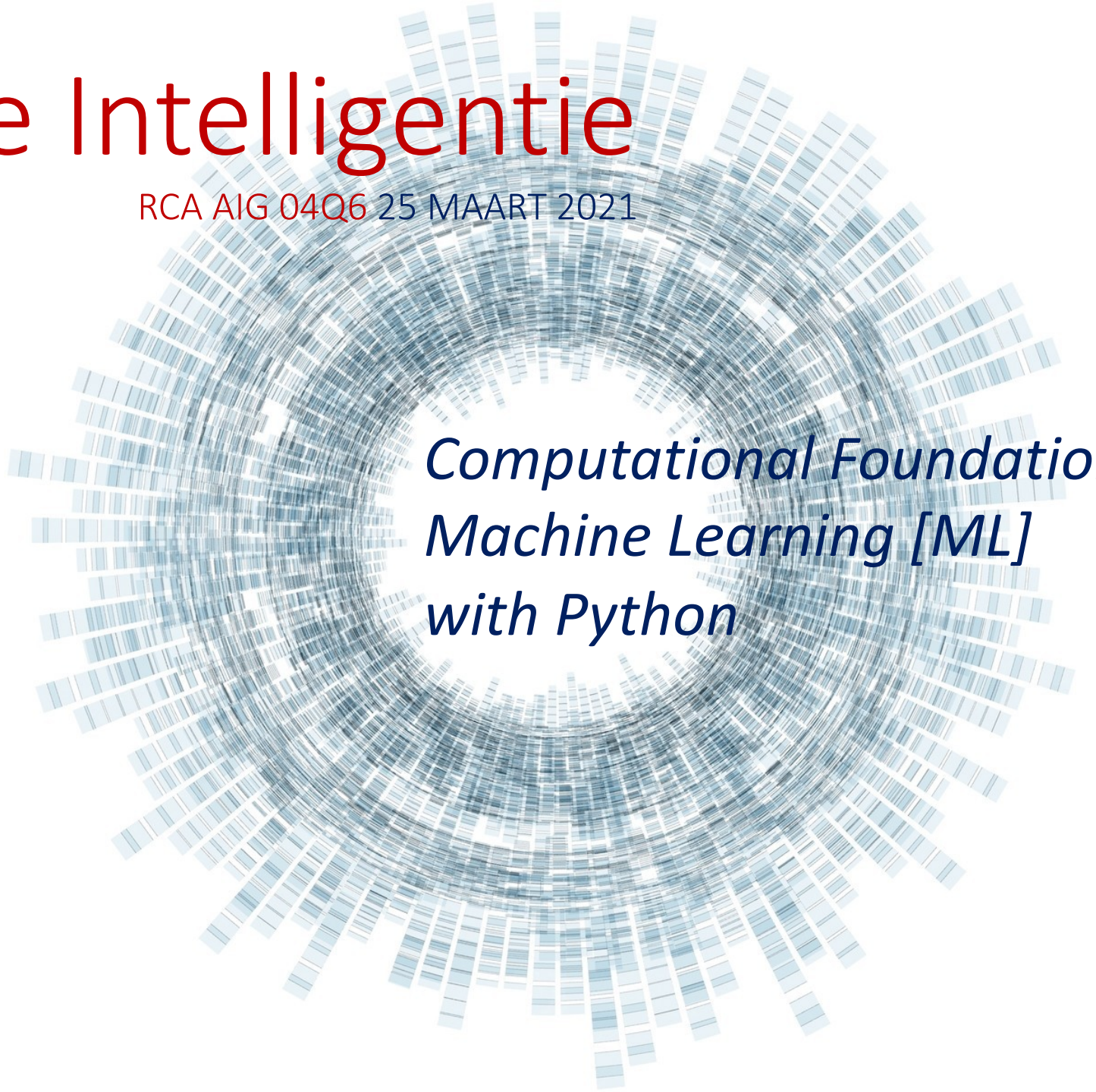
Artificiële Intelligentie

RCA AIG 04Q6 25 MAART 2021



Computational Foundations of Machine Learning [ML] with Python

HOGESCHOOL ROTTERDAM

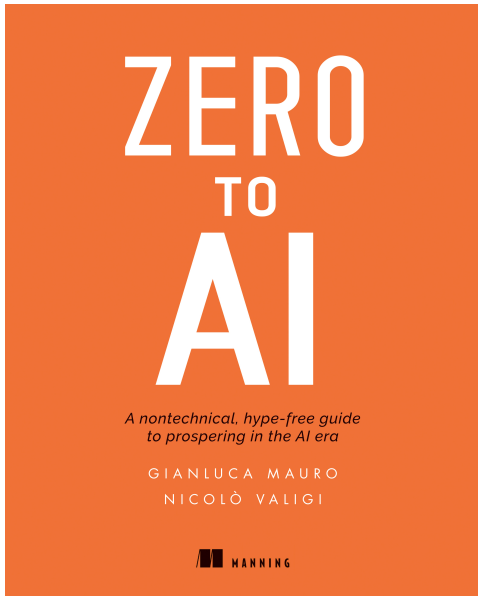
DataLab
Rob van der Willigen

CONTEXT

- Prerequisites: basics in linear algebra, probability, and analysis of algorithms.
- Workload: homework assignments
- GitHub: Start a ML repository at GitHub

This Lecture

- Basic definitions and concepts.
- Introduction to the problem of learning.



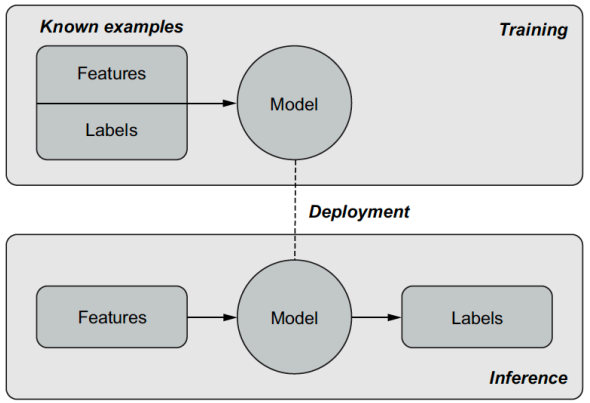


Figure 2.3 The two phases of machine learning: training and inference

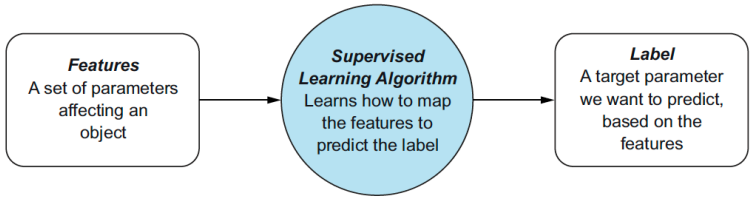
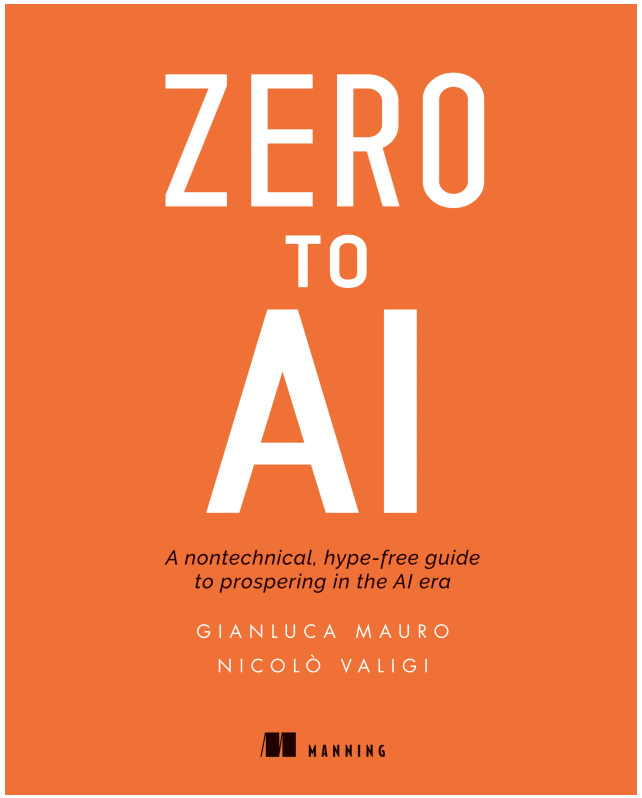


Figure 2.4 The core concept of supervised learning: finding a mapping between a set of features and a label



HOGESCHOOL ROTTERDAM

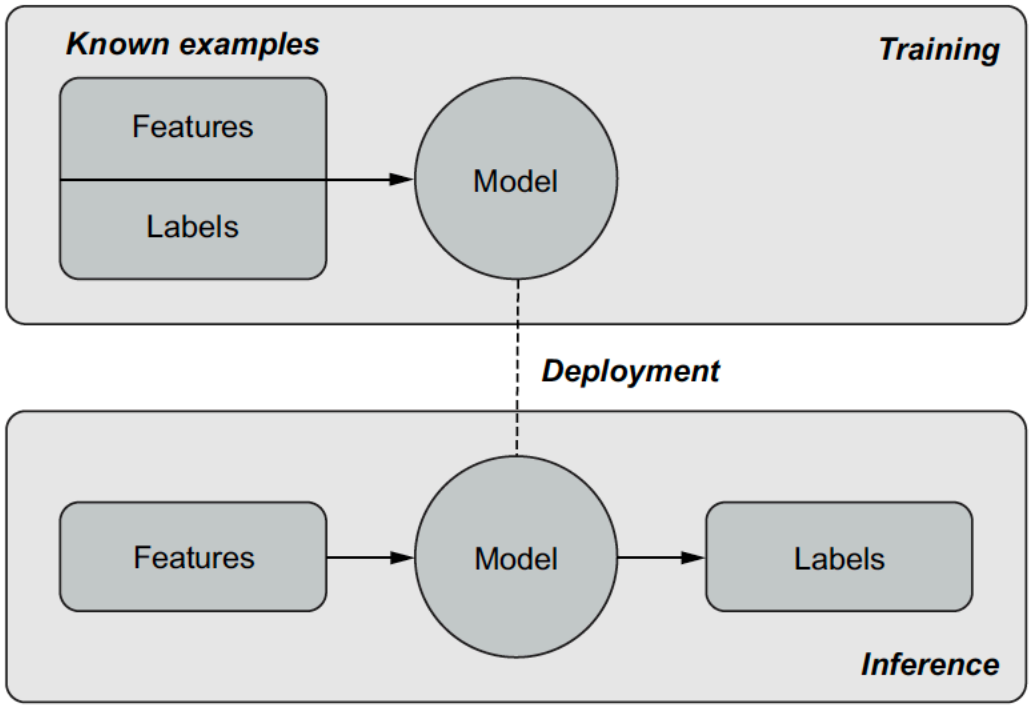


Figure 2.3 The two phases of machine learning: training and inference

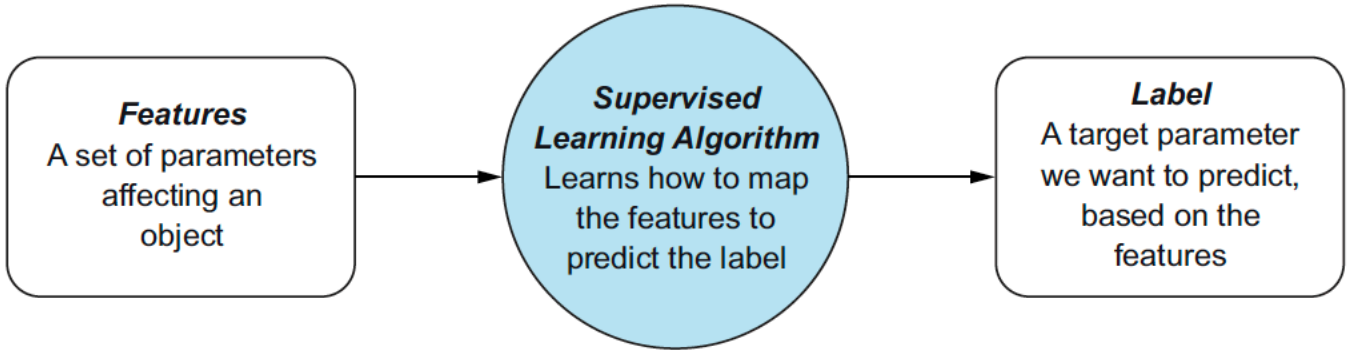
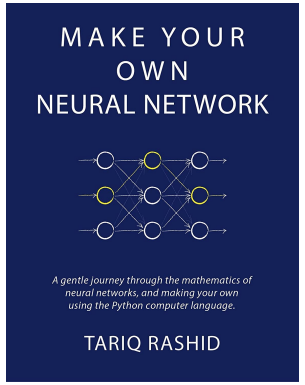
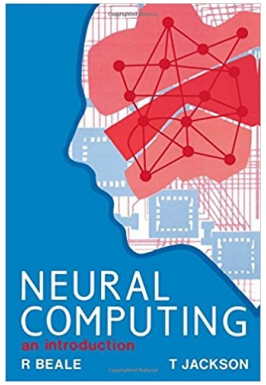
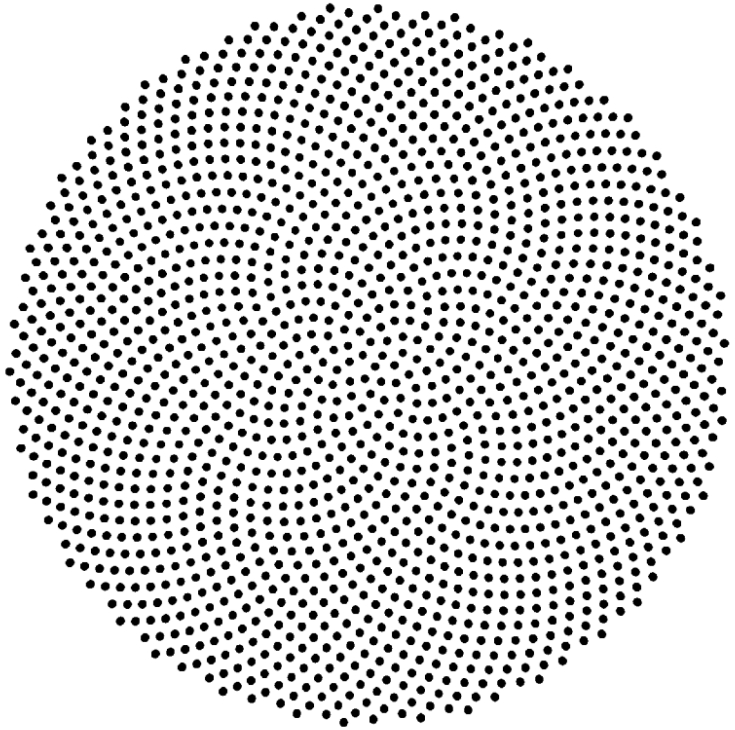


Figure 2.4 The core concept of supervised learning: finding a mapping between a set of features and a label

{01}
Fundamentals

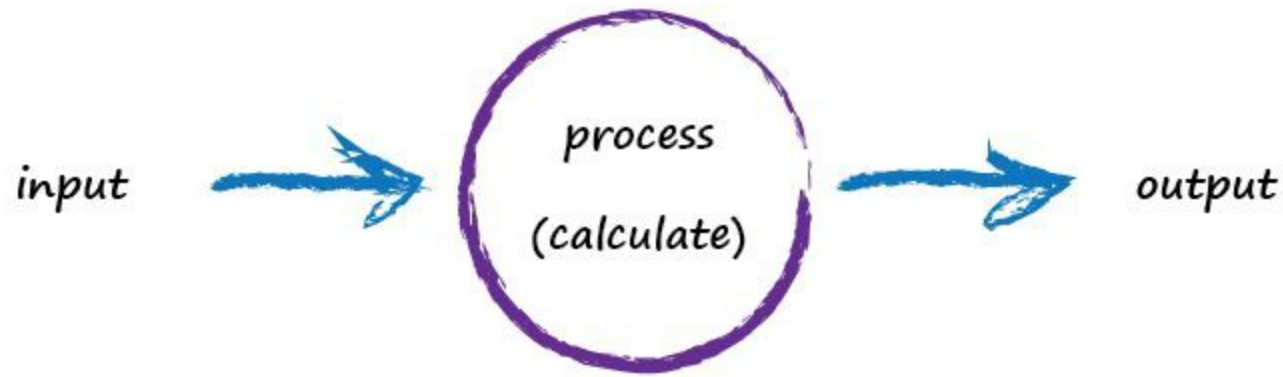


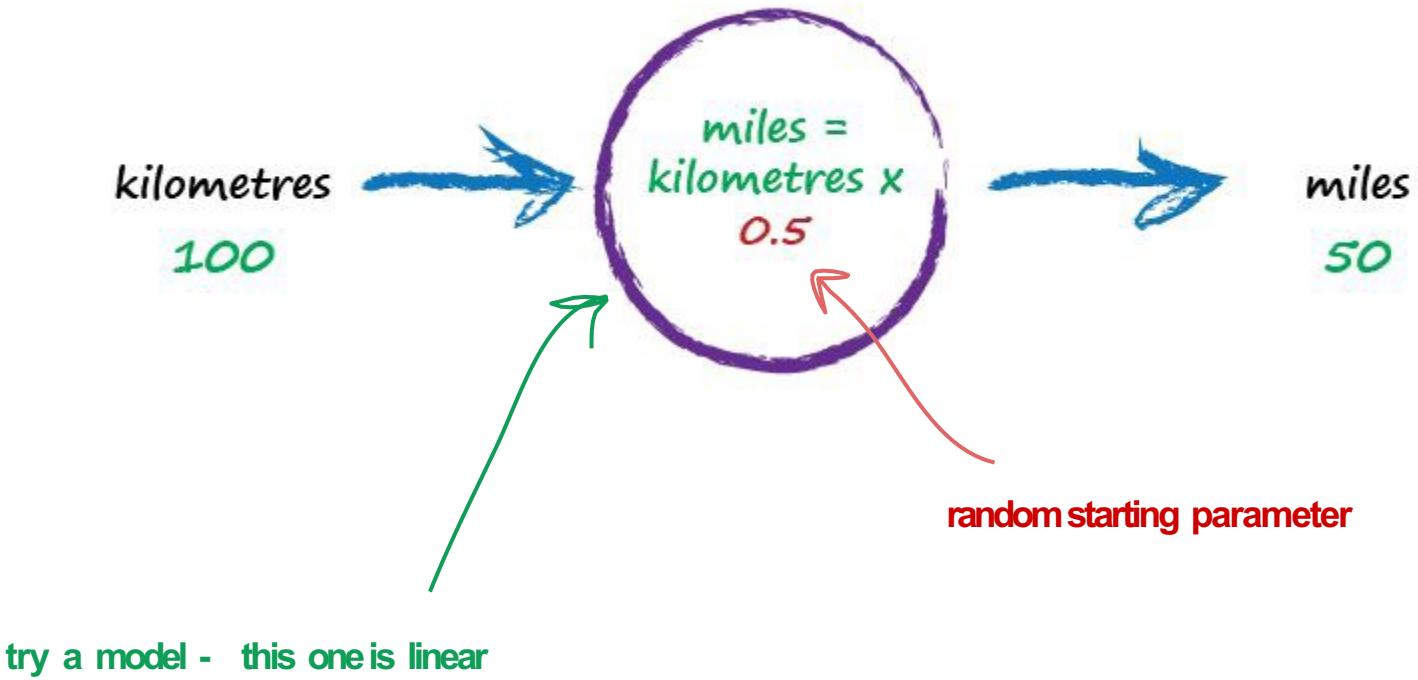


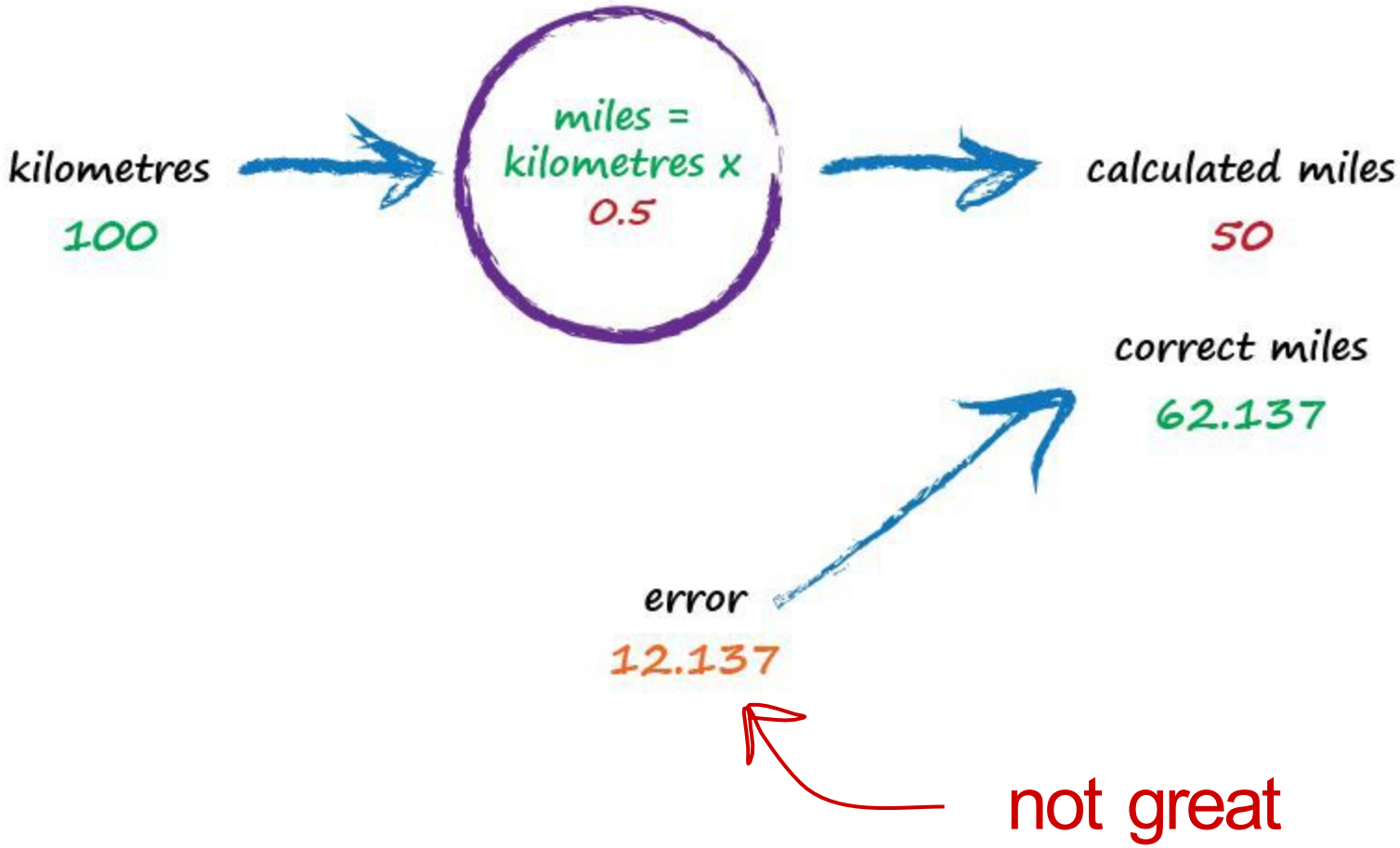


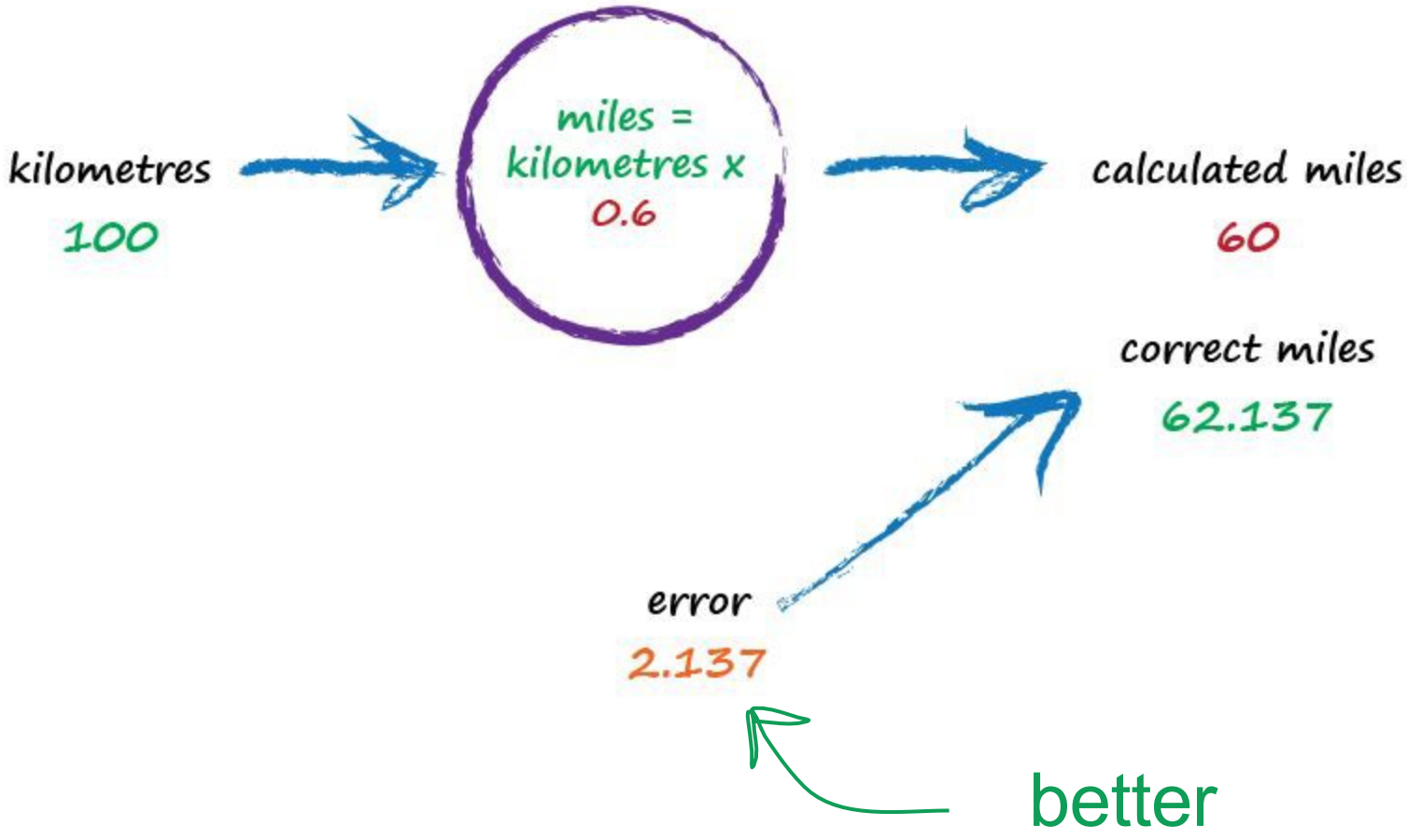
HOGESCHOOL
ROTTERDAM

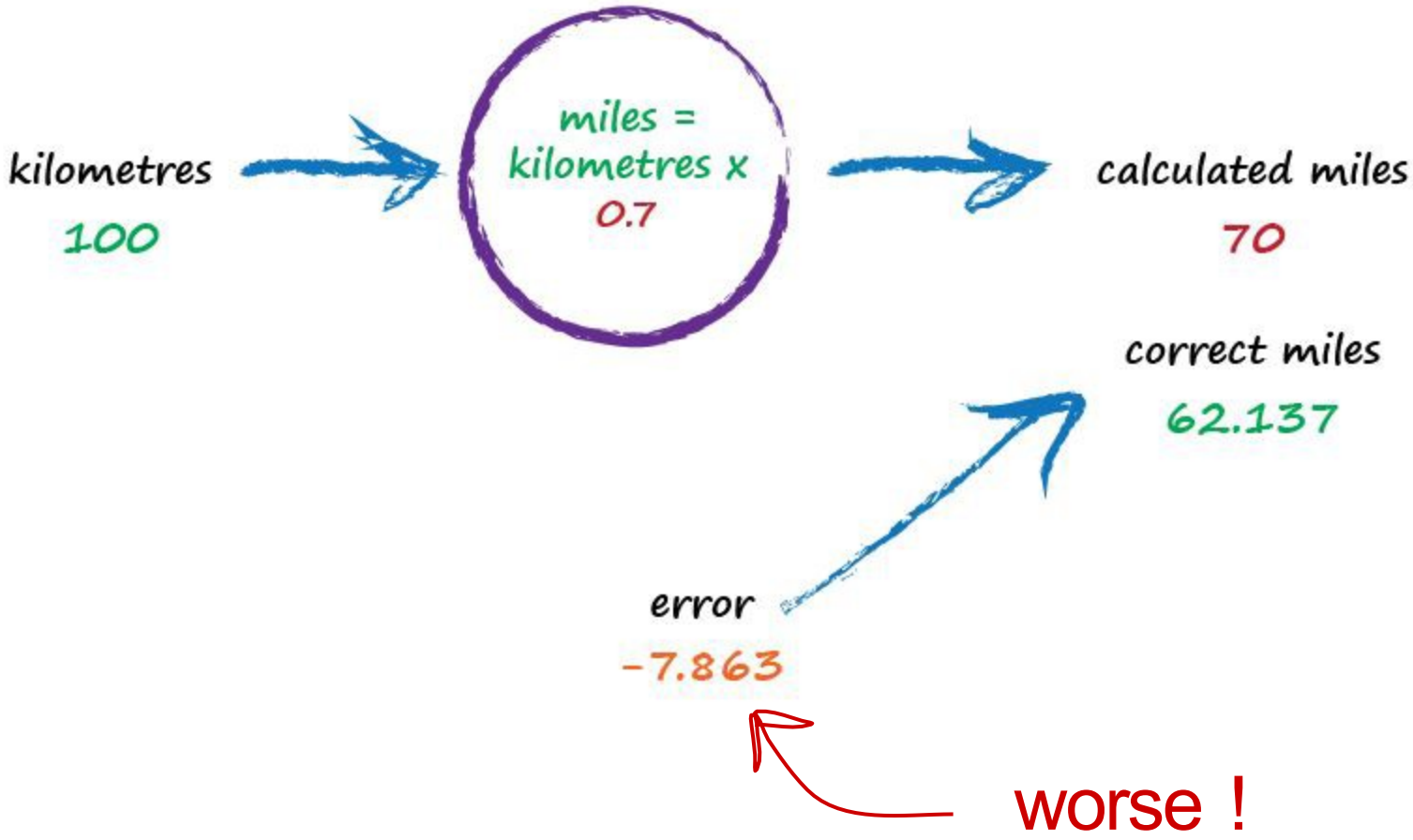

question
think
answer

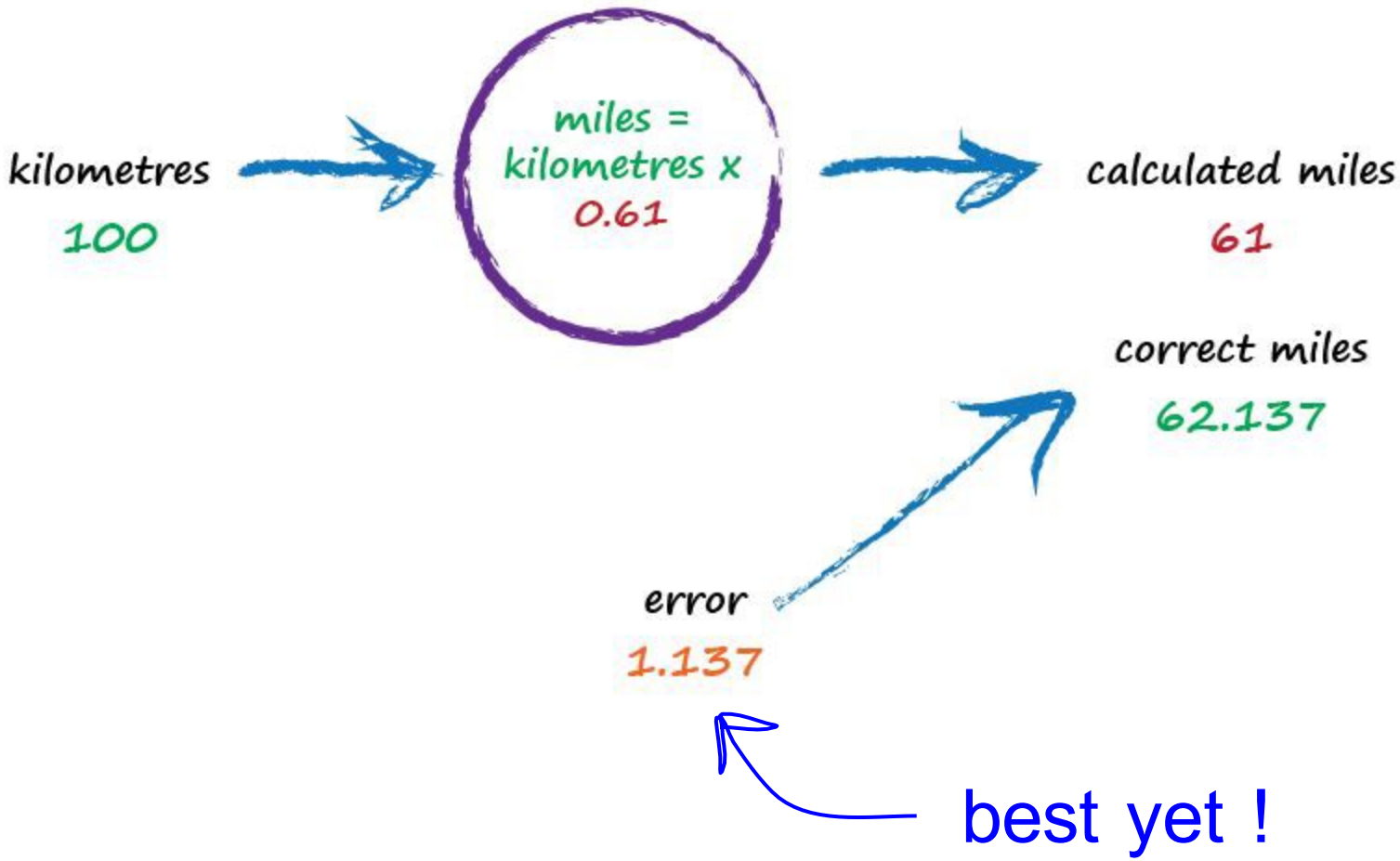

input
process
(calculate)
output


kilometres
100
miles =
kilometres x
0.5
miles
50
random starting parameter
try a model - this one is linear


kilometres
100
miles =
kilometres x
0.5
calculated miles
50
correct miles
62.137
error
12.137
not great

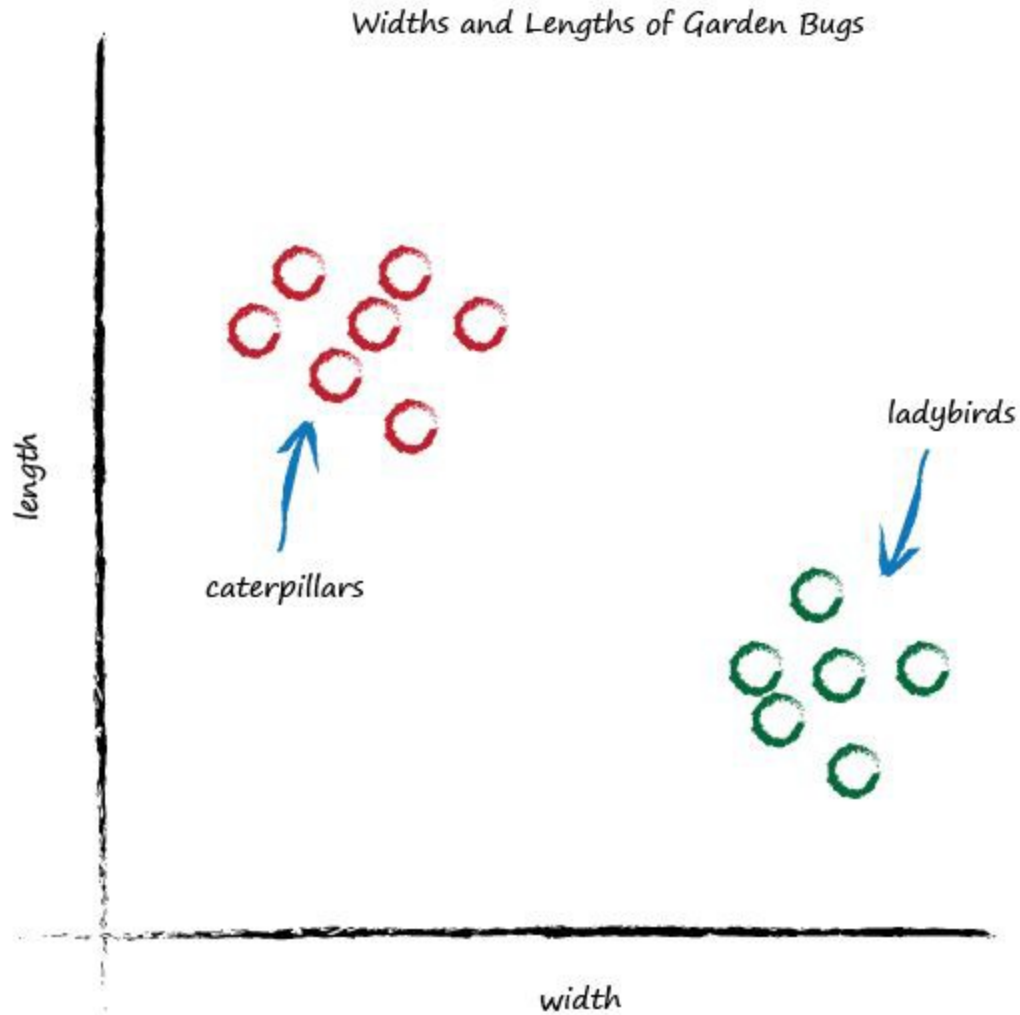

kilometres
100
miles =
kilometres x
0.6
calculated miles
60
correct miles
62.137
error
2.137
better

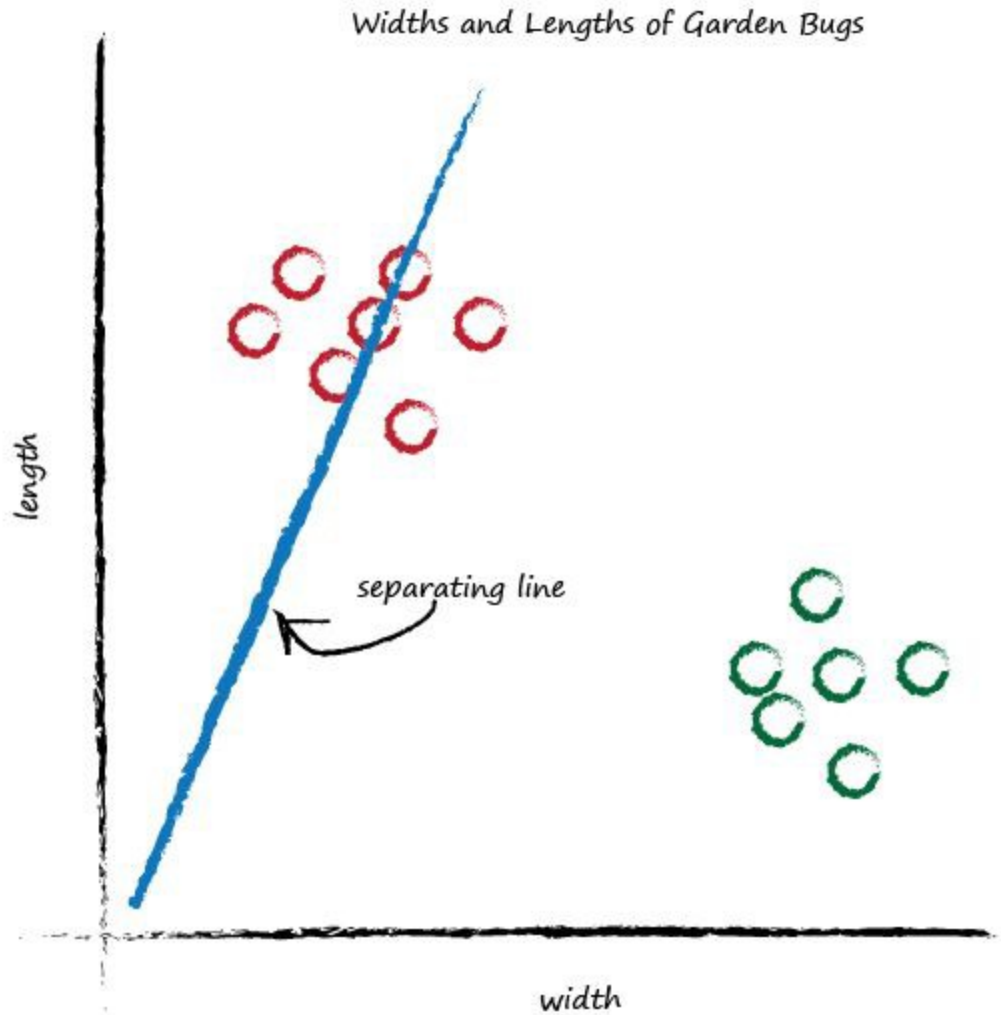

kilometres
100
miles =
kilometres x
0.7
calculated miles
70
correct miles
62.137
error
-7.863
worse !

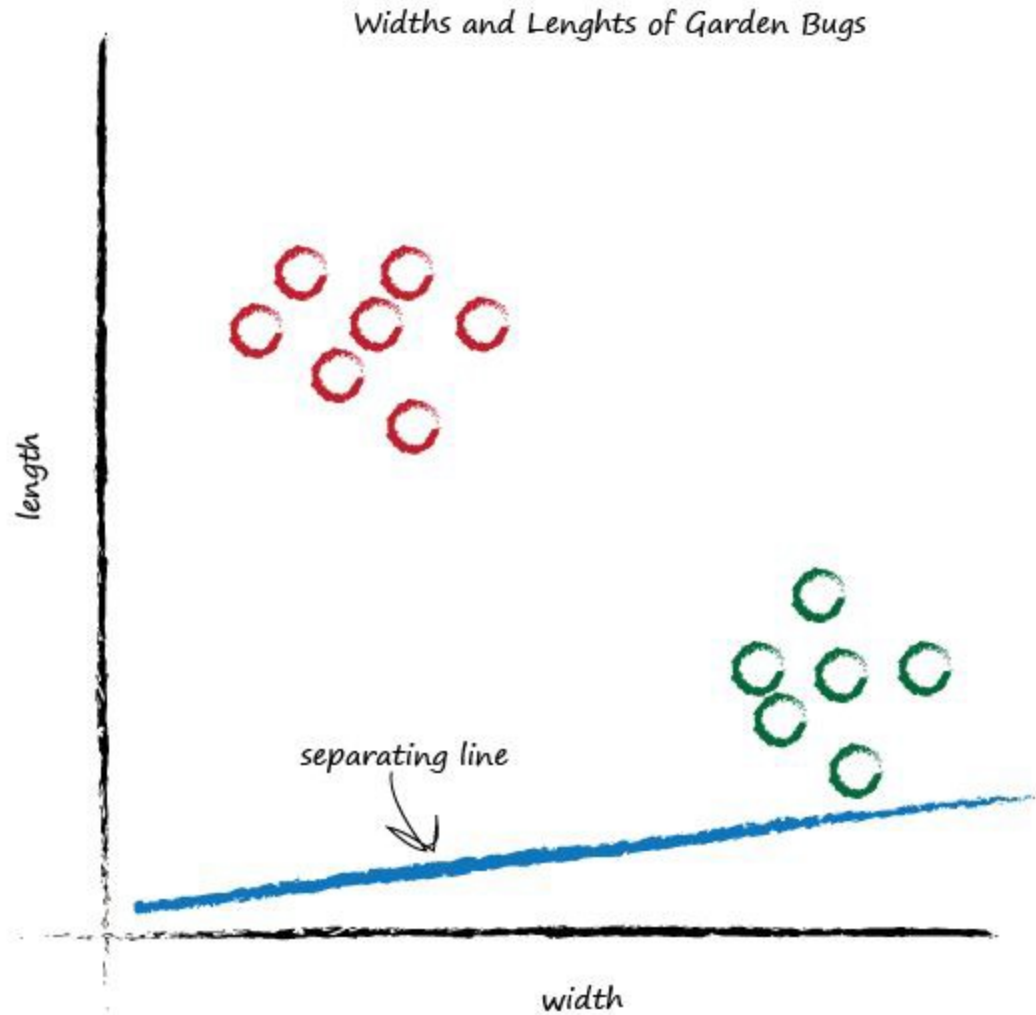

kilometres
100
miles =
kilometres x
0.61
calculated miles
61
correct miles
62.137
error
1.137
best yet !

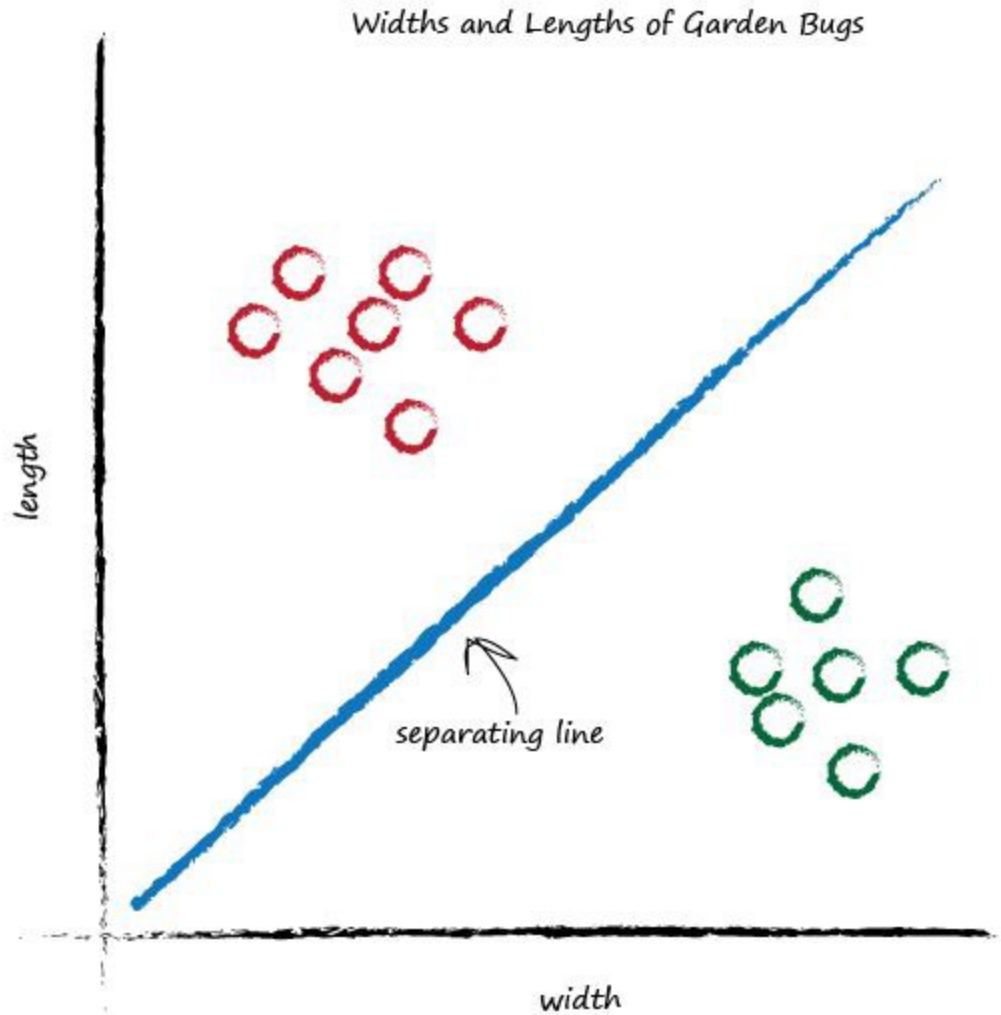
1. Don’t know how something works exactly? Try a **model** with **adjustable** parameters.

2. Use the **error** to refine the parameters.


Widths and Lengths of Garden Bugs
length
width
caterpillars
ladybirds


Widths and Lengths of Garden Bugs
length
separating line
width


Widths and Lenghts of Garden Bugs
length
separating line
width


Widths and Lengths of Garden Bugs
length
separating line
width

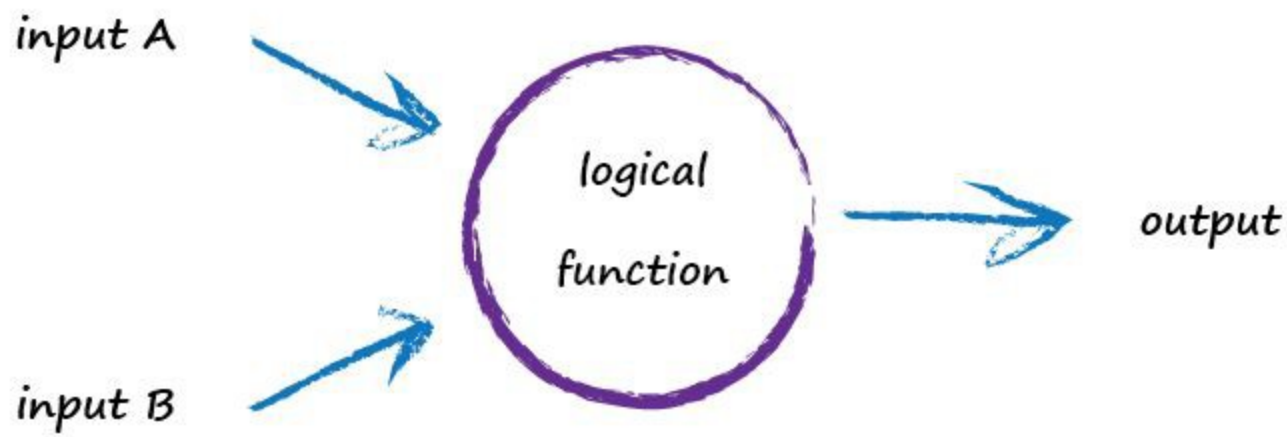
1. **Classifying** things is kinda like **predicting** things.

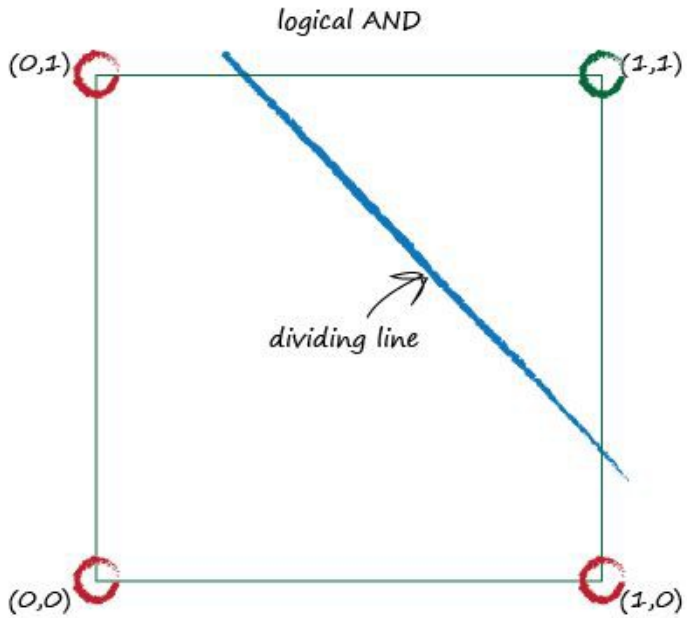
Logic

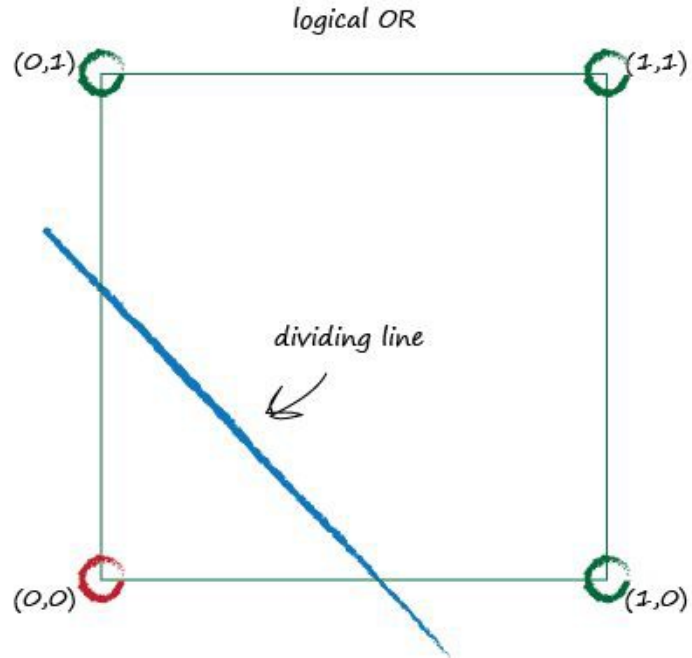
IF I have eaten my vegetables **AND** I am still hungry **THEN** I can have ice cream.

IF it's the weekend **OR** I am on annual leave **THEN** I'll go to the park.

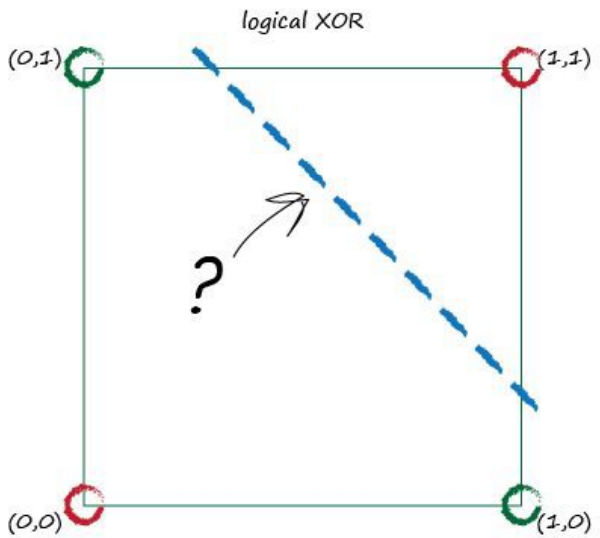
Input A	Input B	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

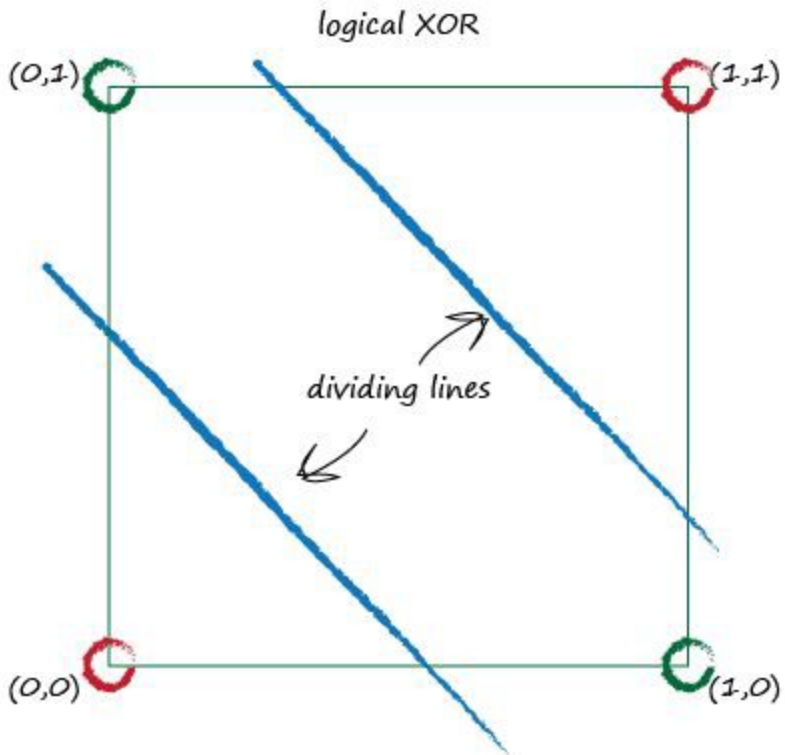

input A
input B
logical
function
output


logical AND
(0,1)
(1,1)
dividing line
(0,0)
(1,0)


logical OR
(0,1)
(1,1)
dividing line
(0,0)
(1,0)

Input A	Input B	XOR
0	0	0
0	1	1
1	0	1
1	1	0





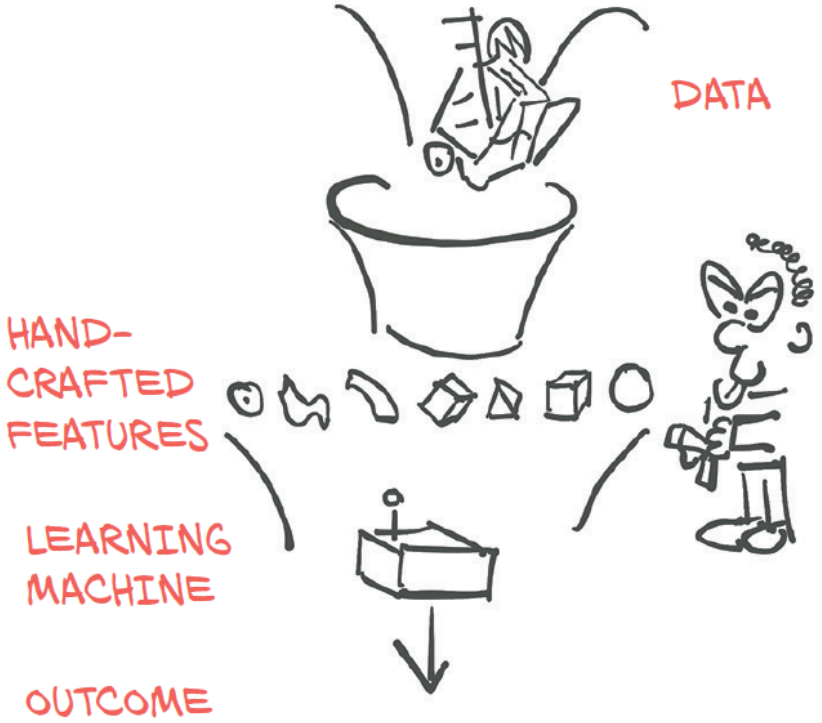
…Use more than one node!

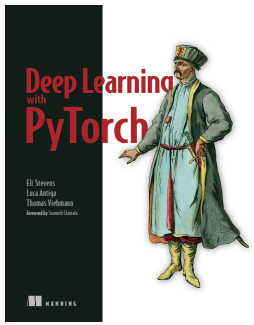
1. Some problems can't be solved with just a single simple linear classifier.

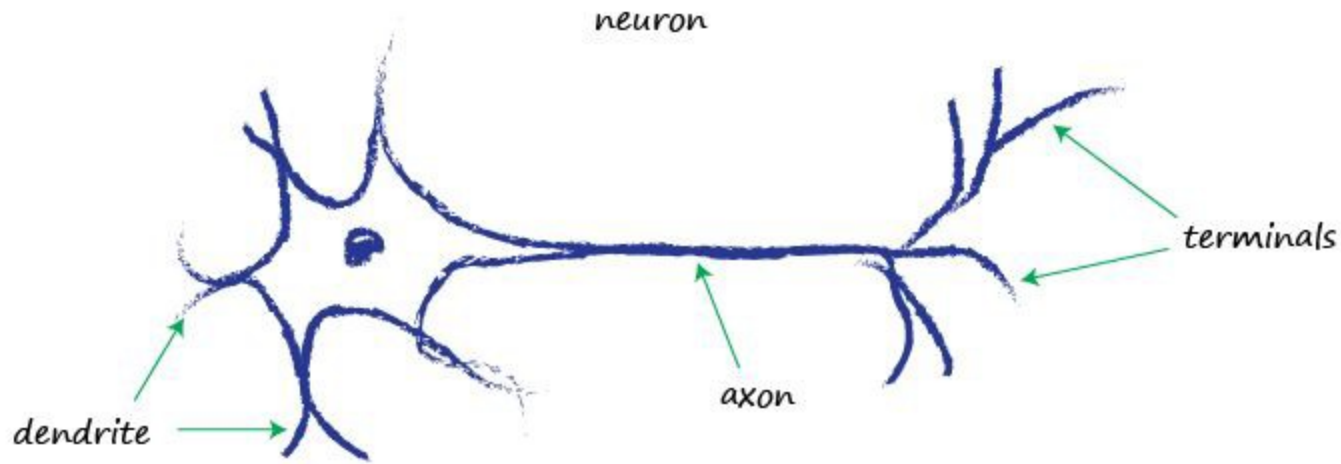
2. You can use **multiple nodes** working together to solve many of these problems.

MACHINE LEARNING [ML]

HOGESCHOOL ROTTERDAM

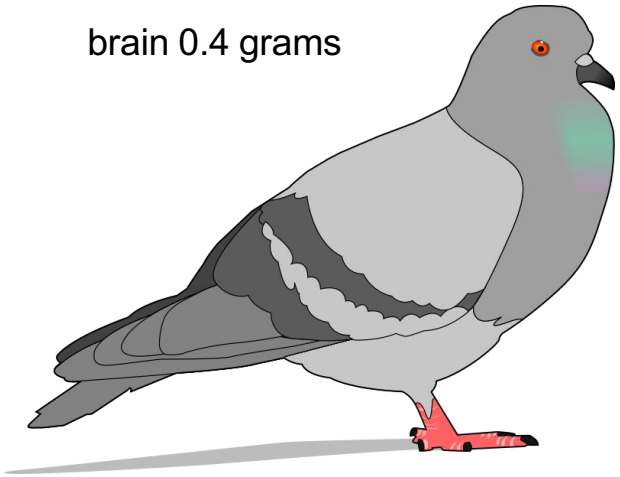




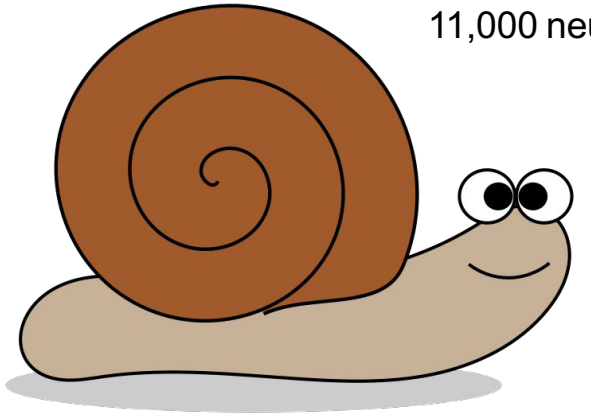

neuron
terminals
axon
dendrite

Nature

brain 0.4 grams

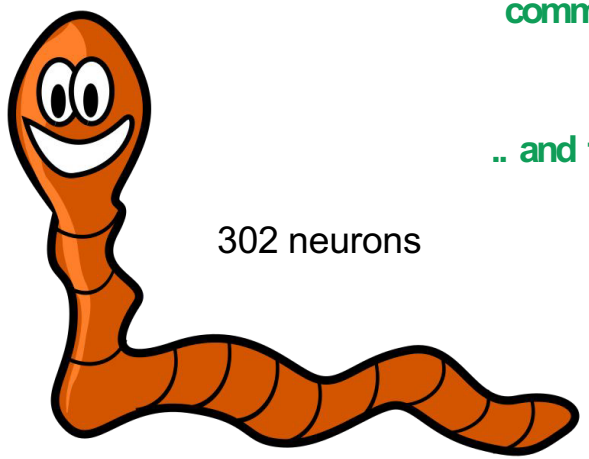


11,000 neurons

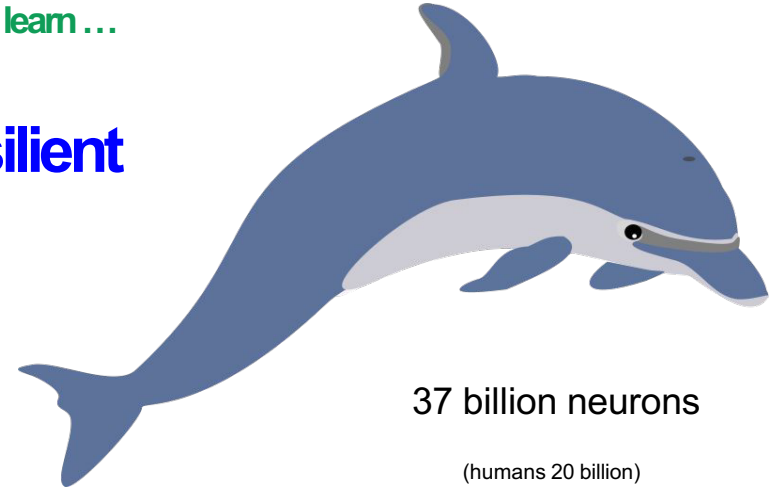


nature's brains can eat, fly, navigate, fight, communicate, play, learn …

.. and they're **resilient**

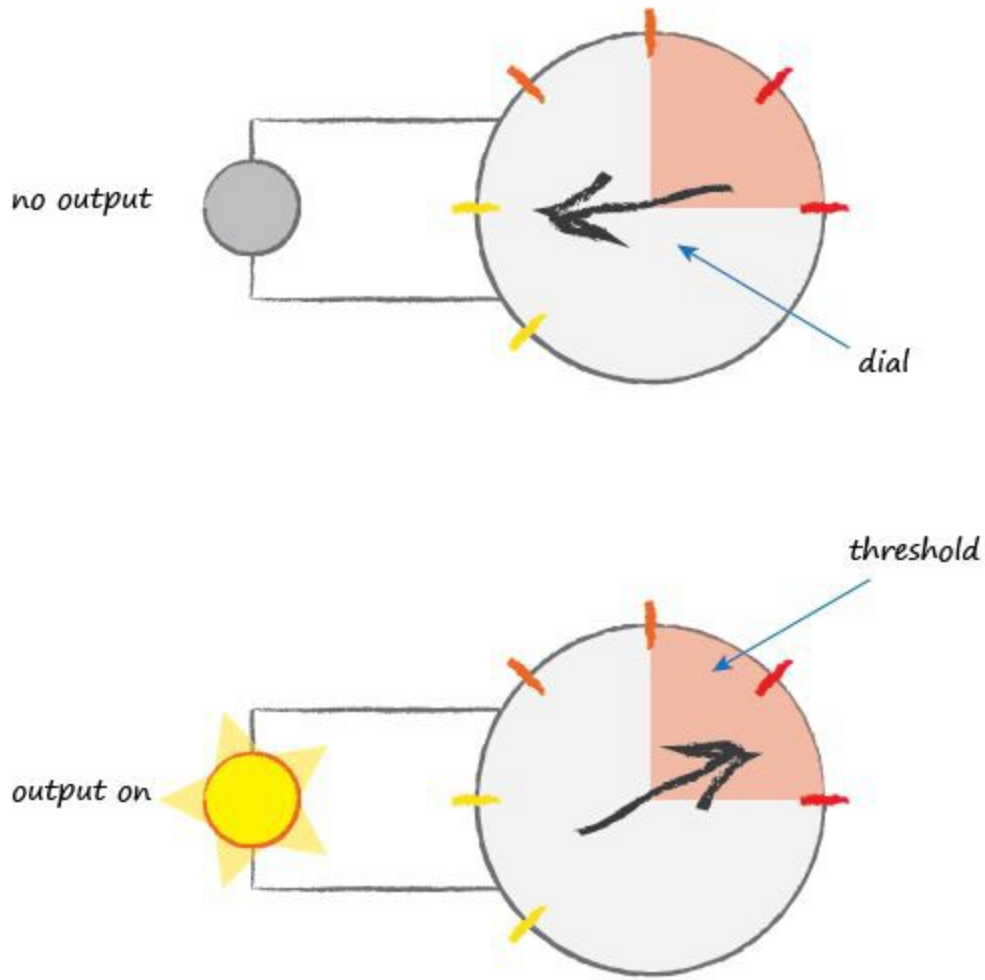


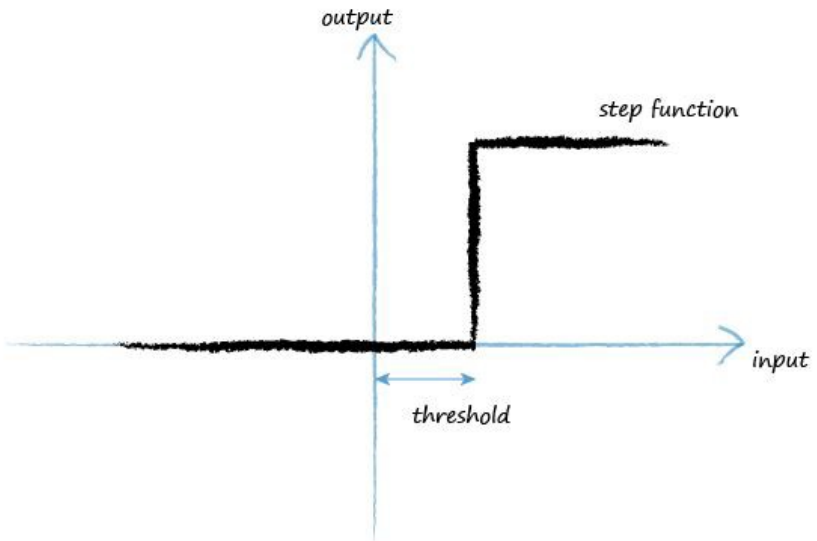
302 neurons

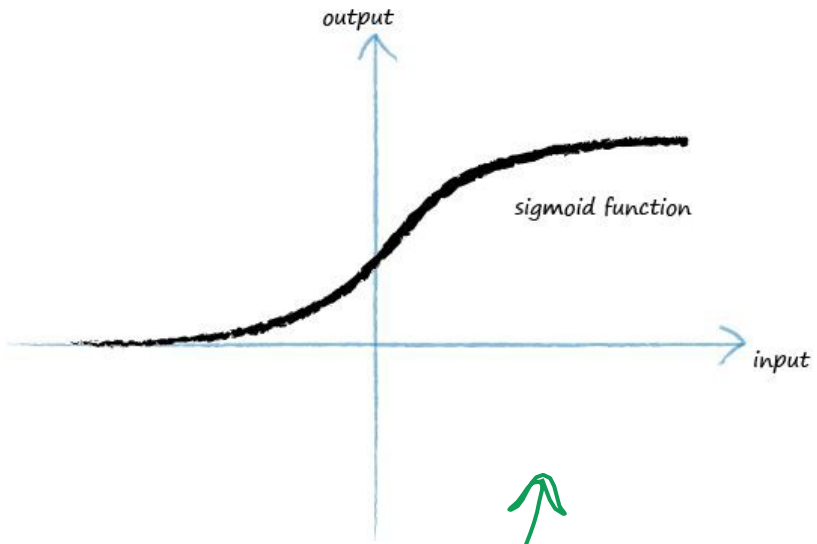


37 billion neurons

(humans 20 billion)

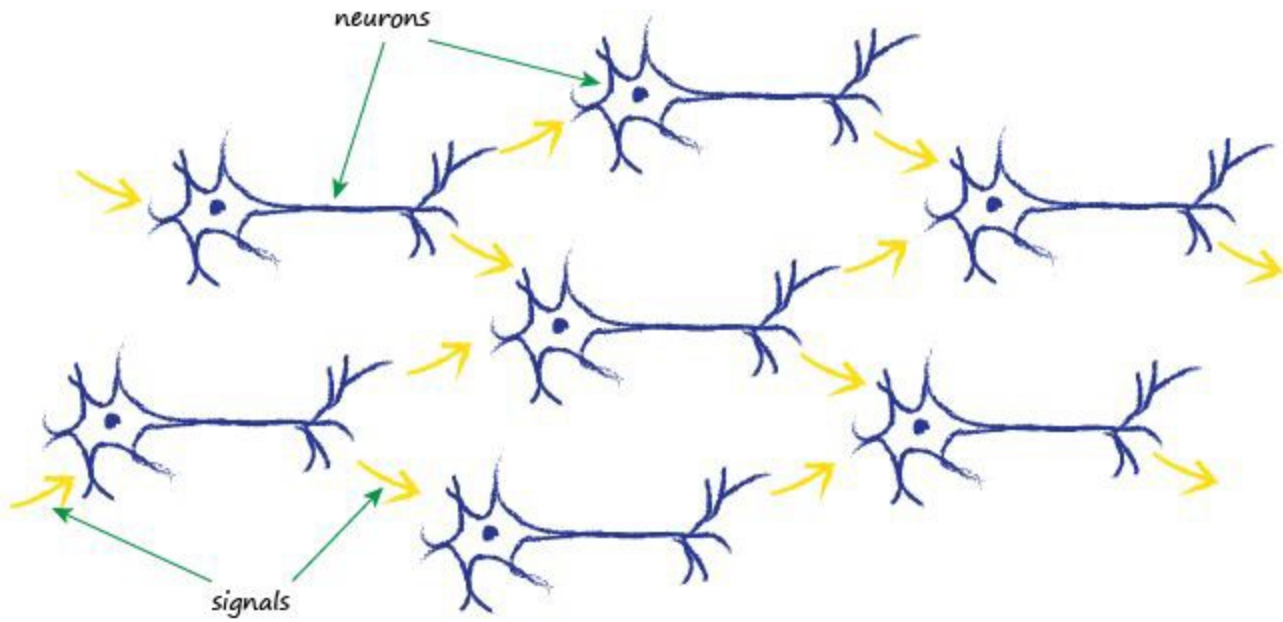

no output
dial
threshold
output on

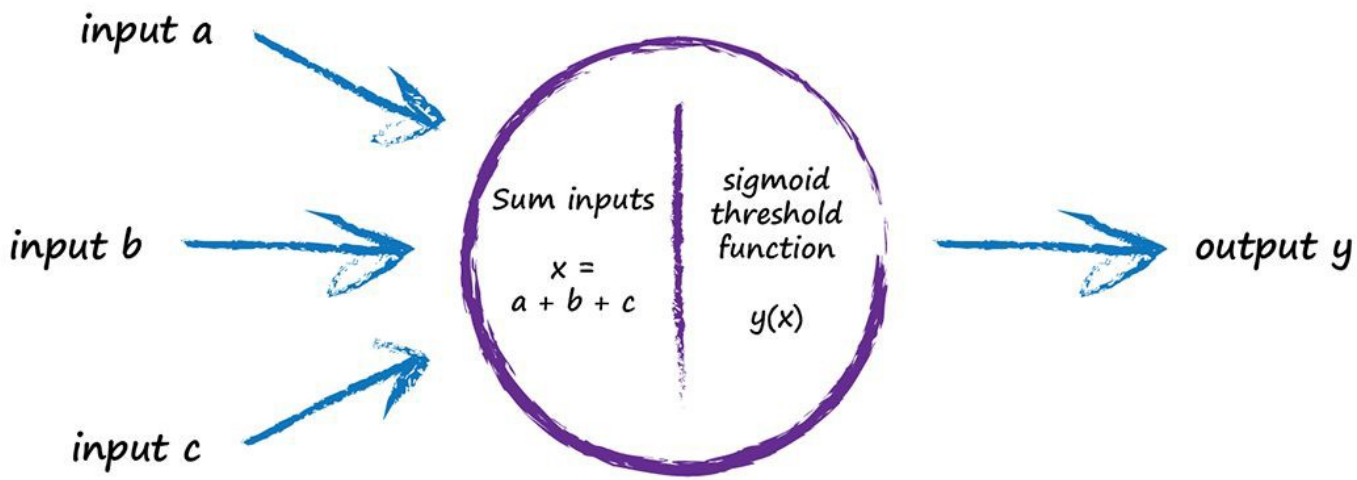


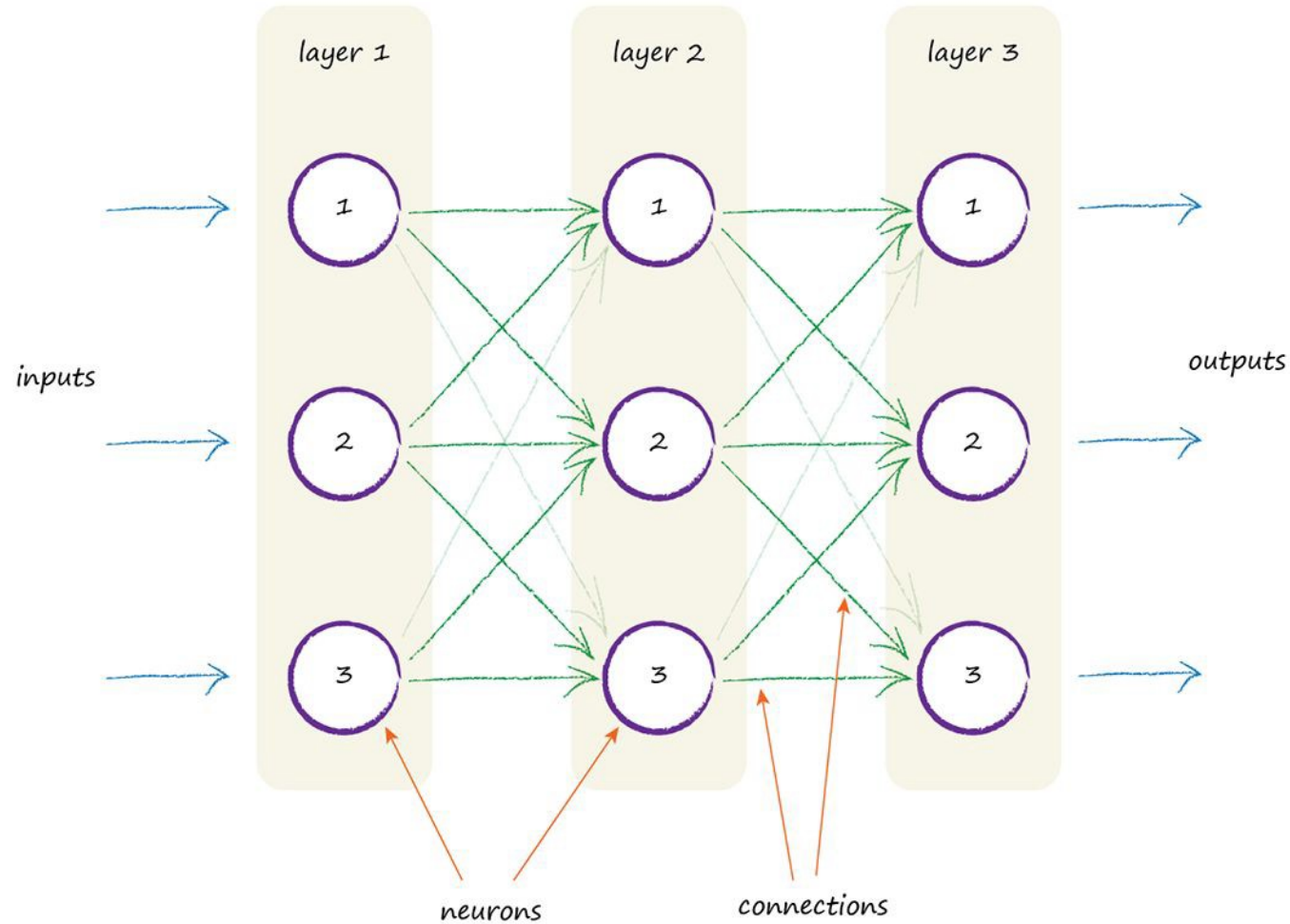


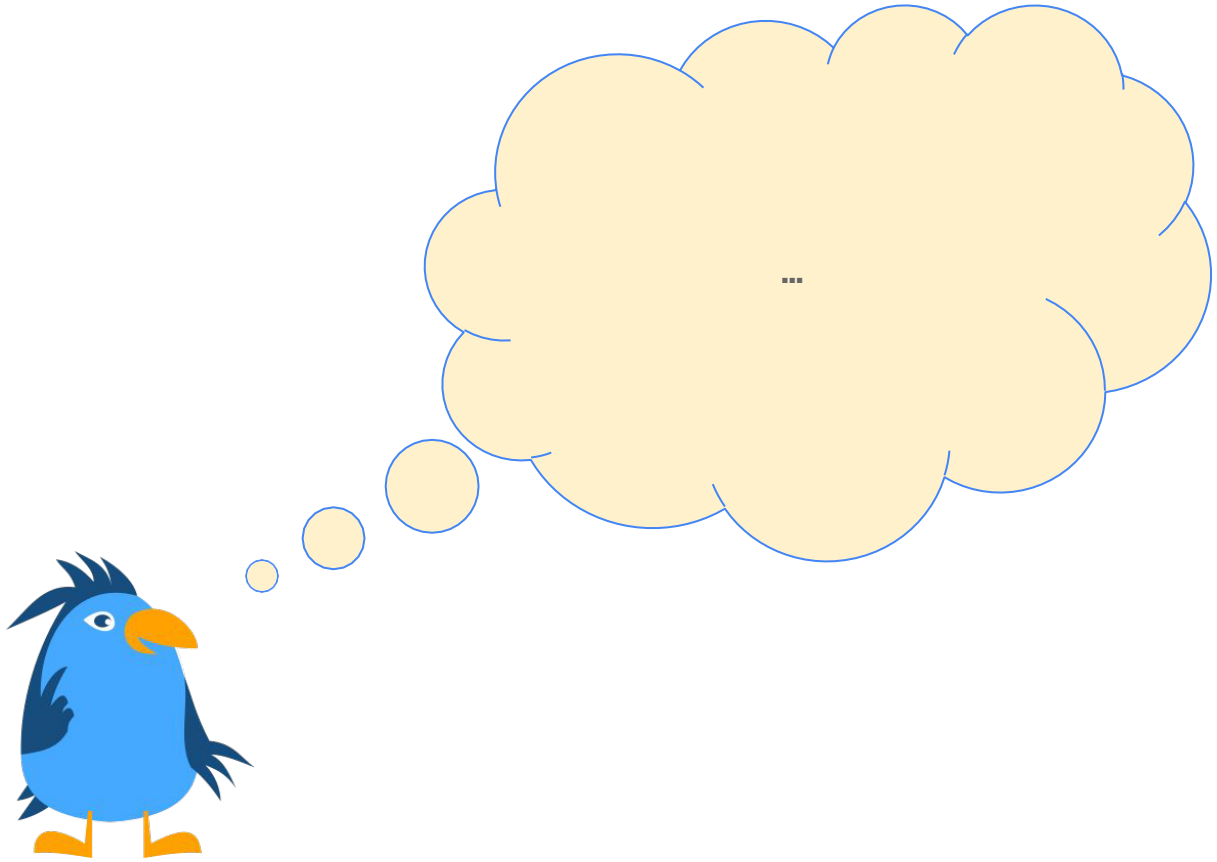
logistic function

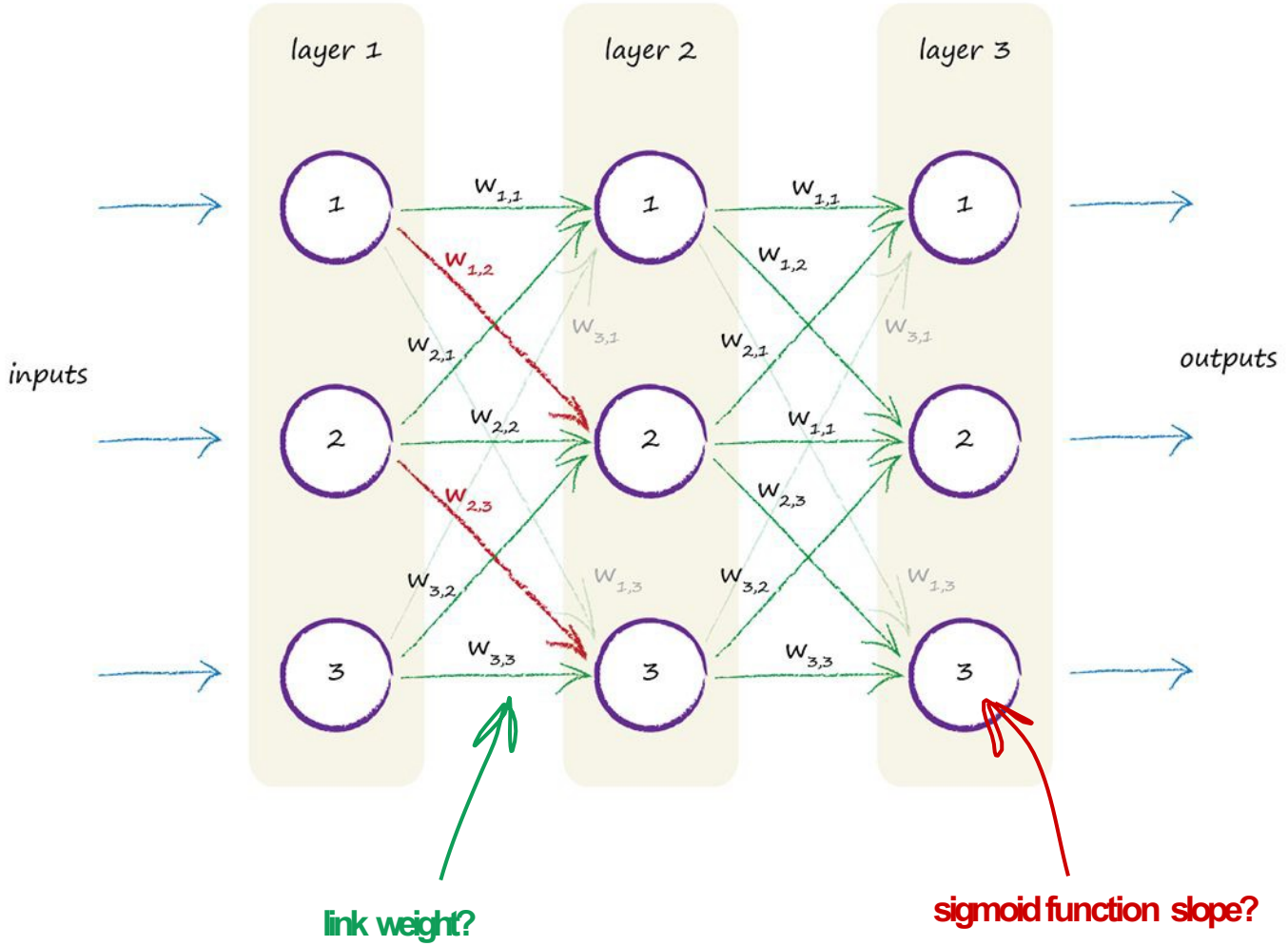
$y = 1 / (1 + e^{-x})$


neurons
signals


input a
input b
input c
Sum inputs
x =
a + b + c
sigmoid
threshold
function
y(x)
output y


layer 1
layer 2
layer 3
inputs
outputs
1
2
3
1
2
3
1
2
3
neurons
connections

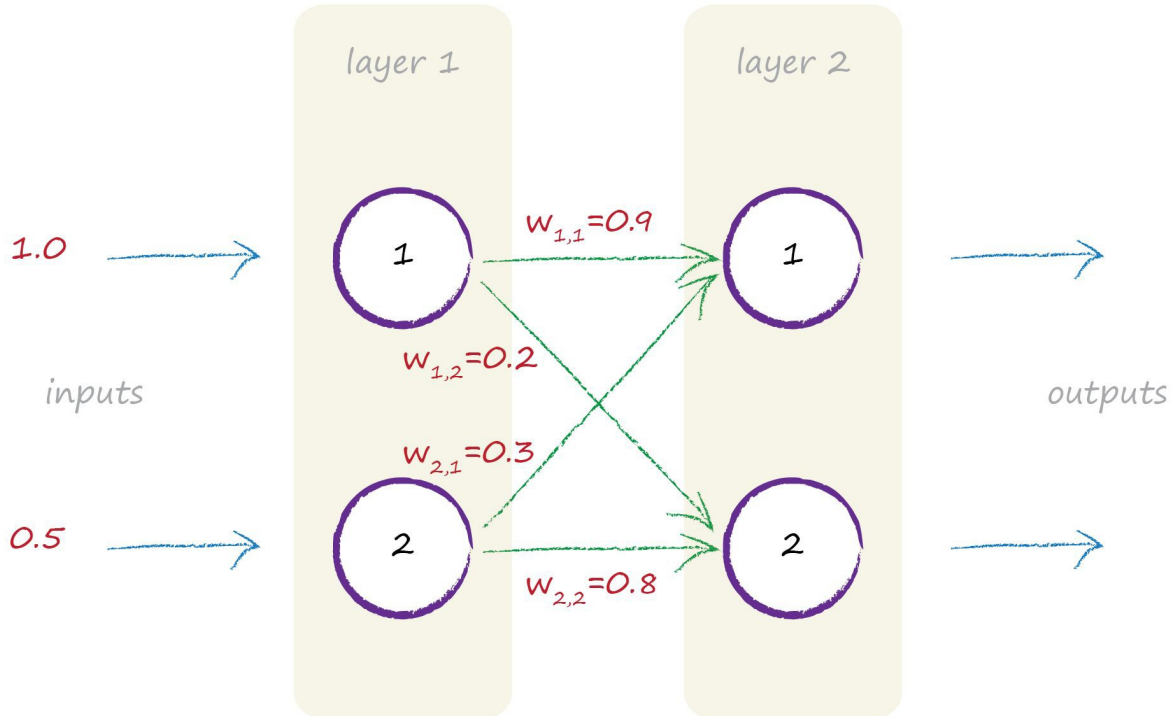


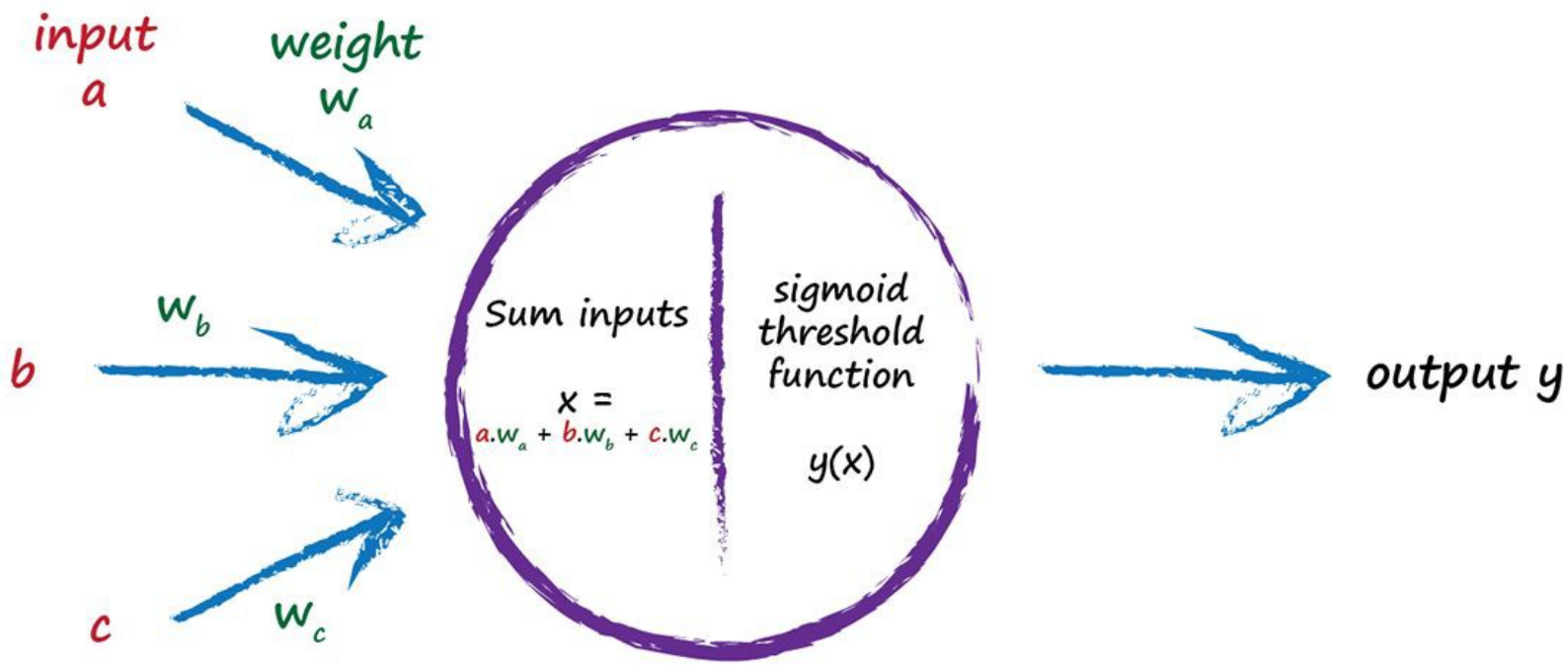

layer 1
layer 2
layer 3
inputs
outputs
1
2
3
$w_{1,1}$
$w_{1,2}$
$w_{2,1}$
$w_{2,2}$
$w_{2,3}$
$w_{3,1}$
$w_{3,2}$
$w_{3,3}$
$w_{1,3}$
link weight?
sigmoid function slope?

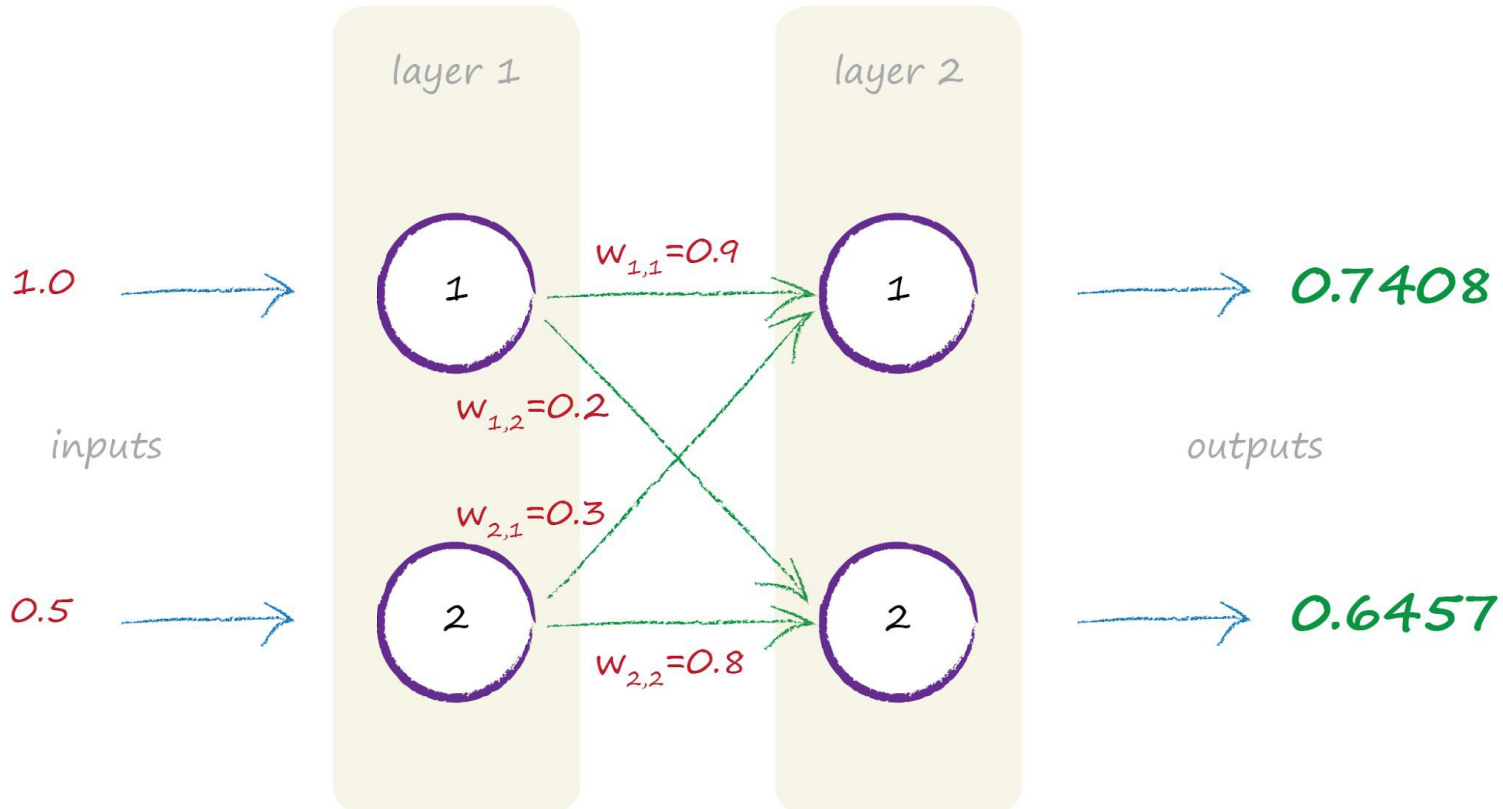
1. Natural brains can do **sophisticated** things, and are incredibly **resilient** to damage and imperfect signals .. unlike traditional computing.

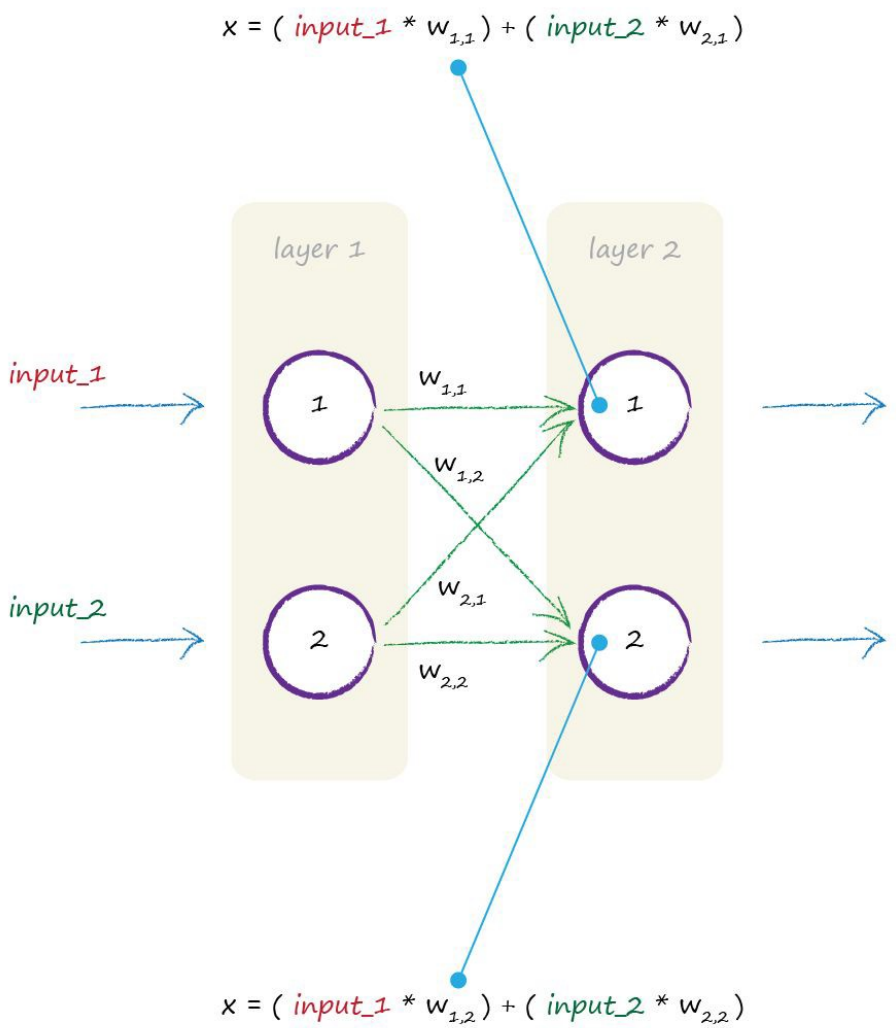
2. Trying to copy biological brains partly inspired **artificial neural networks**.

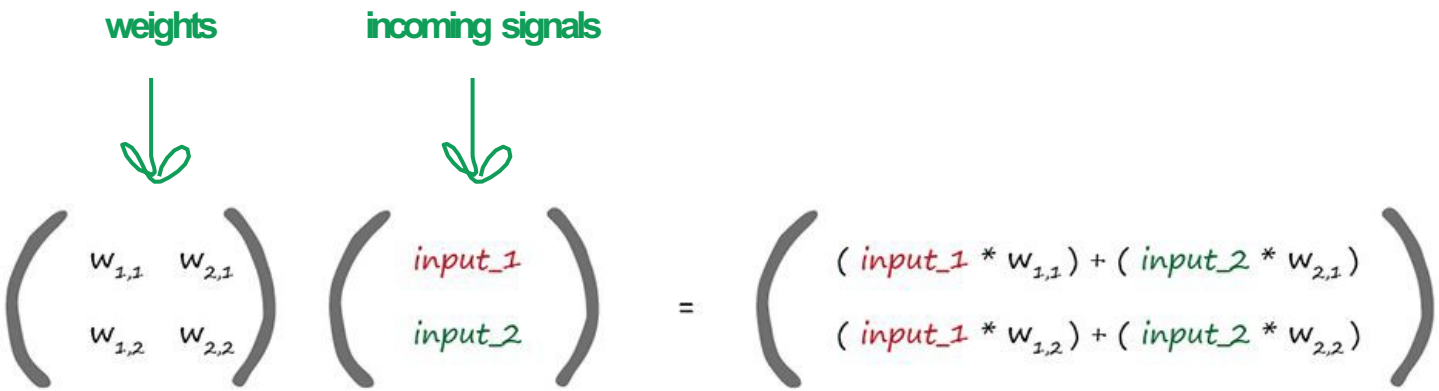
3. **Link weights** are the adjustable parameter - it's where the learning happens.


layer 1
layer 2
1.0
0.5
inputs
1
2
1
2
$w_{1,1}$=0.9
$w_{1,2}$=0.2
$w_{2,1}$=0.3
$w_{2,2}$=0.8
outputs


input
a
weight
w_a
w_b
b
Sum inputs
x =
$a.w_a + b.w_b + c.w_c$
sigmoid
threshold
function
y(x)
output y
c
w_c


layer 1
layer 2
1.0
0.5
inputs
1
2
1
2
$w_{1,1}$=0.9
$w_{1,2}$=0.2
$w_{2,1}$=0.3
$w_{2,2}$=0.8
outputs
0.7408
0.6457


x = (input_1 * w1,1) + (input_2 * w2,1)
layer 1
layer 2
input_1
input_2
1
2
1
2
w1,1
w1,2
w2,1
w2,2
x = (input_1 * w1,2) + (input_2 * w2,2)

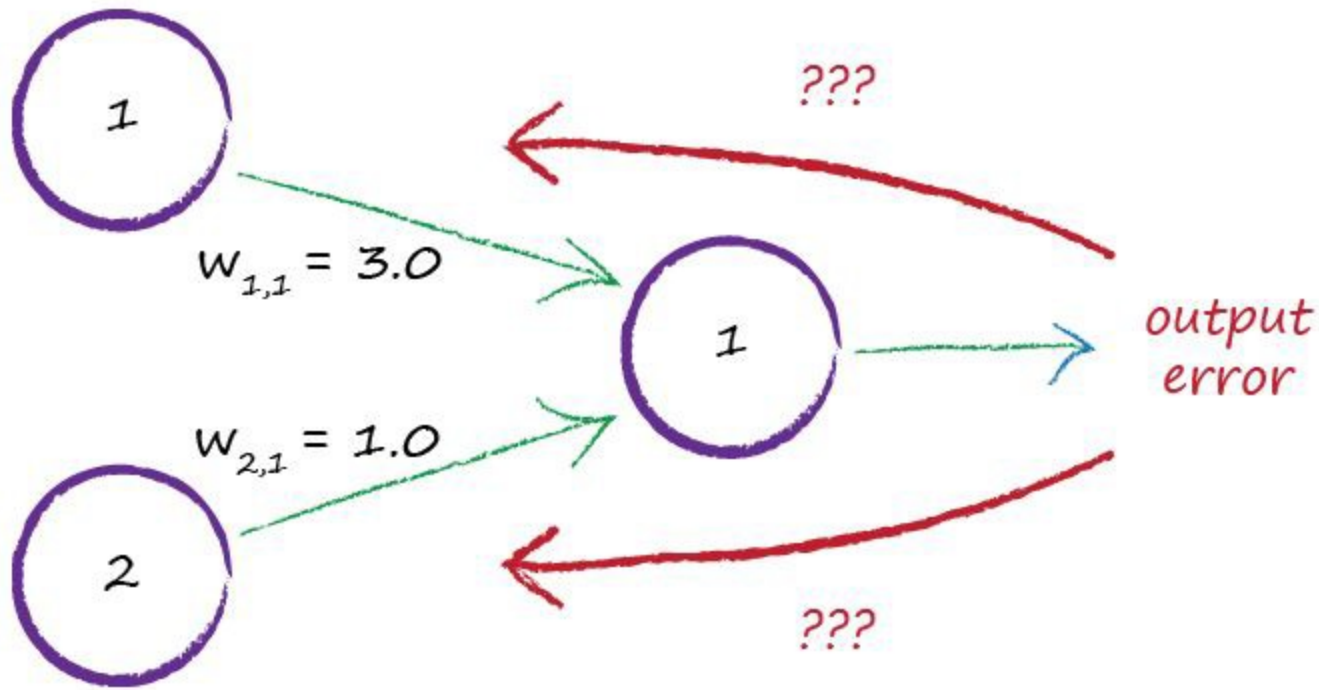


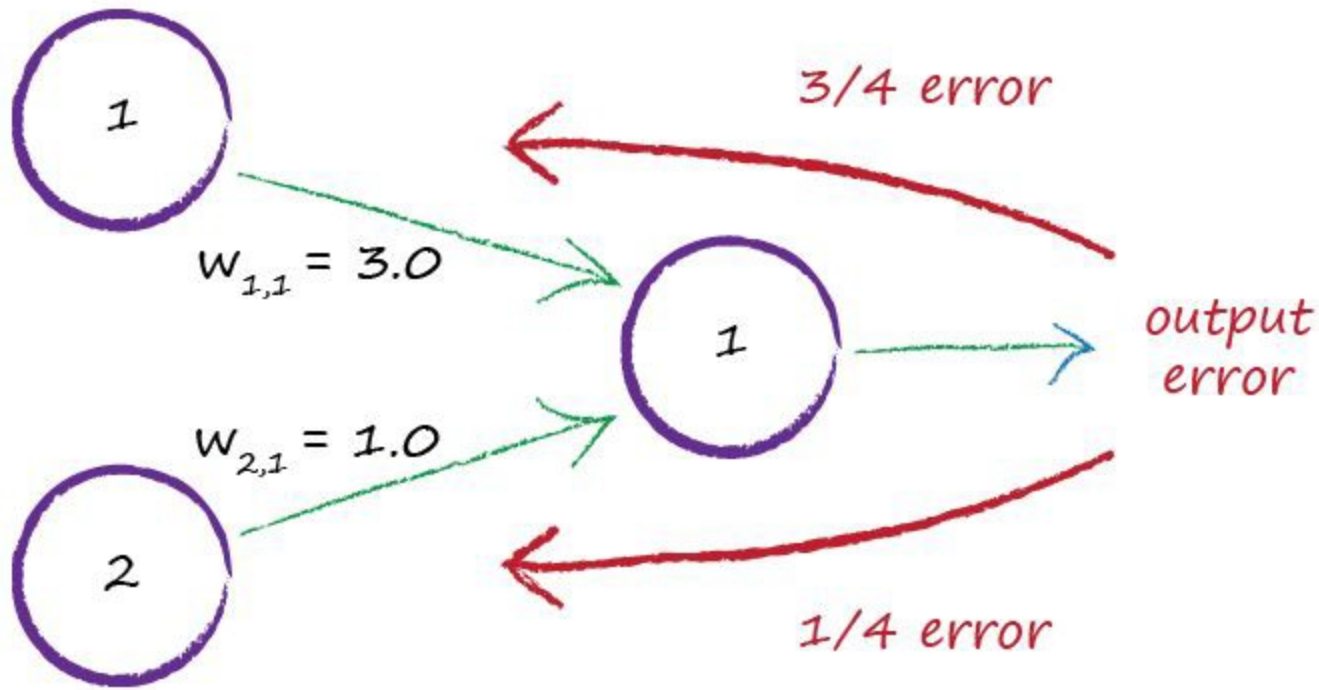
$$W \cdot I = X$$

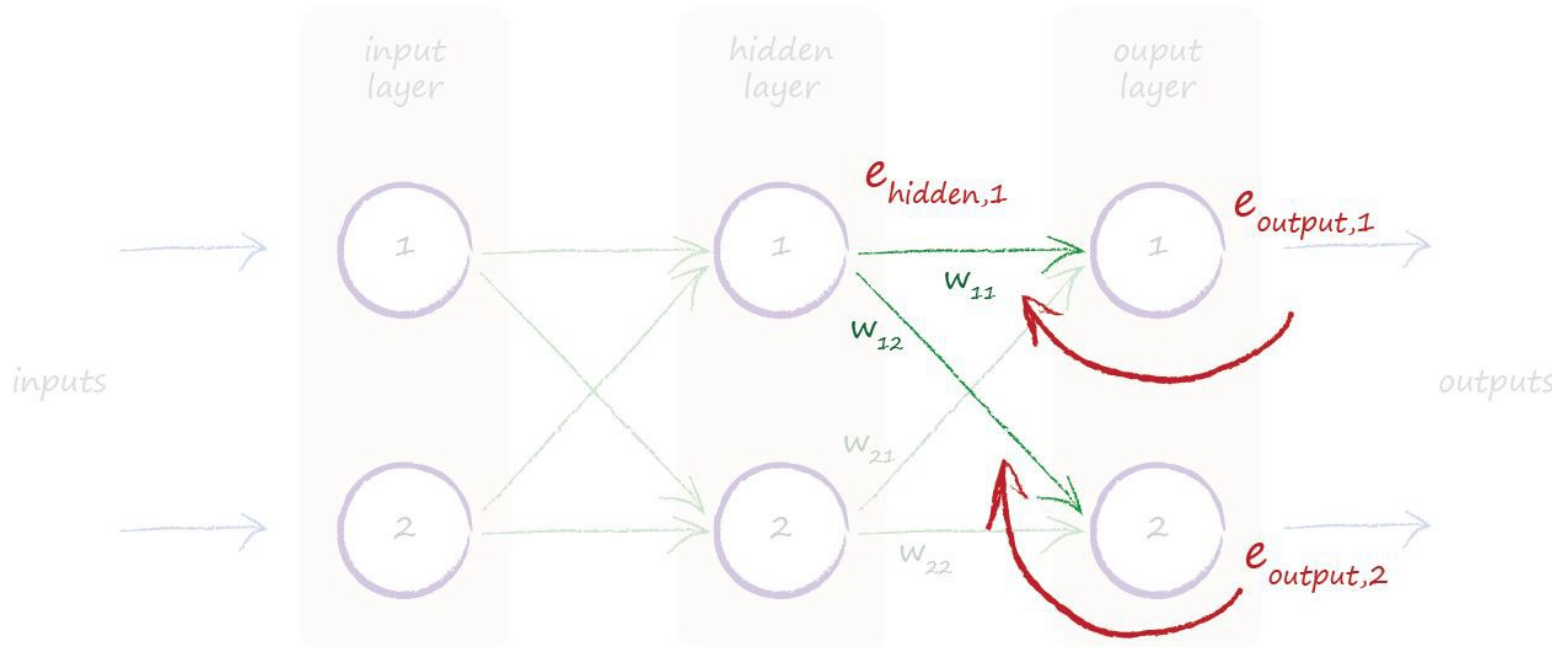
dot product

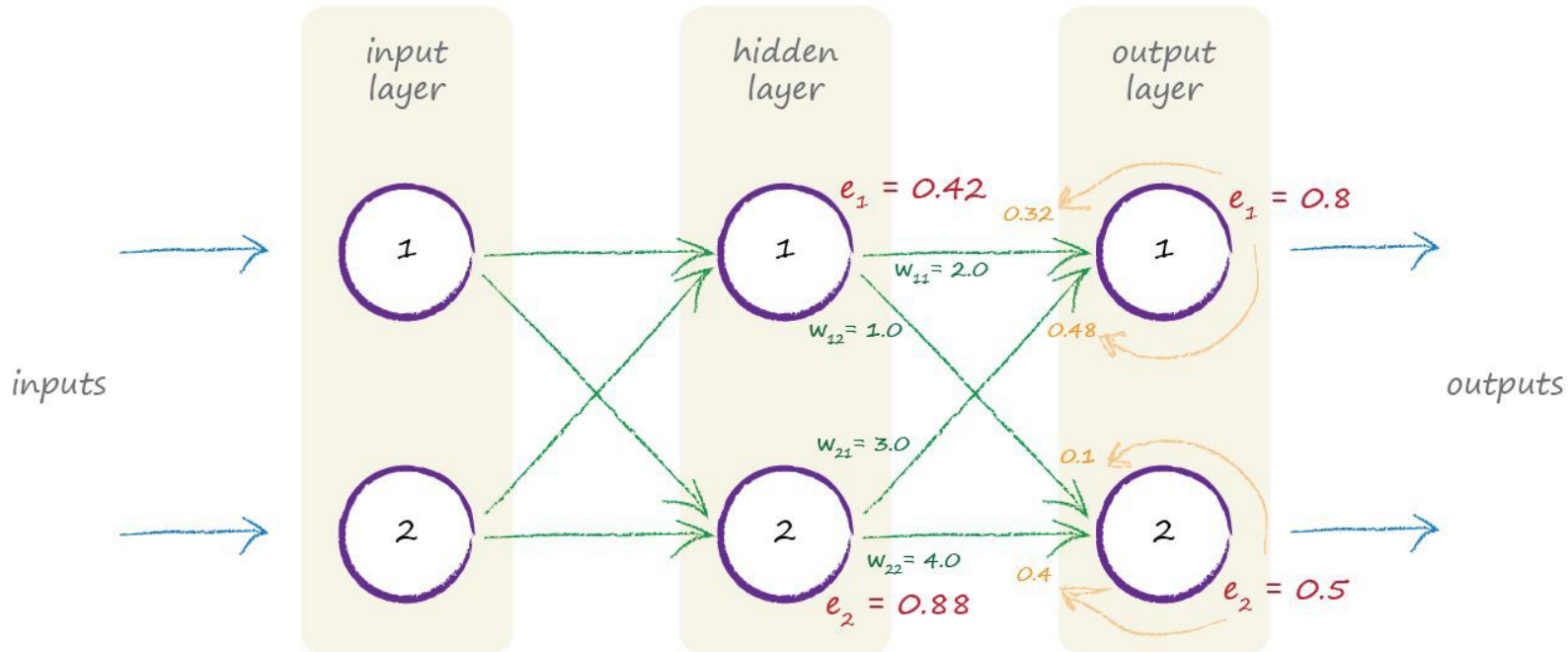
1. The many feedforward calculations can be expressed **concisely** as **matrix multiplication**, no matter what shape the network.

2. Some programming languages can do matrix multiplication really **efficiently** and **quickly**.


1
???
$w_{1,1} = 3.0$
1
output
error
$w_{2,1} = 1.0$
2
???


1
2
1
$w_{1,1} = 3.0$
$w_{2,1} = 1.0$
3/4 error
output error
1/4 error


input layer
hidden layer
ouput layer
inputs
outputs
1
2
1
2
1
2
$e_{hidden,1}$
$e_{output,1}$
$e_{output,2}$
w_{11}
w_{12}
w_{21}
w_{22}


input layer
hidden layer
output layer
inputs
outputs
1
2
1
2
1
2
e_1 = 0.42
e_2 = 0.88
e_1 = 0.8
e_2 = 0.5
w_{11}= 2.0
w_{12}= 1.0
w_{21}= 3.0
w_{22}= 4.0
0.32
0.48
0.1
0.4

$$error_{hidden} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

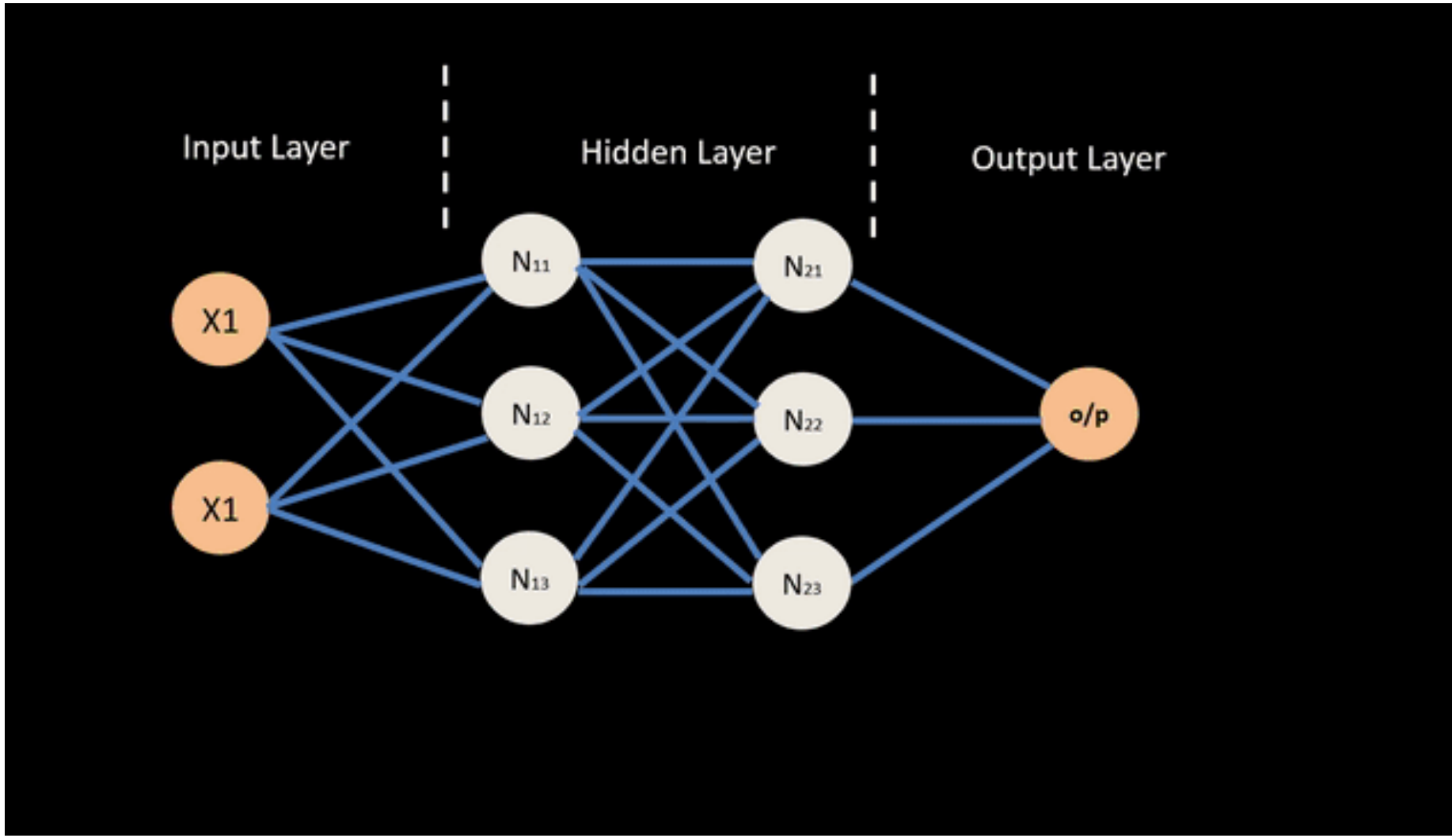
$$error_{hidden} = w^{T}_{hidden\_output} \cdot error_{output}$$

1. Remember we use the **error** to guide how we refine a model's parameter - link weights.

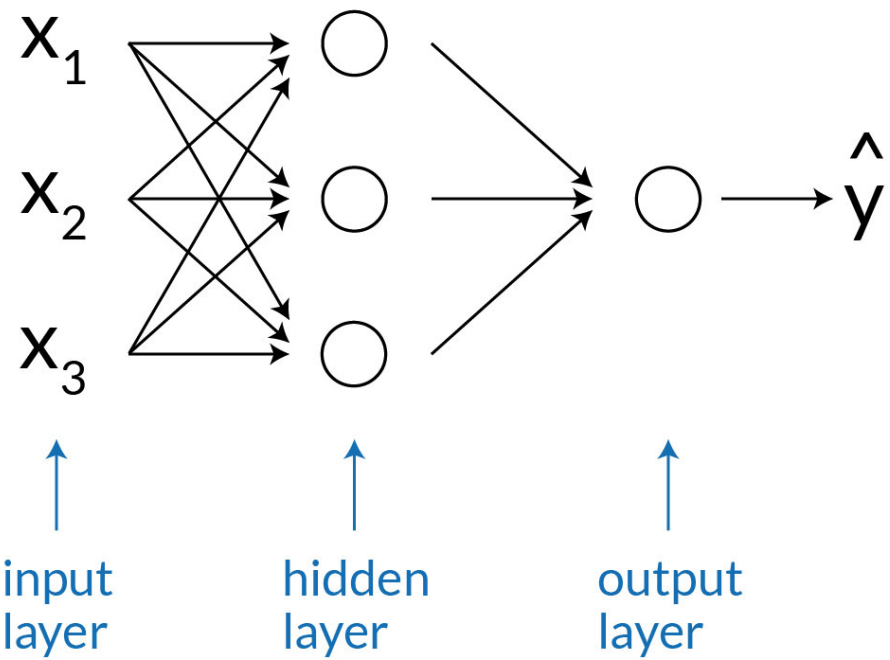
2. The error at the output nodes is easy - the difference between the **desired** and **actual** outputs.

3. The error at internal nodes isn't obvious. A **heuristic** approach is to split it in **proportion** to the link weights.

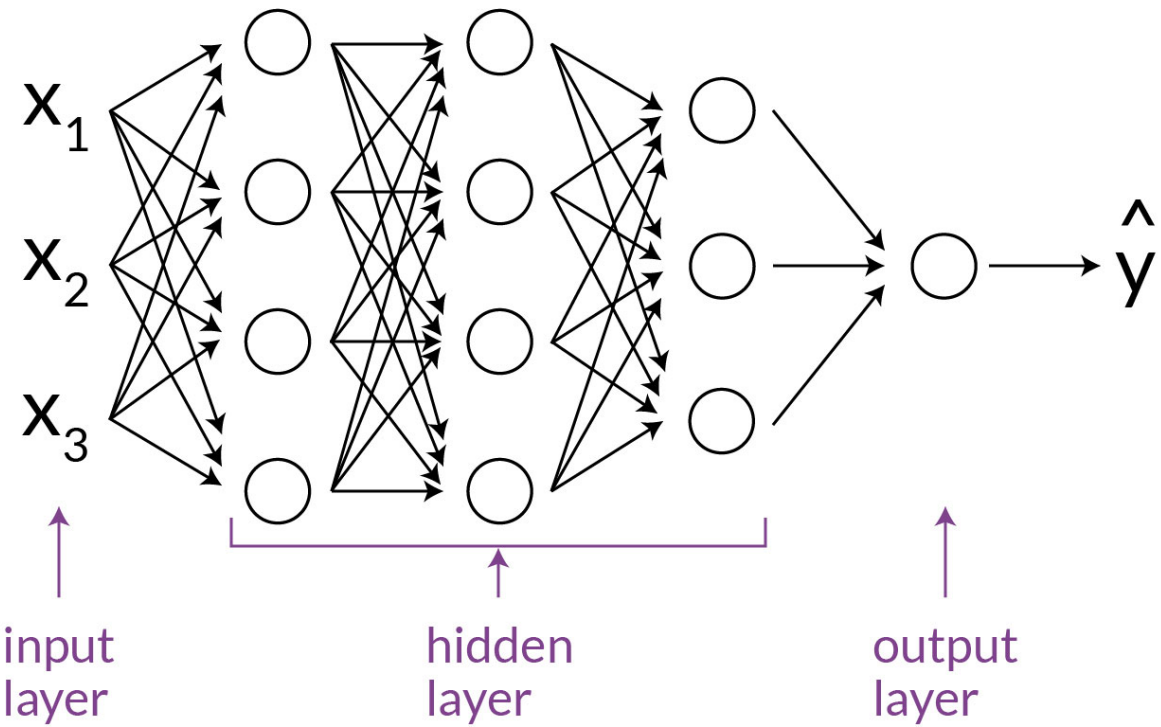
4. … and back propagating the error can be expressed as a **matrix** multiplication too!


Input Layer
Hidden Layer
Output Layer
X1
X1
N11
N12
N13
N21
N22
N23
o/p

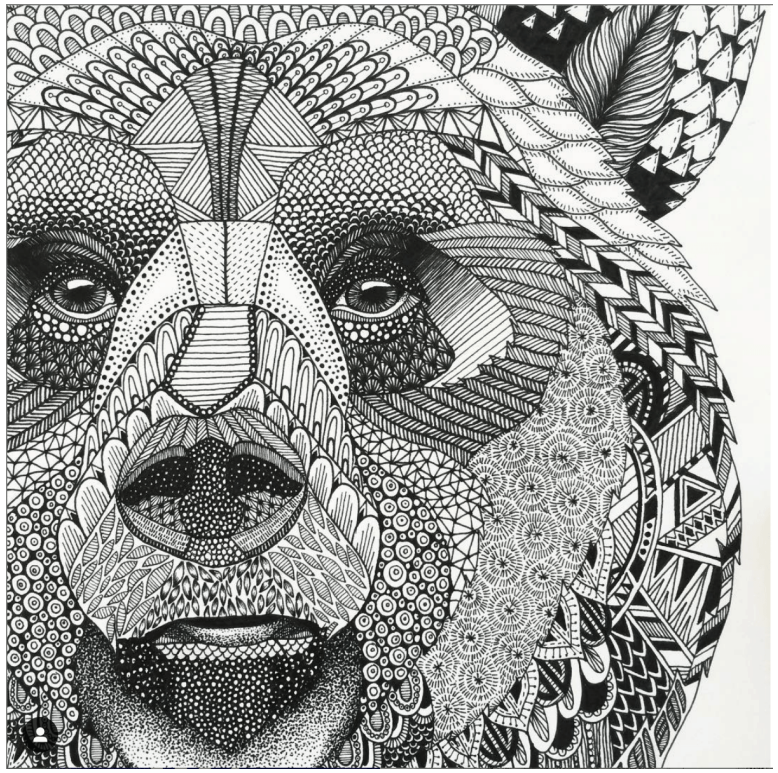
Shallow Neural Network



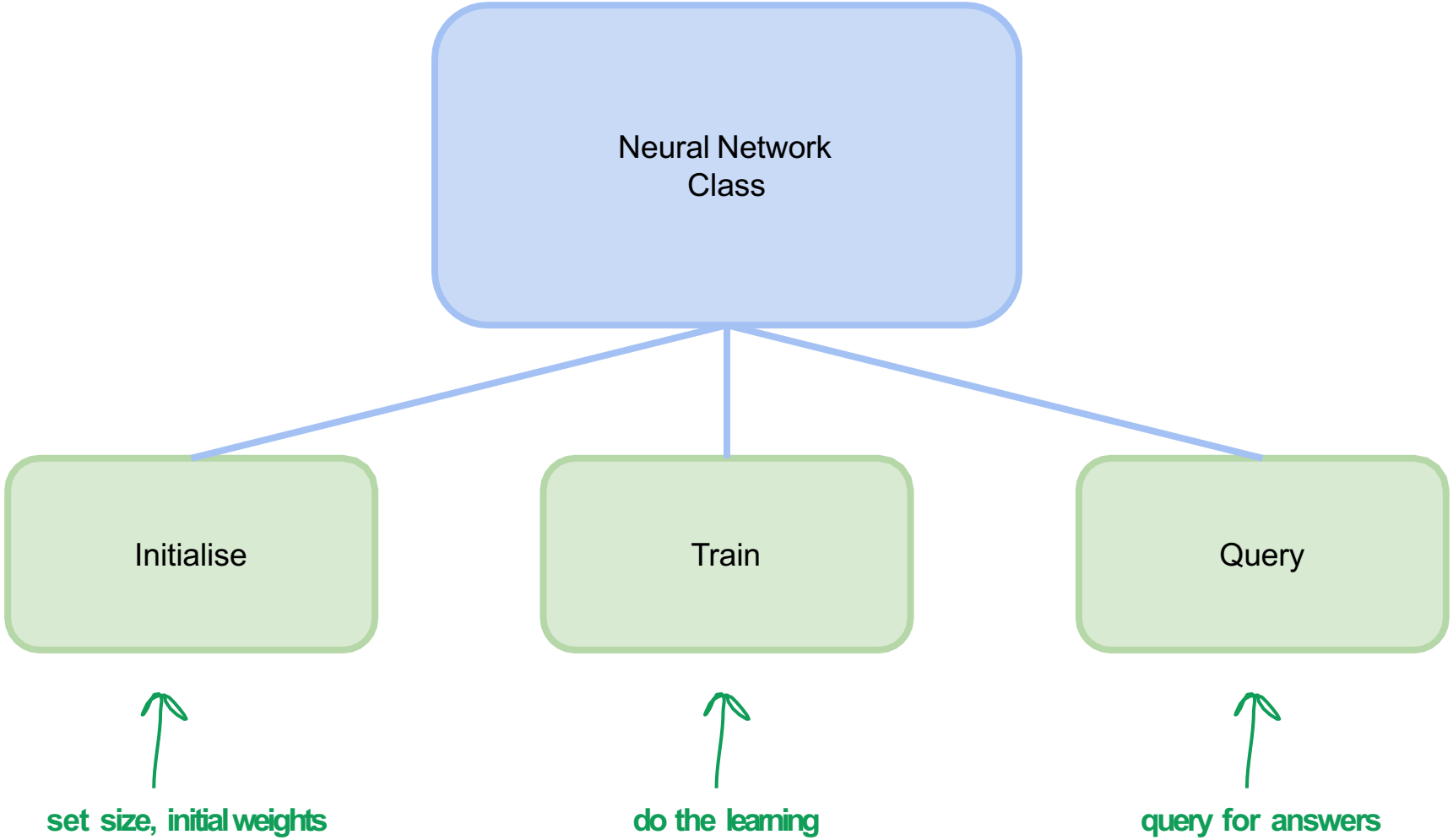
Deep Neural Network



{04}
Do it Yourself



and Functions



Python

matrix maths

numpy scipy

matplotlib

notebook

Function - Initialise

```
# initialise the neural network
def __init_(self, inputnodes,  hiddennodes,  outputnodes,  learningrate):  # set  number
    of nodes in each input, hidden,  output  layer  self.inodes = inputnodes
    self.hnodes = hiddennodes
    self.onodes = outputnodes

    # link weight matrices, wih and who
    # weights inside the arrays are w_i_j, where link is from node i to node j in the next layer  # w11 w21
    # w12 w22 etc
    self.wih = numpy.random.normal(0.0, pow(self.hnodes, -0.5), (self.hnodes, self.inodes))  self.who =
    numpy.random.normal(0.0, pow(self.onodes, -0.5), (self.onodes, self.hnodes))

    # learning rate
    self.lr = learningrate

    # activation function is the sigmoid function
    self.activation_function = lambda x: scipy.special.expit(x)

    pass
```

random initial weights

Function - Query

combined weighted signals into hidden layer

then sigmoid applied

```
# query the neural network
def query(self, inputs_list):
    # convert inputs list to 2d array
    inputs = numpy.array(inputs_list, ndmin=2).T

    # calculate signals into hidden layer
    hidden_inputs = numpy.dot(self.wih, inputs)
    # calculate the signals emerging from hidden layer
    hidden_outputs = self.activation_function(hidden_inputs)

    # calculate signals into final output layer
    final_inputs = numpy.dot(self.who, hidden_outputs)
    # calculate the signals emerging from final output layer
    final_outputs = self.activation_function(final_inputs)

    return final_outputs
```

numpy.dot()

similar for output layer

Function - Train

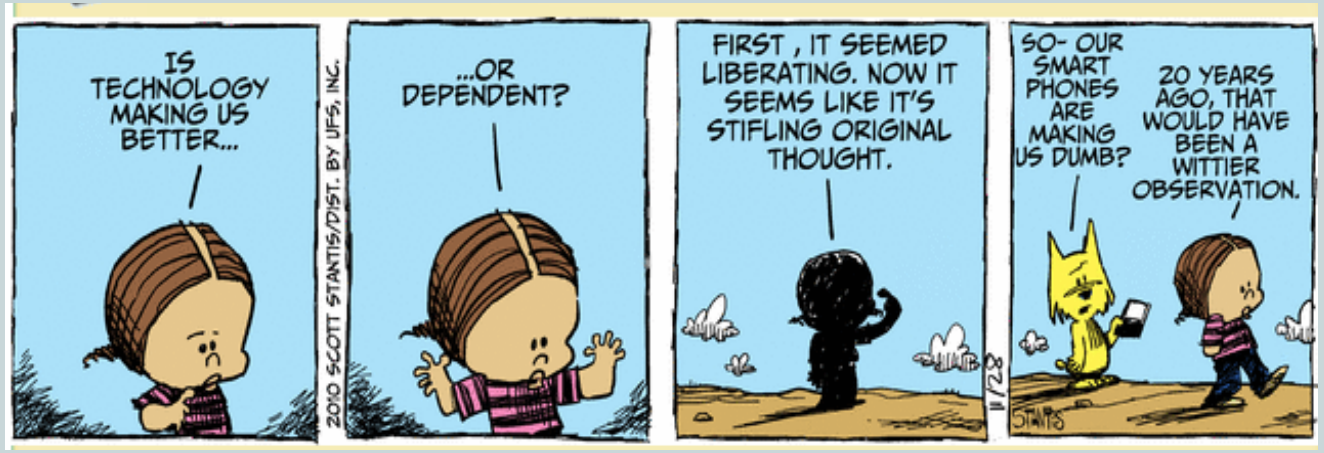
```
    #train the neural network
    def train(self, inputs_list, targets_list):
        #convert inputs list to 2d array
        inputs = numpy.array(inputs_list, ndmin=2).T
        targets = numpy.array(targets_list, ndmin=2).T

        #calculate signals into hidden layer
        hidden_inputs = numpy.dot(self.wih, inputs)
        #calculate the signals emerging from hidden layer
        hidden_outputs = self.activation_function(hidden_inputs)

        #calculate signals into final output layer
        final_inputs = numpy.dot(self.who, hidden_outputs)
        #calculate the signals emerging from final output layer
        final_outputs = self.activation_function(final_inputs)

        #output layer error is the (target - actual)
        output_errors = targets - final_outputs
        #hidden layer error is the output_errors, split by weights, recombined at hidden nodes
        hidden_errors = numpy.dot(self.who.T, output_errors)

        #update the weights for the links between the hidden and output layers
        self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)),
numpy.transpose(hidden_outputs))

        #update the weights for the links between the input and hidden layers
        self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)),
numpy.transpose(inputs))

        pass
```

same feed forward as before

output layer errors

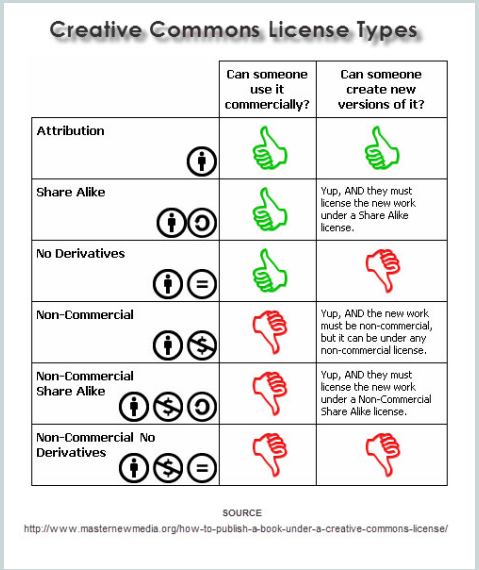
hidden layer errors

update weights



This lesson was developed by:

Robert Frans van der Willigen
CMD, Hogeschool Rotterdam
OKT 2020



http://empoweringthenatives.edublogs.org/2012/03/15/creative-commons-licenses/

http://creativecommons.org/licenses/


INFOGRAPHICS
the functional art
alberto cairo
TECHNOLOGY TRENDS TO WATCH
Christian Leborg
Visual Grammar
RDM
the analytics revolution
Bill Franks
MADE to STICK
generative art
matt pearson
EMPATHIC THINGS
INTERNET OF THINGS!
MAKERS
THE NEW INDUSTRIAL REVOLUTION
CHRIS ANDERSON
MARITIEME MAAKINDUSTRIE TERUG NAAR RDM
WebGL
TV AND VIDEO
An analysis of evolving consumer habits
THE DISRUPTION MACHINE
Learning to Program with MATLAB
BUILDING GUI TOOLS
Port Management Program
R Data Analysis
10 TRENDS FOR 2014
A giant cage
resonate
Design for Information
HACKATHON
OPEN HAVEN
1 VOLLE DAG
7 SEPTEMBER
WERELDHAVENDAGEN 2013
the myths of innovation
scott berkun
O'REILLY
Cool Infographics
Effective Communication with Data Visualization and Design
The VISUAL
PHIL SIMON
INVENT TO LEARN
Making, Tinkering, and Engineering in the Classroom
IORE SECRETS with Big Data Analytics
OUR WORK HERE IS DONE
VISIONS OF A ROBOT ECONOMY
Processing
A Programming Handbook for Visual Designers and Artists
THE WALL STREET JOURNAL.
Guide to Information Graphics
Dona M. Wong
Customer Journey Mapping
Wearable Technology Design
The Hidden Pitfall in your Design
Make: Books
Making Things See
Learn by Discovery
TEACHING that STICKS
Chip Heath & Dan Heath
Protecting Data and Privacy in the Cloud
DISCOVER METEOR
The Internet of Things
Business Adapts to a New Style of Computer
Making Things Talk
EVOLUTION OF AGILE