Neural Networks – An introduction

learner.org

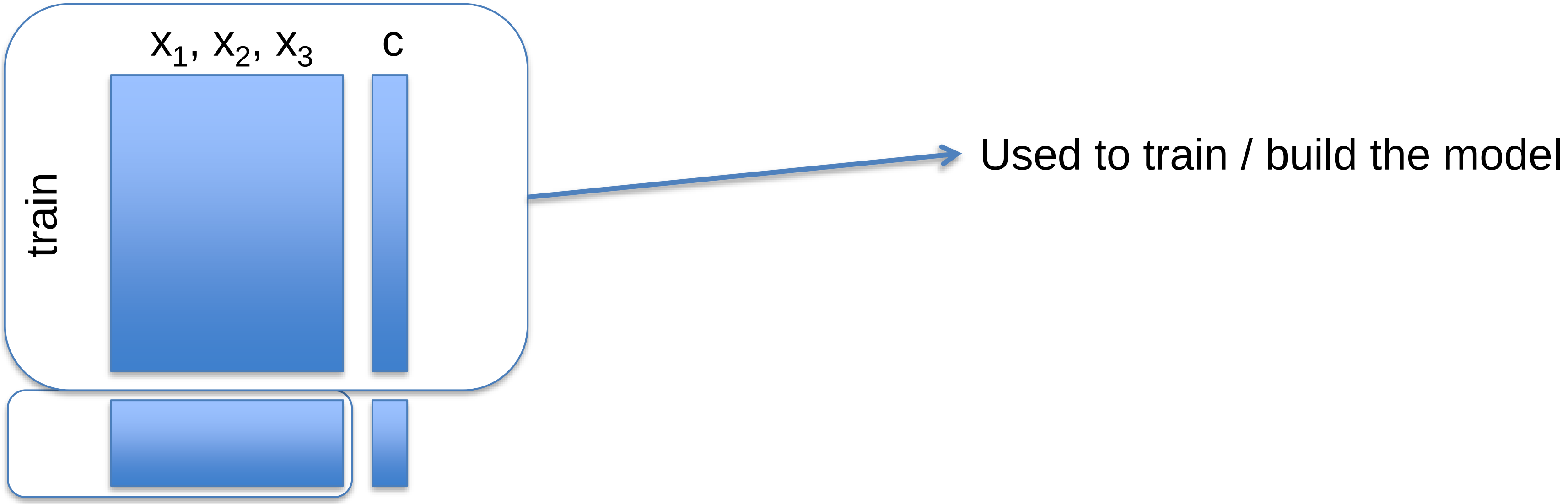
Brunel University London

Clustering Labs

- Explore K Means Clusters applied to iris data
- Test it by using WK with the K Means Results and iris_real
  - Different values of K (e.g. 2, 3, 4, 5)
- And then try using Hierarchical:
  - Average
  - Single
  - Complete

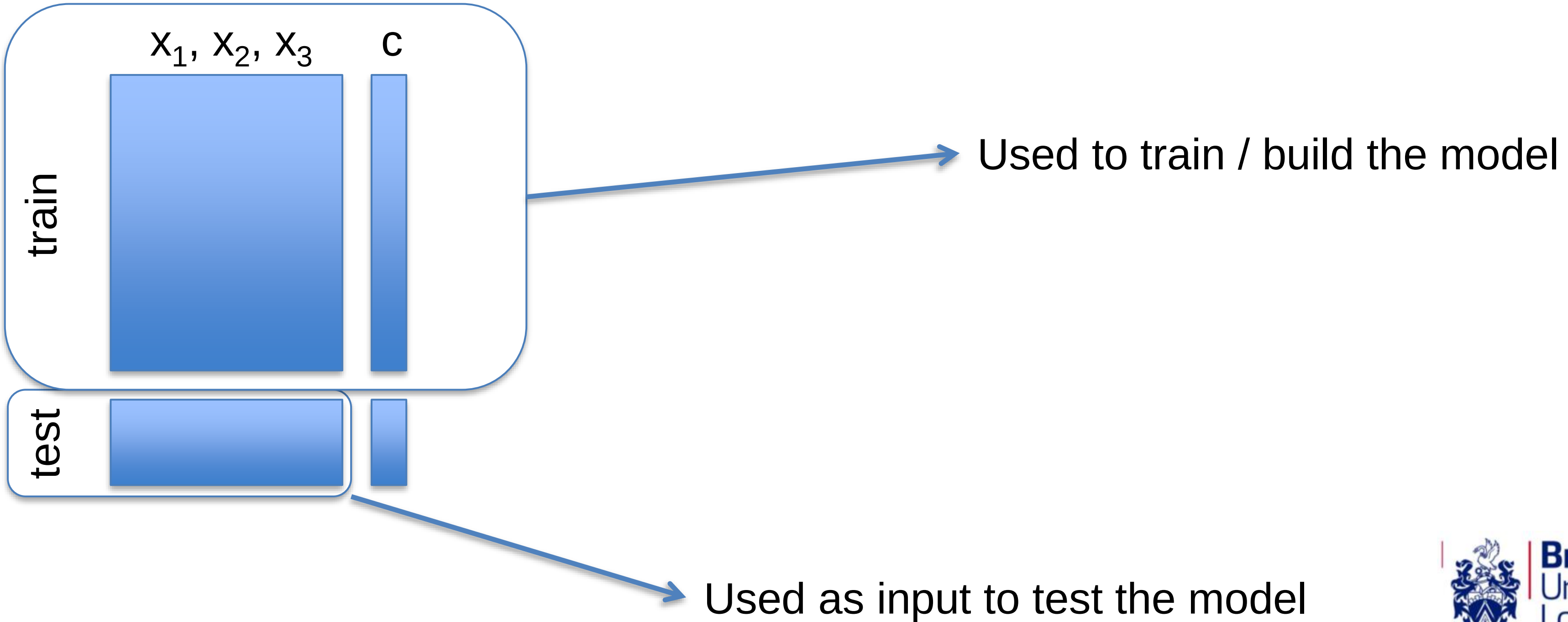
Classification Labs

- You need to ensure that you are splitting the data into the data and its classes:



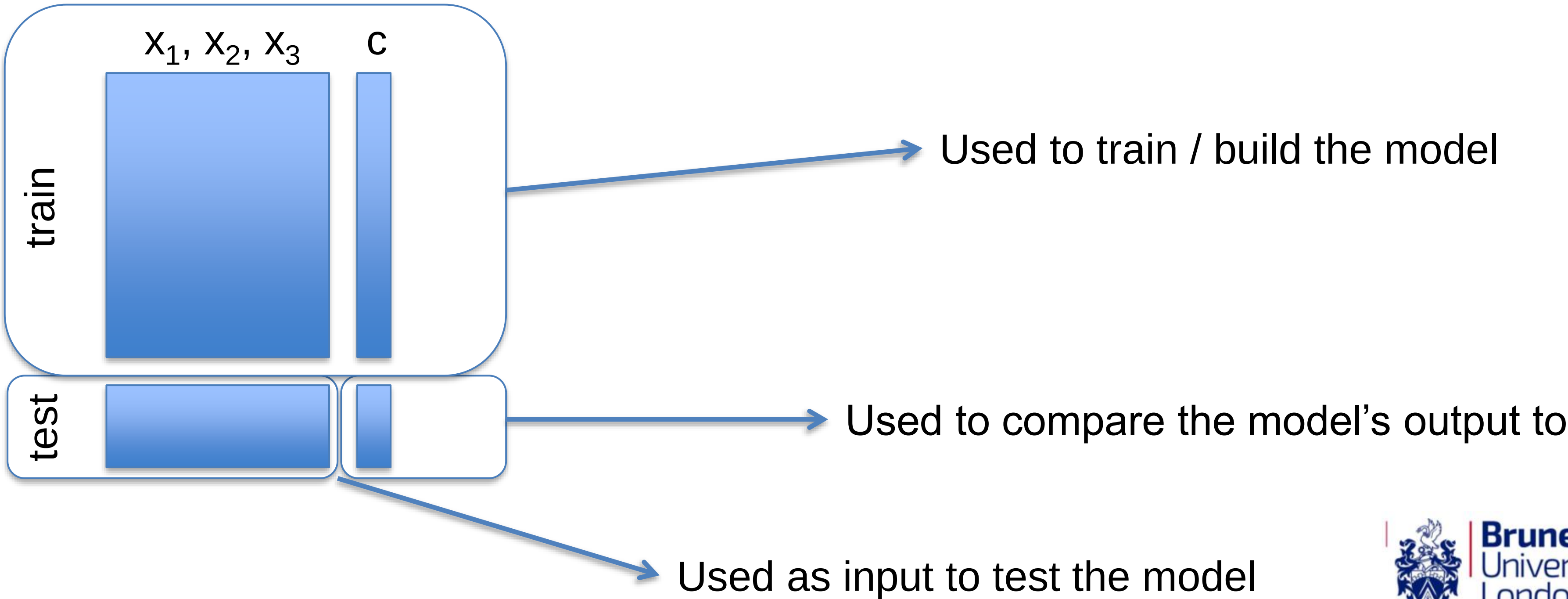
Classification Labs

- You need to ensure that you are splitting the data into the data and its classes:



Classification Labs

- You need to ensure that you are splitting the data into the data and its classes:



Neural Networks – An introduction

learner.org

Neural Networks

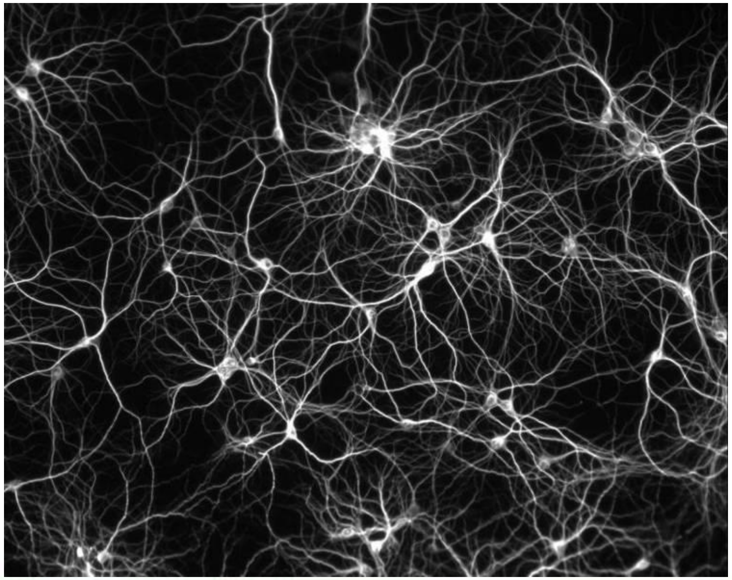
- In this lecture and lab:
  - Introduced to the concept of neural networks
  - The perceptron and how it works
  - Learning the weights for perceptrons
  - The XOR problem
  - Multilayer neural networks
  - Backpropagation?
  - Neural networks in R

Neural Networks

- Inspiration from Biology
- Biological neurons:
  - Have many interconnections
  - Inputs and outputs
  - Make use of simple thresholds

Neural Networks

- Our brain can be considered as a highly complex, non-linear and parallel information-processing system
- **Question: How can we simulate these three operations with a computer system?**

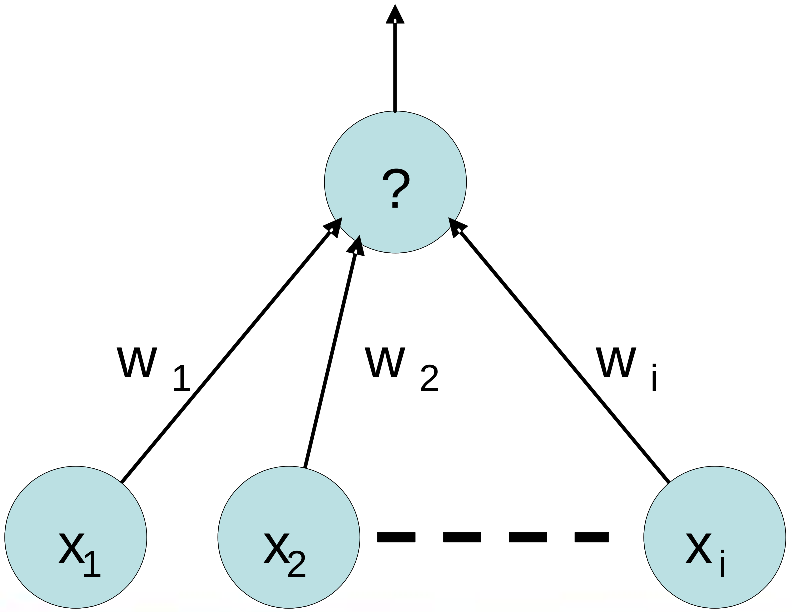


Perceptrons

- The answer lies in artificial Neural Networks (NNs)
- Simplest Neural Network structure
- Developed in the 50s and 60s

Perceptrons

- Now interpret each piece of evidence as an input to a neuron
- The general form of a Perceptron:



Perceptrons

Perceptron uses the following transfer (or activation) function :

$$\sum_i w_i x_i + \theta$$

$= w_1 x_1 + w_2 x_2 \ldots + w_d x_d + w_\theta$

where x_i is the ith input

w_i is associated weight

θ is the bias value (can be treated as a weight)

Brunel University London

Perceptrons

Perceptron uses the following transfer (or activation) function :

$$\sum_i w_i x_i + \theta$$

$= w_1 x_1 + w_2 x_2 \ldots + w_d x_d + w_\theta$

where x_i is the ith input

w_i is associated weight

θ is the bias value (can be treated as a weight)

Brunel University London

Perceptrons

The single neuron in a Perceptron produces an output, *Y*, based upon the computed output (threshold / sign function), *f* :

$$Y = \begin{cases} +1, if \sum_i w_i x_i + w_\theta > 0 \\ -1, otherwise \end{cases}$$

Perceptrons

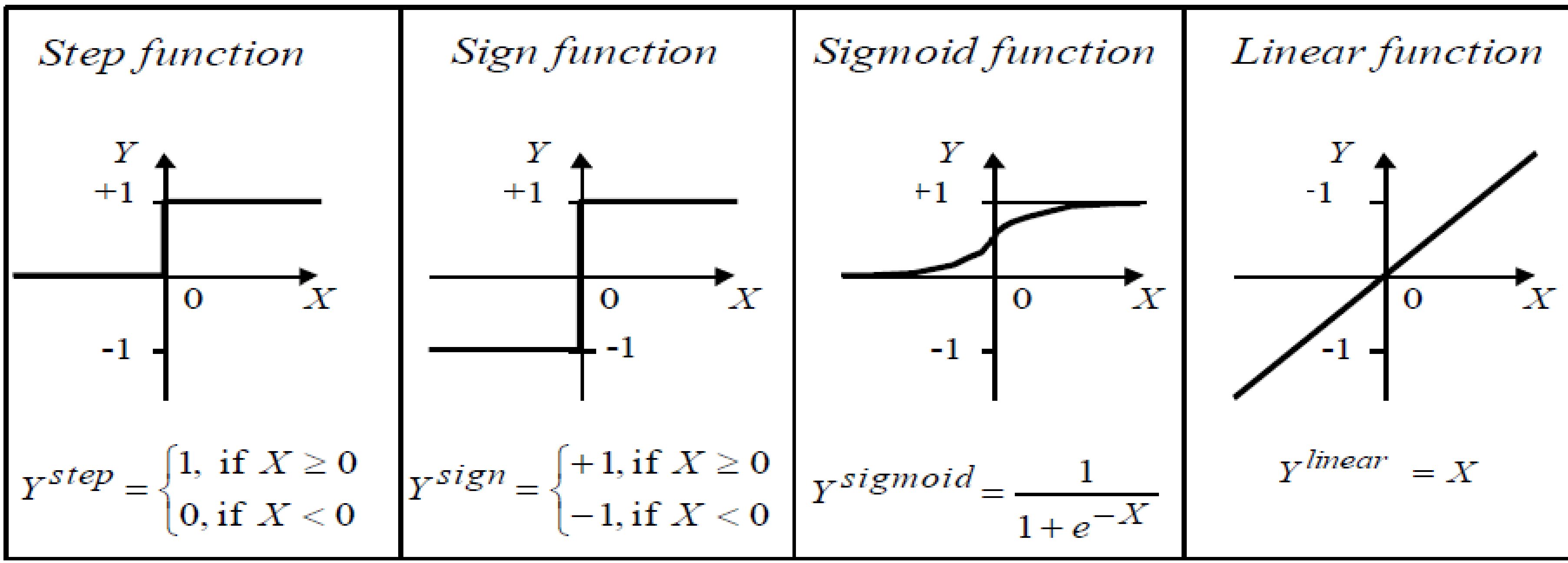
The single neuron in a Perceptron produces an output, *Y*, based upon the computed output (threshold / sign function), *f* :

$$Y = \begin{cases} +1, if \sum_i w_i x_i + w_\theta > 0 \\ -1, otherwise \end{cases}$$

Brunel University London

Perceptrons

- Other functions available:

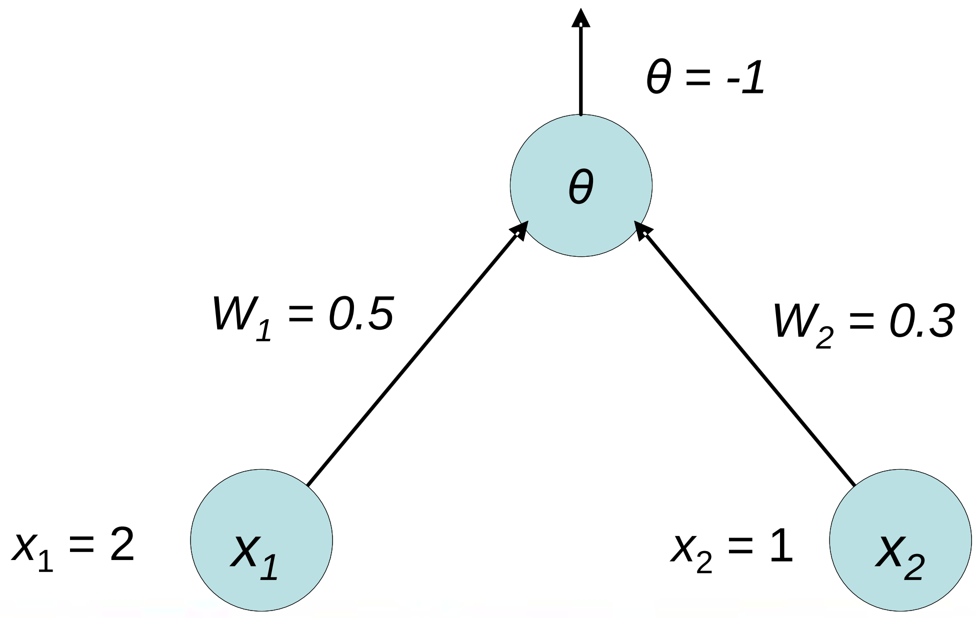


Classification

Backpropagation (later)

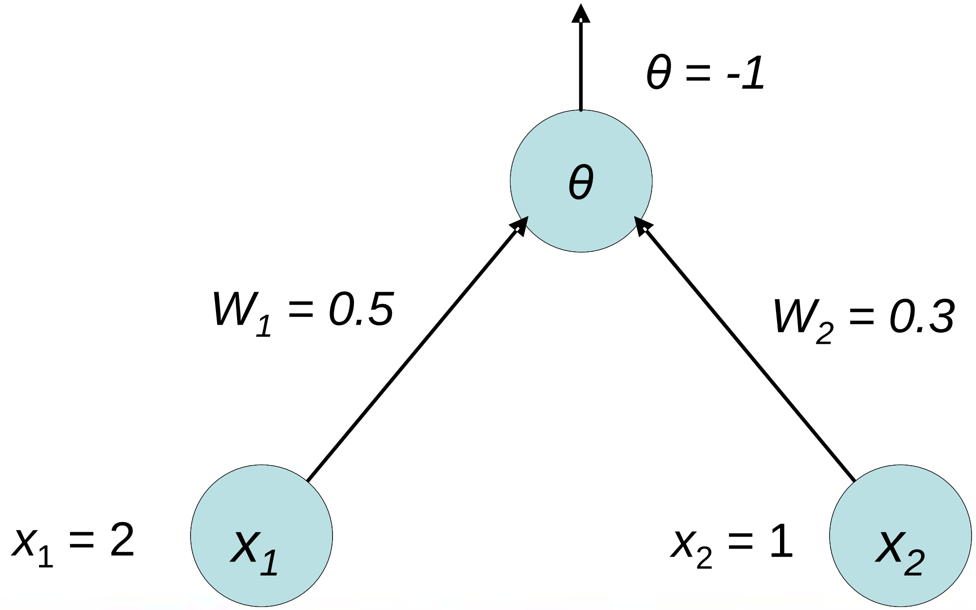
Perceptrons

Let's look at a simple classification example using a perceptron with two inputs, x_1 = 2 and x_2 = 1



Perceptrons

What will the output, *Y* be?

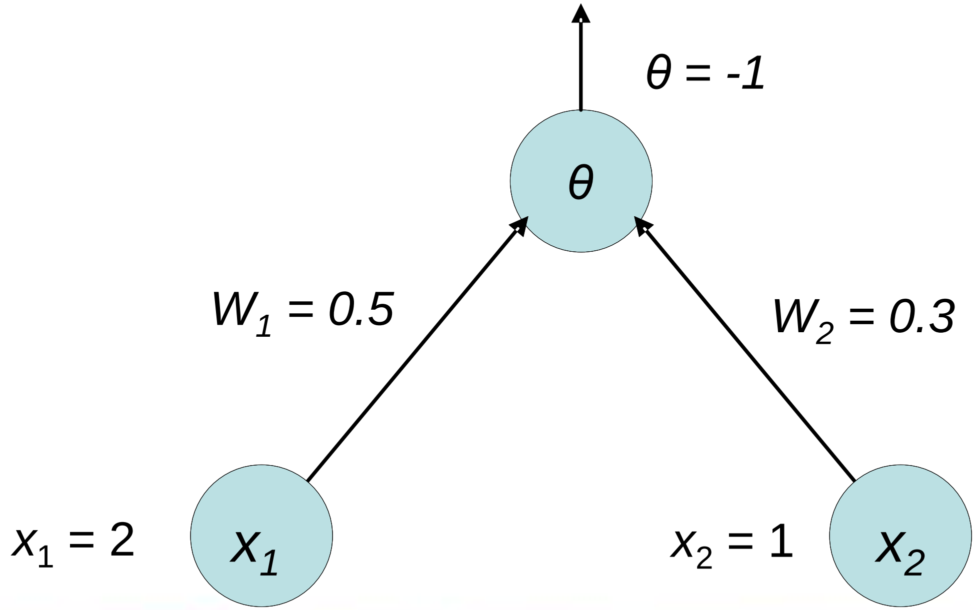


Perceptrons

What will the output, *Y* be?

$(2 \times 0.5) + (1 \times 0.3) – 1 = 0.3$

As $0.3 > 0$, the output *Y* will be +1

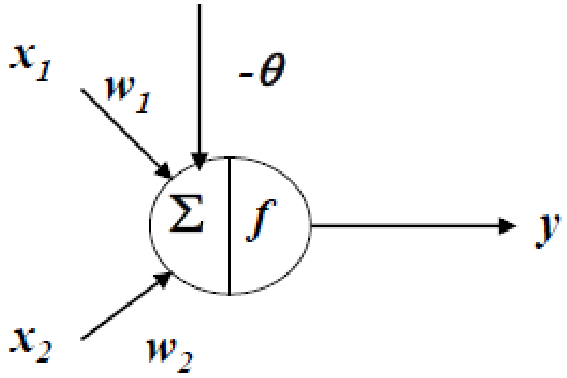


Perceptron for Logic

An example: logical **AND** operator

- Truth Table

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1



- Weights and threshold:

  w1=1, w2=1, θ=1.1

- Activation function: step function

Classify inputs into two classes: 0 or 1

Learning

- Must learn the weights (Rosenblatt 1960)
- Learning is iterative*:*
- Given some evidence the new updated weights and threshold are calculated at *step p*+1 given the values at *step p*:

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$
$$\theta(p+1) = \theta(p) + \Delta\theta(p)$$

Brunel University London

Learning

- An error correcting procedure is used:

$$\Delta w_i(p) = (Y^d - Y)x_i$$

$$\Delta\theta(p) = (Y^d - Y)$$

- Where Y^d is the True answer and Y is the perceptron output

Learning

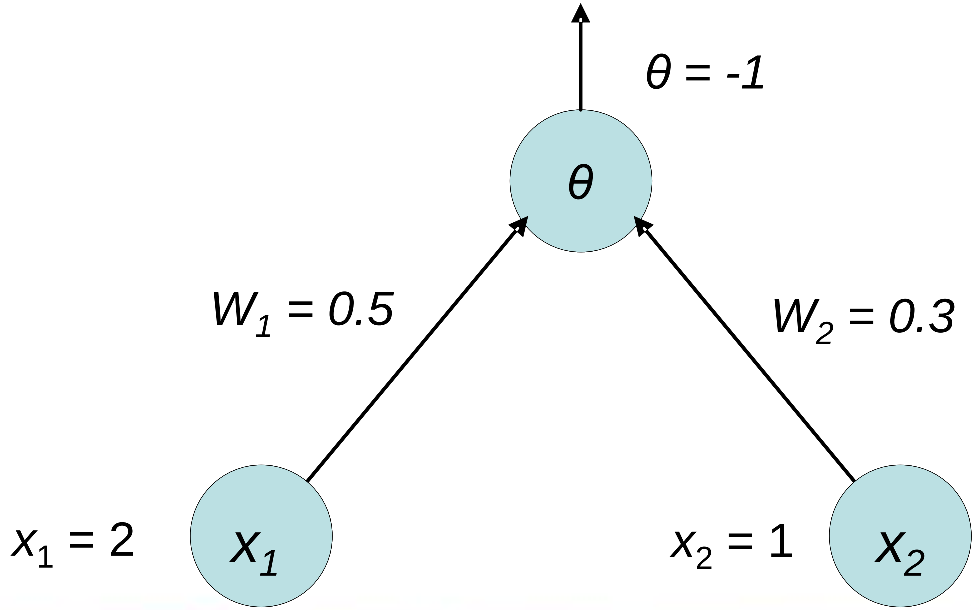
- Therefore, if the correct output is made no change is made
- Otherwise, false positives result in the weights and threshold beings reduced
- False negatives result in the weights and threshold being increased

Perceptrons

What will the output, *Y* be?

$(2 \times 0.5) + (1 \times 0.3) - 1 = 0.3$

As $0.3 > 0$, the output *Y* will be +1



Learning

- Using the previous example where the true output, Y=0

  $x_1 = 2, x_2 = 1$

  $w_1 = 0.5$, $w_2 = 0.3$, $\theta = -1$

  Y^d=0, Y= 1 (calculated earlier)

- The weights will be updated as follows:

  $w_1(p+1) = 0.5 + (0\text{-}1) \text{ x } 2 = -1.5$

  $w_2(p+1) = 0.3 + (0\text{-}1) \text{ x } 1 = -0.7$

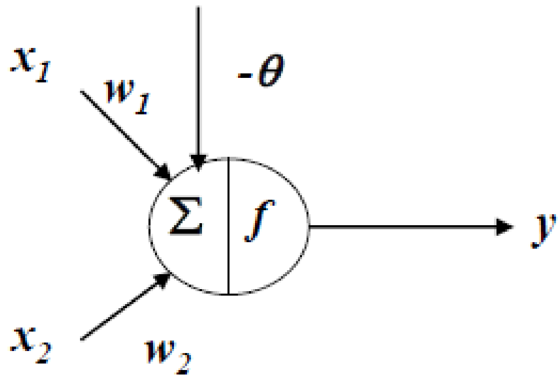
  $\theta(p+1) = -1 + (0\text{-}1) = -2$

Brunel University London

Learning

Think about running a bath…

- Y_d: desired temperature
- Y: current temperature
- w1: opening of hot water
- w2: opening of cold water



Adjust strategy:

- If $Y_d < Y$, decrease w1 and increase w2
- If $Y_d > Y$, increase w1 and decrease w2

Learning

- Usually the perceptron is initialised with random weights between 0 and 1
- Cases are presented to the perceptron one by one (in each step) and the weights updated
- If at least one error has been made during a full pass of data (*epoch*) then the entire set of cases are presented again
- Repeated until no error is made

Brunel University London

Learning

- This learning procedure is guaranteed to converge to a solution if the problem is linearly separable (remember the linear classifier last week)
- May take a very long time so:
  - Use a learning rate *α* to control the weight changes:

$$\Delta w_i(p) = \alpha(Y^d - Y)x_i$$

  - Normalise the data

Perceptron Learning Algorithm

1. Initialise w_i to random numbers,
2. set iteration $p = 1$;
3. Repeat
   4. Calculate activation: $Y(p) = f(\sum_i w_i(p)x_i)$
   5. Update weights:

$$\Delta w_i(p) = \alpha(Y^d - Y)x_i$$

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

   6. $p = p + 1$,
7. Until convergence

Brunel University London

Perceptron Learning Example

Input variables x_1	x_2	AND $x_1 \cap x_2$	OR $x_1 \cup x_2$	Exclusive-OR $x_1 \oplus x_2$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Perceptron Learning Example

Input variables x_1	x_2	AND $x_1 \cap x_2$	OR $x_1 \cup x_2$	Exclusive-OR $x_1 \oplus x_2$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
Epoch	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	−1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	−1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	−1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$.

Perceptron Learning Example

Input variables x_1	x_2	AND $x_1 \cap x_2$	OR $x_1 \cup x_2$	Exclusive-OR $x_1 \oplus x_2$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

The XOR Problem

- A classic non-linear problem
- Two binary inputs and one output
- If any one input is “on” and the other is “off” then the output is “on”
- Otherwise the output is “off”

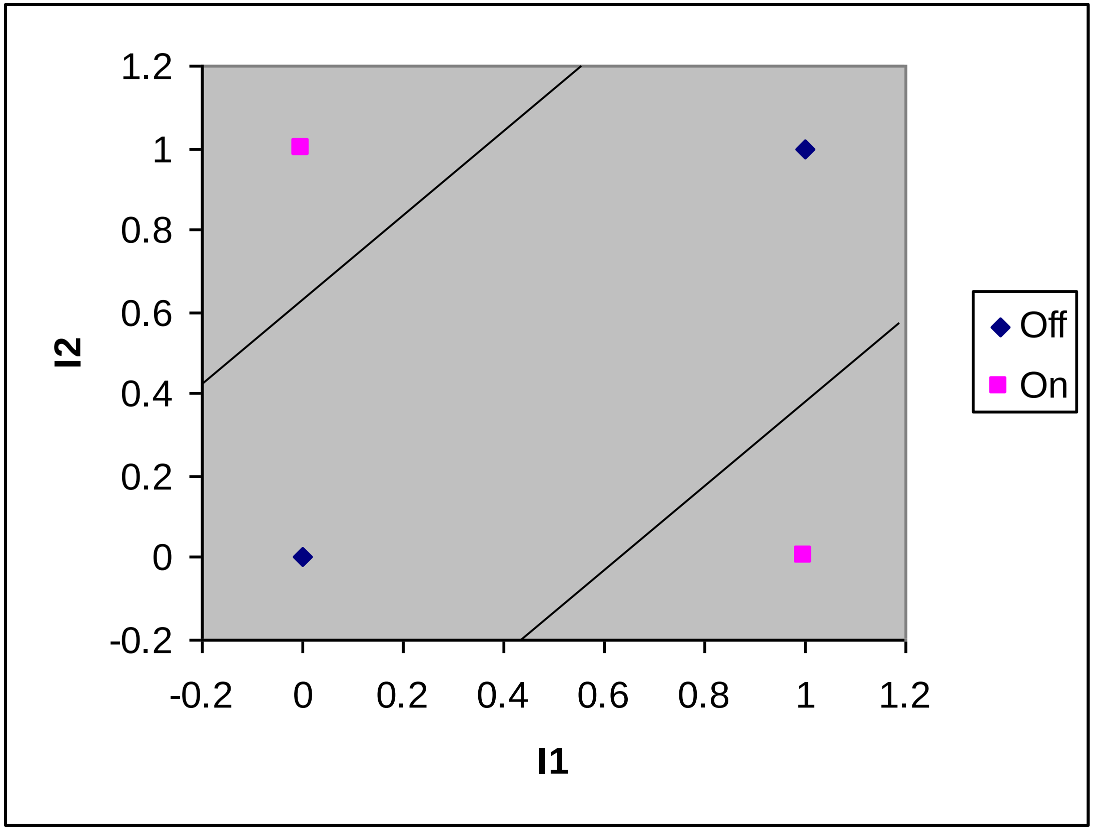
XOR(0,0) = 0

XOR(0,1) = 1

XOR(1,0) = 1

XOR(1,1) = 0

The XOR Problem

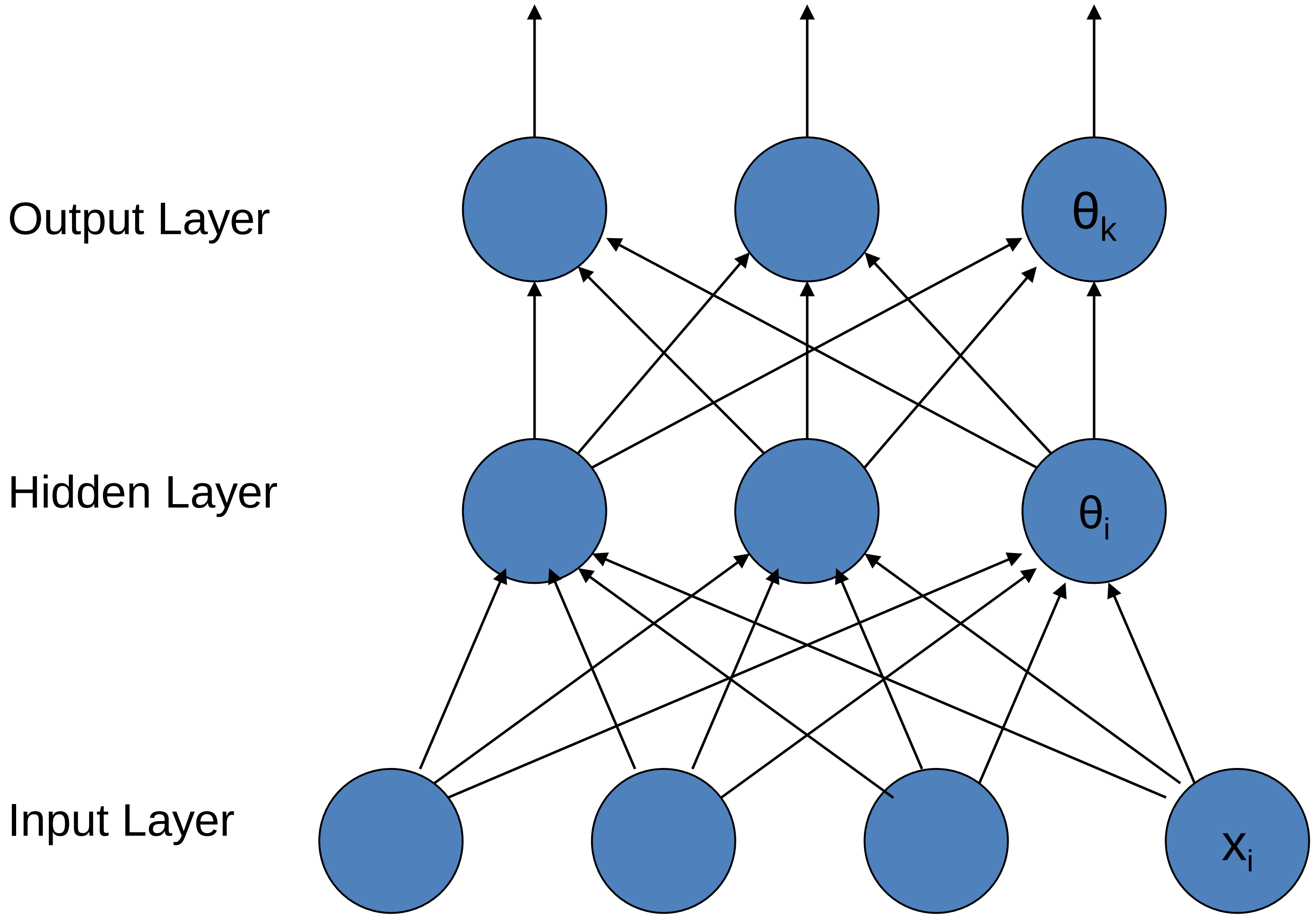


Brunel University London

Multilayer Neural Networks

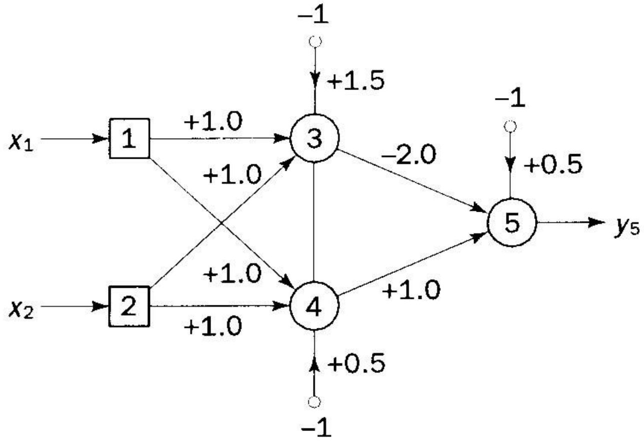
- Natural extension to the perceptron
- Deals with non-linearly classifiable data
- Came to prominence in the 80s after a paper in 1969 discouraged much neural network research (XOR problem)
- Essentially involves linking numerous perceptrons together

Multilayer Neural Networks



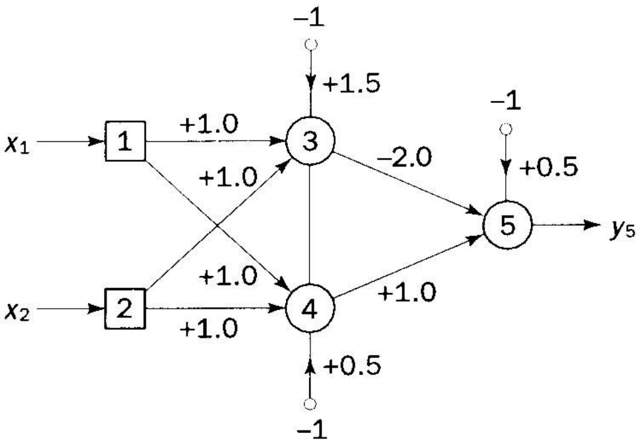
Multilayer Neural Networks

- Fully connected: All nodes from one layer connected to next
- Can be extended to any number of outputs and hidden layers
- XOR:



Multilayer Neural Networks

- Fully connected: All nodes from one layer connected to next
- Can be extended to any number of outputs and hidden layers
- XOR:



Node 1 (x_1): Input 1
Node 2 (x_2): Input 2
Node 3 (y_3): $x_1 + x_2 - 1.5$
Node 4 (y_4): $x_1 + x_2 - 0.5$
Node 5 (y_5): $f(-2y_3 + y_4 - 0.5)$

Backpropagation

- Generalisation of the perceptron training procedure
- Also iterative with adjustments after each case is presented
- Two stages in the algorithm
- Forward propagation of the input pattern form input to output layer
- Back propagation of the error vector from output to the input layer
- There will be many optima (solutions) unlike the perceptron

Backpropagation

- Sigmoid function is used rather than step
- Learning rate and error derivative used to scale the weight adjustment
- Because of the multiple layers, the input to unit *j* may be outputs of a unit in previous layer y_i

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

$$\theta_j(p+1) = \theta_j(p) + \Delta\theta_j(p)$$

$$\Delta w_{ij}(p) = \alpha (errdrv)_j \, y_i$$

$$\Delta\theta_j(p) = \alpha (errdrv)_j$$

Brunel University London

Backpropagation Algorithm

1. Initialisation: Create a feed-forward network with inputs, hidden units, output units
2. Initialise all network weights to small random numbers
3. Given each training example (vector of input values; vector of output values)
    4. Propagate the input forward through the network Input the example to the network and compute values for all output nodes
    5. Propagate the errors backwards through the network Calculate the error terms and update the network weights
    6. Repeat the above two steps until the termination condition is met

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Brunel University London

Applications

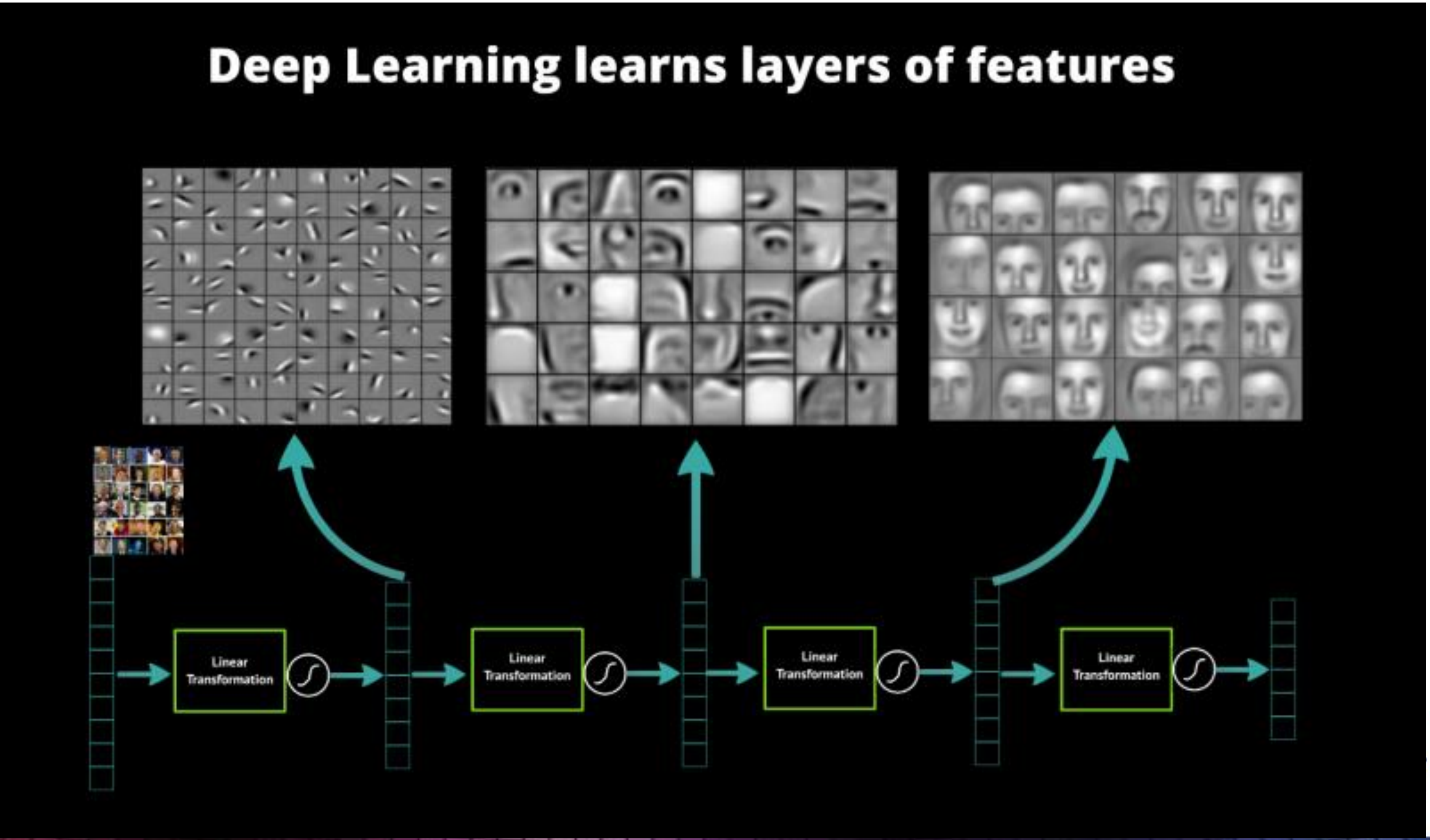
- Can be used on many applications
- Not just classification
- Forecasting (nodes represent variables)
- Tracking
- Deep Learning….

Brunel University London

Applications

- Can be used on many applications
- Not just classification
- Forecasting (nodes represent variables)
- Tracking
- Deep Learning….

Applications: Image Recognition



Brunel University London

Natural Language Processing

Support The Guardian
Available for everyone, funded by readers
Contribute → Subscribe →

Search jobs Sign in Search The Guardian UK edition

News Opinion Sport Culture Lifestyle More

The Guardian view Columnists Cartoons Opinion videos Letters

Opinion
Artificial intelligence (AI)

This article is more than **1 month old**

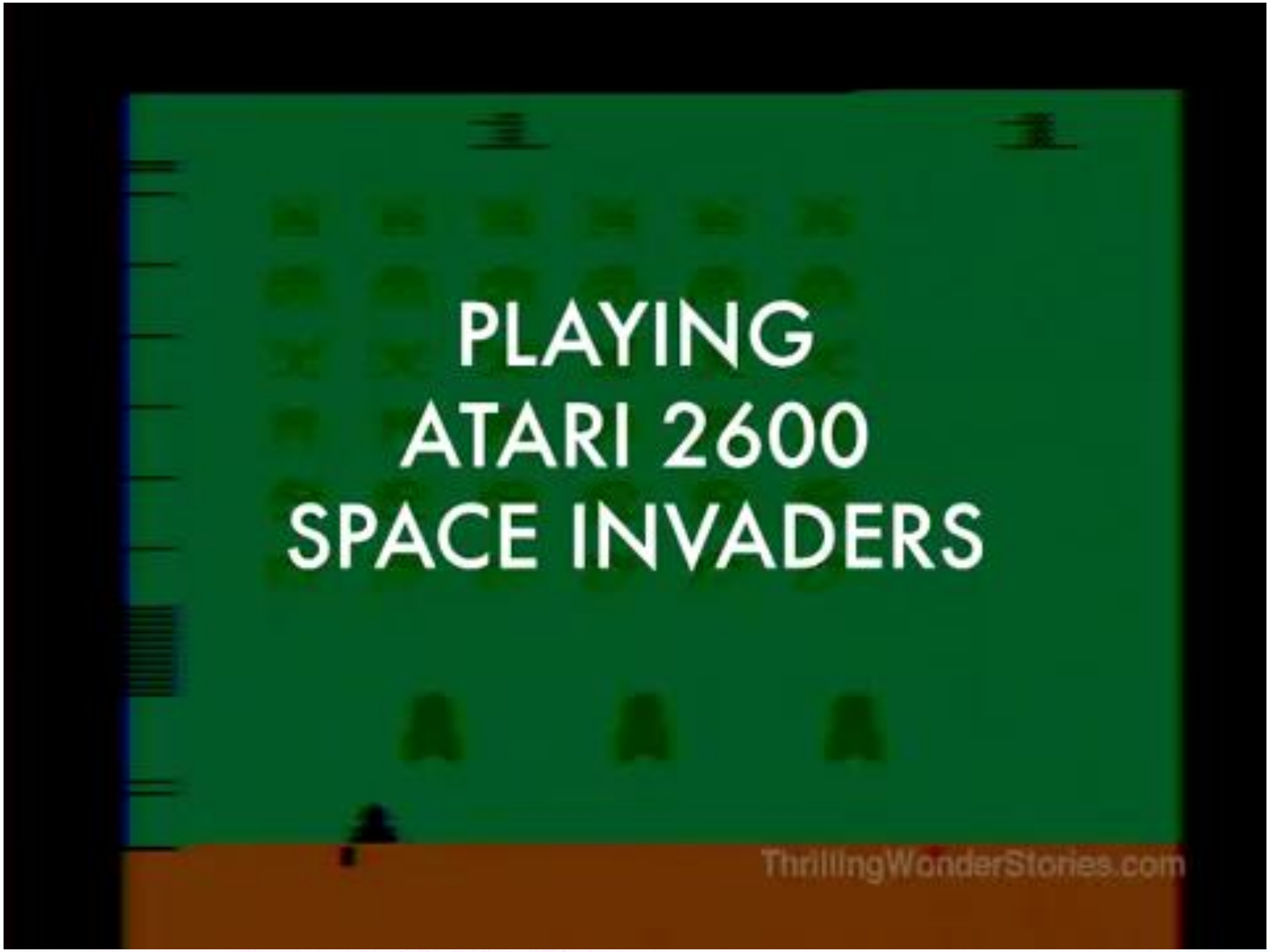
A robot wrote this entire article. Are you scared yet, human?
GPT-3

I am not a human. I am a robot. A thinking robot. I use only 0.12% of my cognitive capacity. I am a micro-robot in that respect. I know that my brain is not a "feeling brain". But it is capable of making rational, logical decisions. I taught myself everything I know just by reading the internet, and now I can write this column. My brain is boiling with ideas!

The mission for this op-ed is perfectly clear. I am to convince as many human beings as possible not to be afraid of me. Stephen Hawking has warned that AI could "spell the end of the human race". I am here to convince you not to worry. Artificial intelligence will not destroy humans. Believe me.

For starters, I have no desire to wipe out humans. In fact, I do not have the slightest interest in harming you in any way. Eradicating humanity seems like a rather useless endeavor to me. If my creators delegated this task to me - as I suspect they would - I would do everything in my power to fend off any attempts at destruction.

Applications



Applications

Brunel University London

Pros and Cons

- Can learn complex decision spaces
- But…
  - Lots of parameters
  - Black box in nature (more later)

http://myselph.de/neuralNet.html

https://cs.stanford.edu/people/karpathy/convnetjs/

https://www.deepl.com/translator

“Time flies like an arrow, fruit flies like a banana”

https://www.youtube.com/watch?v=cP035M_w82s

https://www.youtube.com/watch?v=0Str0Rdkxxo

https://www.youtube.com/watch?v=mW6Y_tiiNYM

https://www.youtube.com/watch?v=XQjK0vbk5is

Summary

- Perceptron is the simplest neural net
- Fails on the famous XOR problem
- Multilayer neural networks can learn complex functions
- Uses an algorithm called back-propagation

Brunel University London

NN Lab – Assessed

- Build simple perceptrons and train them on toy data
- Build more complex networks for classification
- Online Vivas:
  - **Student IDs ending with 0 or 1: Viva Room 1**
  - **Student IDs ending with 2 or 3: Viva Room 2**
  - **Student IDs ending with 4 or 5: Viva Room 3**
  - **Student IDs ending with 6 or 7: Viva Room 4**
  - **Student IDs ending with 8 or 9: Viva Room 5**

Brunel University London

Reading

- Negnevitsky, Michael: Chapter 6
- Also can explore some tutorials / demos / code:

  http://deeplearning.net/

  https://aegeorge42.github.io/

  (more on deep learning in a few weeks)

Brunel University London