SDU Summer School

Deep Learning

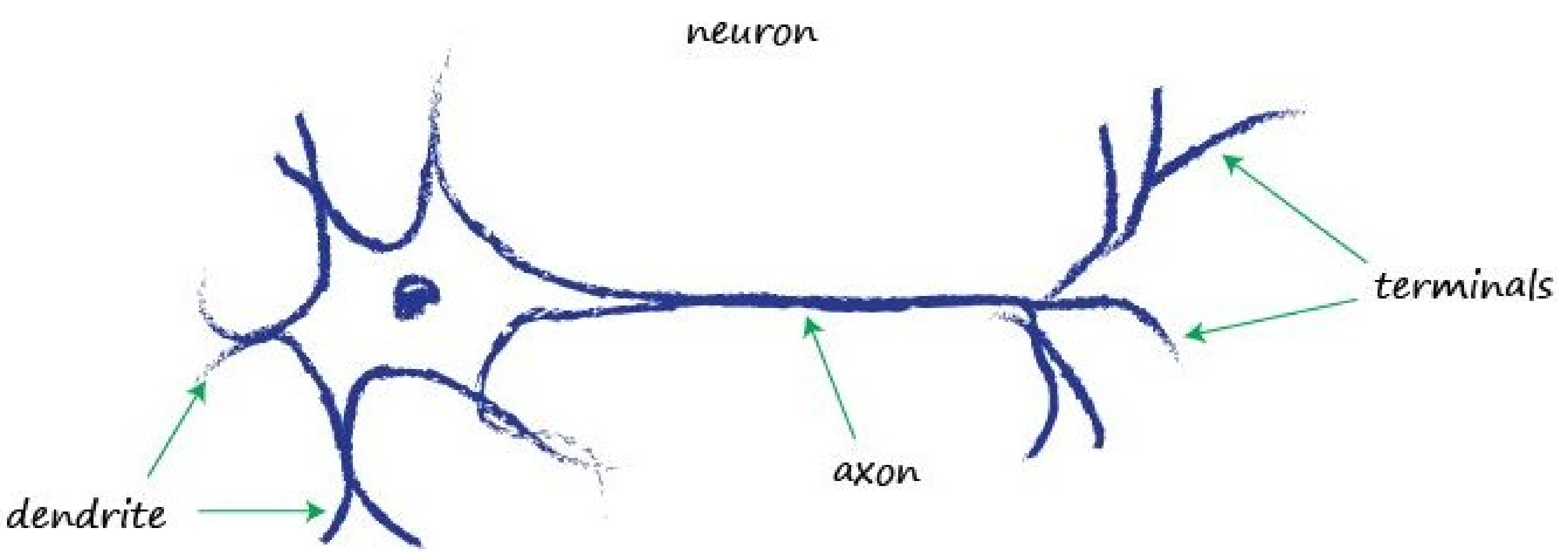
Summer 2022

Welcome to the Summer School

Deep Feedforward Networks Part I

- PART I
  - **What is a Neuron?**
  - How to build a Network
- Intermezzo: Some Math
- PART II
  - Networks
  - Output Units
  - Hidden Units
  - Architecture Design

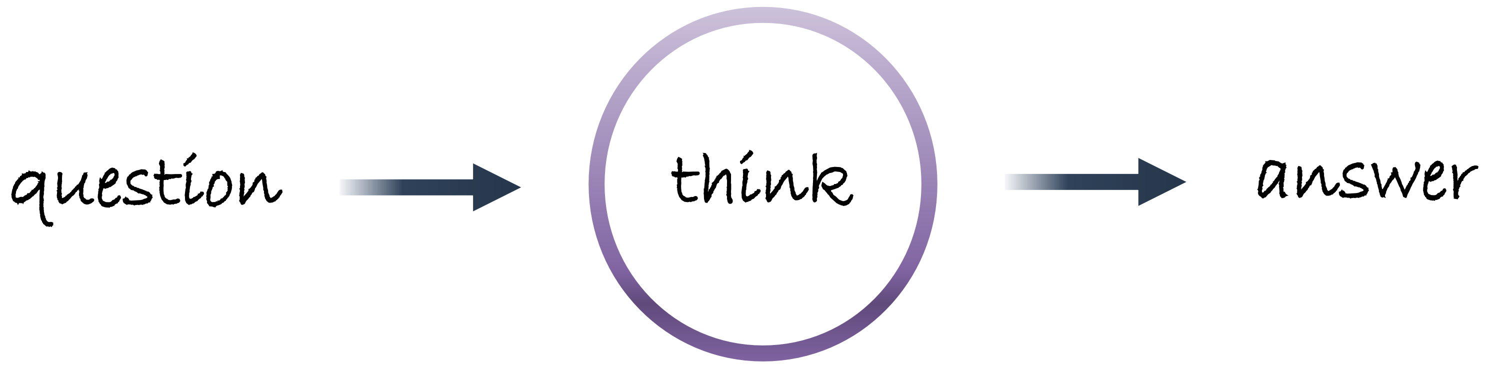
The Neuron … in Nature



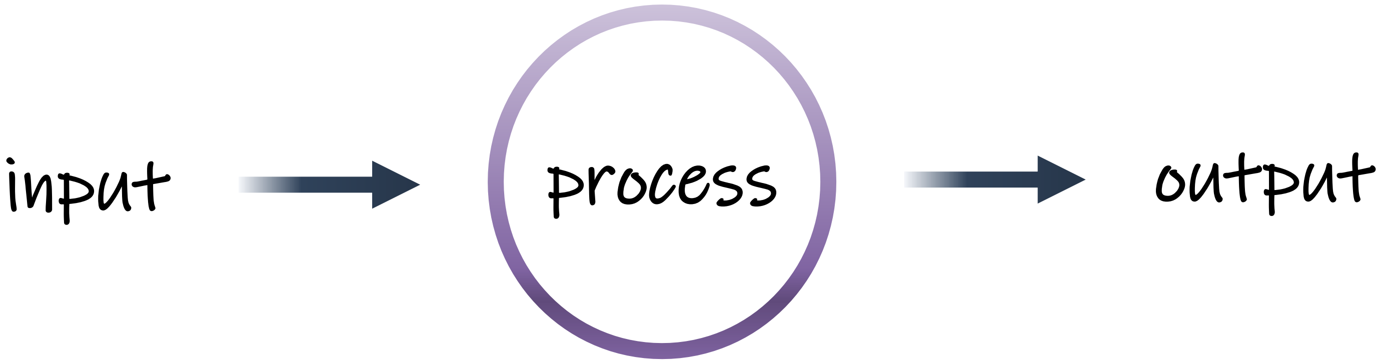
That is nice ... but

- We will use Neurons as basic simple "prediction machines"

- What does that mean?



Example



Example

kilometers

100

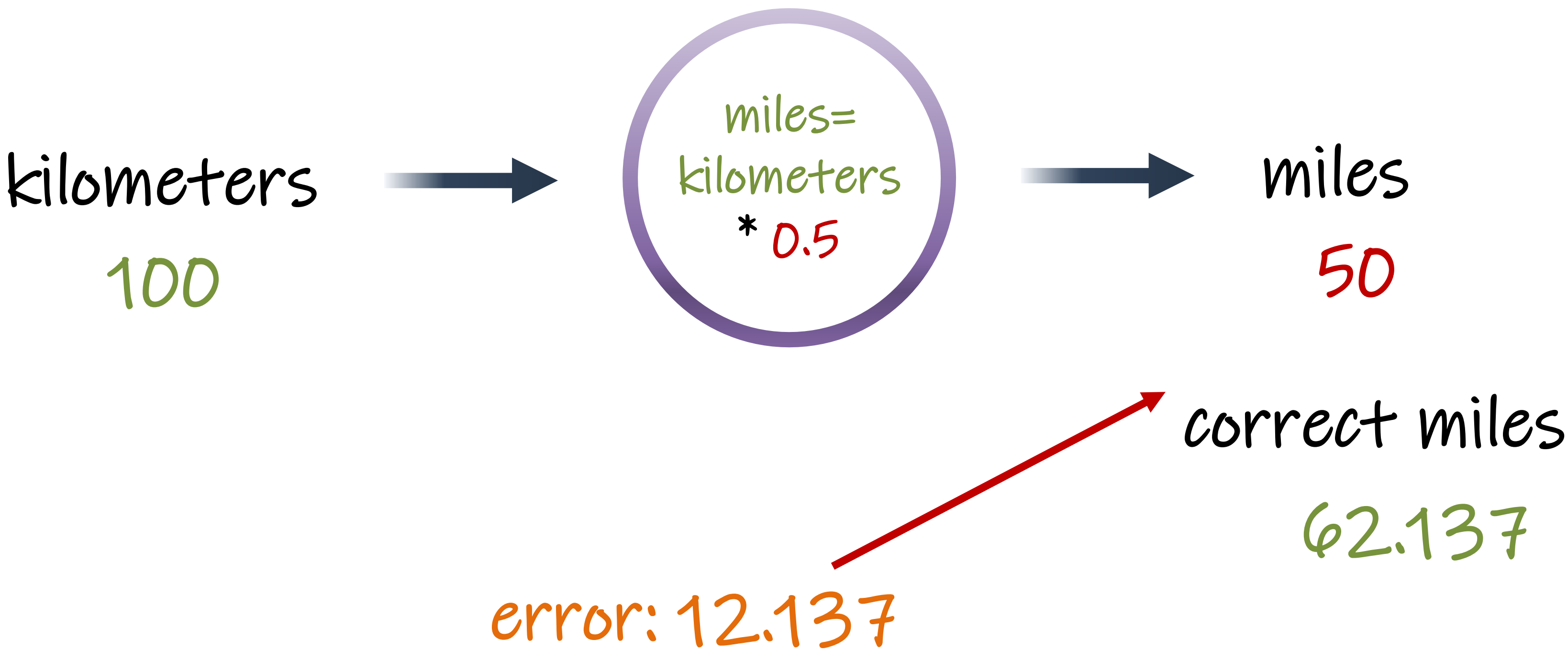
miles=
kilometers
* 0.5

miles

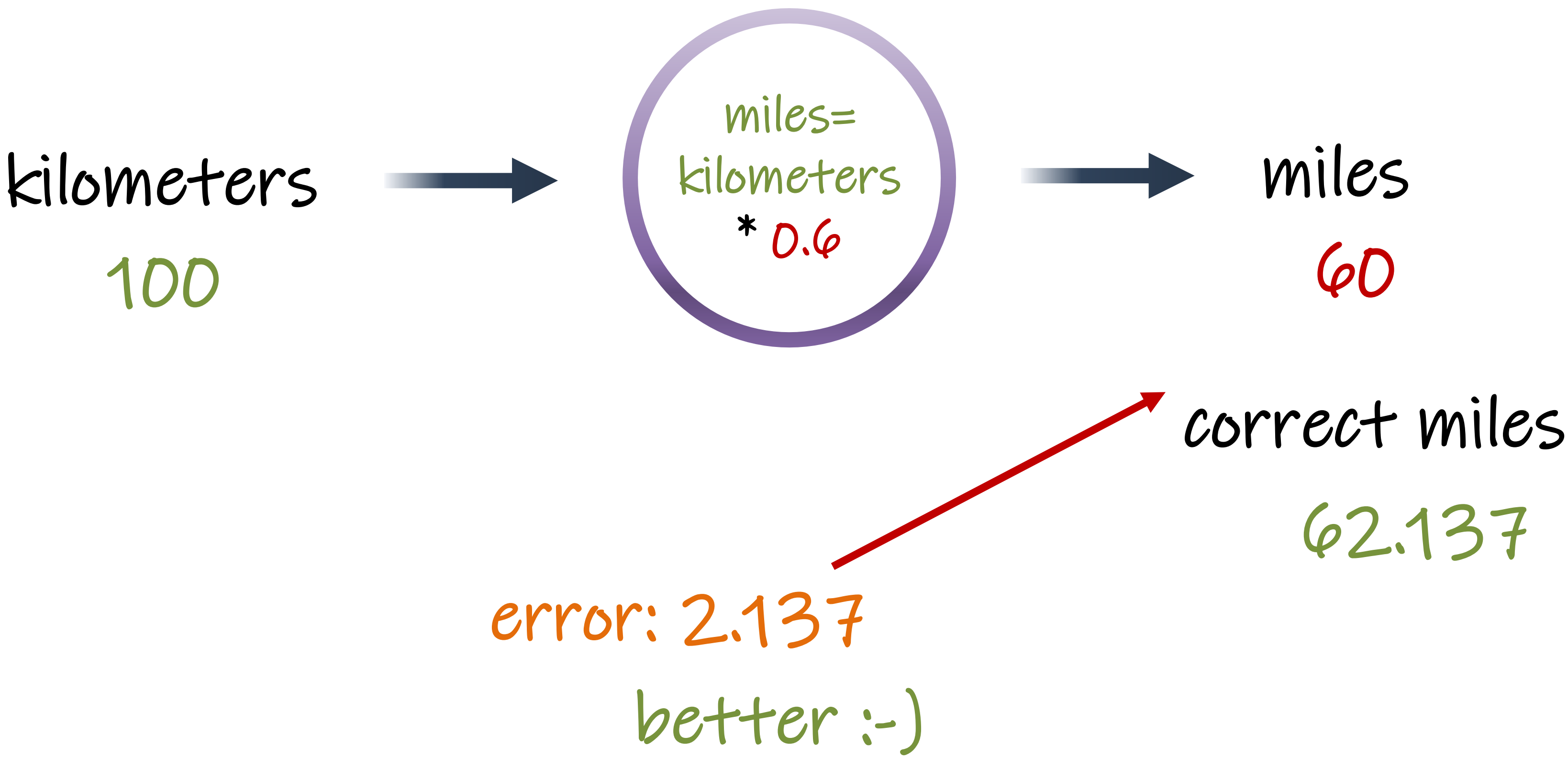
50

random starting point,
only model Parameter

Example


kilometers
100
miles=
kilometers
* 0.5
miles
50
correct miles
62.137
error: 12.137

Example: Learning



Example: Learning

kilometers
100

miles=
kilometers
* 0.6

miles
70

correct miles
62.137

error: -7.863
dang, too far

Now with more inputs

- This was extremely simplistic

- Now let's look at a more complicated thing:
  - You are thinking whether you should attend a sports event

- You base the decision on the following factors:
  1. Temperature?
  2. Ticket Price?
  3. Travel time?

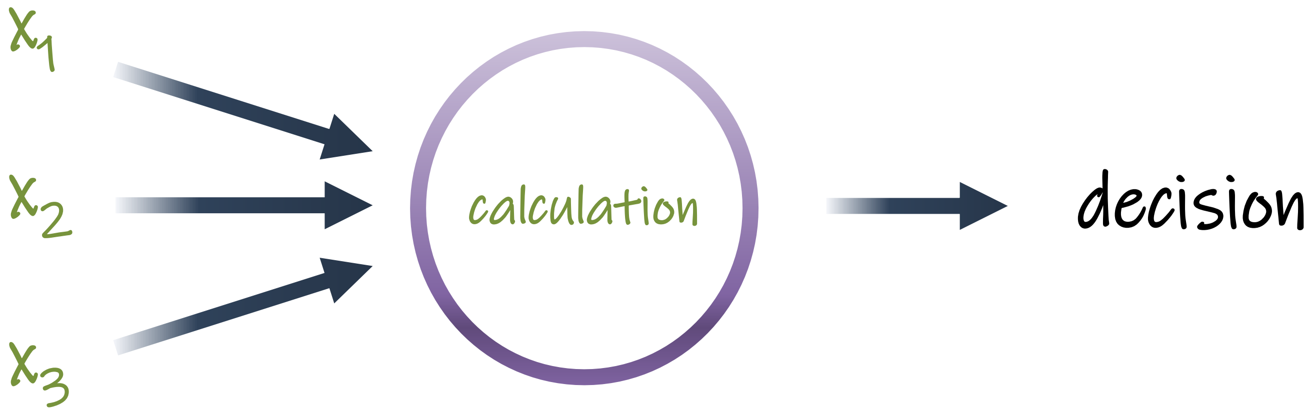
Now with more inputs

- This was extremely simplistic

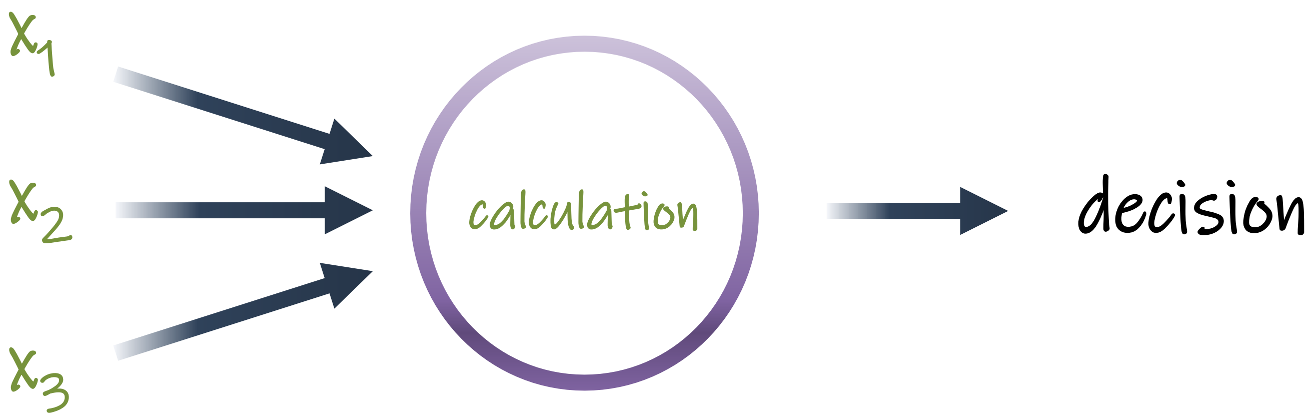
- Now let's look at a more complicated thing:
  - You are thinking whether you should attend a sports event

- You base the decision on the following factors:
  1. Temperature? x_1
  2. Ticket Price? x_2
  3. Travel time? x_3

Now with more inputs



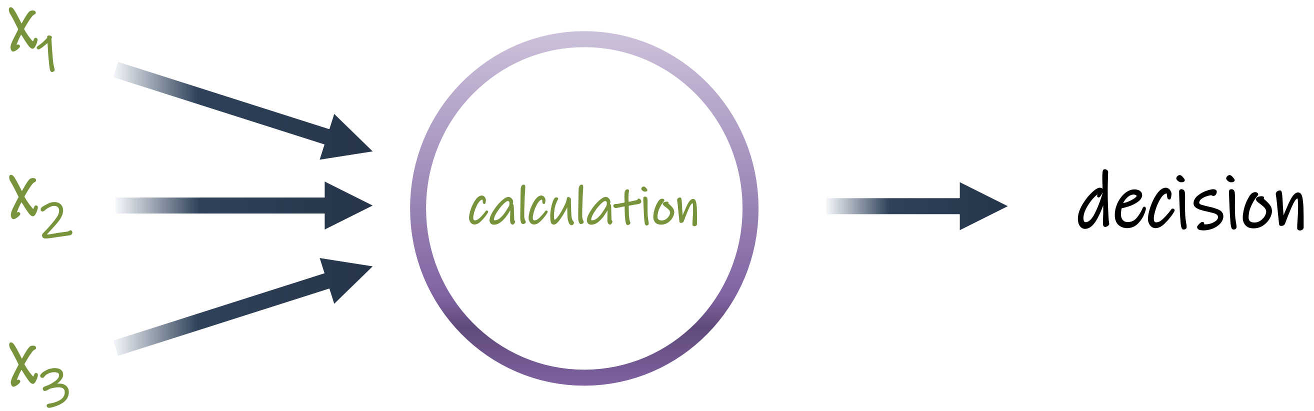
Now with more inputs



- How to make the decision?

$$x_1 + x_2 + x_3 > \text{threshold?}$$

Now with more inputs

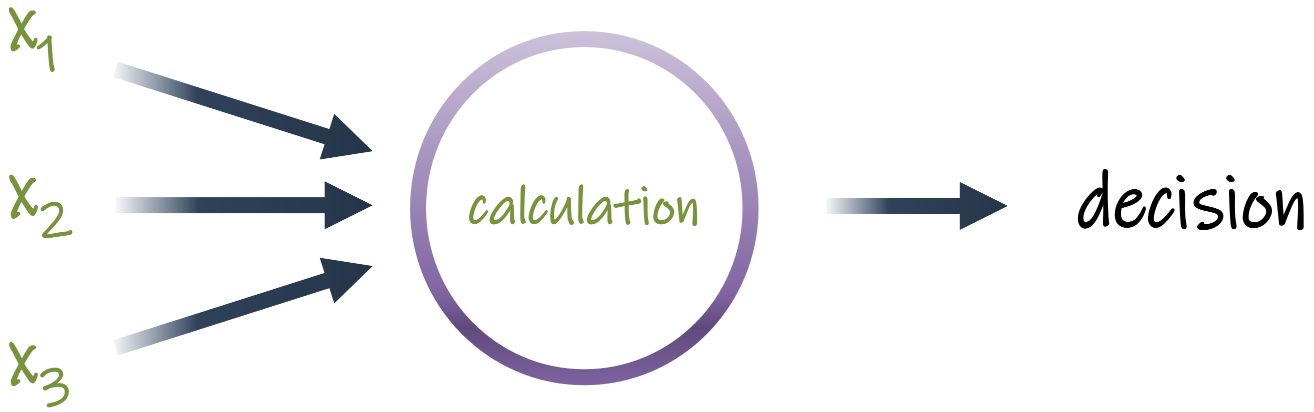


- How to make the decision?

$$x_1 + x_2 + x_3 > \text{threshold?}$$

- That would mean, the higher the price and the farther away, the more likely you are to go …

Now with more inputs

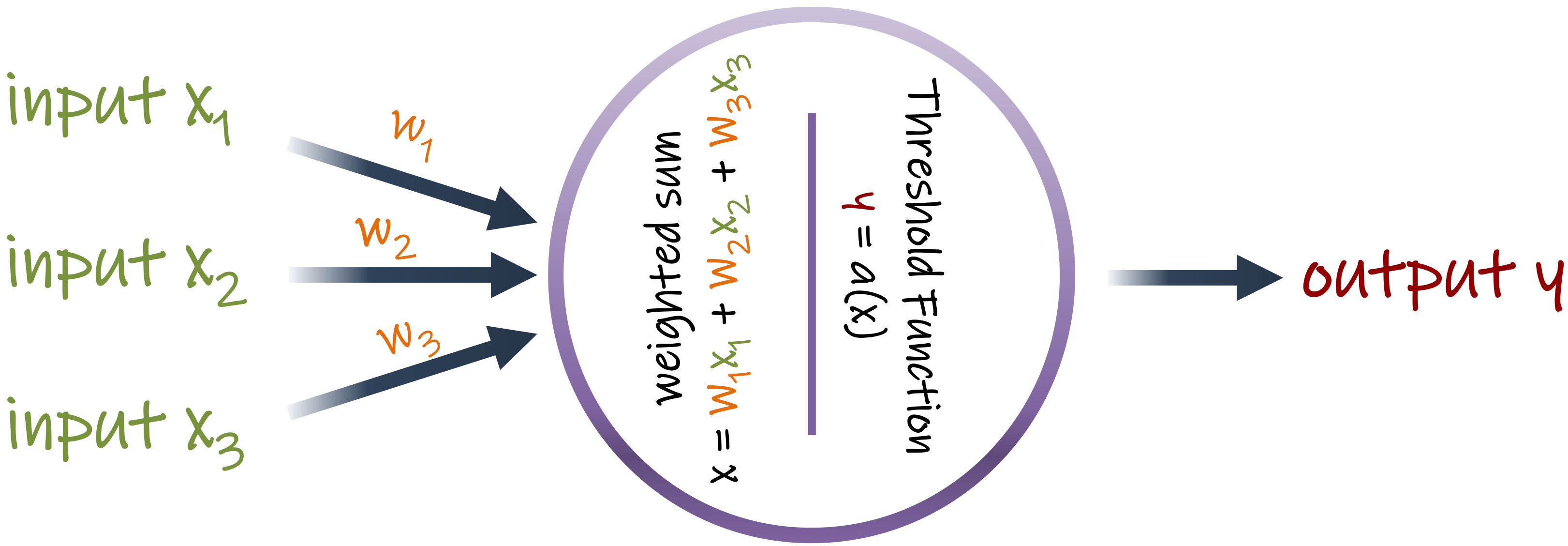


- How to make the decision?
- Therefore, introduce weights:

$$w_1x_1 + w_2x_2 + w_3x_3 > \text{threshold?}$$

- Now, you can set the weights and the threshold according to your preference, e.g., 1, -4, -1

This is in fact how neurons work

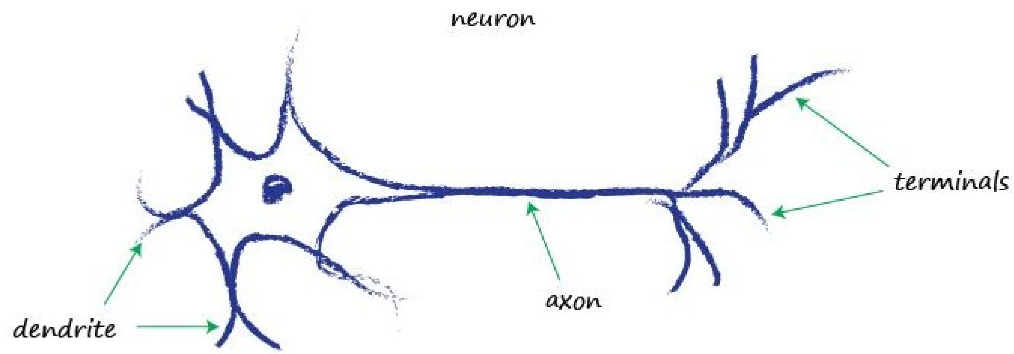


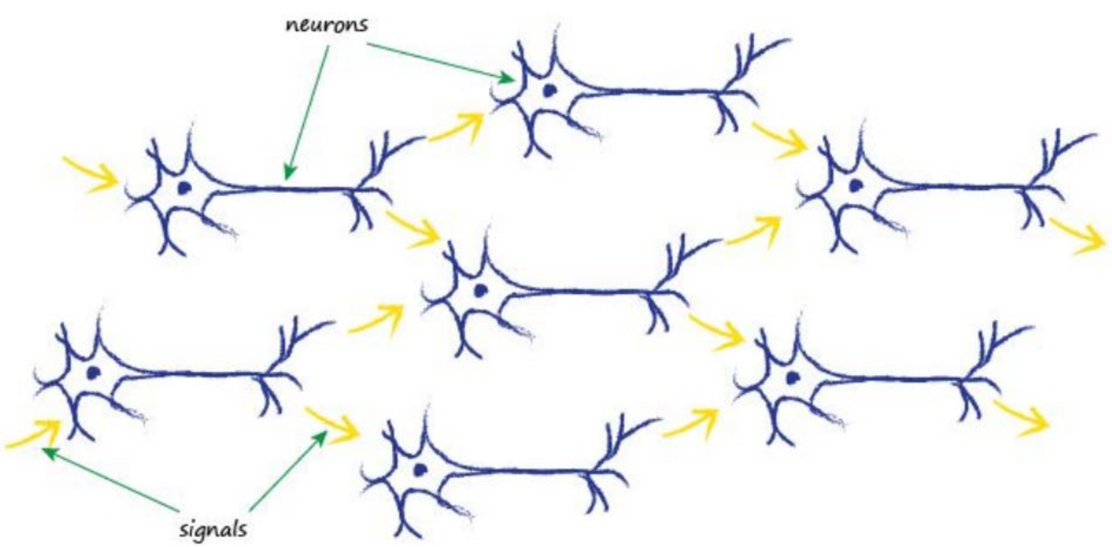
- An artificial Neuron consists of
  - A number of weighted inputs
  - An activation function
  - The generated output

Deep Feedforward Networks
Part I

- **PART I**
  - What is a Neuron?
  - **How to build a Network**
- **Intermezzo: Some Math**
- **PART II**
  - Networks
  - Output Units
  - Hidden Units
  - Architecture Design

From Neuron to "Brain"

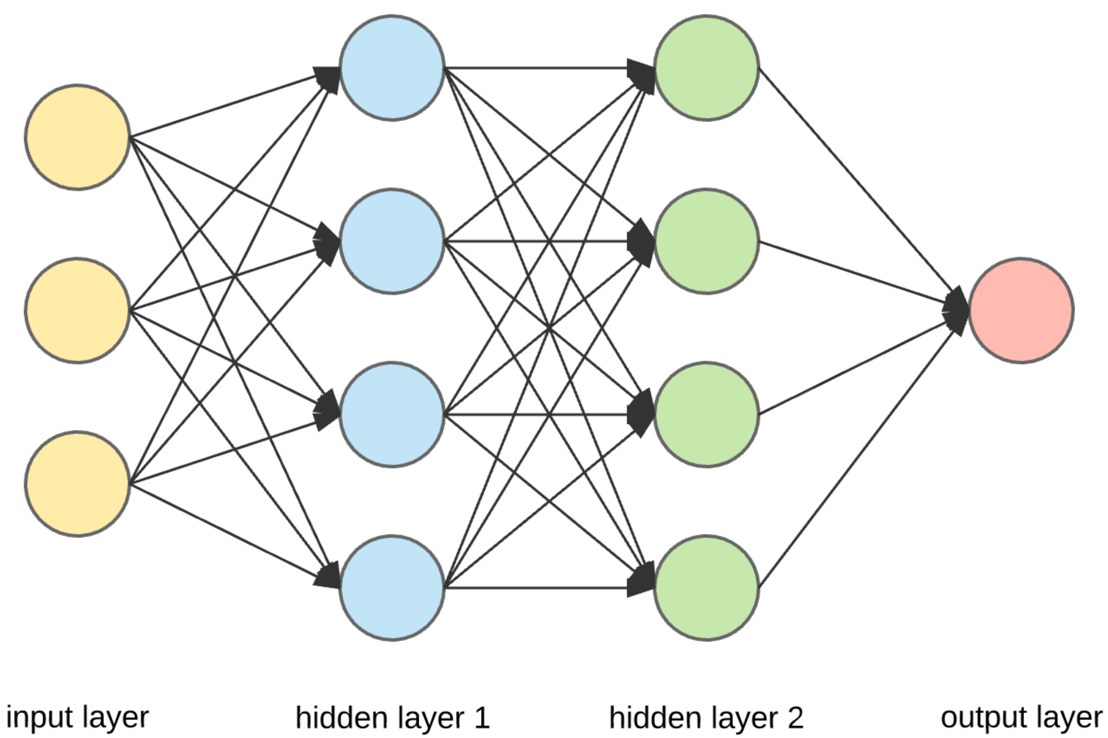




- We now can connect neurons
- The output of one neuron becomes the input of other neurons

In a more structured way

- We normally have more than one node
- Multiple Nodes are arranged in layers
- Each layer receives the generated output from the previous layer



How To Build these Networks?

- **Define the Architecture**
  - How many inputs?
  - How many hidden layers?
    - How many neurons per layer?
    - What activation function to use?
  - What is the desired output? Define output neurons
    - Binary classification: Sigmoid
    - General classification: Softmax
    - Regression: linear output
- **Define the loss function**
  - Dependent on the output
- **Train the network**

How is a Network Trained?

- **General Procedure:**
  - We present the network with an example where we know the answer
  - We observe the answer of the network and adjust the weights accordingly

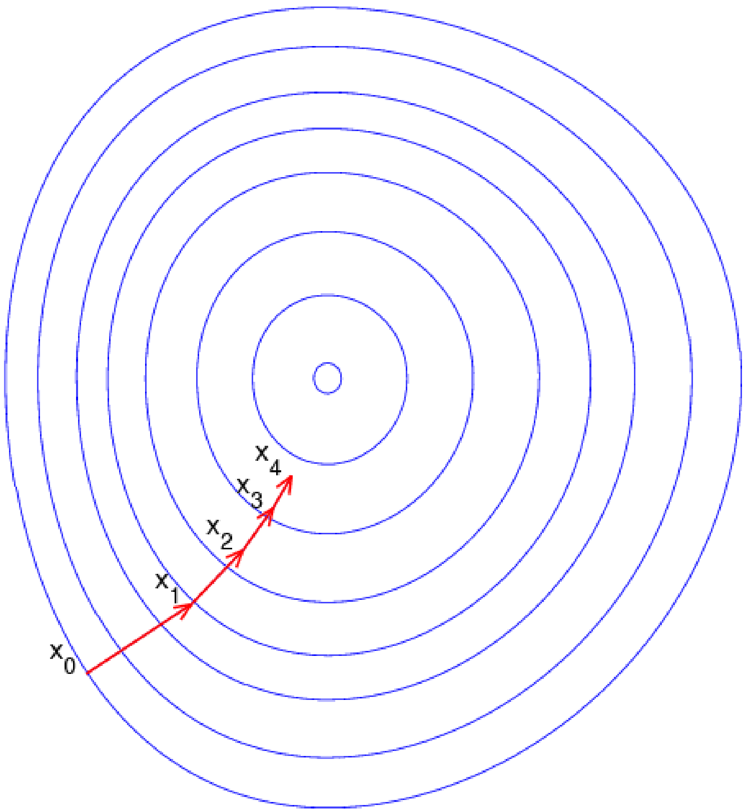
- **Ideal world:**
  - We simply look at the dataset and could exactly calculate the weights

- **Reality:**
  - We define a **cost-function**, the so-called **loss function**
  - We iteratively approximate the best setting by trying to successively minimize the cost functions
  - We do this by a process called **gradient descent**

Gradient Based Learning

The Central Idea

- Update the model parameters following the steepest slope of the loss function



More Mathematically

- Suppose function $y = f(x)$
- Derivative of function denoted: $f'(x)$ or as dy/dx
  - Derivative $f'(x)$ gives the slope of $f(x)$ at point x
  - It specifies how to scale a small change in input to obtain a corresponding change in the output:

$$f(x + \varepsilon) \approx f(x) + \varepsilon f'(x)$$

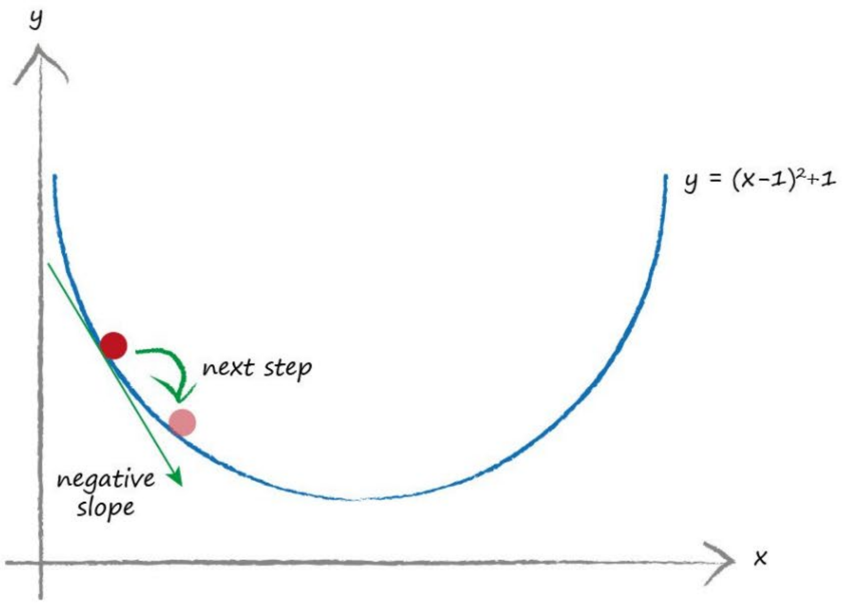
- We know that

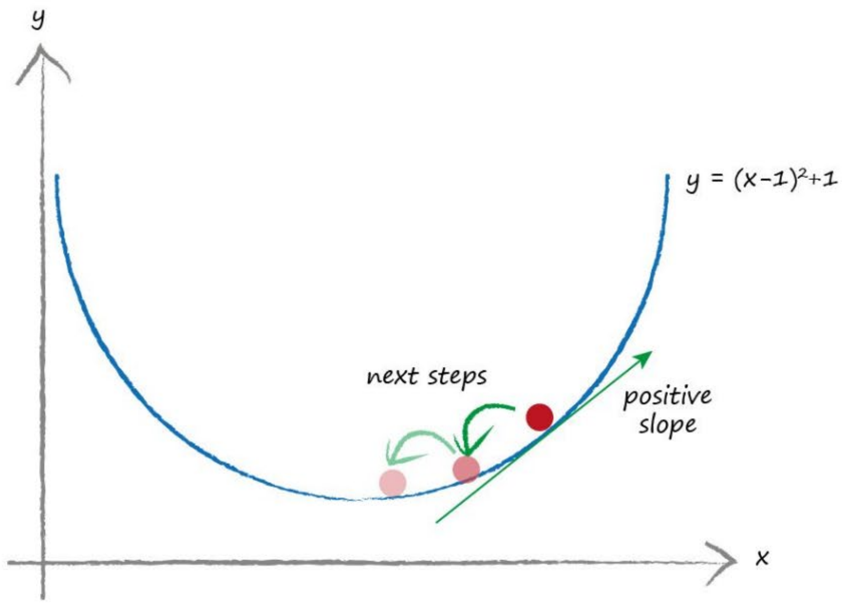
$$f(x - \varepsilon \mathrm{sign}(f'(x)))$$

is less than $f(x)$ for small ε.

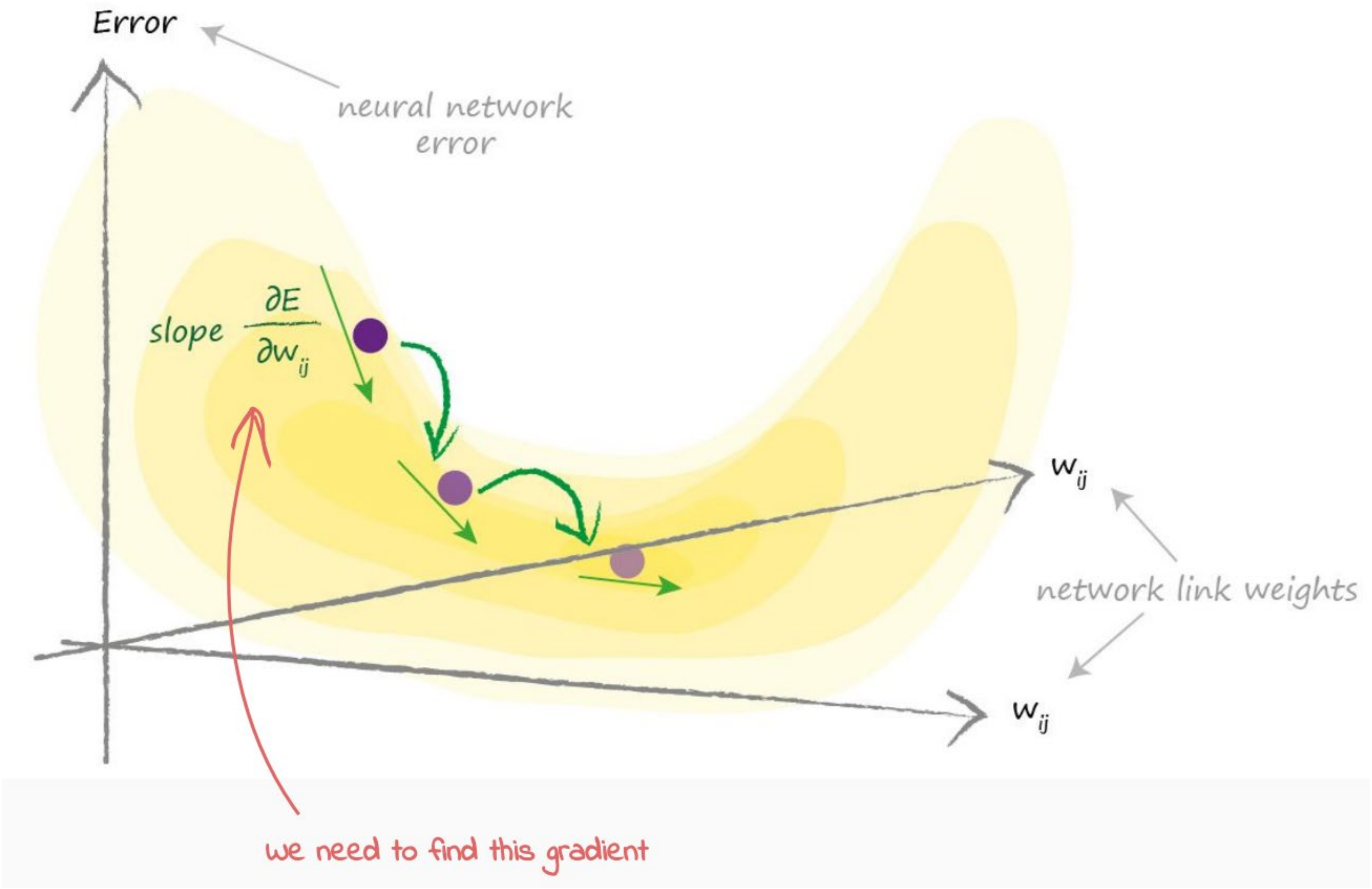
  - Thus we can reduce $f(x)$ by moving x in small steps with opposite sign of derivative
  - This technique is called gradient descent (Cauchy 1847)

Gradient Descent





Gradient Descent



A bit more formally

- Let us assume we a data dataset $X = \{x_1, \dots, x_N\}$
- We have define a neural network with the parameters $\boldsymbol{\theta}$
- We further have defined a cost function J:

$$J(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y})$$

  - Which give us per sample $\boldsymbol{x}$ and the true label $\boldsymbol{y}$, and the current parameters of the model a certain cost
    - If we, e.g., misclassify x we will get high costs, if we are correct, very low costs

A bit more formally

- For each object $\boldsymbol{x}_1$ we can now calculate the gradient:

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{x}_1, \boldsymbol{y}_1)$$

- This gradient tells us how to modify the weights in order to achieve a lesser weight, but is noise and overly depended on $\boldsymbol{x}_1$
- Since we want to improve on the entire dataset, we calculate the overall gradient:

$$\boldsymbol{g} = \frac{1}{N} \sum_{X} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{x}_i, \boldsymbol{y}_i) = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y})$$

A bit more formally

- We now modify the weights according to the gradient descent method:

$$\boldsymbol{g} = \frac{1}{N}\sum_{X} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{x}_i, \boldsymbol{y}_i) = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y})$$

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{old} - \epsilon \boldsymbol{g}$$

  - ϵ is called the learning rate
  - Note, most of the time, we do not compute the gradient for all available data, but for randomly selected small portions of the dataset.
  - This is call the **Stochastic Gradient Descent**

Don't Panic

- This was just a quick introduction to these networks
- We will discuss the different components later
- You will gradually understand in detail what was shown here