A Gentle Introduction to Neural Networks
(with Python)

Tariq Rashid @rzeta0

July 2018

Background

Ideas

DIY

Handwriting

Background

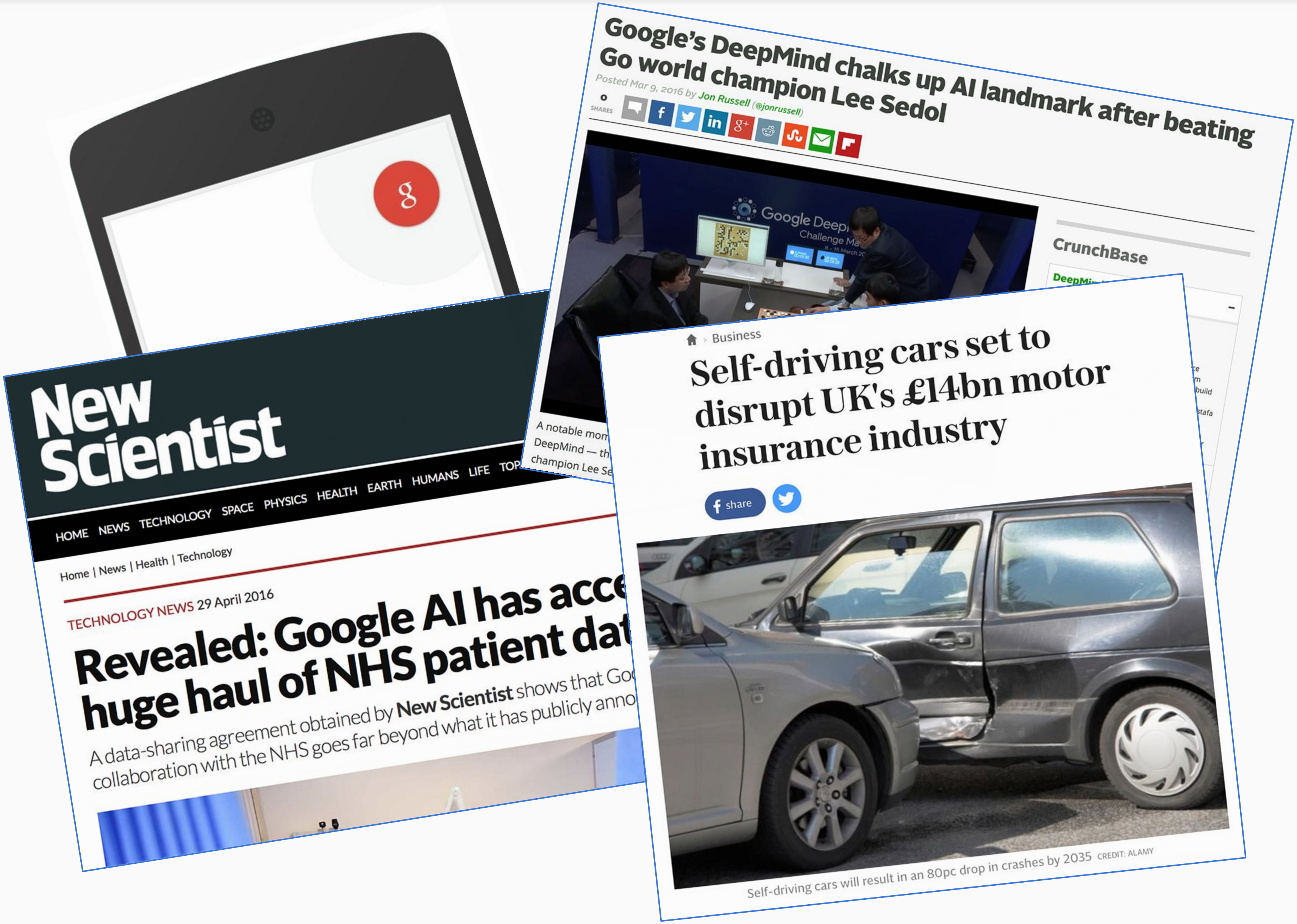
Start With Two Questions

locate people in this photo

add these numbers

```
    2403343781289312
+   2843033712837981
+   2362142787897881
+   3256541312323213
+   9864479802118978
+   8976677987987897
+   8981257890087988

                 = ?
```

AI is Huge!



Google's DeepMind chalks up AI landmark after beating Go world champion Lee Sedol
Posted Mar 9, 2016 by Jon Russell (@jonrussell)
CrunchBase
New Scientist
HOME NEWS TECHNOLOGY SPACE PHYSICS HEALTH EARTH HUMANS LIFE
Home | News | Health | Technology
TECHNOLOGY NEWS 29 April 2016
Business
Self-driving cars set to disrupt UK's £14bn motor insurance industry
share
Self-driving cars will result in an 80pc drop in crashes by 2035 CREDIT: ALAMY

Google's and Go

nature International weekly journal of science

ARTICLE PREVIEW

view full access options ▸

NATURE | ARTICLE

日本語要約

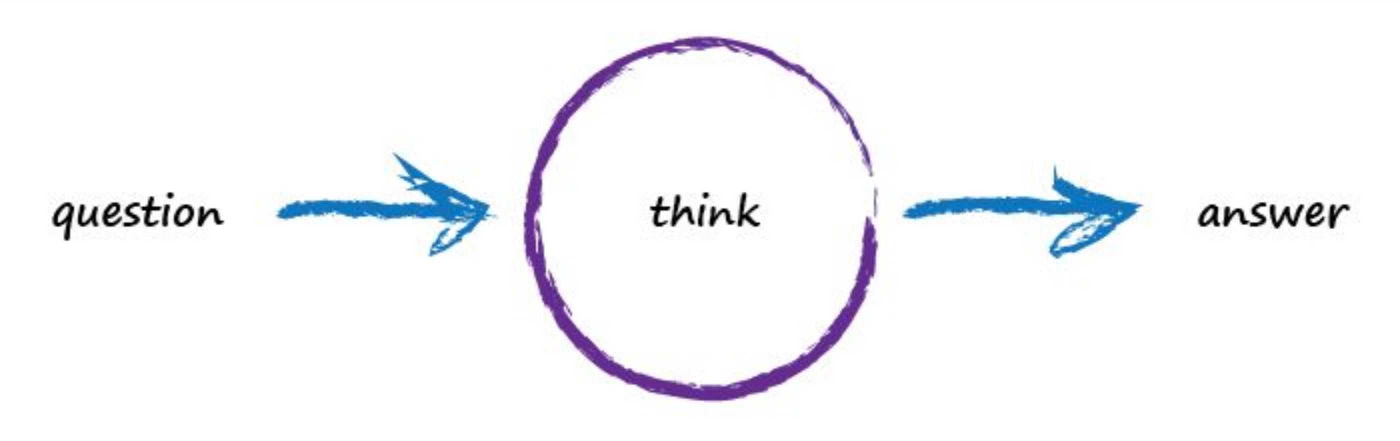
Mastering the game of Go with deep neural networks and tree search

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel & Demis Hassabis

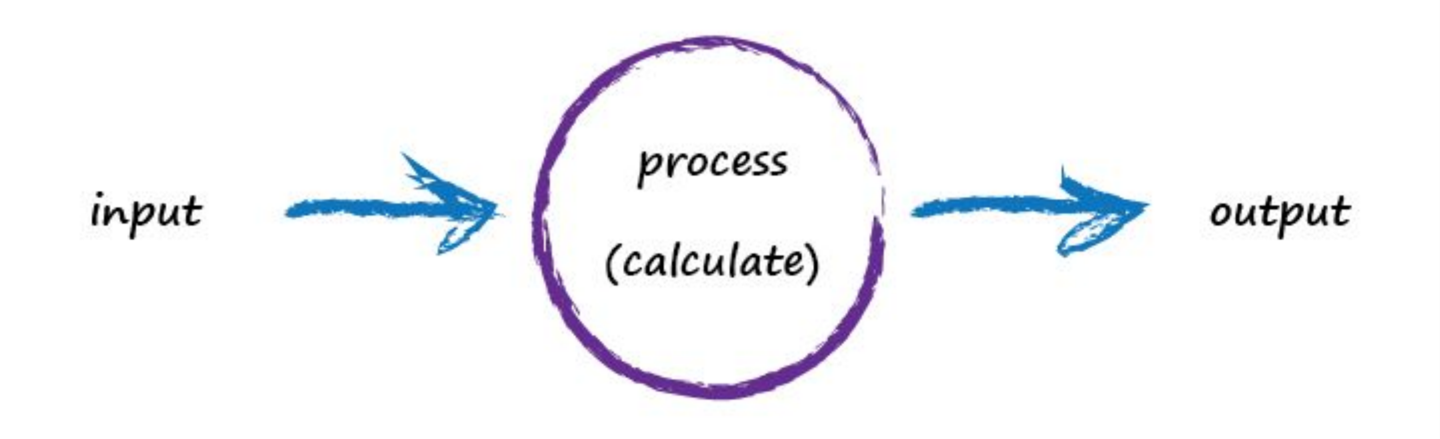
Affiliations | Contributions | Corresponding authors

Ideas

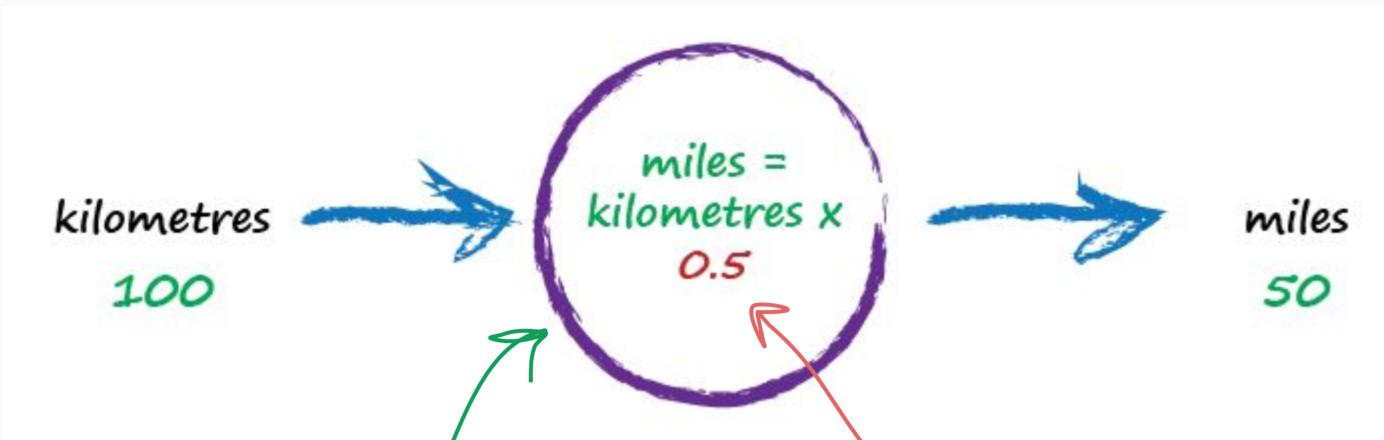
Simple Predicting Machine



Simple Predicting Machine



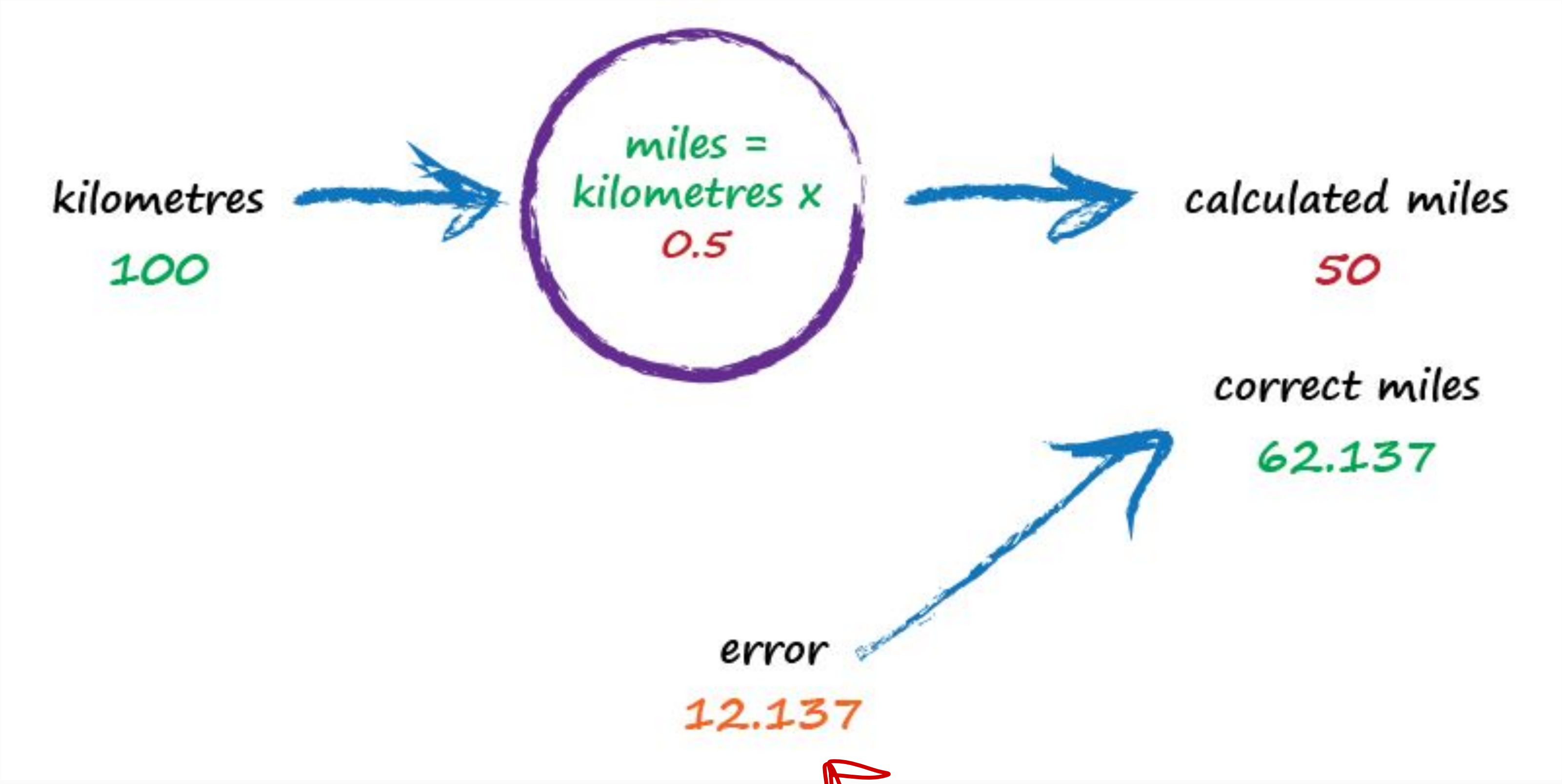
Kilometres to Miles



random starting parameter

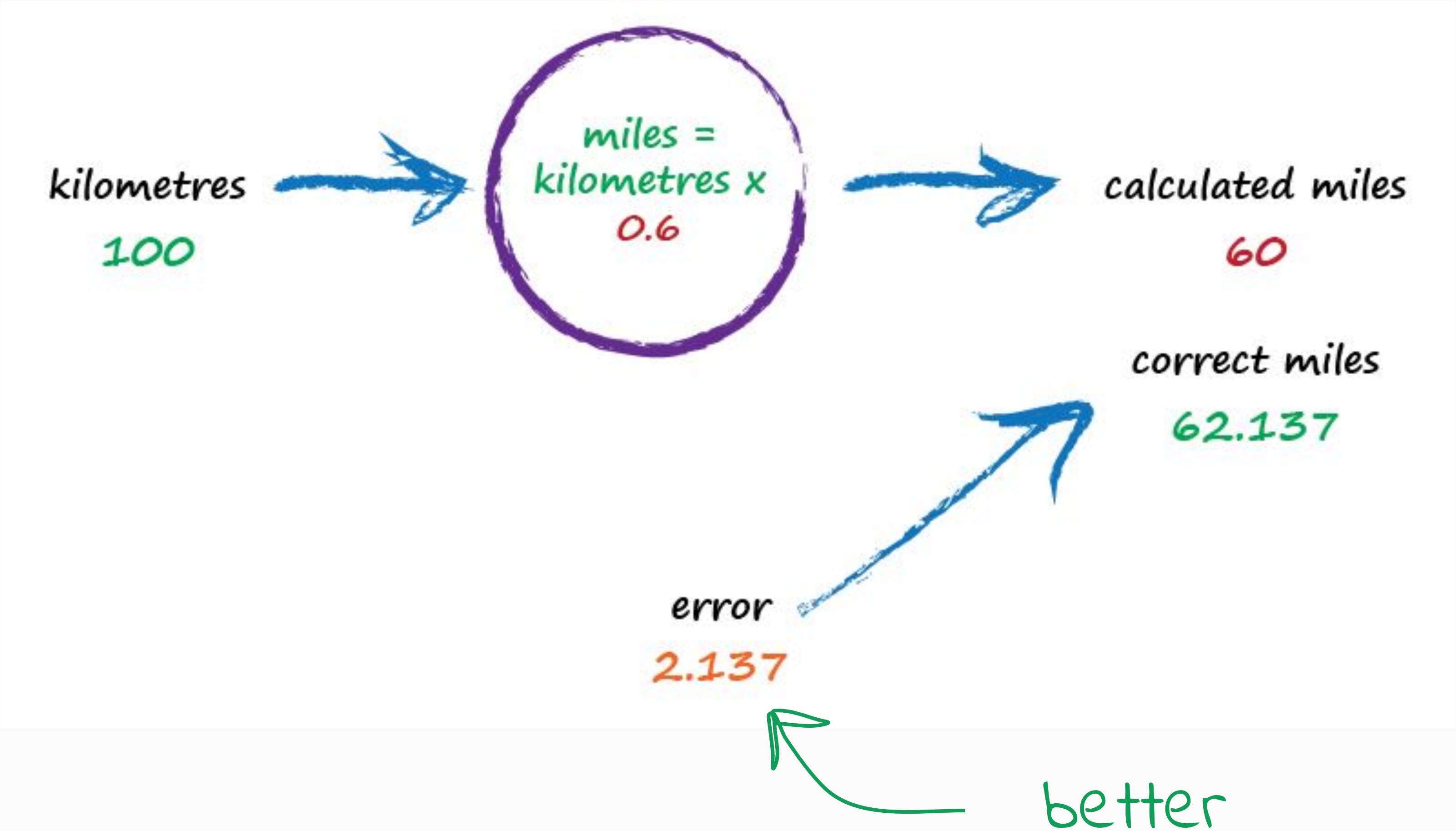
try a model - this one is linear

Kilometres to Miles

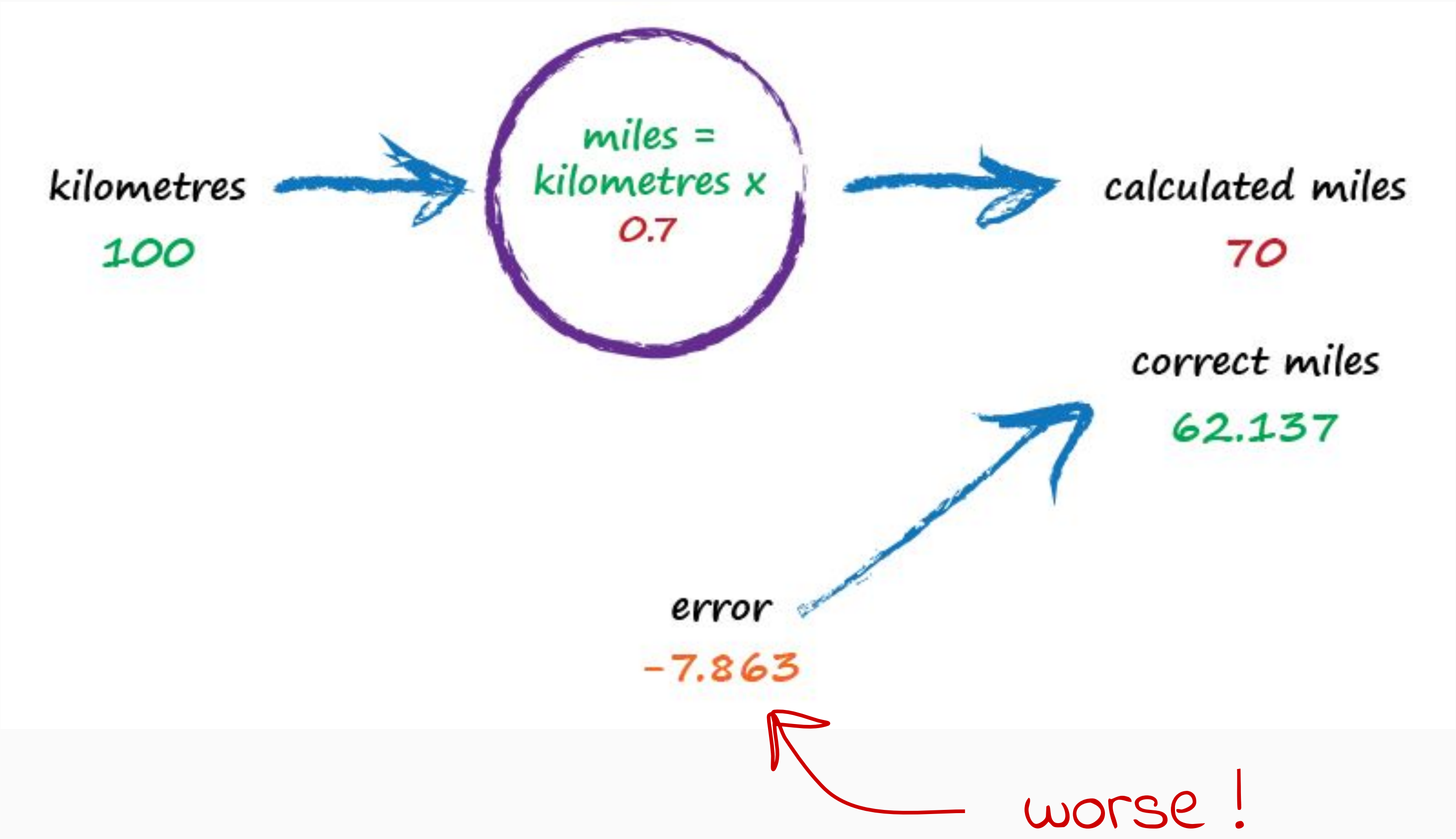


not great

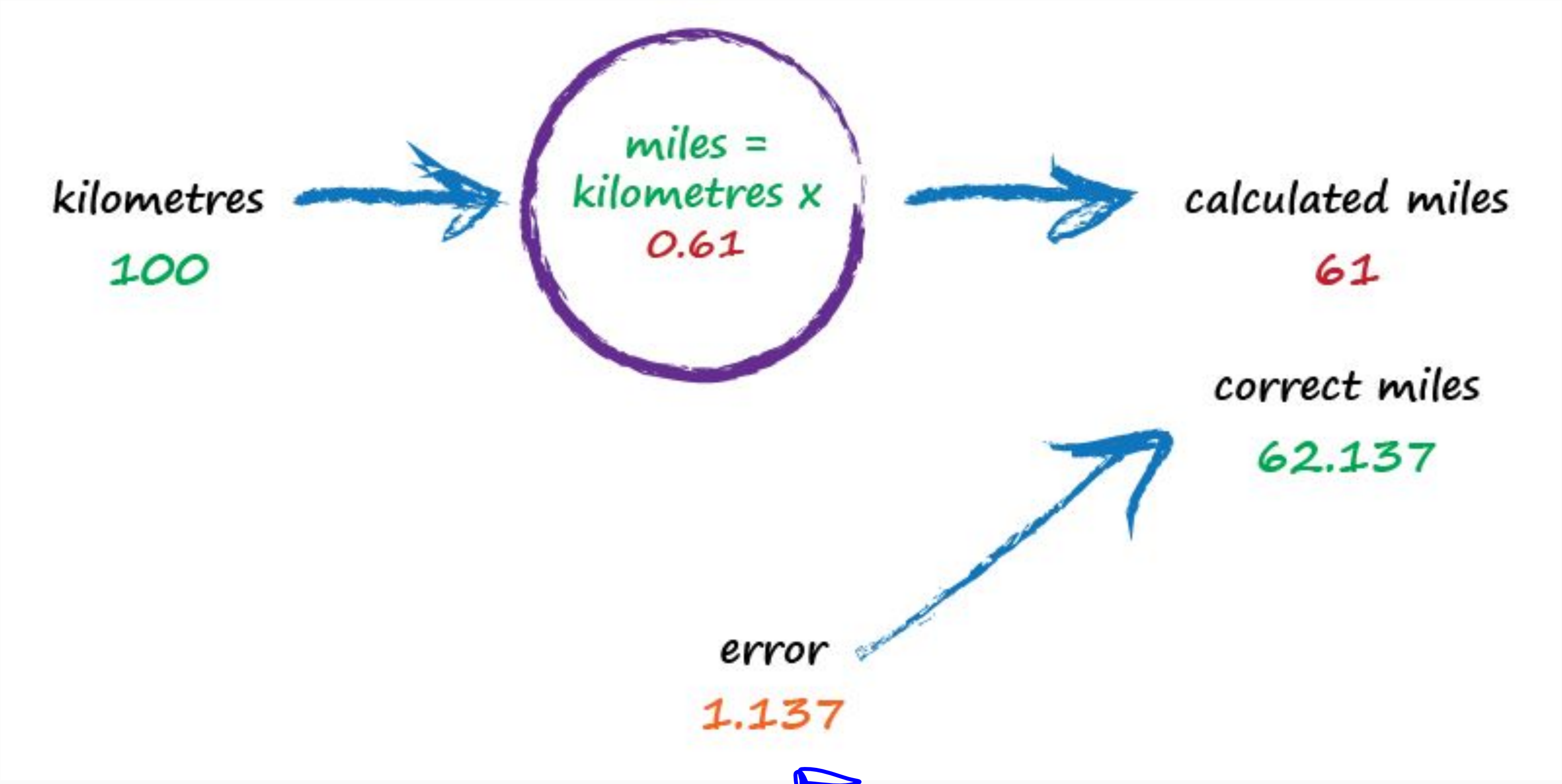
Kilometres to Miles



Kilometres to Miles



Kilometres to Miles



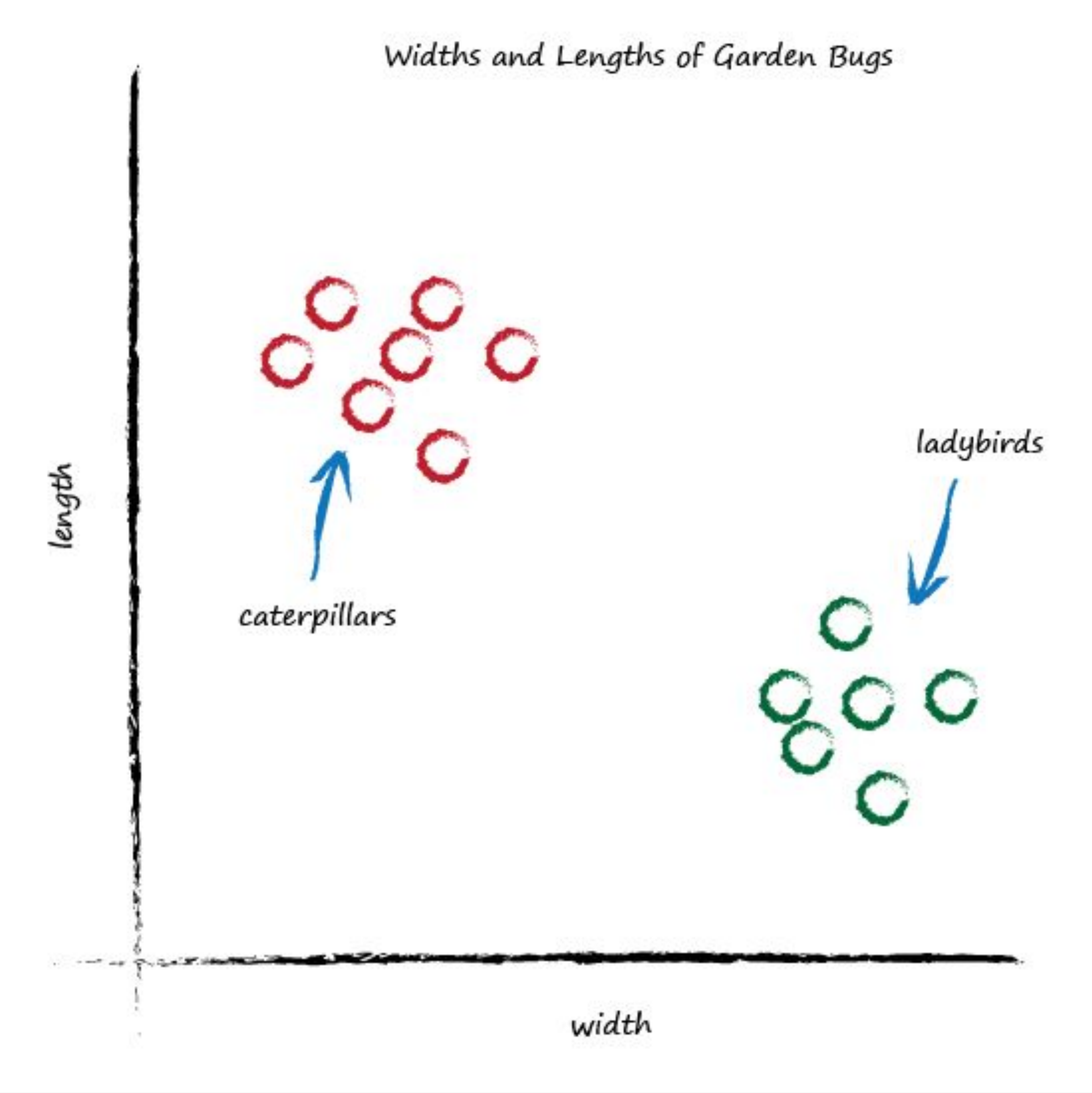
best yet !

Key Points

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

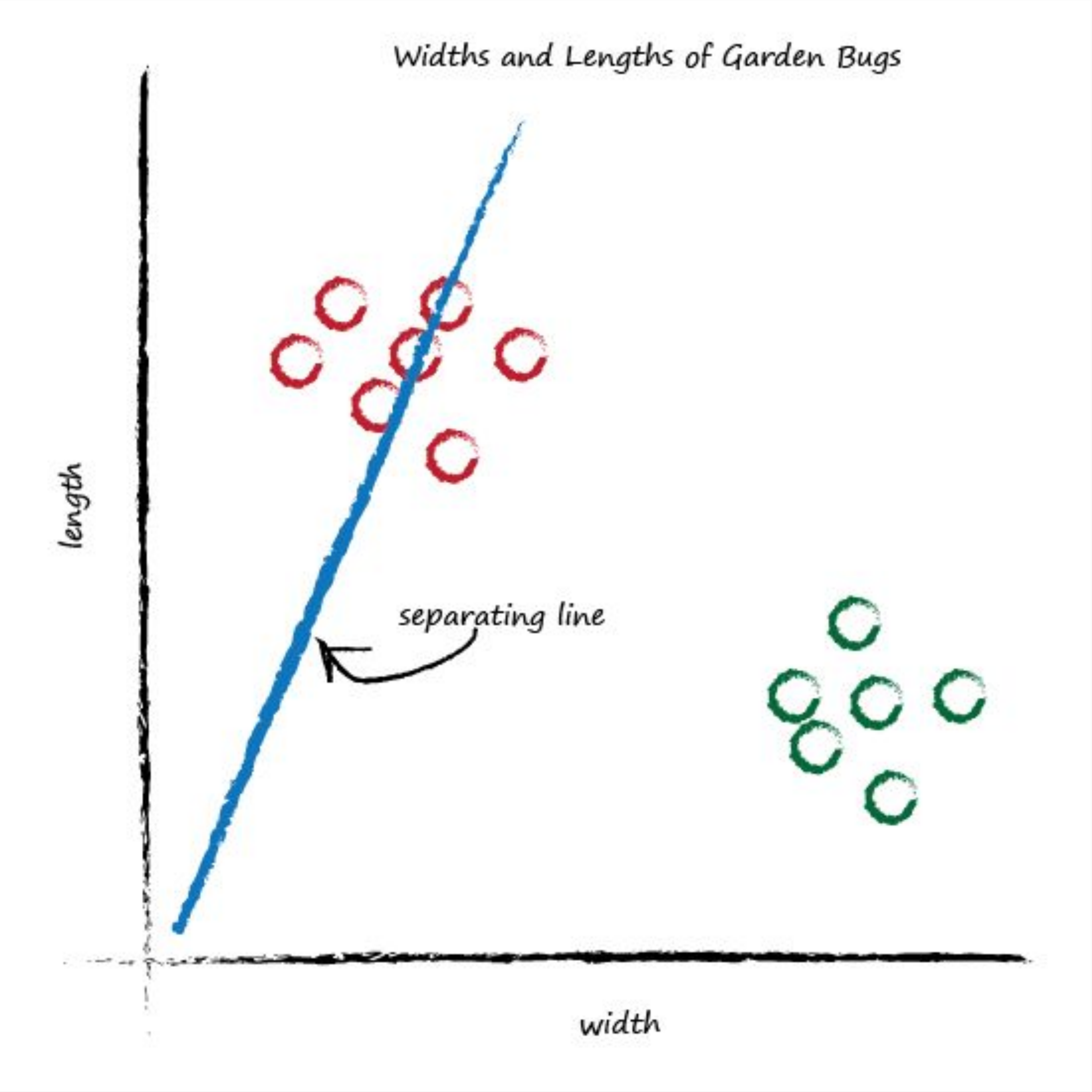
2. Use the **error** to refine the parameters.

Garden Bugs

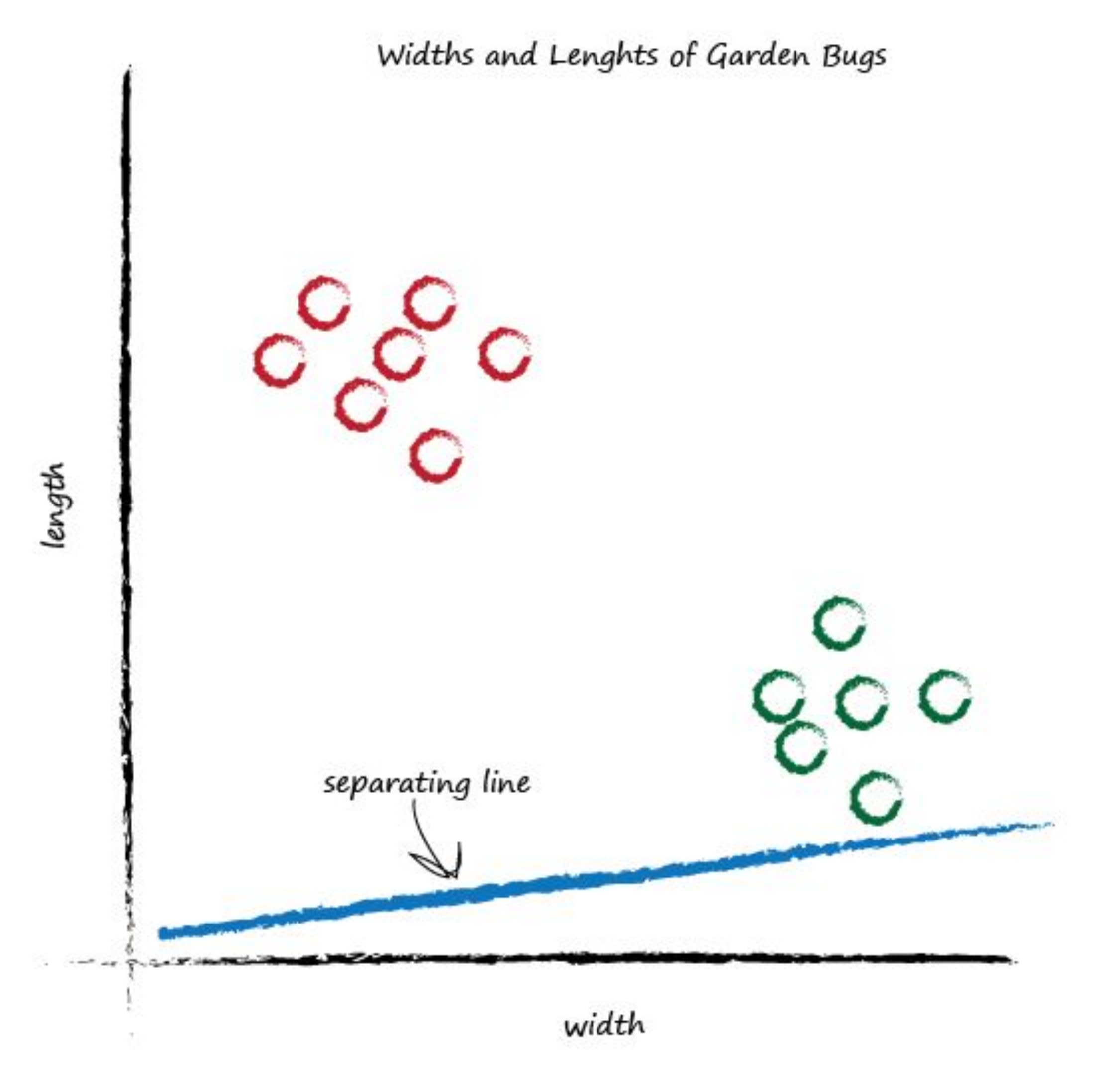


Widths and Lengths of Garden Bugs
length
caterpillars
ladybirds
width

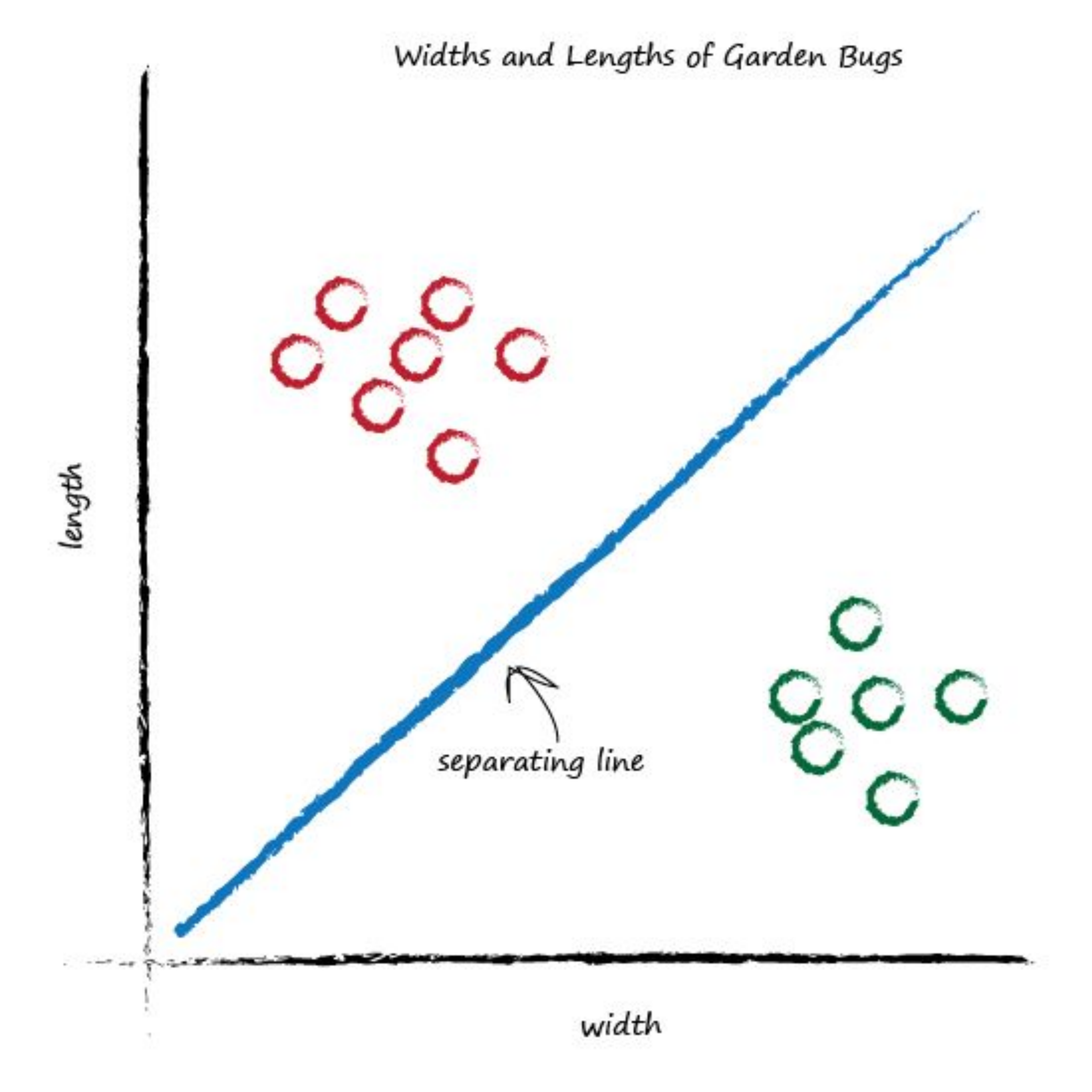
Classifying Bugs



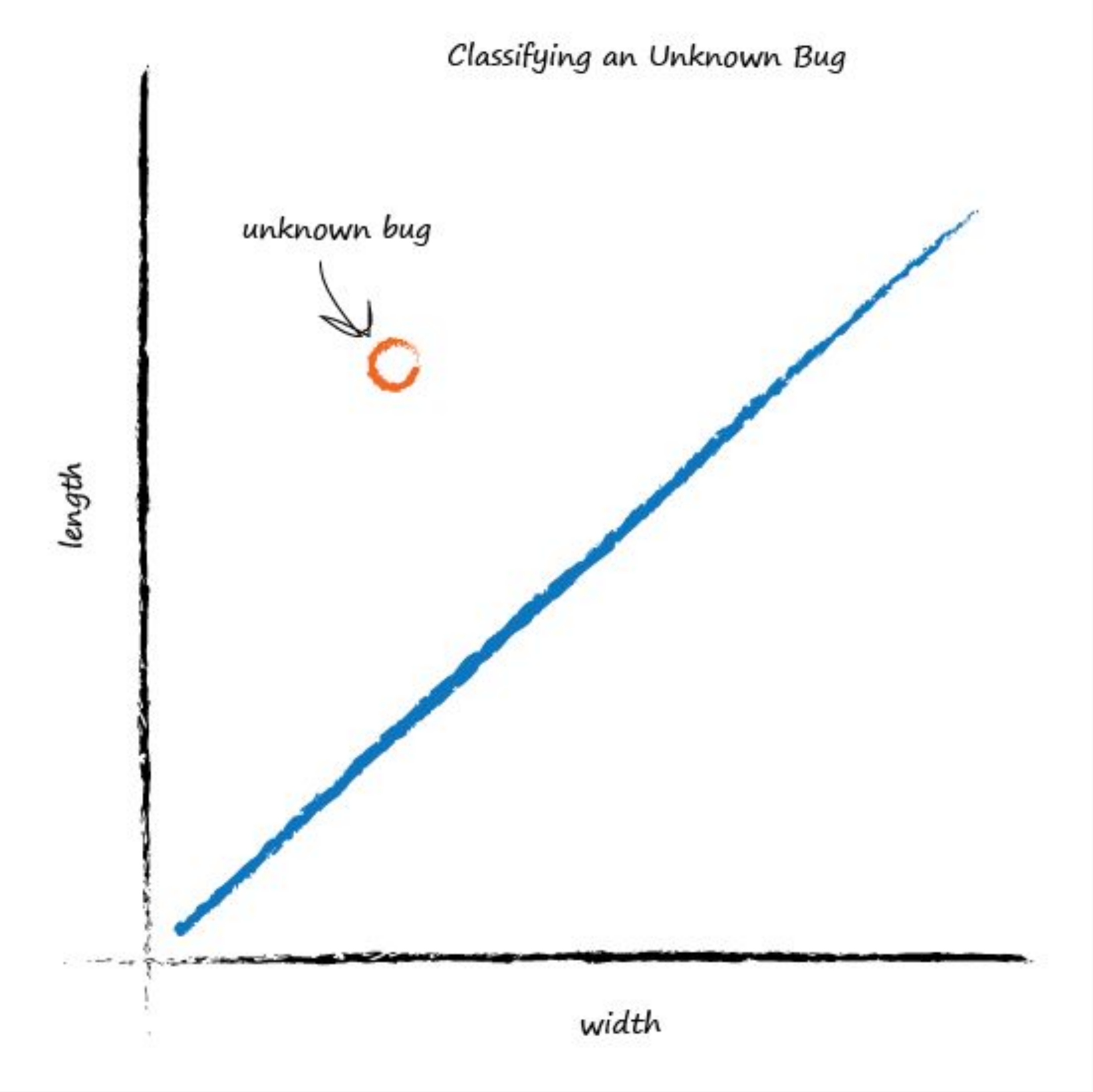
Classifying Bugs



Classifying Bugs



Classifying Bugs



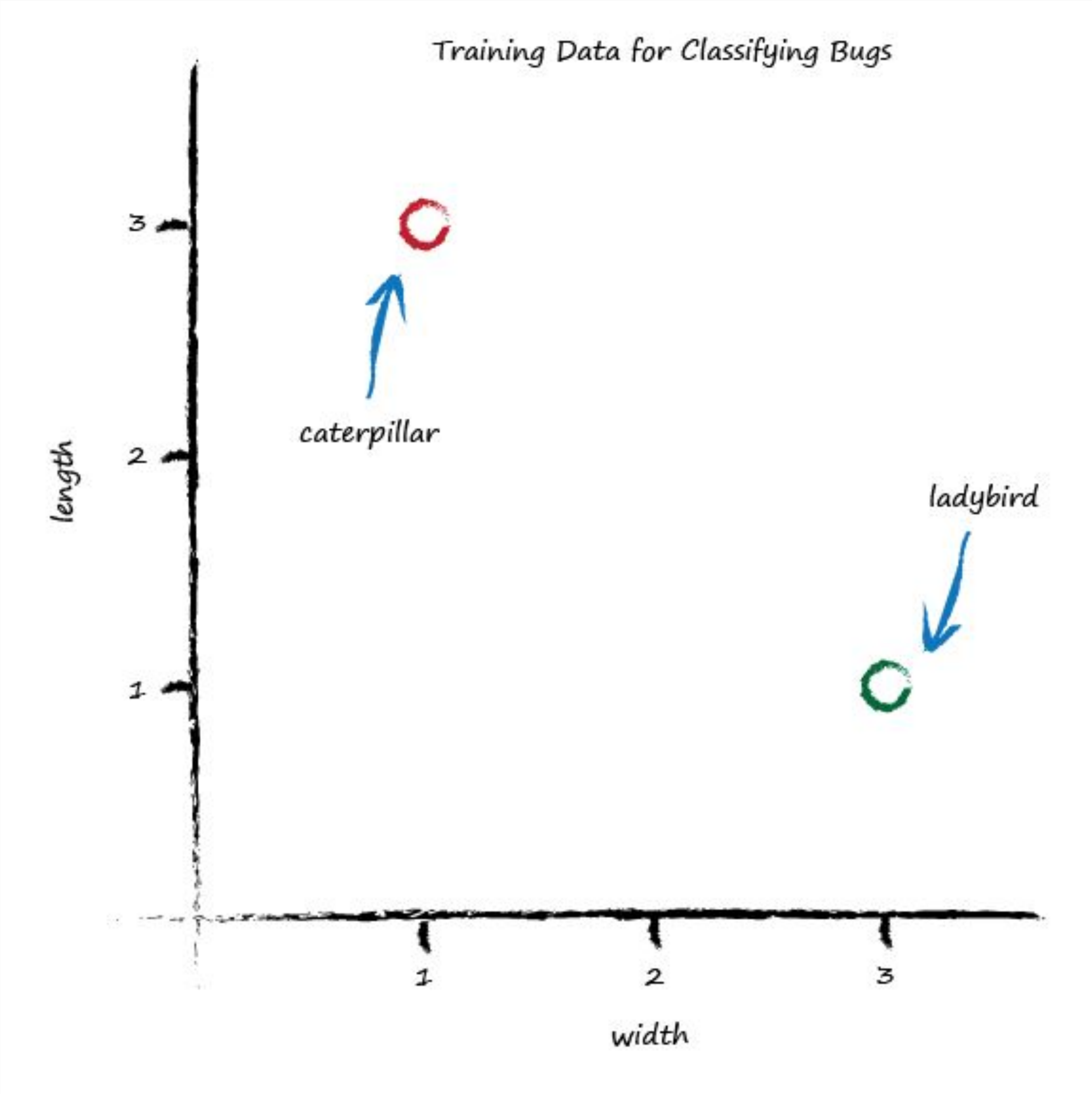
Key Points

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

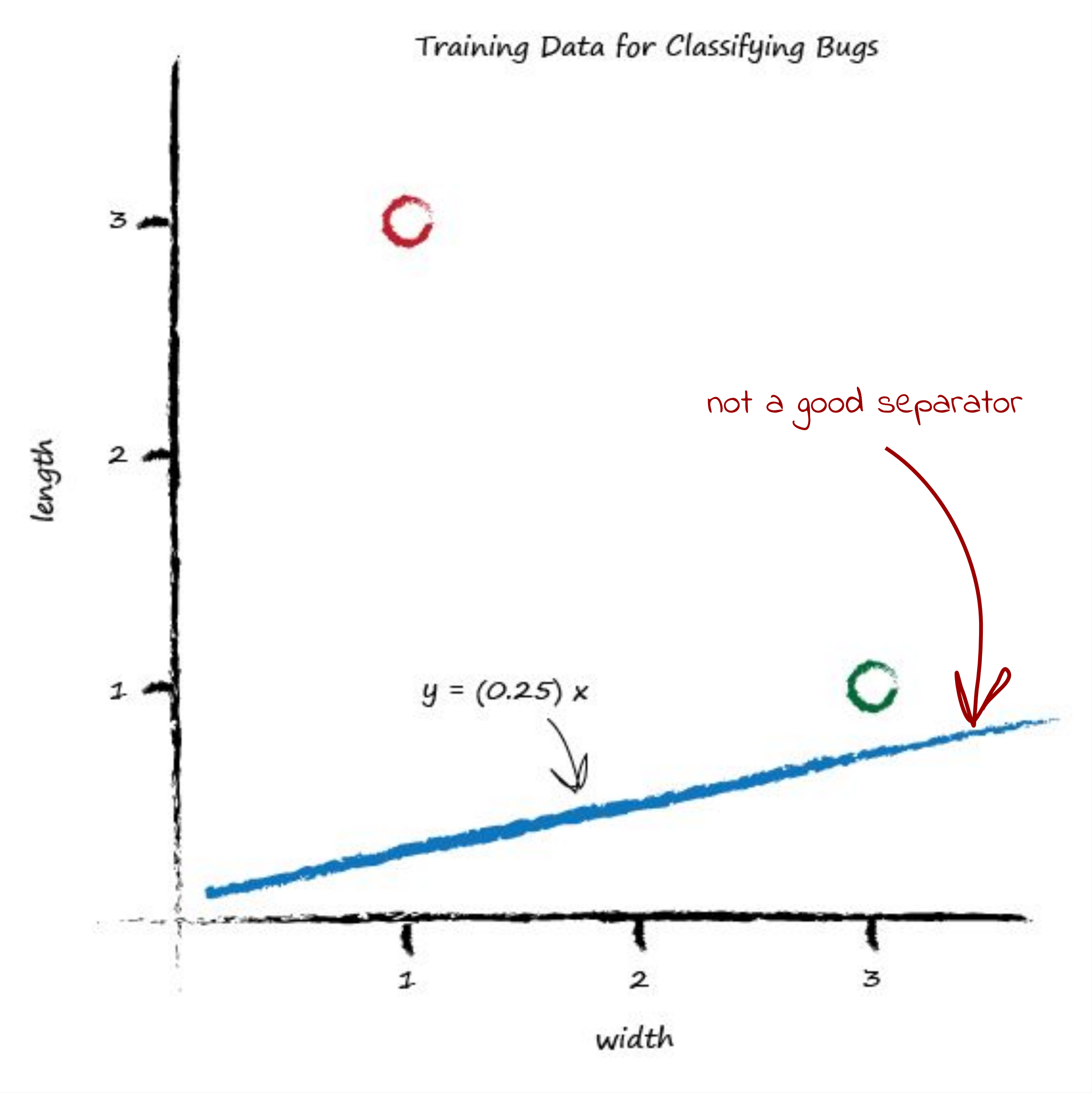
Learning from Data

Example	Width	Length	Bug
1	3.0	1.0	ladybird
2	1.0	3.0	caterpillar

Learning from Data

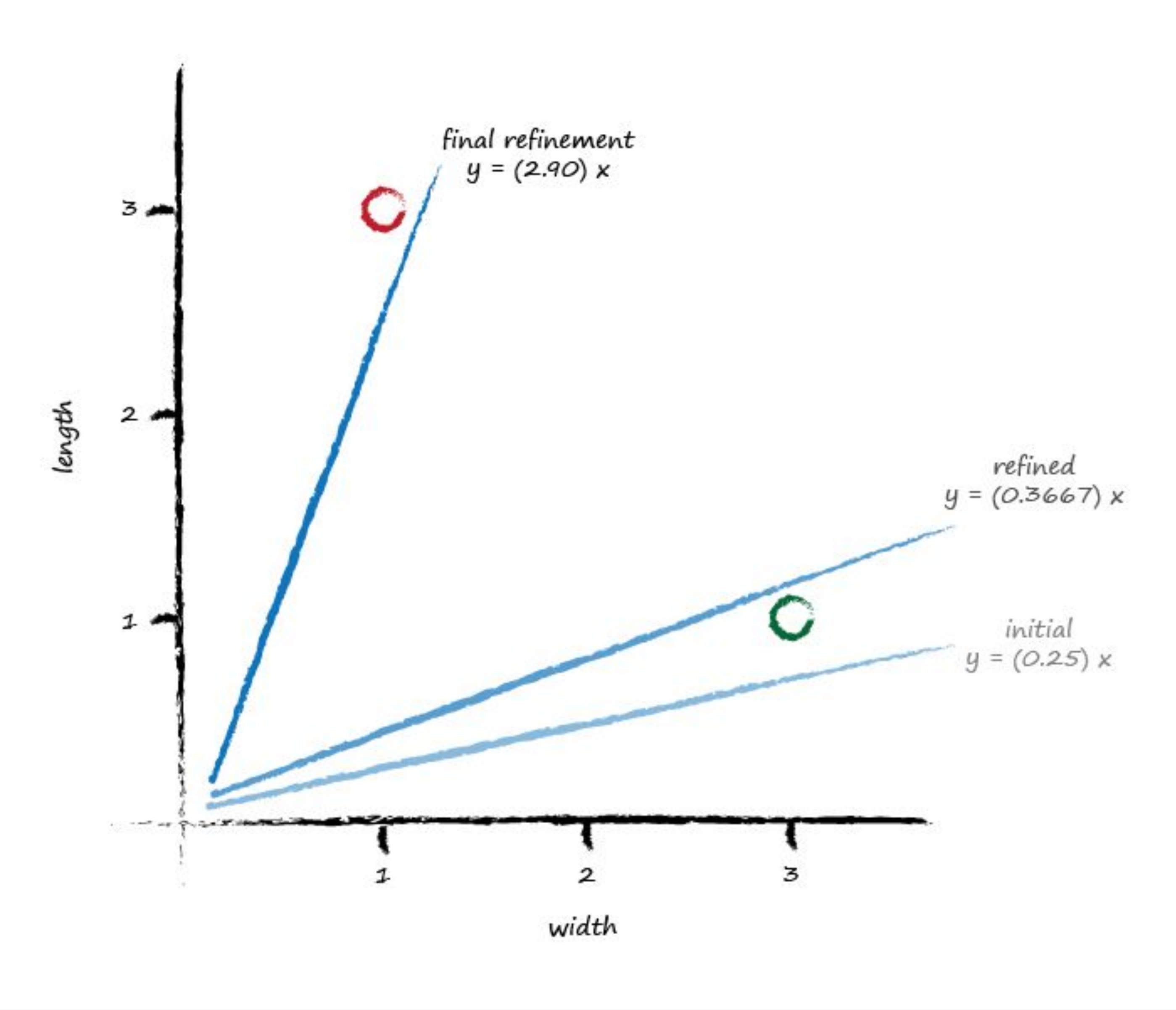


Learning from Data

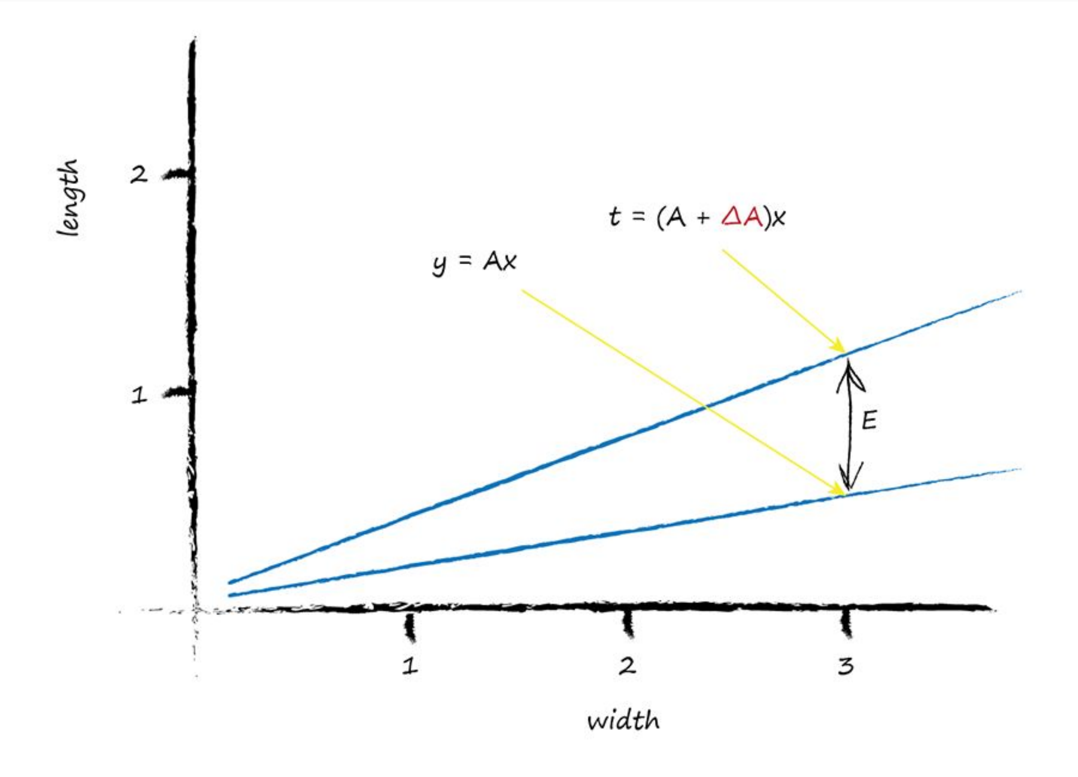


Learning from Data

Learning from Data



How Do We Update The Parameter?

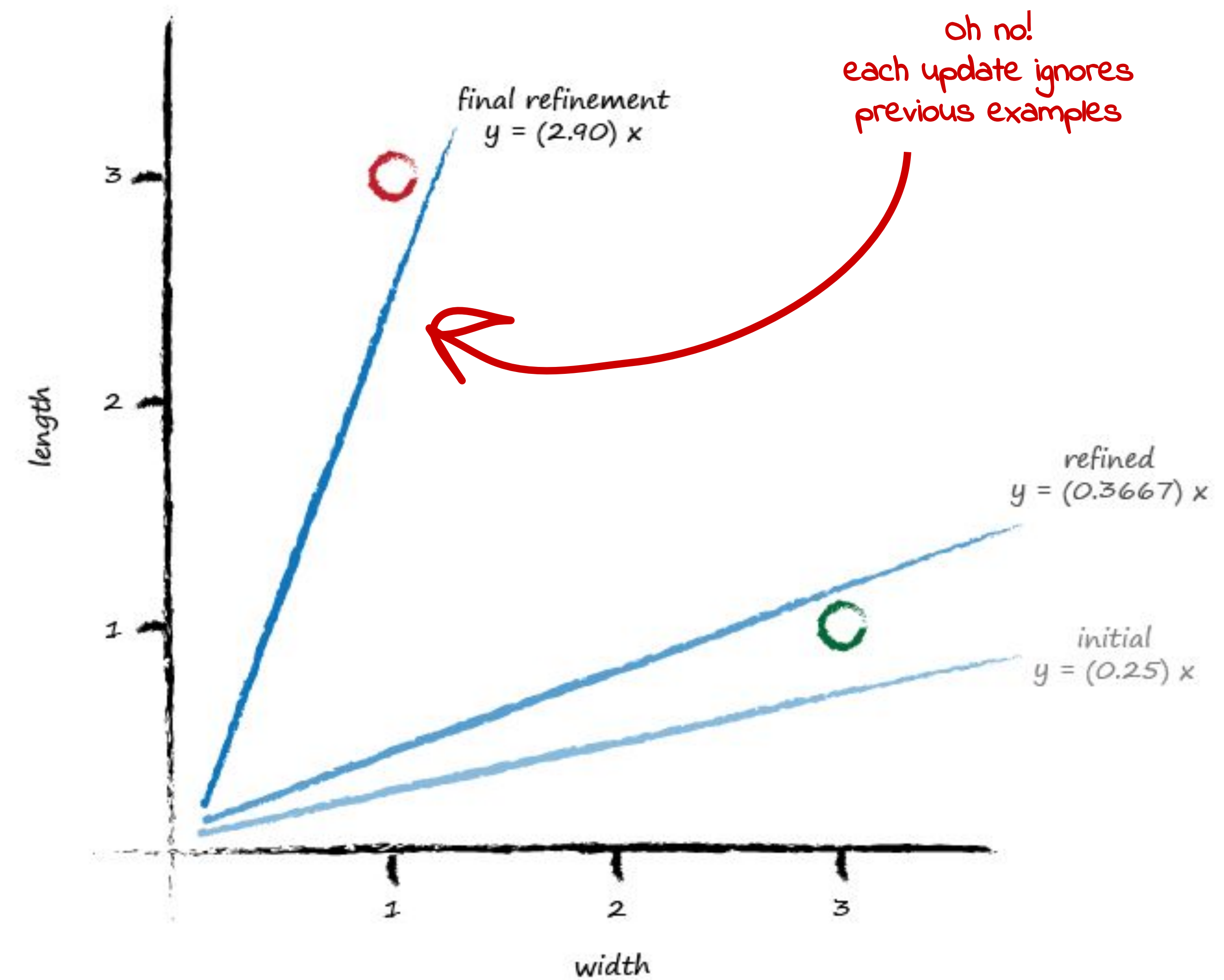


$$\text{error} = \text{target} - \text{actual}$$

$$E = (A + \Delta A)x - Ax$$

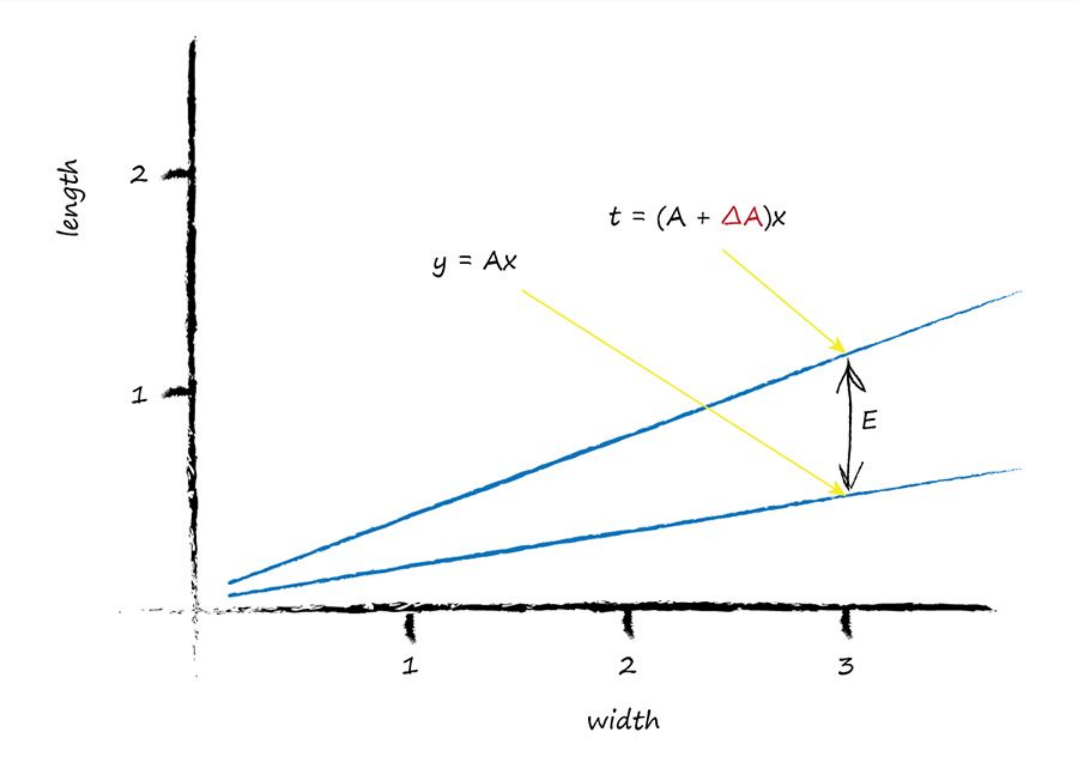
$$\Delta A = E / x$$

Hang On!



oh no!
each update ignores
previous examples
final refinement
y = (2.90) x
refined
y = (0.3667) x
initial
y = (0.25) x
length
width
1
2
3
1
2
3

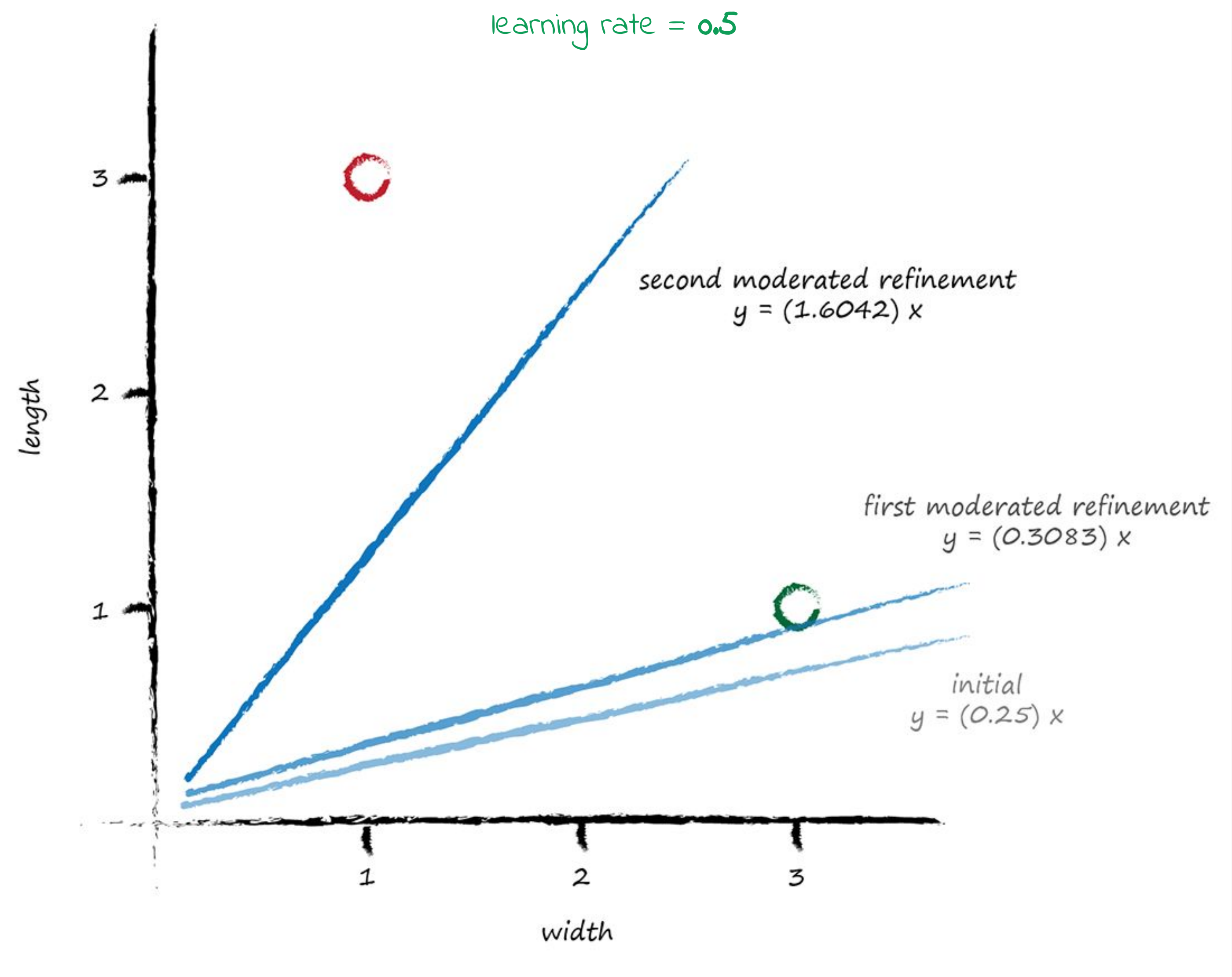
Calm Down the Learning



$$\Delta A = L \cdot (E / x)$$

learning rate

Calm Down the Learning



Key Points

1. **Moderating** your learning is good - ensures you learn from all your data, and reduces impact of outliers or noisy training data.

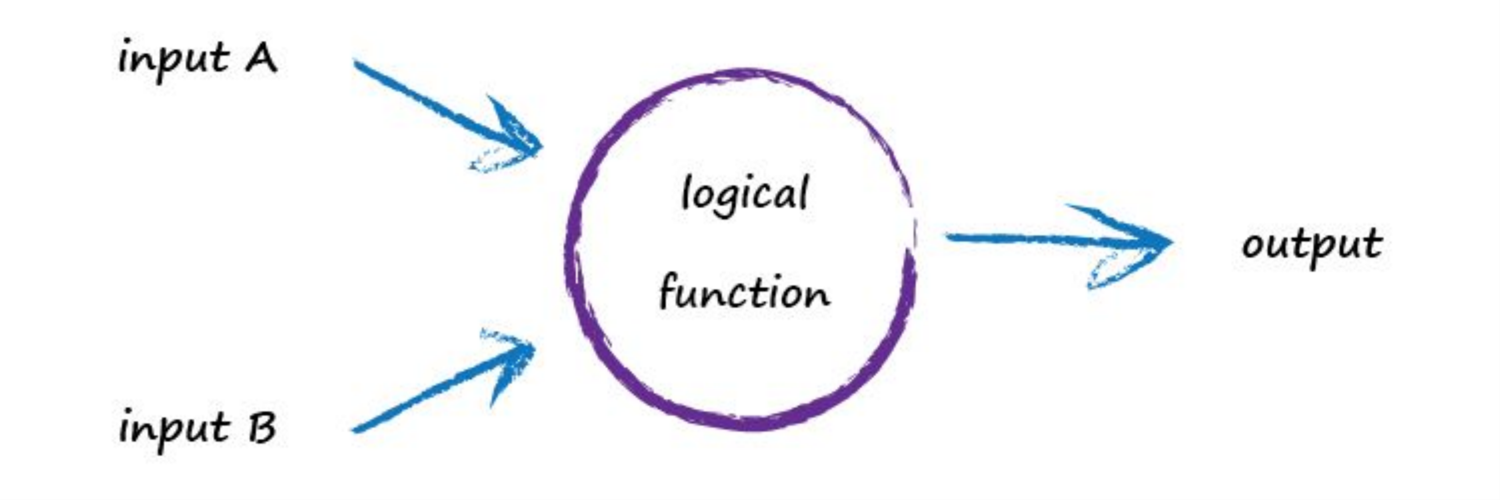
Boolean Logic

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

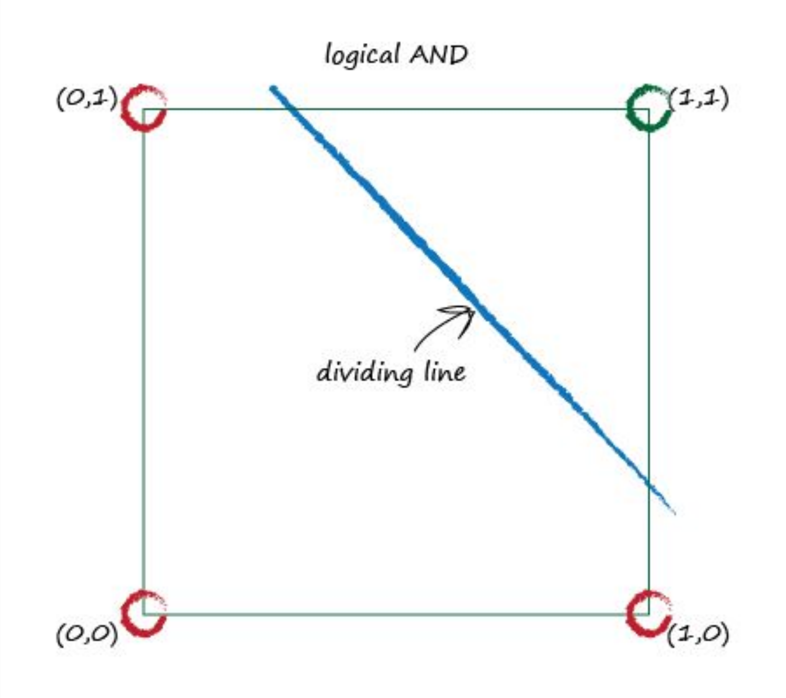
IF it's the weekend **OR** I am on annual leave **THEN** I'll go to the park.

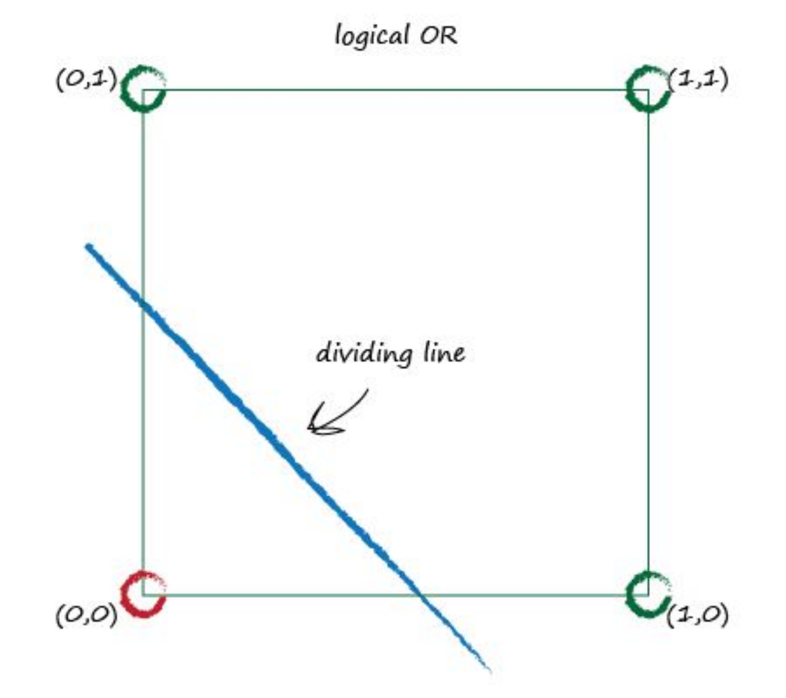
Input A	Input B	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Boolean Logic



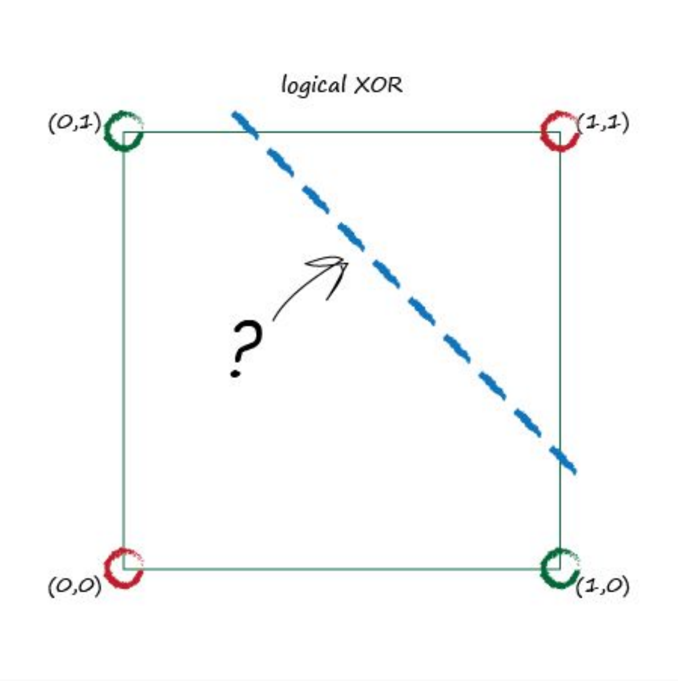
Boolean Logic



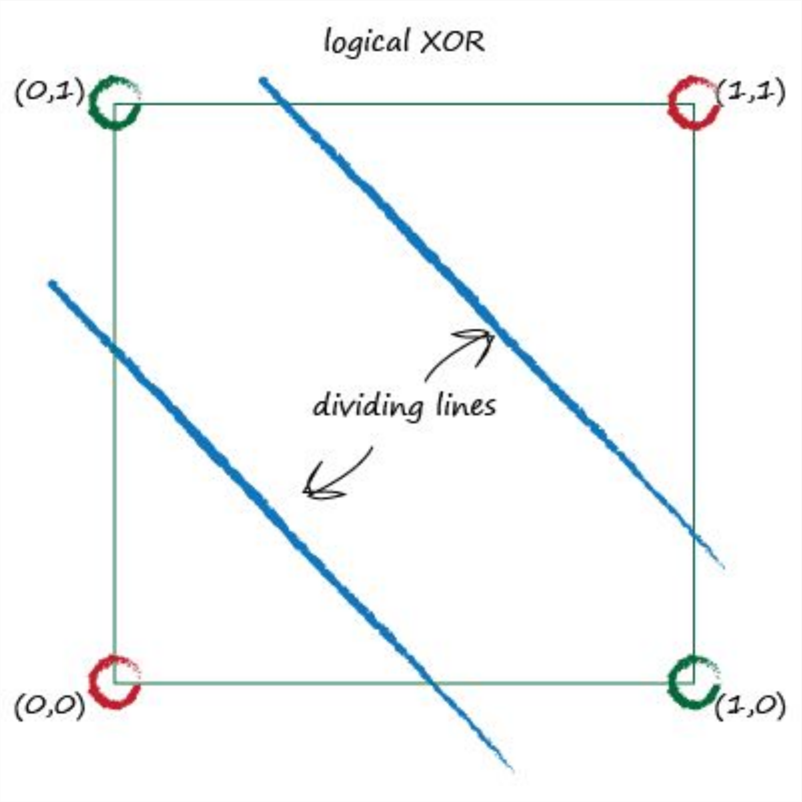


XOR Puzzle!

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



XOR Solution!

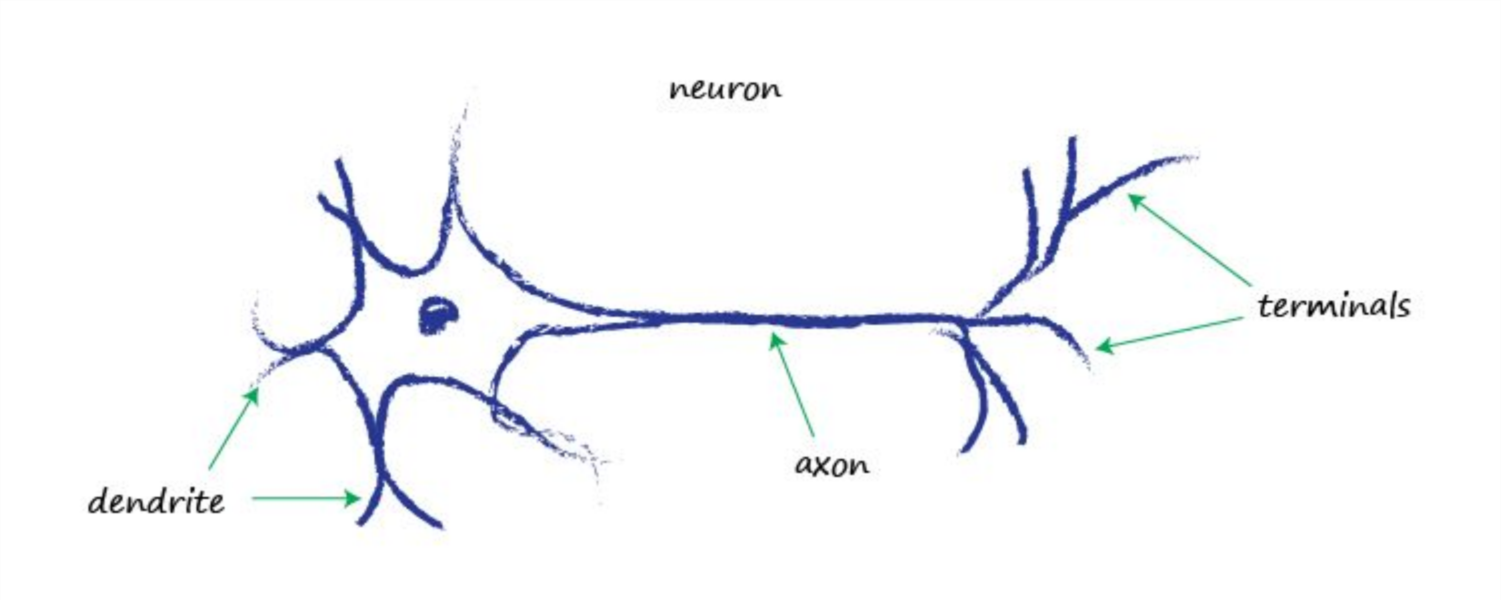


... use more than one node!

Key Points

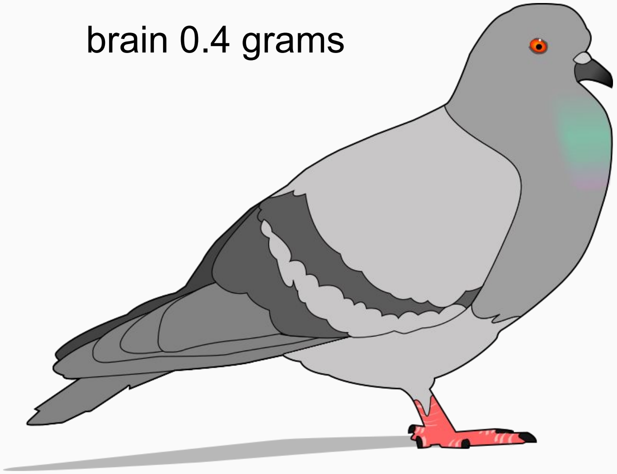
1. Some problems can't be solved with just a single simple linear classifier.
2. You can use **multiple nodes** working together to solve many of these problems.

Brains in Nature

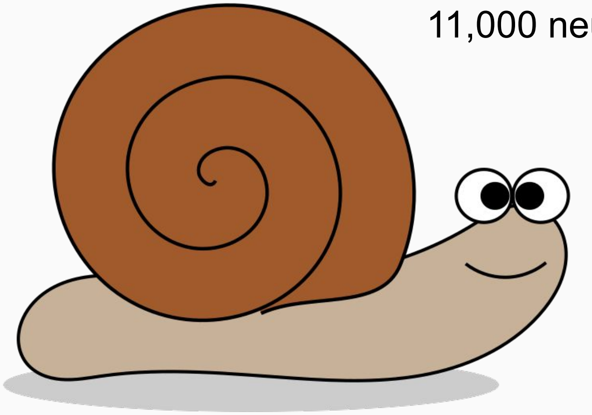


Brains in Nature

brain 0.4 grams

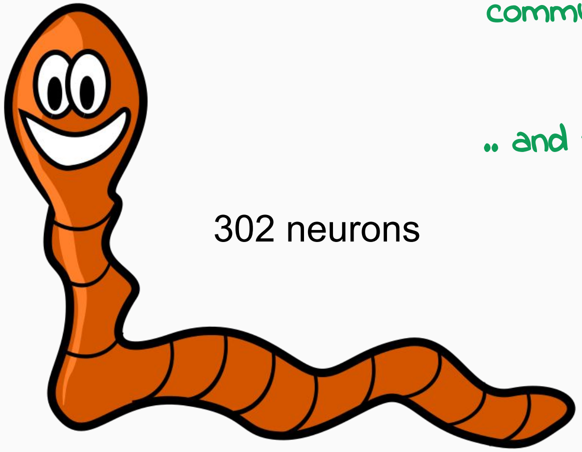


11,000 neurons

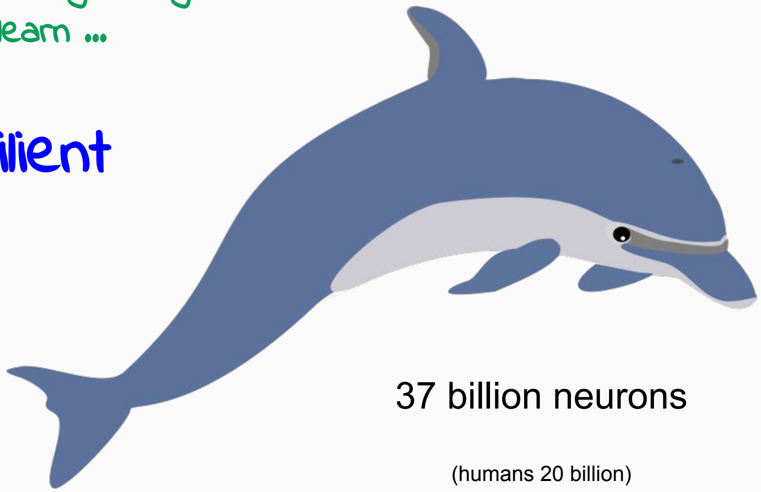


nature's brains can eat, fly, navigate, fight, communicate, play, learn ...

.. and they're **resilient**



302 neurons

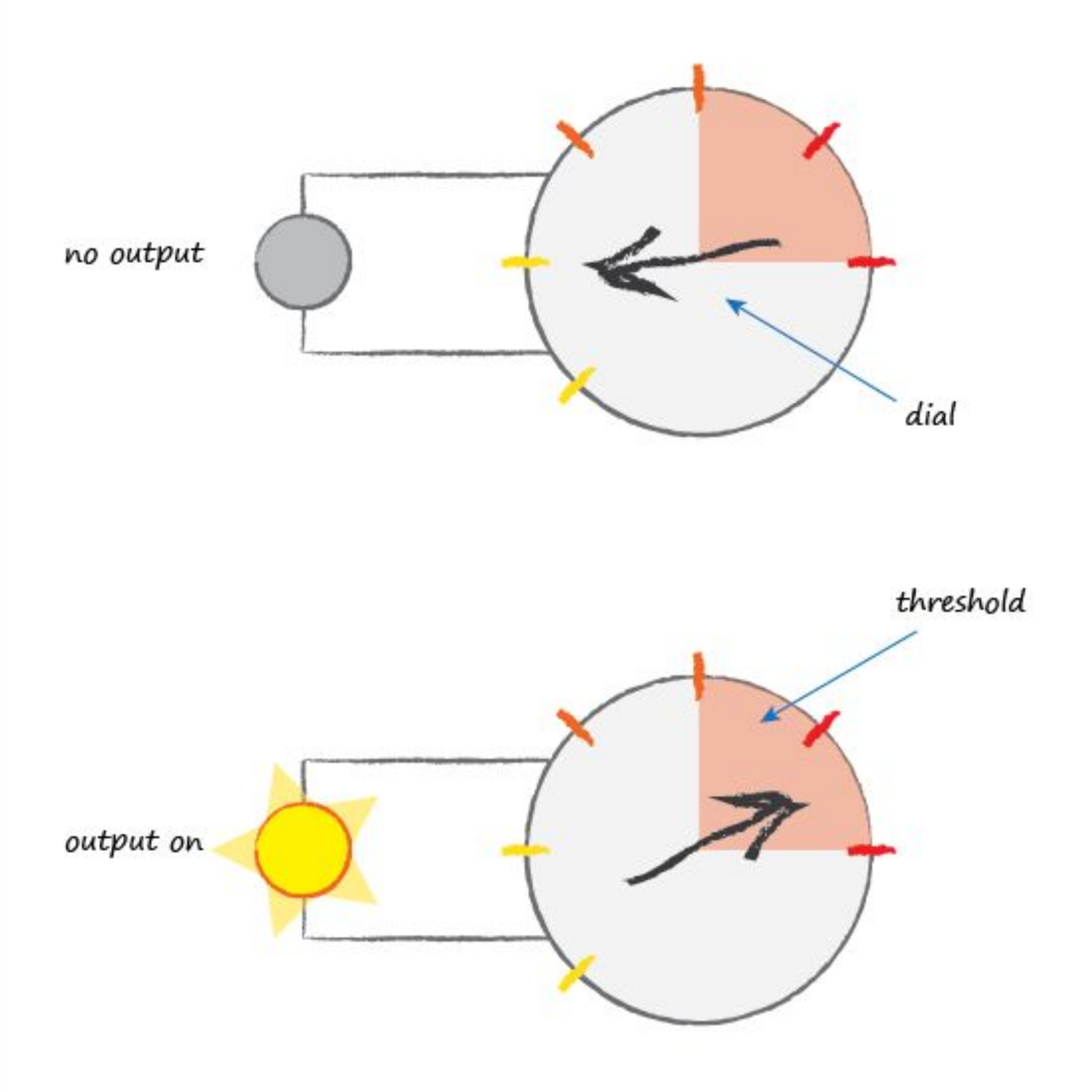


37 billion neurons

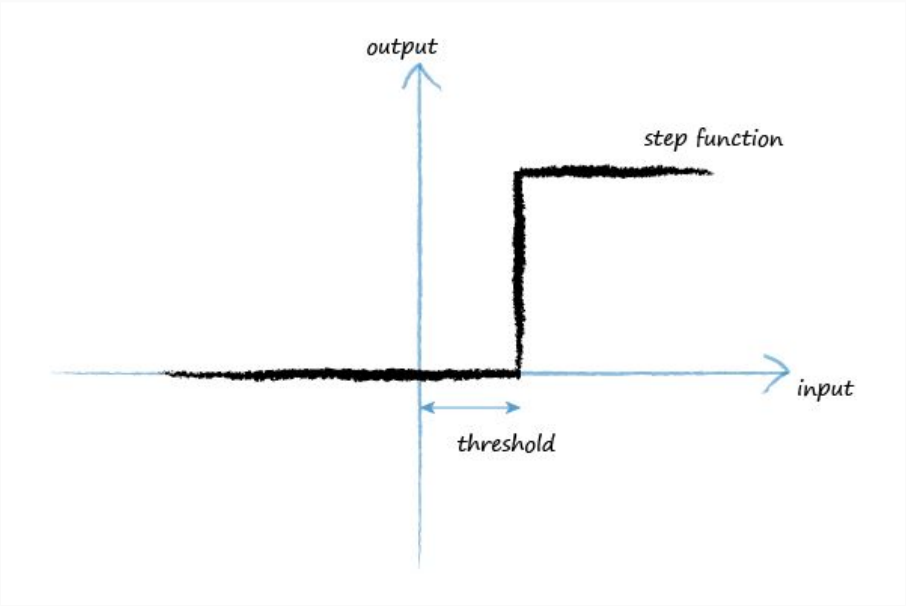
(humans 20 billion)

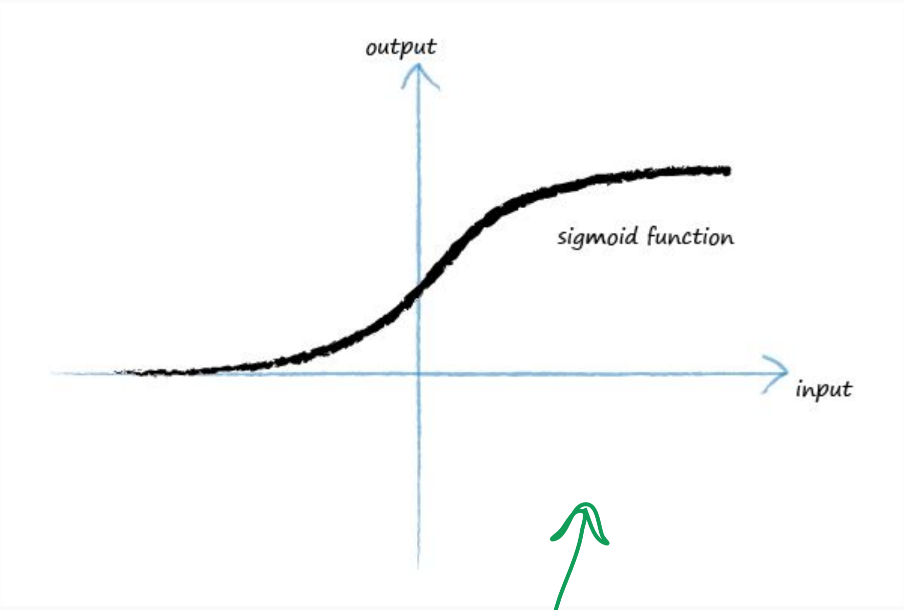
https://en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons
https://faculty.washington.edu/chudler/facts.html

Brains in Nature



Brains in Nature

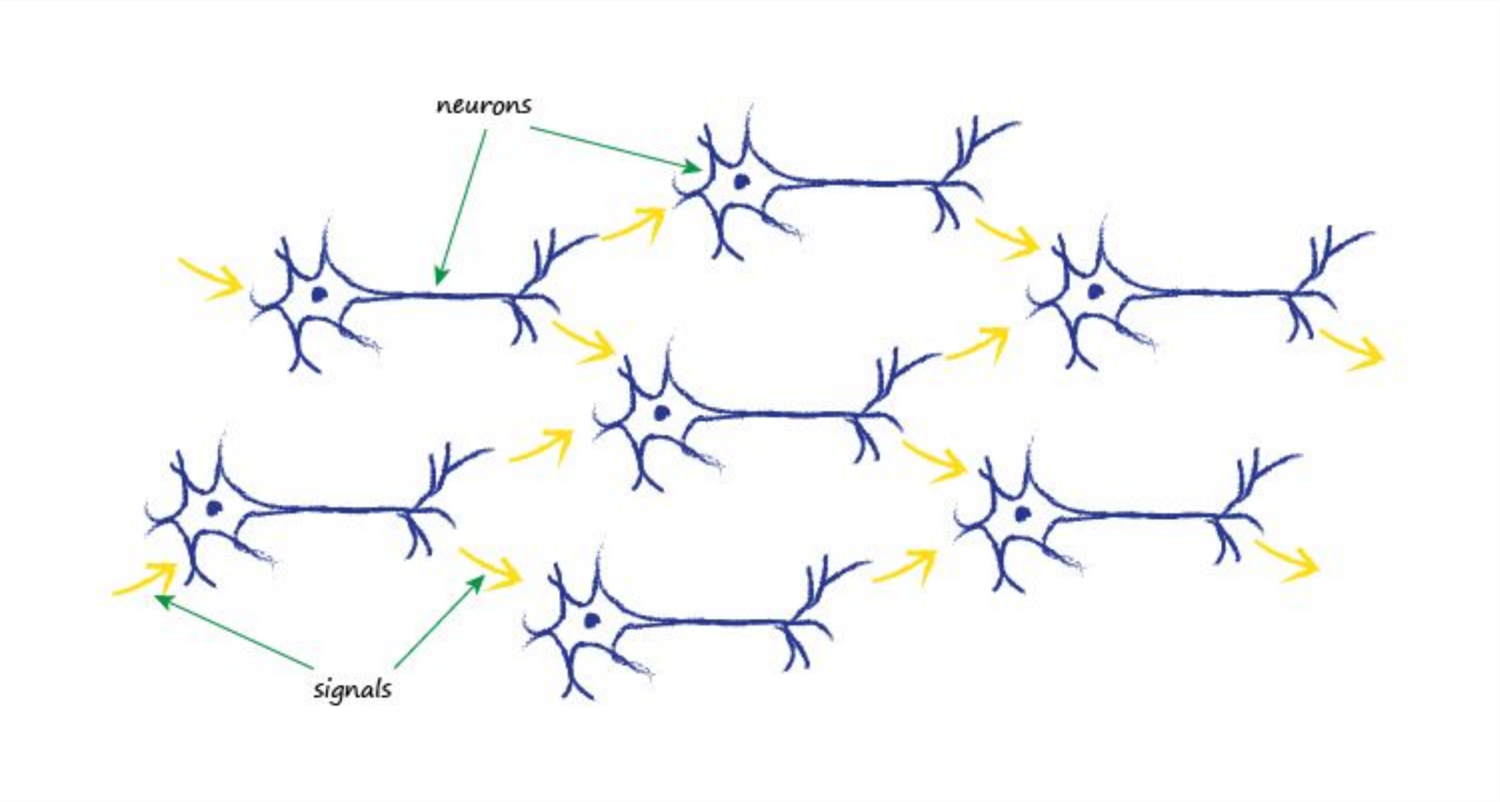




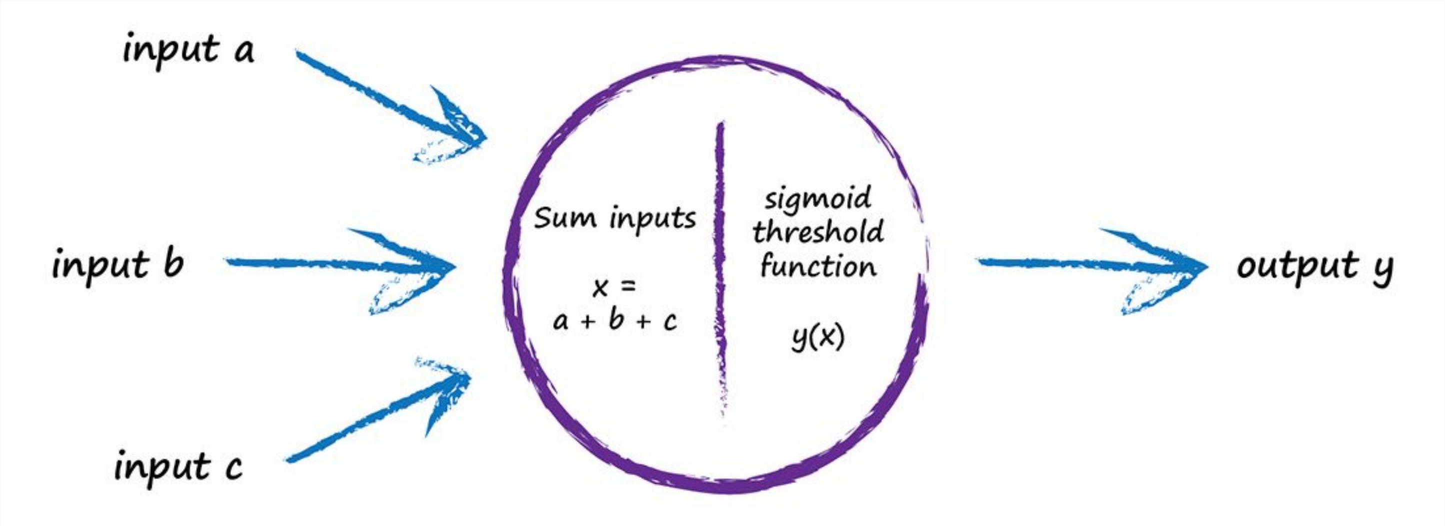
logistic function

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

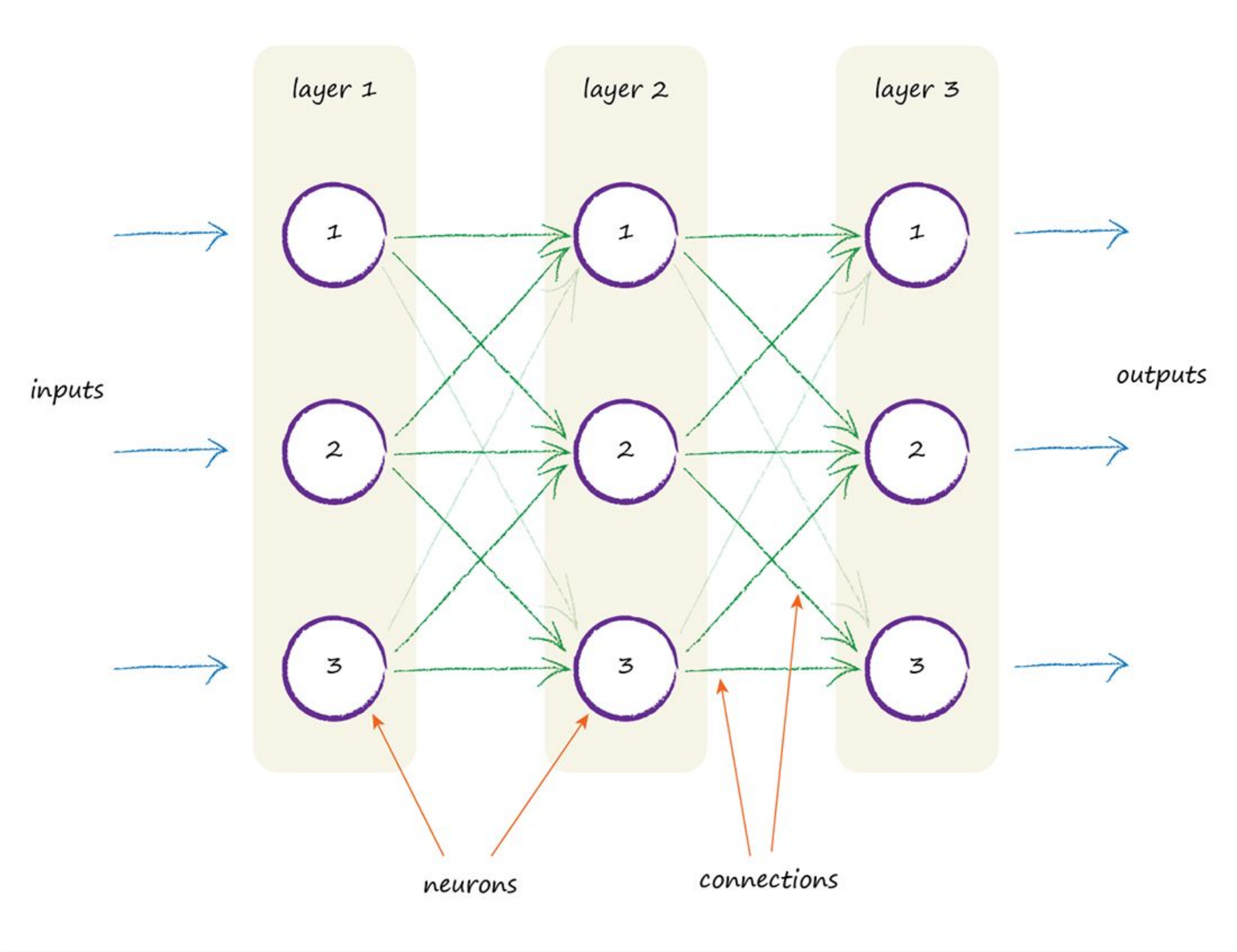
Brains in Nature



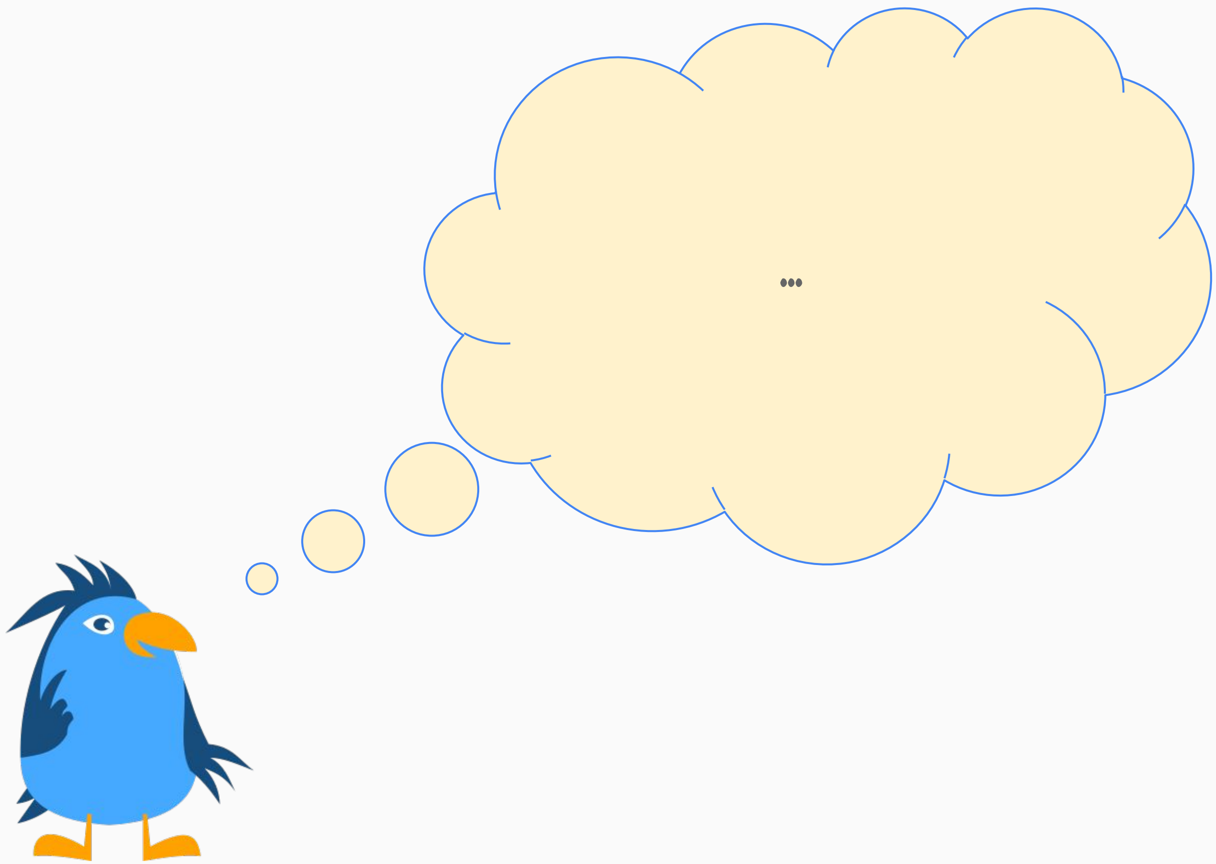
Artificial Neuron



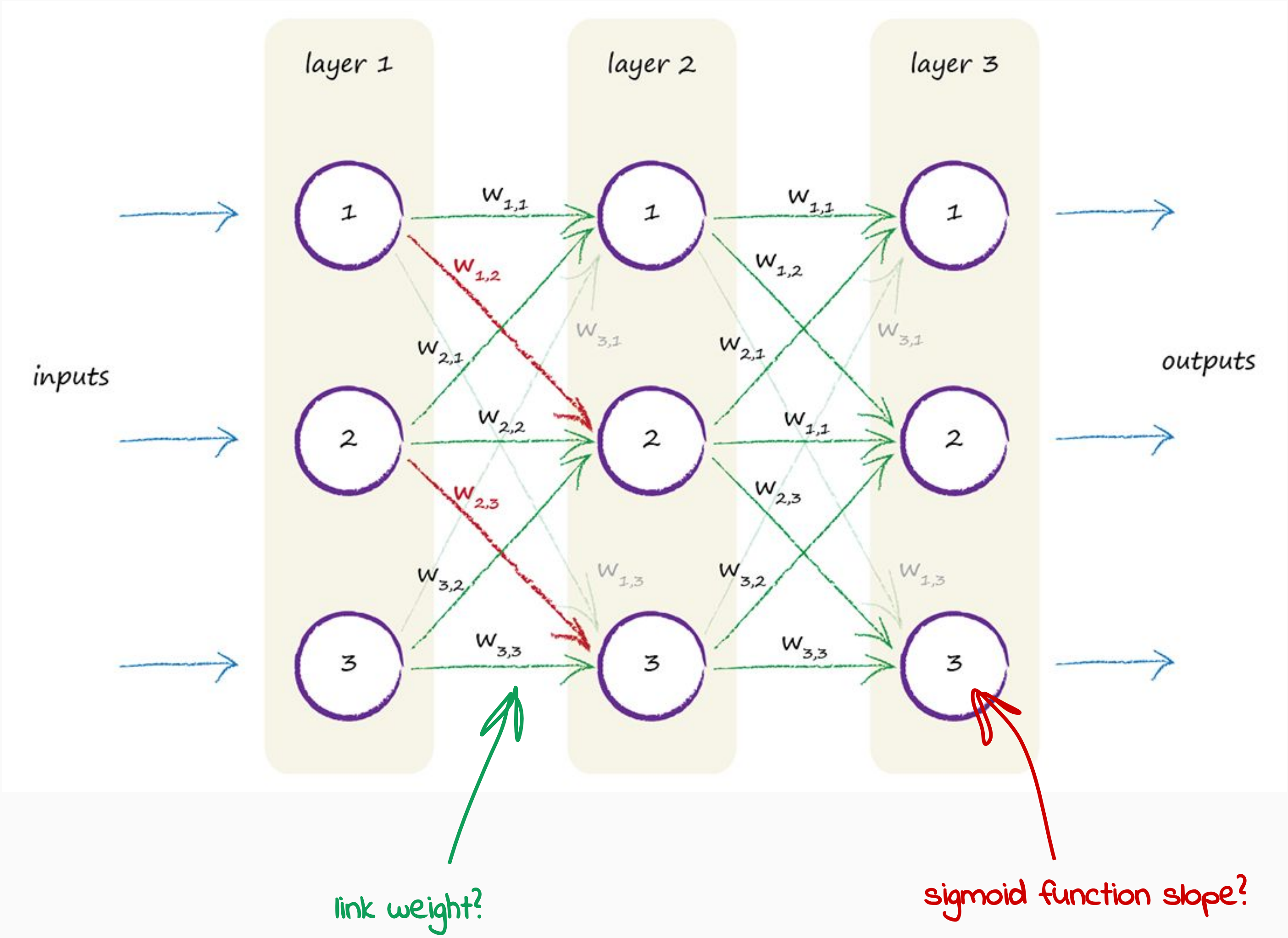
Artificial Neural Network .. finally!



Pause.



Where Does The Learning Happen?



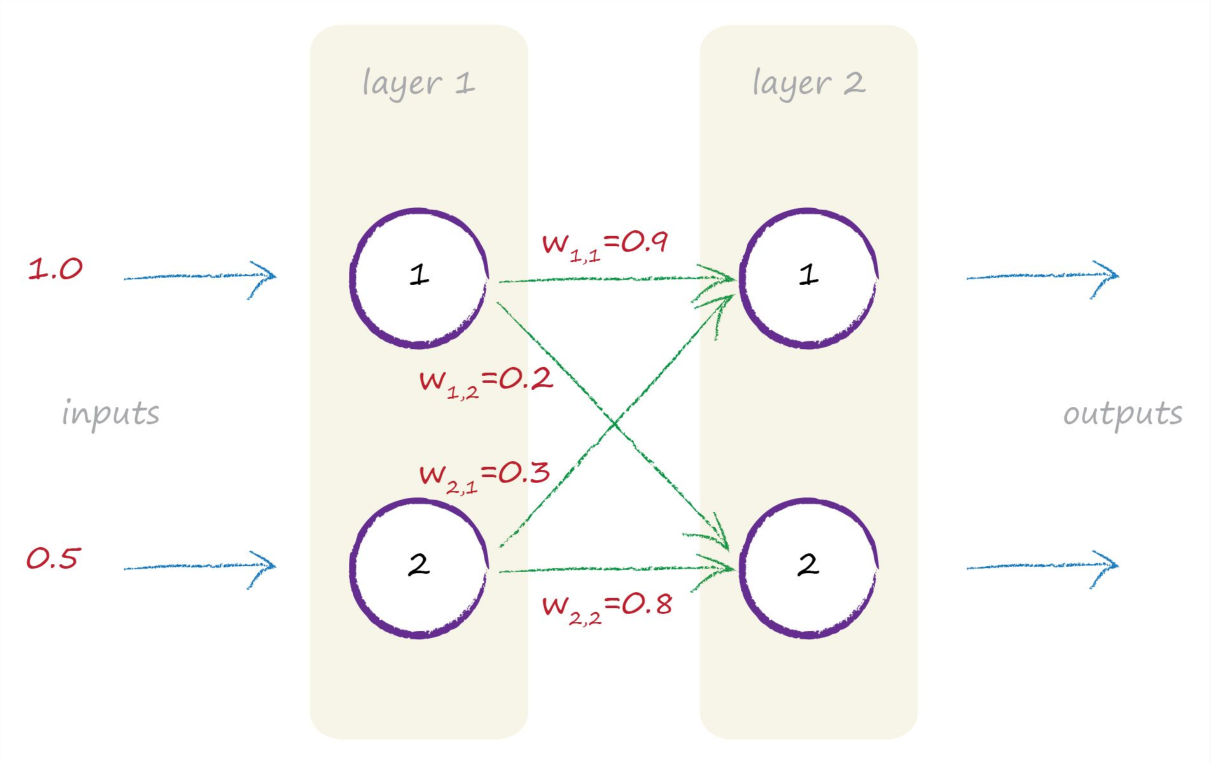
Key Points

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

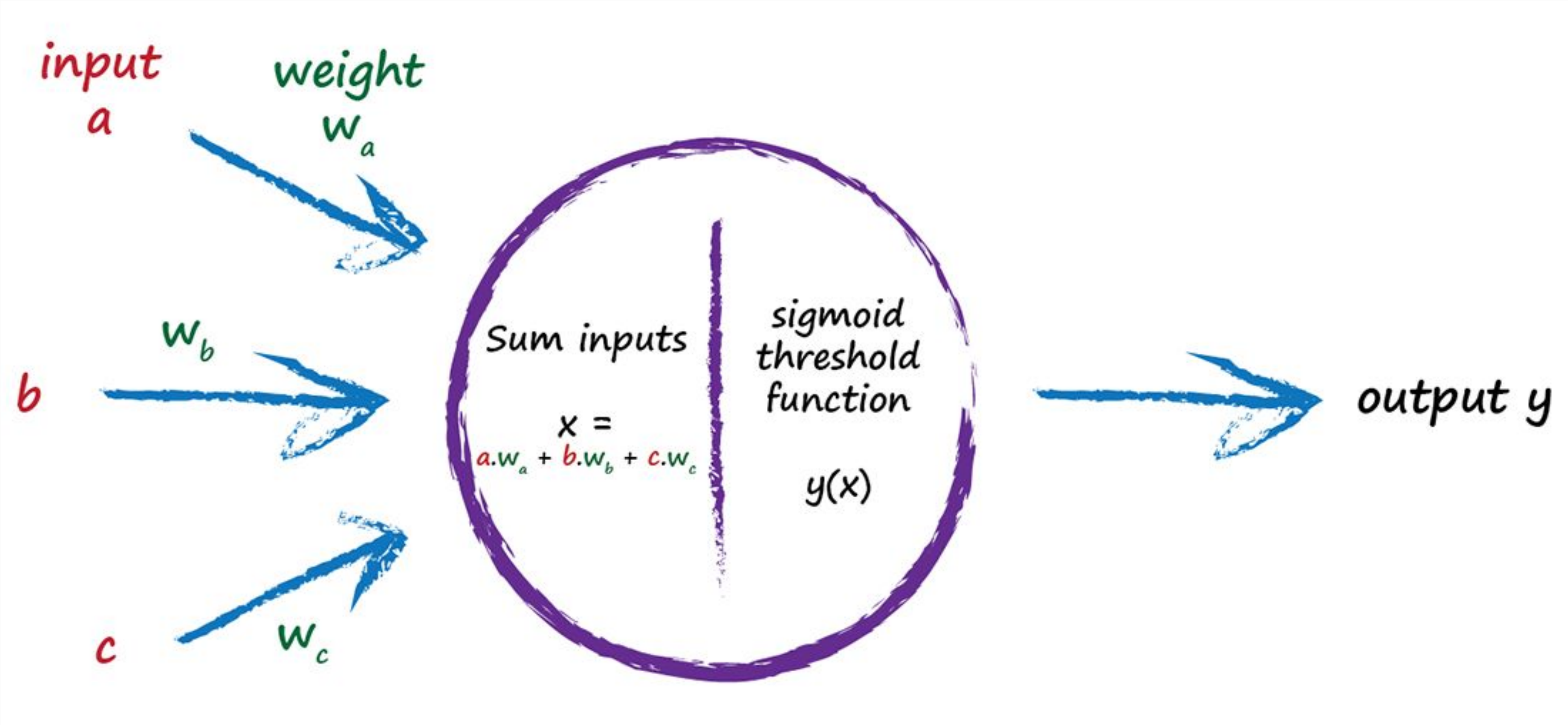
2. Trying to copy biological brains partly inspired **artificial neural networks**.

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

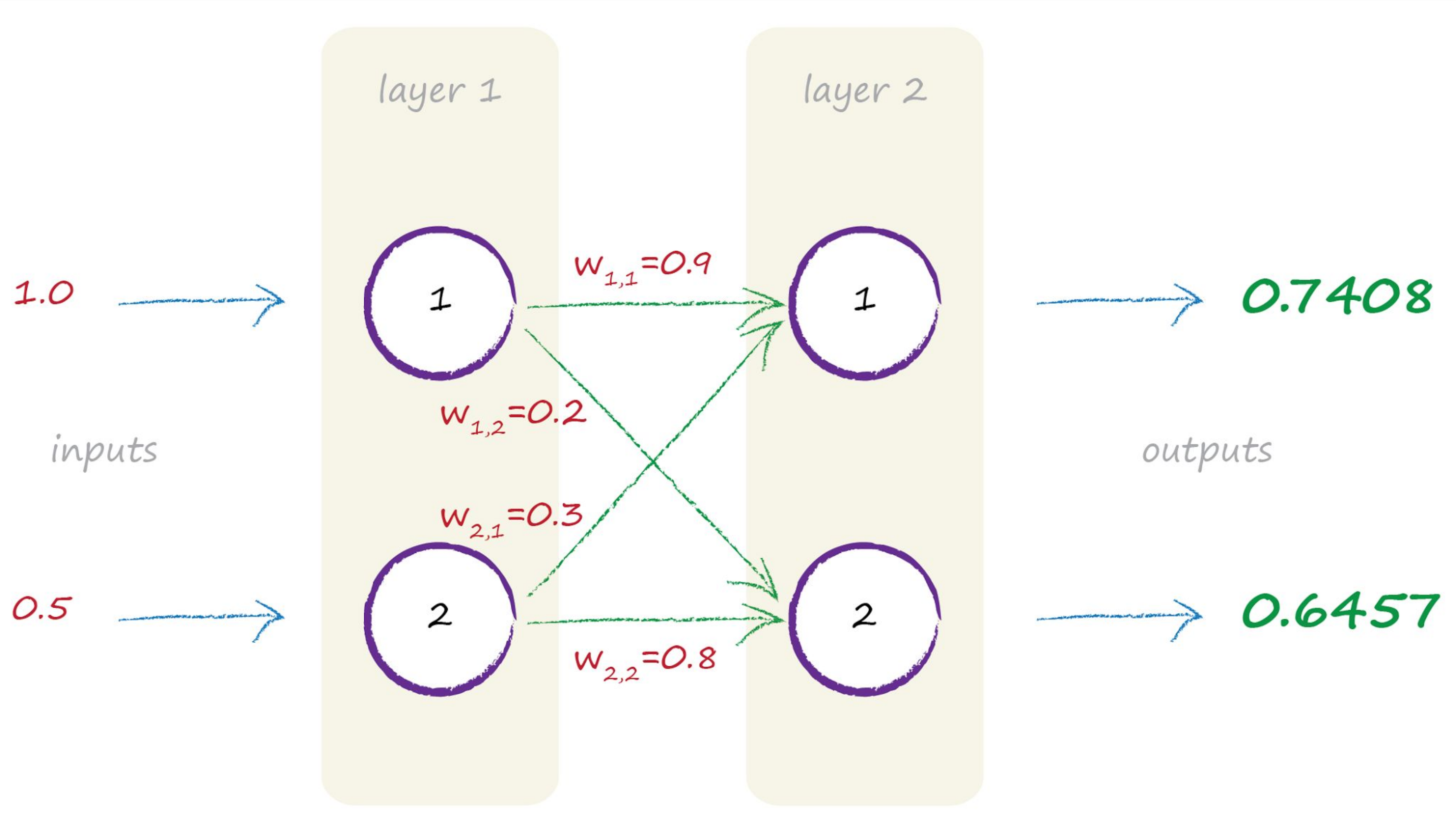
Feeding Signals Forward



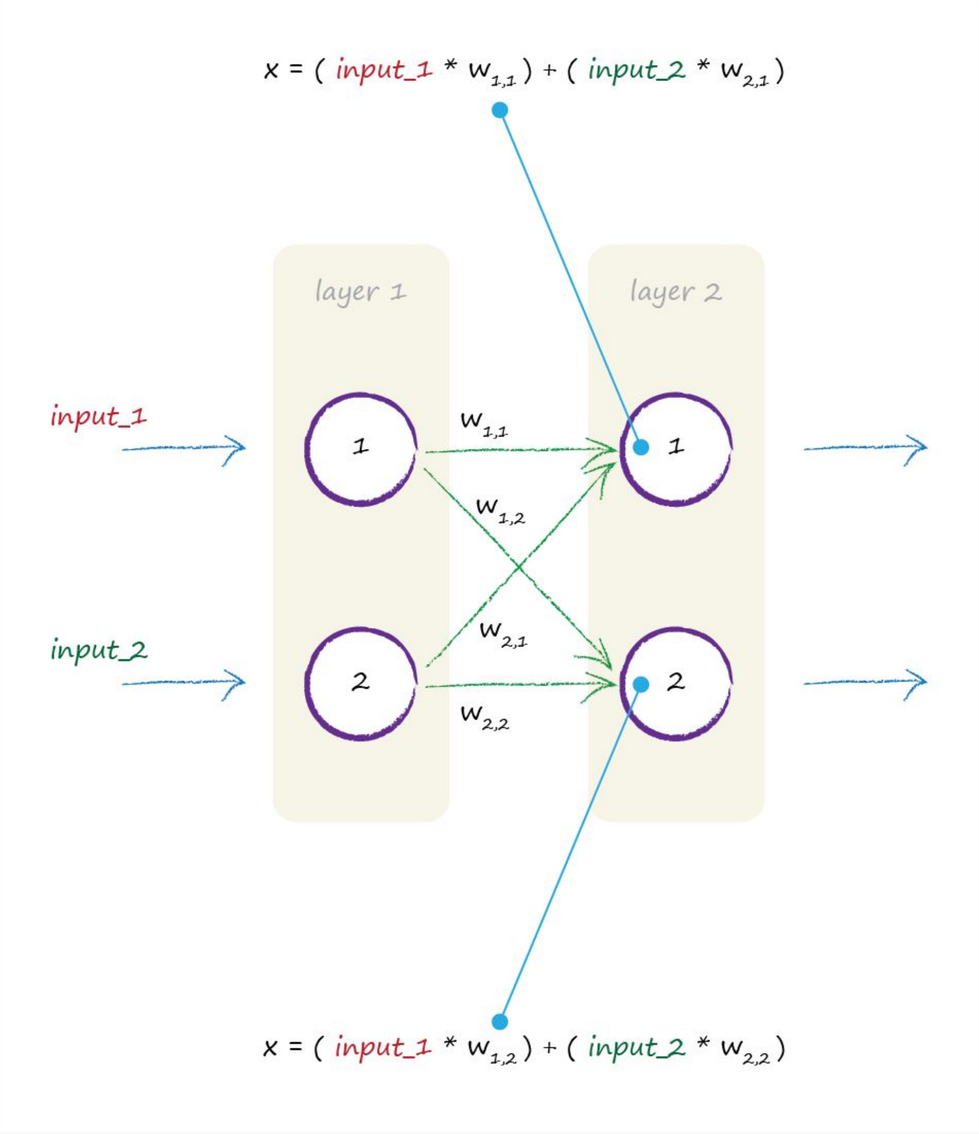
Feeding Signals Forward



Feeding Signals Forward



Matrix Multiplication



Matrix Multiplication

weights   incoming signals

$$\begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix} \begin{pmatrix} input\_1 \\ input\_2 \end{pmatrix} = \begin{pmatrix} (input\_1 * w_{1,1}) + (input\_2 * w_{2,1}) \\ (input\_1 * w_{1,2}) + (input\_2 * w_{2,2}) \end{pmatrix}$$

$$W \cdot I = X$$

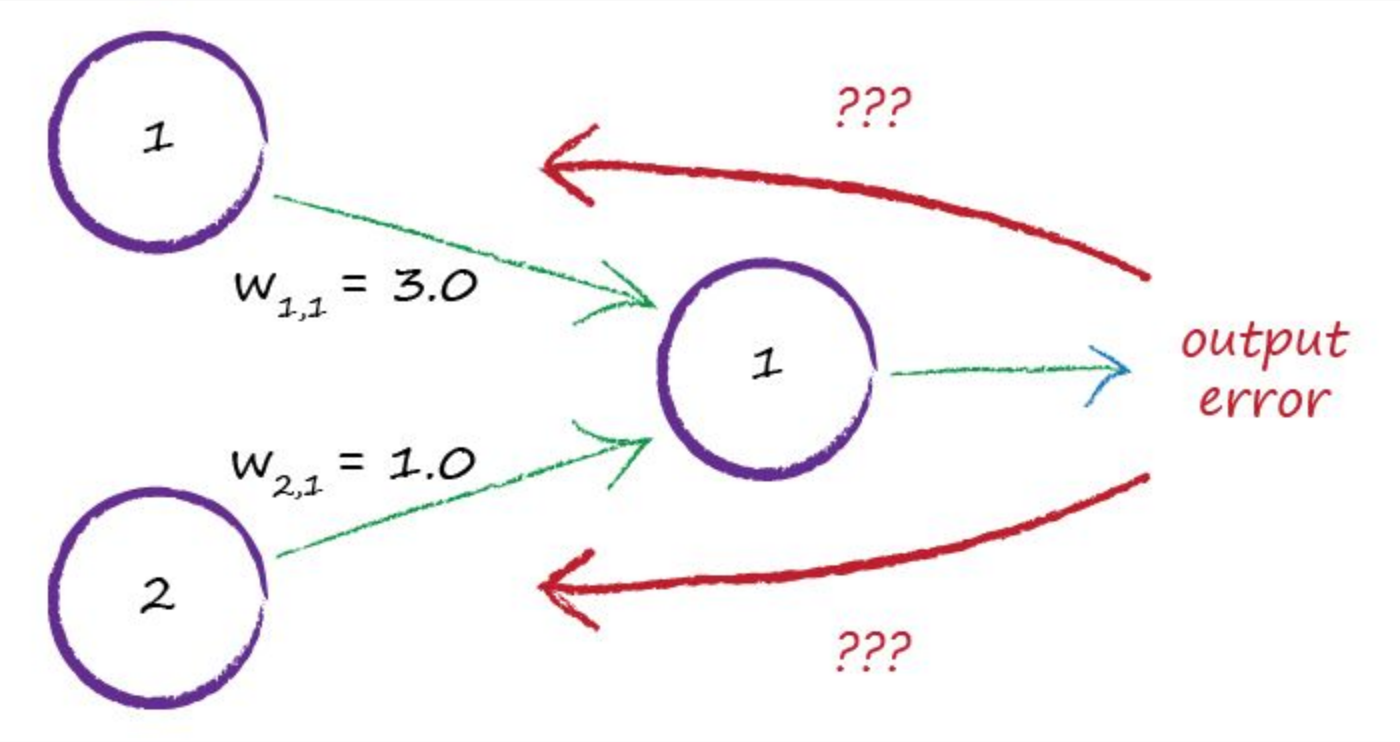
dot product

Key Points

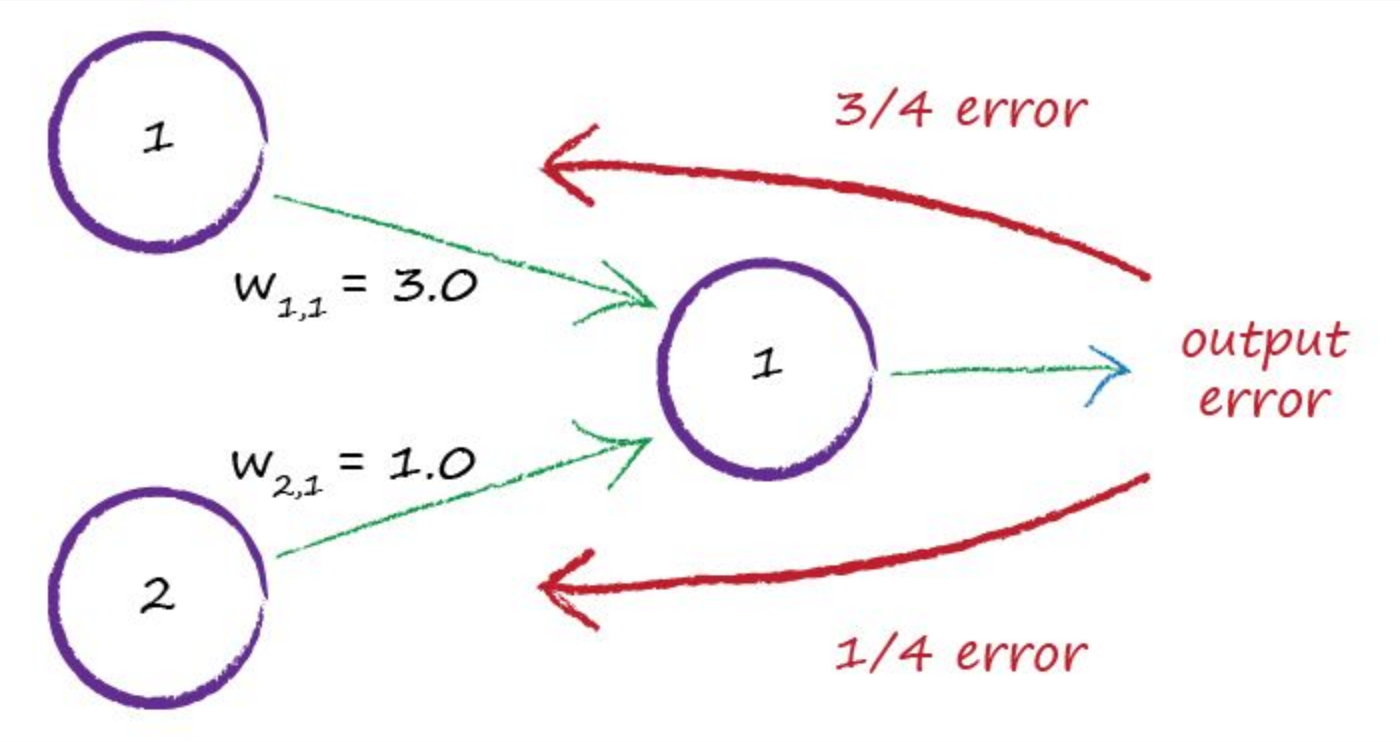
1. The many feedforward calculations can be expressed **concisely** as **matrix multiplication**, no matter what shape the network.

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

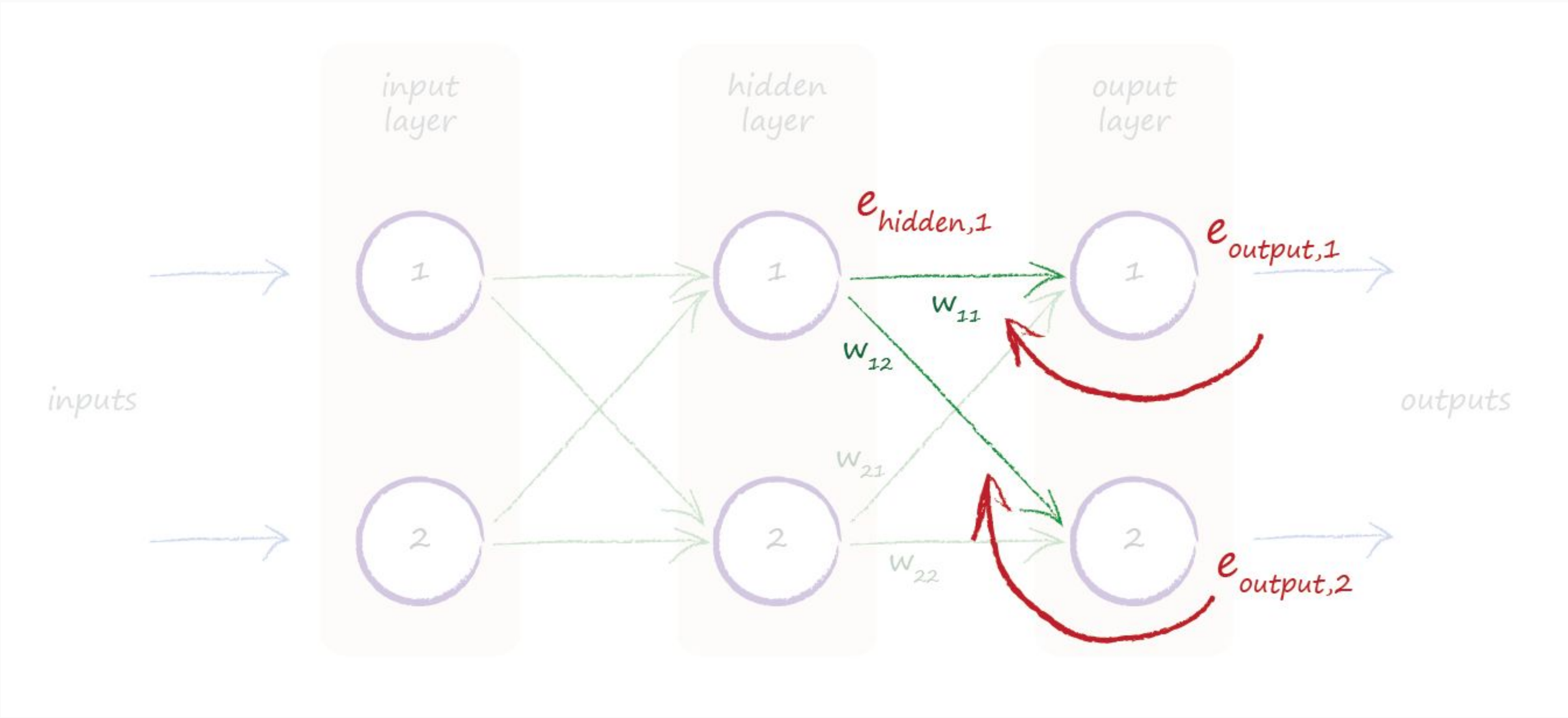
Network Error



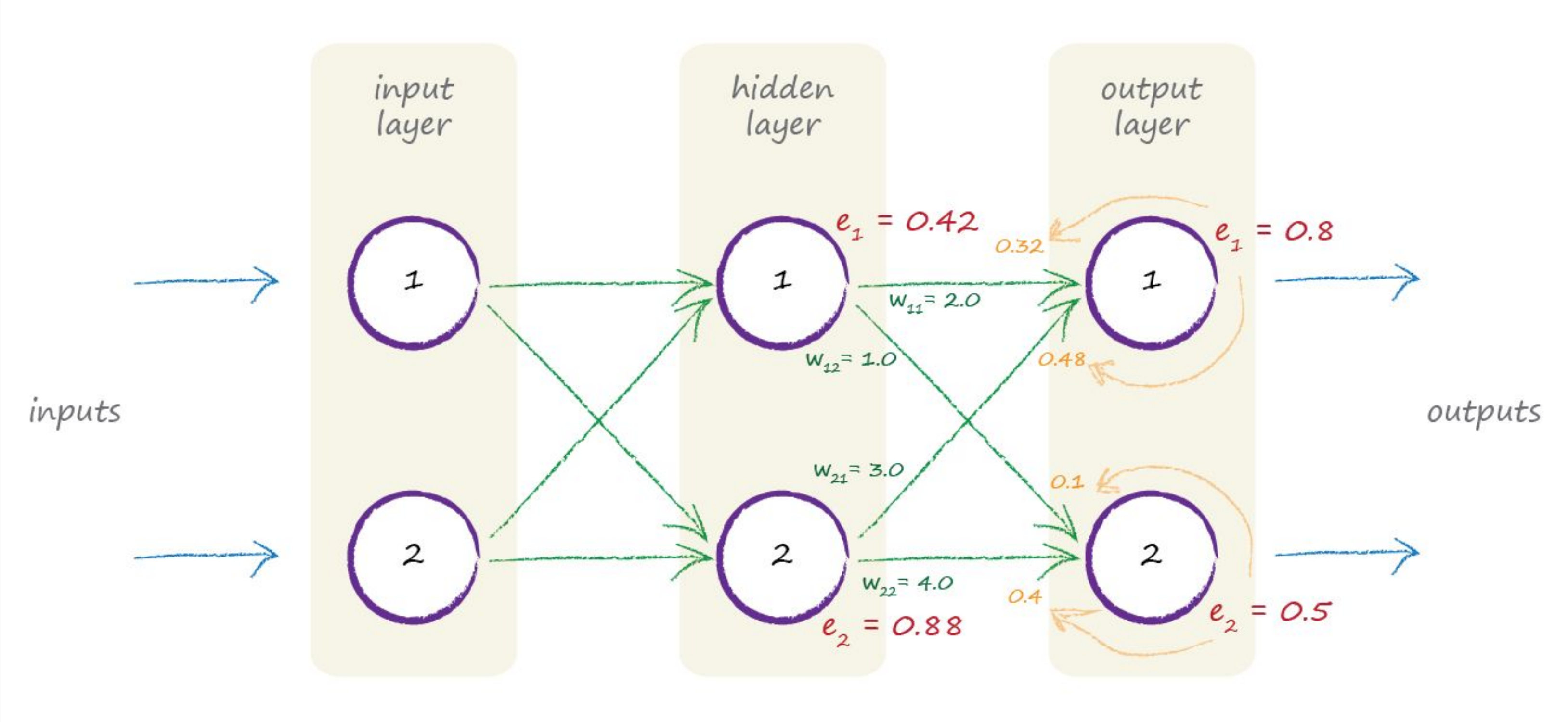
Network Error



Internal Error



Internal Error



Matrices Again!

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

$$error_{hidden} = W^T_{hidden\_output} \cdot error_{output}$$

Key Points

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.

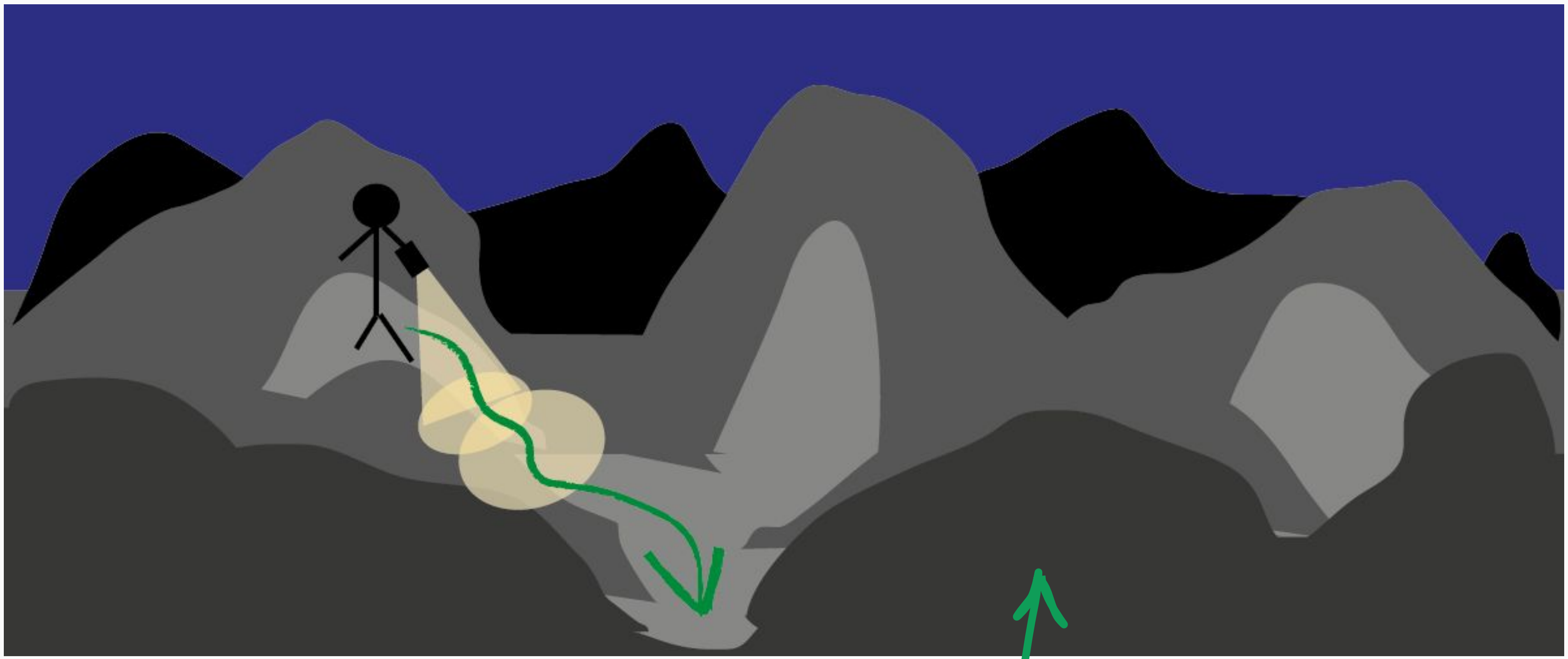
4. … and back propagating the error can be expressed as a **matrix** multiplication too!

Yes, But How Do We Actually Update The Weights?

$$o_k = \frac{1}{1 + e^{-\sum_{j=1}^{3}\left(w_{j,k} \cdot \frac{1}{1+e^{-\sum_{i=1}^{3}(w_{i,j} \cdot x_i)}}\right)}}$$

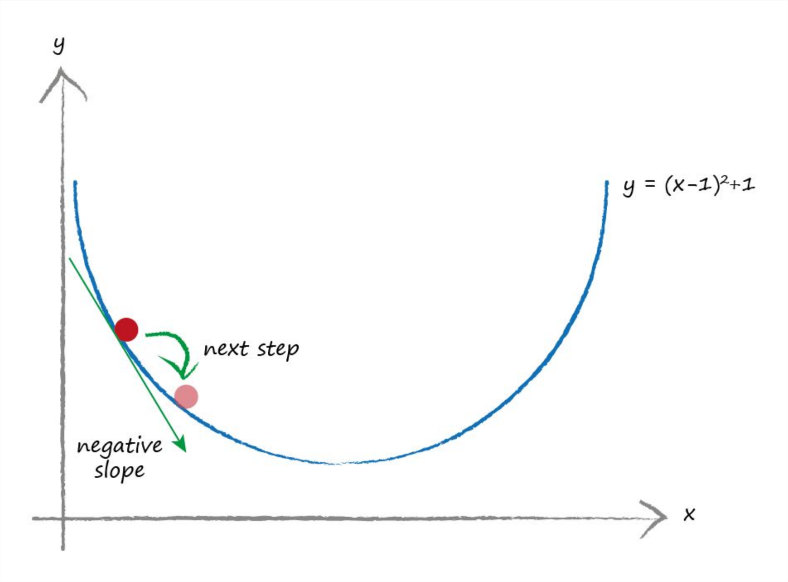
Aaarrrggghhh !!

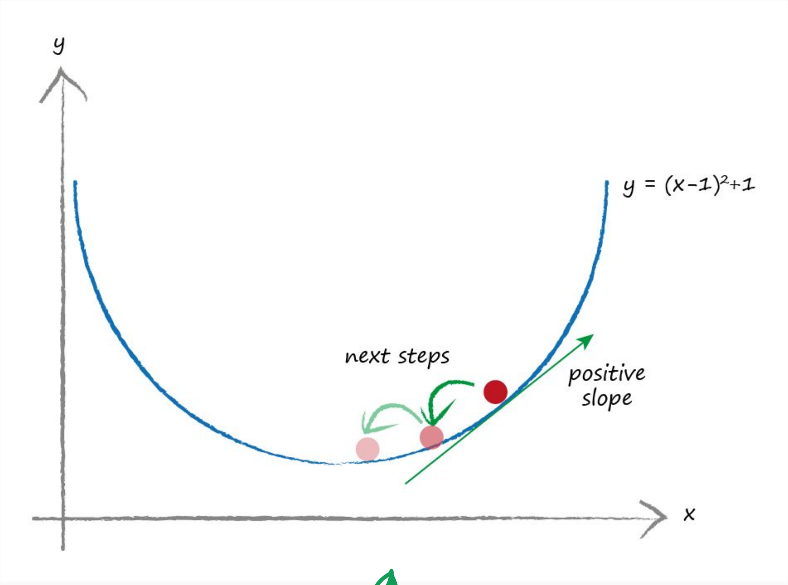
Perfect is the Enemy of Good



landscape is a complicated difficult mathematical function ..
... with all kinds of lumps, bumps, kinks ...

Gradient Descent





smaller gradient .. you're closer to the bottom ... take smaller steps?

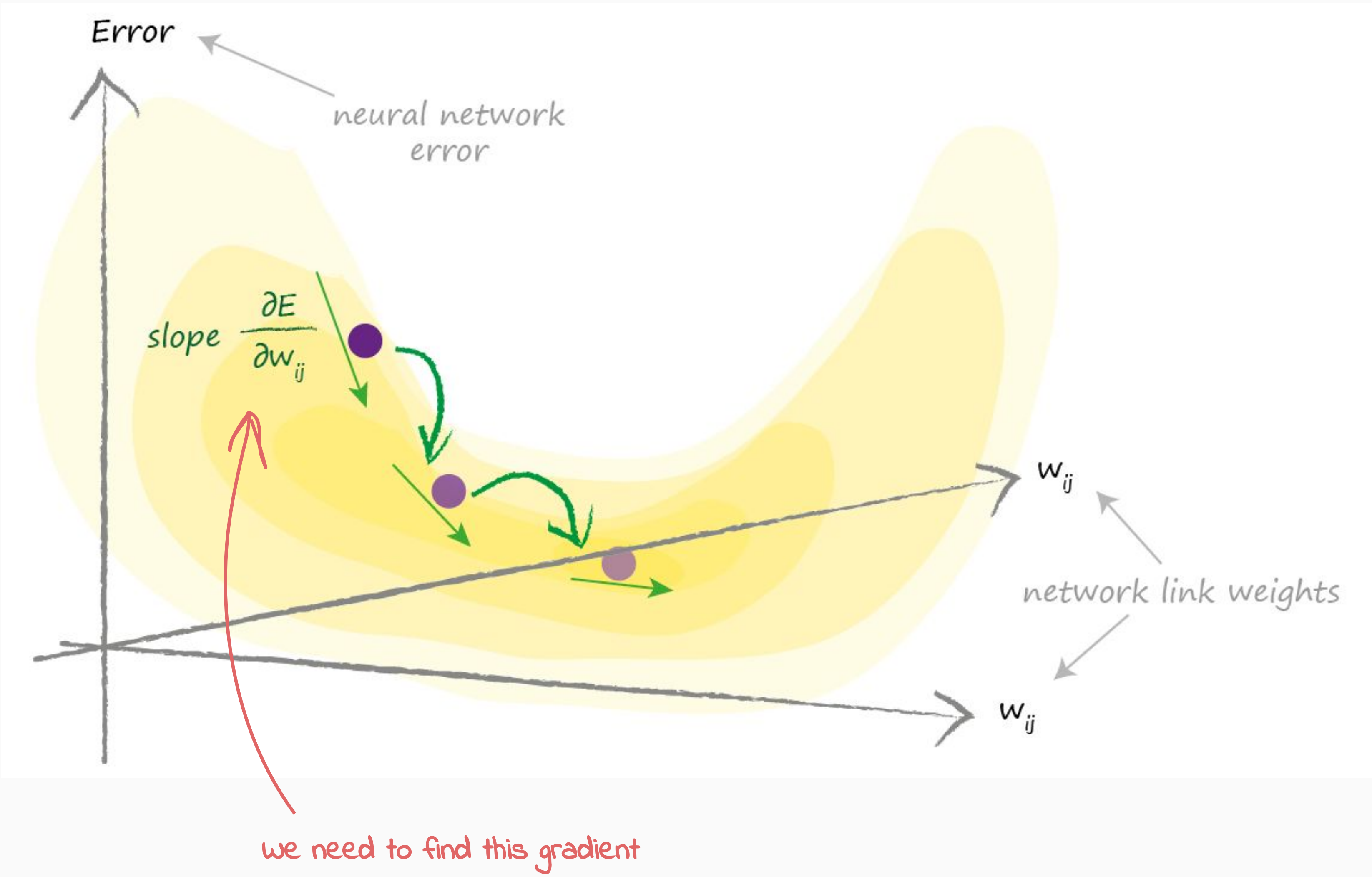
Key Points

1. **Gradient descent** is a practical way of finding the minimum of **difficult** functions.

2. You can avoid the chance of ***overshooting*** by taking smaller steps if the gradient gets shallower.

3. The error of a neural network is a **difficult** function of the link weights … so maybe gradient descent will help ...

Climbing Down the Network Error Landscape



Error Gradient

$$E = (\text{desired} - \text{actual})^2$$

school level calculus (chain rule)

$$dE/dw_{ij} = - e_j \cdot o_j \cdot (1 - o_j) \cdot o_i$$

previous node

A gentle intro to calculus

http://makeyourownneuralnetwork.blogspot.co.uk/2016/01/a-gentle-introduction-to-calculus.html

Updating the Weights

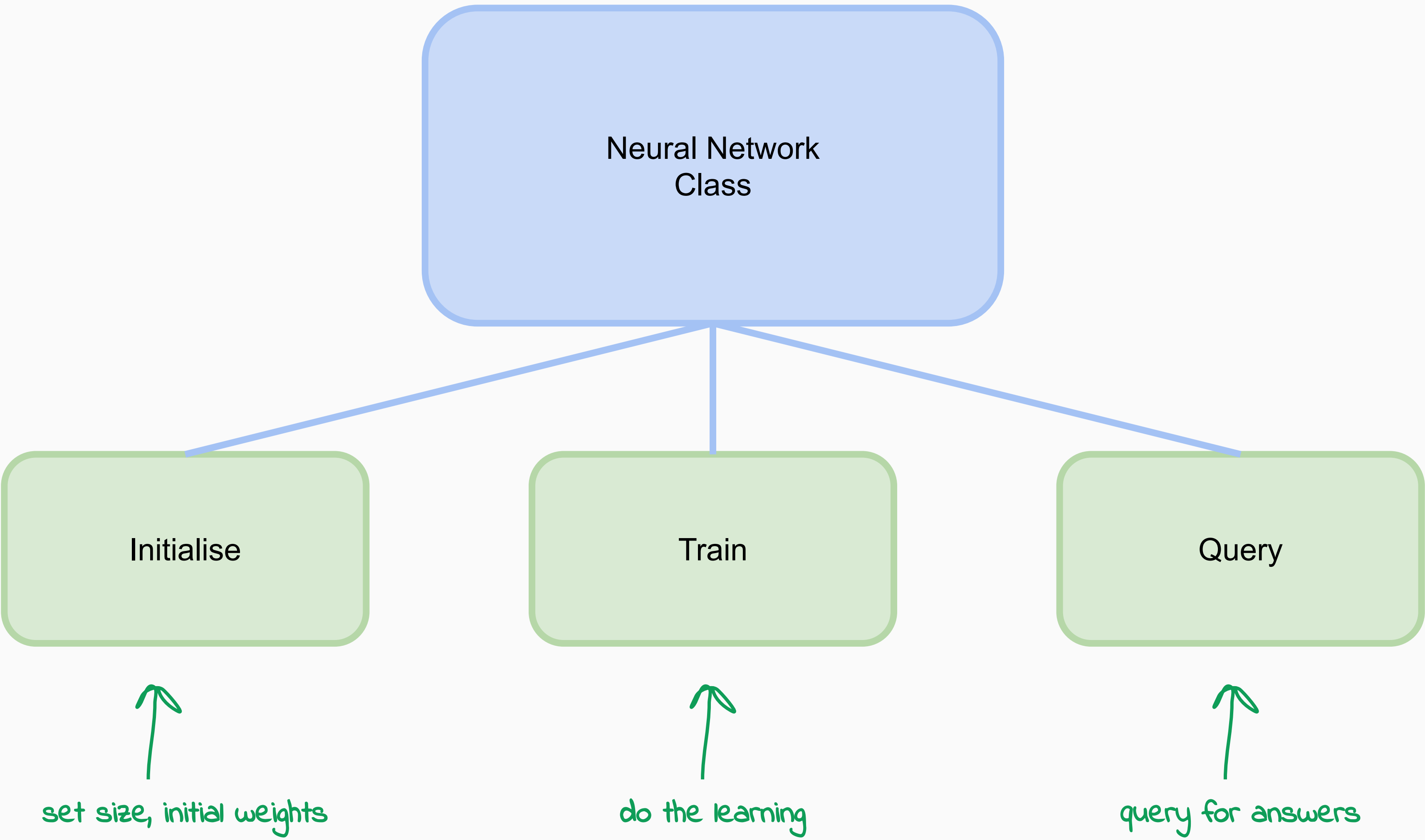
move w_{jk} in the opposite direction to the slope

$$\text{new } w_{jk} = \text{old } w_{jk} - \alpha \cdot \frac{\partial E}{\partial w_{jk}}$$

remember that learning rate

DIY

Python Class and Functions



Python has Cool Tools

matrix maths

numpy

scipy

matplotlib

notebook

Function - Initialise

```python
# 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

```python
# 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
```

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

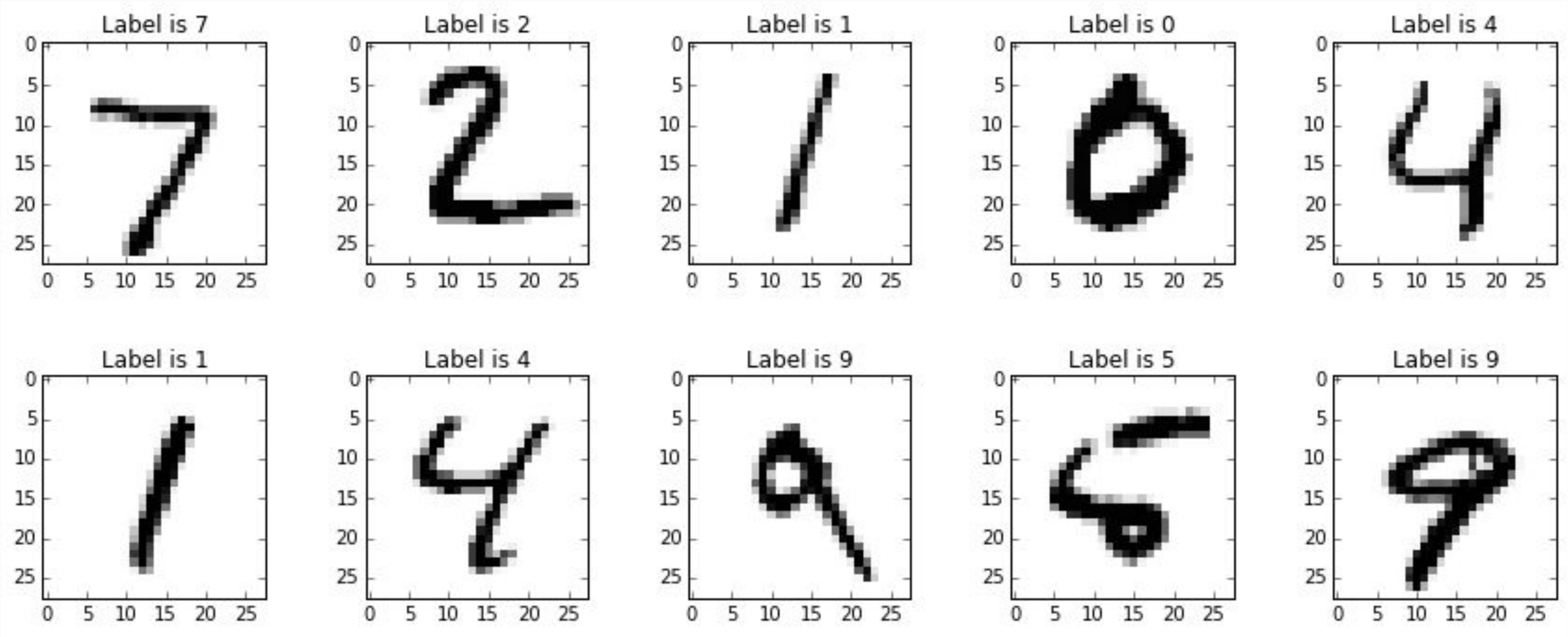
output layer errors

hidden layer errors

update weights

Handwriting

Handwritten Numbers Challenge



MNIST Datasets

MNIST dataset:

60,000 training data examples

10,000 test data examples

```
In [8]: data_file = open("mnist_dataset/mnist_train_100.csv", 'r')
        data_list = data_file.readlines()
        data_file.close()

In [9]: len(data_list)

Out[9]: 100

In [10]: data_list[0]

Out[10]: '5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,18,18,18,126,136,175,26,166,255,247,127,0,
0,0,0,0,0,0,0,0,0,0,0,30,36,94,154,170,253,253,253,253,253,225,172,253,242,195,64,0,0,0,0,0,0,0,0,0,0,0,49,238,253,25
3,253,253,253,253,253,253,251,93,82,82,56,39,0,0,0,0,0,0,0,0,0,0,0,0,18,219,253,253,253,253,253,198,182,247,241,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,80,156,107,253,253,205,11,0,43,154,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,14,1,154,253,
90,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,139,253,190,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11
,190,253,70,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,35,241,225,160,108,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,81,240,253,253,119,25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,45,186,253,253,150,27,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,93,252,253,187,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,249,253,249,64,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,46,130,183,253,253,207,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,148,229,253,
253,253,250,182,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,24,114,221,253,253,253,253,201,78,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,23,66,213,253,253,253,253,198,81,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,18,171,219,253,253,253,253,195,80,9,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,55,172,226,253,253,253,253,244,133,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,136,253,253,253,212,13
5,132,16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\n'
```

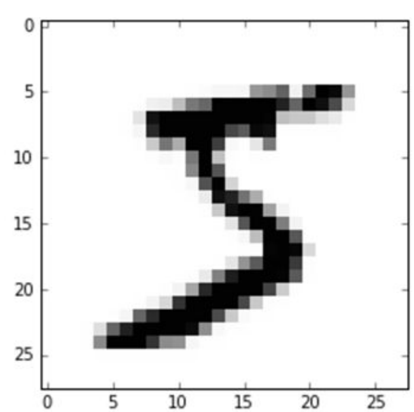
MNIST Datasets

```
In [8]: data_file = open("mnist_dataset/mnist_train_100.csv", 'r')
        data_list = data_file.readlines()
        data_file.close()

In [9]: len(data_list)

Out[9]: 100

In [10]: data_list[0]

Out[10] '5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,18,18,18,126,136,175,26,166,255,247,127,0,
0,0,0,0,0,0,0,0,0,0,0,30,36,94,154,170,253,253,253,253,253,225,172,253,242,195,64,0,0,0,0,0,0,0,0,0,0,0,49,238,253,25
3,253,253,253,253,253,253,251,93,82,82,56,39,0,0,0,0,0,0,0,0,0,0,0,0,18,219,253,253,253,253,253,198,182,247,241,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,80,156,107,253,253,205,11,0,43,154,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,14,1,154,253,
90,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,139,253,190,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11
,190,253,70,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,35,241,225,160,108,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,81,240,253,253,119,25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,45,186,253,253,150,27,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,93,252,253,187,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,249,253,249,64,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,46,130,183,253,253,207,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,148,229,253,
253,253,250,182,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,24,114,221,253,253,253,253,201,78,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,23,66,213,253,253,253,253,198,81,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,18,171,219,253,253,253,253,195,80,9,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,55,172,226,253,253,253,253,244,133,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,136,253,253,253,212,13
5,132,16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\n'
```

label

784 pixels values

```
In [32]: all_values = data_list[0].split(',')
         image_array = numpy.asfarray(all_values[1:]).reshape((28,28))
         matplotlib.pyplot.imshow(image_array, cmap='Greys', interpolation='None')

Out[32]: <matplotlib.image.AxesImage at 0x108818cc0>
```

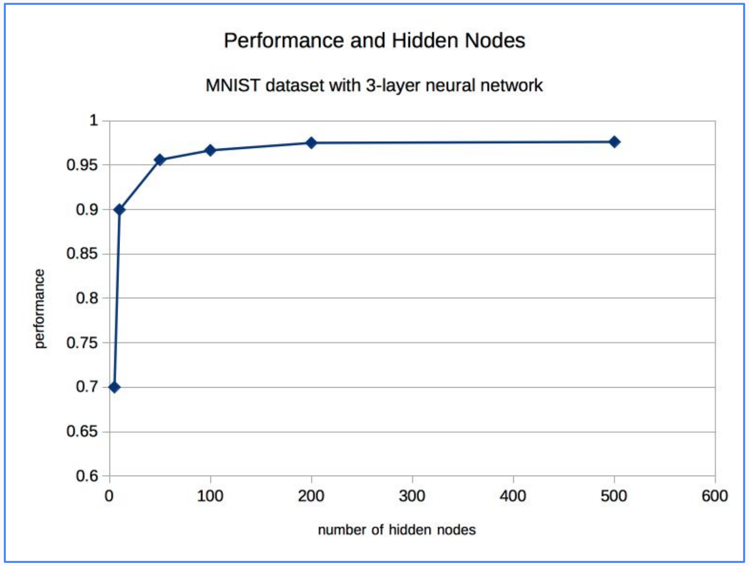


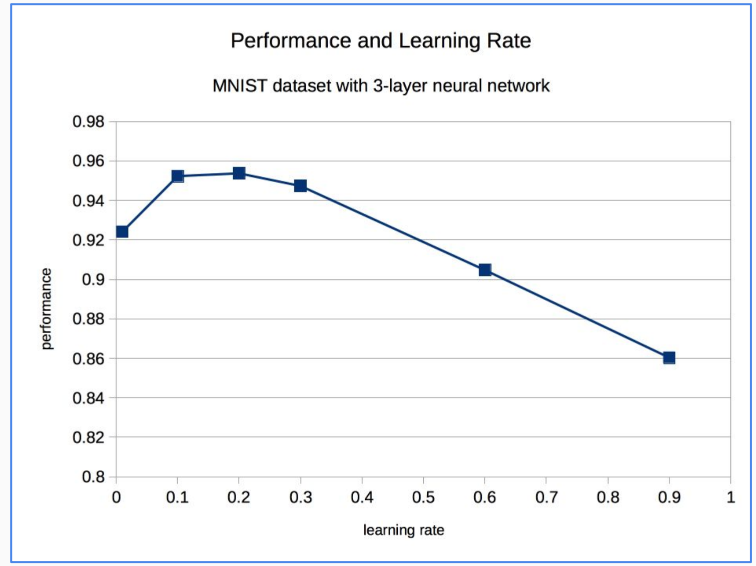
28 by 28 pixel image

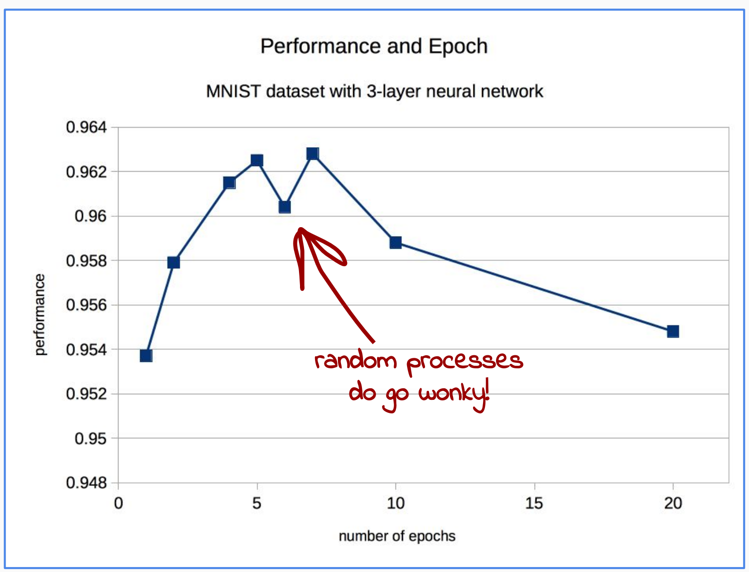
Output Layer Values

output layer	label	example "5"	example "0"	example "9"
0	0	0.00	0.95	0.02
1	1	0.00	0.00	0.00
2	2	0.01	0.01	0.01
3	3	0.00	0.01	0.01
4	4	0.01	0.02	0.40
5	5	0.99	0.00	0.01
6	6	0.00	0.00	0.01
7	7	0.00	0.00	0.00
8	8	0.02	0.00	0.01
9	9	0.01	0.02	0.86

Experiments



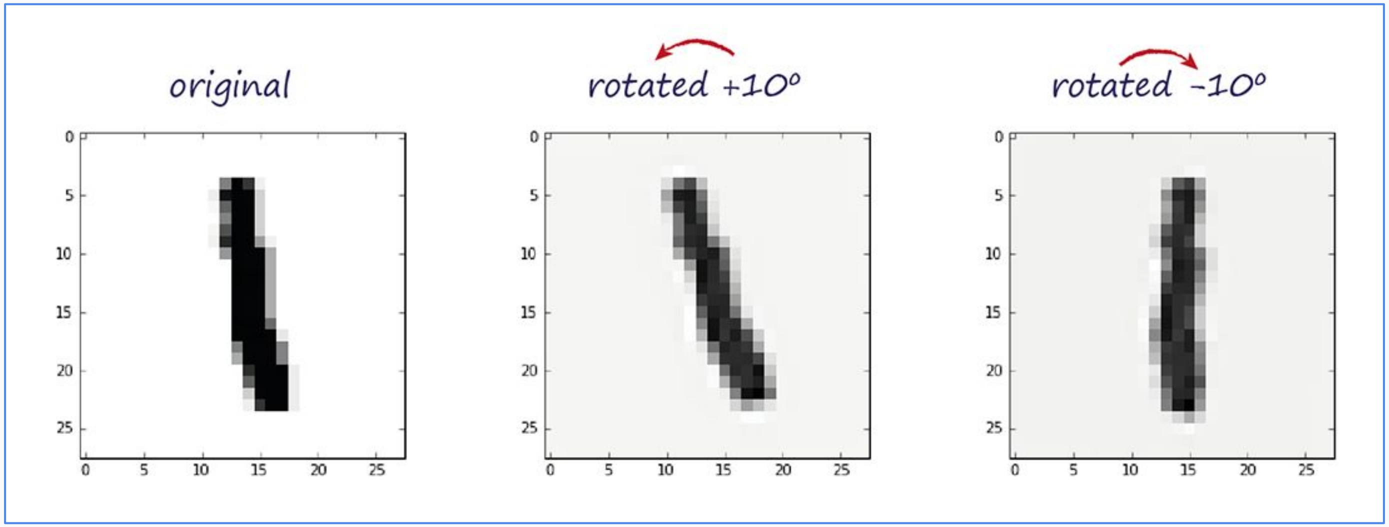


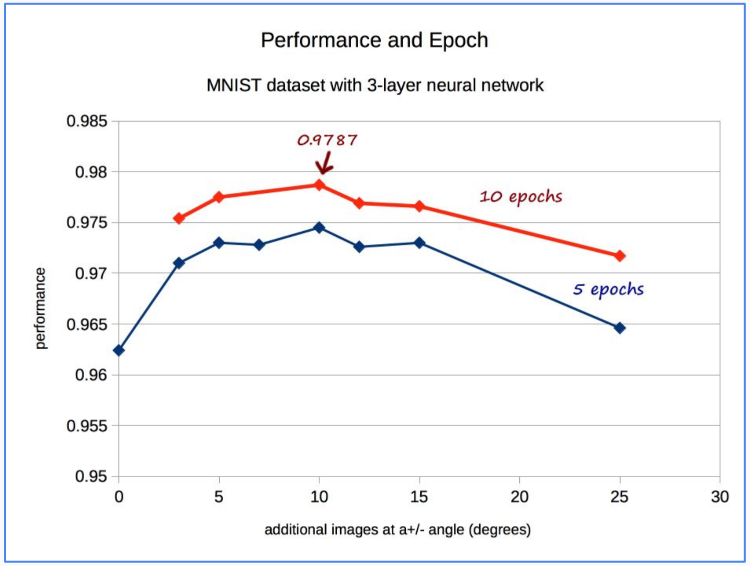


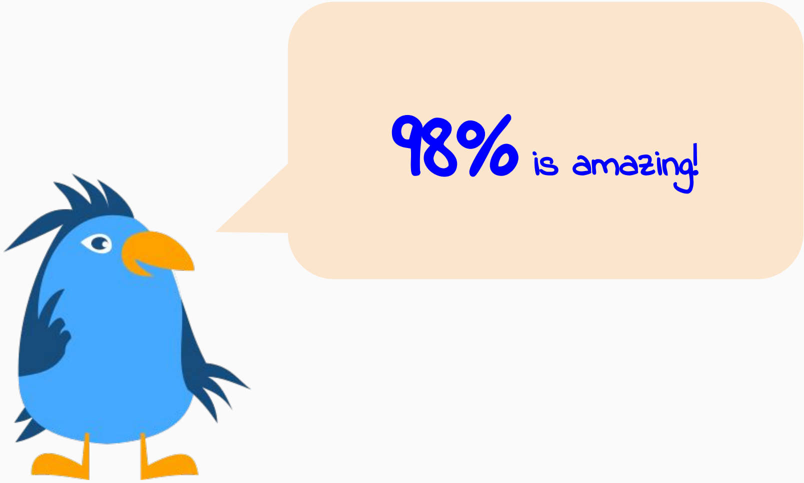
96% is very good!

we've only used simple ideas and code

More Experiments

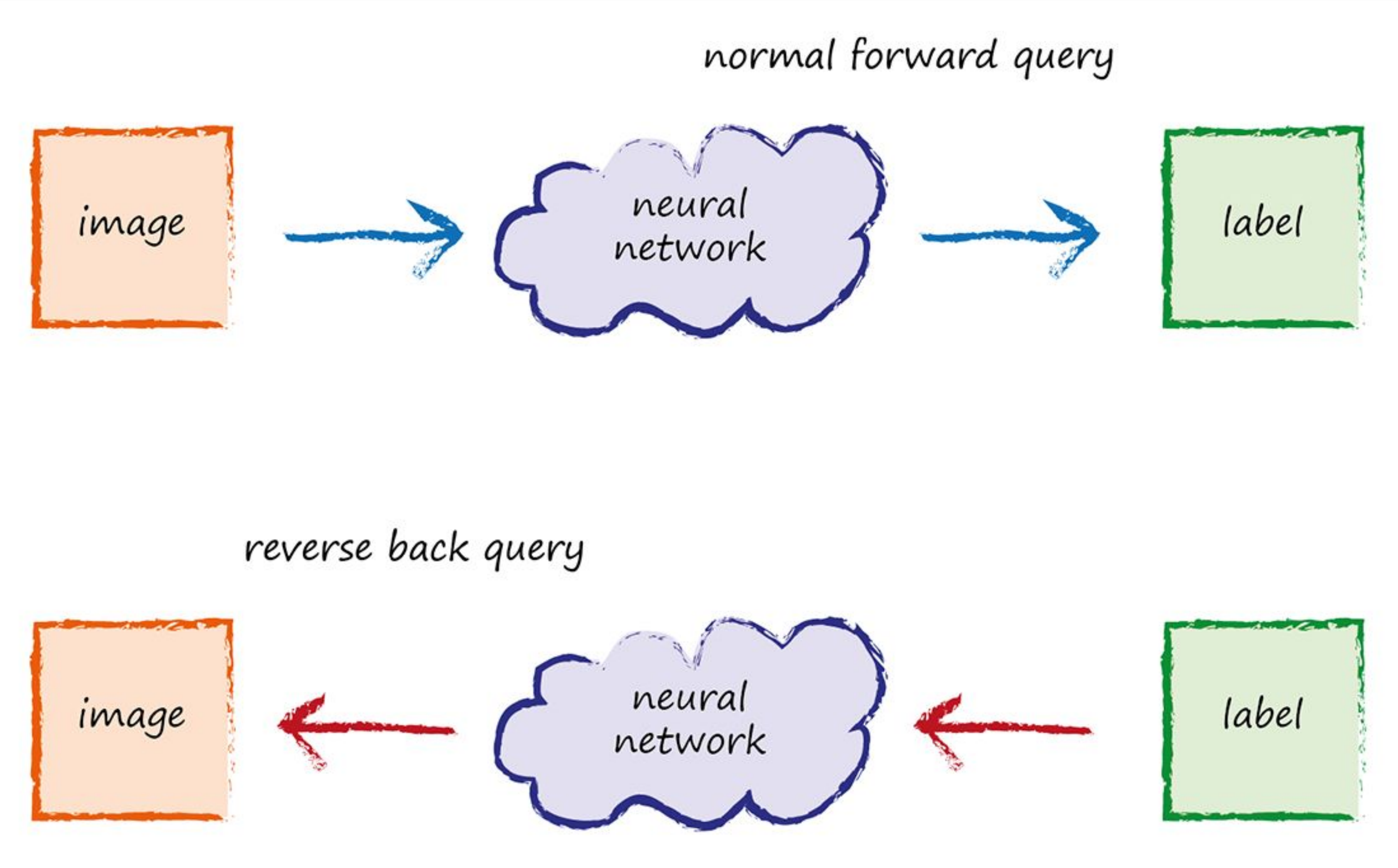




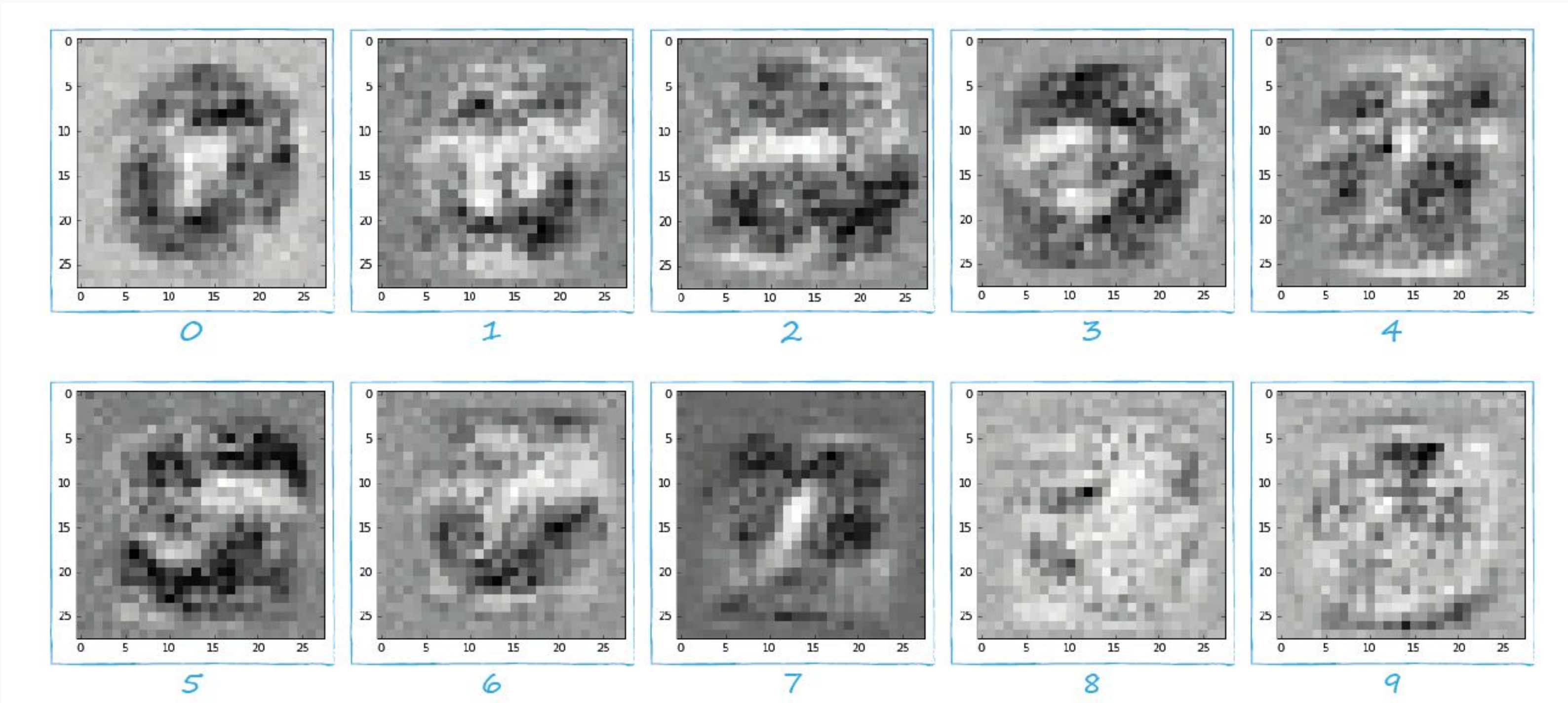


Thoughts

Peek Inside The Mind Of a Neural Network?



Peek Inside The Mind Of a Neural Network?



Thanks!

Finding Out More

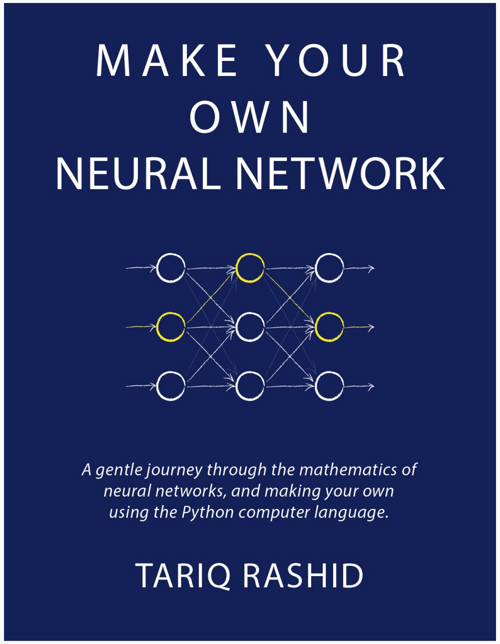
makeyourownneuralnetwork.blogspot.co.uk

github.com/makeyourownneuralnetwork

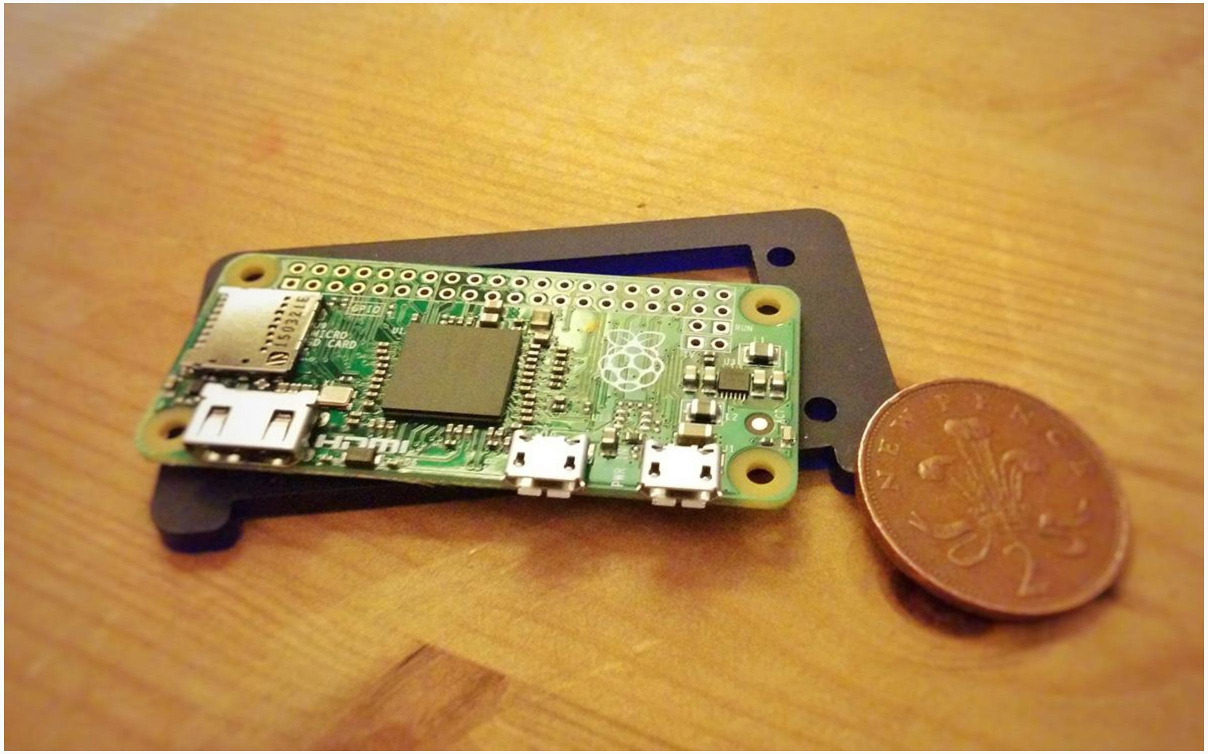
www.amazon.co.uk/dp/B01EER4Z4G

twitter.com/myoneuralnet

slides goo.gl/JKsb62



Raspberry Pi Zero



It all works on a Raspberry Pi Zero
... and it only costs £4 / $5 !!

Data Science
Cornwall