharo
```

"Step 1." "Download Iris csv from url" irisCSV := (ZnEasy get: 'https://agileartificialintelligence.github.io/Datasets/i
contents.

```
# Step 2 - Convert iris types names to numbers
    - ```Pharo
"Step 2."
lines := irisCSV lines allButFirst. "Initialize the variable irisData."
tLines := lines
        collect: [ :line |
            | valuesOnThisLine |
            valuesOnThisLine := line substrings: '',''.
            (valuesOnThisLine allButLast collect: [ :w | w asNumber ])
                , (Array with: valuesOnThisLine last) ].

irisData := tLines
        collect: [ :row |
            | l |
            row last = 'setosa' ifTrue: [ l := #(0) ].
            row last = 'versicolor' ifTrue: [ l := #(1) ].
            row last = 'virginica' ifTrue: [ l := #(2) ].
            row allButLast , l ].
^ irisData
```

Chapter 13

Step 3 - Create Neural Network

```
- ```Pharo
```

“Step 3.” network := NNetwork new. network configure: 4 hidden: 6 nbOfOutputs: 3. network learningRate: 0.3 . ^ network

```
# Step 4 - Train the network
    - ```Pharo
"Step 4."
 " Repeat the script "
network train: irisData nbEpochs: 1000.
^ network
```

Chapter 14

Step 5 - Try

- ```Pharo

“Step 5.” network feed: #(5.6 2.9 3.6 1.3)

`<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4ifQ== -->`{=html}

```{=html}
<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4iLCJib29rSXRlbVR5cGUiOiJjaGFwdGVyIiwiYm9va0l0ZW
```

Chapter 15

Magic-behind-NNetwork

Title: The 'magic' behind a Neural Network #The 'magic' behind a Neural Network

We saw that a **Neural Network** (let's shorten it to **NNetwork**) is a combination of **Neuron**s. In a previous chapter we saw how a Neuron can 'learn'. When combining Neurons in a Neural Network we want the network also be able to learn. You may be able to imagine that that is more complicated. ## How does 'learning' work in a NNetwork? - Well, it is based on an algorithm called 'Backpropagation'. The background is mathematical and made up by very smart people, **although** if you did HAVO (for the Netherlands) or another secondary school in another country, you *have* learned all mathematics you need to understand backpropagation! - To **use** NNetworks you don't need to know exactly how backpropagation works, but below you will find a really nice video that clearly explains backpropagation. - Previously we learned that the number of input neurons corresponds to the dataset being used (in this case, 28 x 28 = 784), while the number of output neurons matches the number of classes (in this case, 10). Additionally, two hidden layers, each containing 16 neurons, were chosen. The number of hidden layers and neurons per layer were selected arbitrarily. - Once the structure of the neural network and the activation function for each neuron (e.g., sigmoid or ReLU) are determined, the next step is to assign values to the weights of the connections between neurons and the biases of each neuron. This results in a large number of parameters to assign (about 13,000 in this case). This process happens automatically during training. - The goal of training a neural network is to adjust the weights and biases so that, when a specific input is provided, the expected output neuron will activate (while the others remain inactive). Training involves feeding the network with many examples from the dataset, along with their corresponding expected outputs. As the network processes each example, the weights and biases are updated based on a learning rule. - In the next video, the concept of gradient descent is introduced, which is the algorithm used to train neural networks. It

also discusses the importance of splitting the dataset into a training set and a test set. The training set is used to train the network, while the test set is used to evaluate how well the network has learned by using a cost function. A preview of the next module on Convolutional Neural Networks is also provided.

The Most Important Algorithm in Machine Learning

Chapter 16

Wanna know more on neural networks?

Title: More on Training and Learning #More on Training and Learning

There is a lot to learn about the algorhithms involved in making a NNetwork able to ***learn***. We learned that the number of input neurons corresponds to the dataset being used (in this case, 28 x 28 = 784), while the number of output neurons matches the number of classes (in this case, 10). Additionally, two hidden layers, each containing 16 neurons, were chosen. The number of hidden layers and neurons per layer were selected arbitrarily. Once the structure of the neural network and the activation function for each neuron (e.g., sigmoid or ReLU) are determined, the next step is to assign values to the weights of the connections between neurons and the biases of each neuron. This results in a large number of parameters to assign (about 13,000 in this case). This process happens automatically during training. The goal of training a neural network is to adjust the weights and biases so that, when a specific input is provided, the expected output neuron will activate (while the others remain inactive). Training involves feeding the network with many examples from the dataset, along with their corresponding expected outputs. As the network processes each example, the weights and biases are updated based on a learning rule. In the next video, the concept of gradient descent is introduced, which is the algorithm used to train neural networks. It also discusses the importance of splitting the dataset into a training set and a test set. The training set is used to train the network, while the test set is used to evaluate how well the network has learned by using a cost function. A preview of the next module on Convolutional Neural Networks is also provided.

Gradient descent, how neural networks learn | DL2

The core algorithm how neural networks learn is backpropagation. An intuitive

explanation of the backpropagation algorithm is presented in the next video. In addition, it is explained how the training set is divided into so-called mini-batches to speed-up the algorithm.

Backpropagation, step-by-step | DL3

Backpropagation calculus | DL4

16.1 Multivariable calculus

In the second video, multivariable calculus was used to explain gradient descent. For the interested student, more on multivariable calculus can be found in the next video from Kahn Academy:

Multivariable functions and multivariable calculus (Khan Academy):

Multivariable functions | Multivariable calculus | Khan Academy

Chapter 17

Wanna know more on neural networks?

In one of the videos by 3Blue1Brown the following freely available digital book was mentioned:

Neural Networks and Deep Learning by Michael Nielsen: neuralnetworksanddeeplearning.com

Chapter 18

Recurrent neural network

Title: Different types of Neural Networks #Different types of Neural Networks

Although the basic idea of a neural network is more or less the same, a lot of people have tried all kinds of variations: We only looked at networks where the outputs of **all** perceptrons/neurons are connnected to the inputs of **all** perceptrons/neurons of the next layer. You can think of networks that do not connect all of these! Also you can think of an output that is leaded back into a previous layer or neuron. As activation function we looked at the **Step** function and at the **Sigmoid**, but there are a lot more options. For visual recognition also some additions were invented. … maybe you have had some ideas/questions while looking and thinking about it: maybe investigate on it…? # Recurrent neural network - attachments/93mdx43cuyk9ozqsnka2rswnb/2021-AI-General.049.png # Convolutional neural network - Succesful in visual recognition - attachments/93mdx45vgrq9ult7mc0ug29re/2021-AI-General.052.png - attachments/93mdx46vvditkr1u9395xa7an/2021-AI-General.050.png - attachments/93mdx42tkirmyu6xep5pninz4/2021-AI-General.051.png

Part III

Next Steps

Chapter 19

OK-Next

Title: OK, now what could I do next? #OK, now what could I do next?

By now you have seen a bit of the idea behind Artificial Neural Networks. Maybe you're appetite if satisfied and you know what you want to know. Maybe you want to dive in the Mathematics behind it. Maybe you want to get hands-on in some programming language. The scripts in this workshop come from the excellent book 'Agile AI in Pharo' by Alexandre Bergel. In the video 'Introduction to neural networks, genetic, and neuroevolution' Alexandre Bergel himself tells the story using Pharo, so there you can find a lot of information: video 'Introduction to neural networks, genetic, and neuroevolution'). After Neural Networks he talks about Genetic Algorithms and in the end about Neuroevolution using NEAT. Or you could follow a tutorial in (for example) Python, Keras, Tensorflow… Coding Lane: Image Classification using CNN in Keras InvesTime: Deep-learning in Health care 3Blue1Brown - But what is a convolution?

Chapter 20

Learning-Python

Title: Learning Python #Learning Python

To really dive into AI at one point you will want to learn how to program. The most-used programming language in the AI Field is Python. Online you can find a lot of courses and tutorials. One that I really like is given by Andrew Ng (a famous AI teacher!) Andrew Ng: python for beginners For this course an online coding environment is available, complete with a chatbot to ask questions about programming!

Chapter 21

Finding-Data

Title: Finding data #Finding data

For training models you need data. Here are some pointers where you can find data: - Eindhoven open data - kaggle - datasetsearch.research.google.com

Chapter 22

Used Sources on AI, and the inspi for this workshop:

Title: References #References

Chapter 23

Used Sources on AI, and the inspi for this workshop:

```
- + [Book: Agile Artificial Intelligence in Pharo, by Alexandre Bergel, 2020](https://link.spring
- + [Video: Introduction to Neural Networks, genetic Algorithm, and neuroevolution](https://tube.
- + [GitHub repository: https://github.com/Apress/agile-ai-in-pharo](https://github.com/Apress/ag
```

on [[Traveling Salesman]], [[Maze Robots]], [[Zoomorphic Creatures]] , [[NeuroEvolution]] , [[concluding]] . # Sources on GToolkit, Pharo and Smalltalk - + Pharo site - + GToolkit site - + 'Glamorous Toolkit Book' in this environment (tab 'gt') - + Books on Pharo