5-day Hands-on Workshop on:

Python for Scientific Computing

and

TensorFlow for Artificial Intelligence

By Dr Stephen Lynch FIMA SFHEA

Holder of Two Patents

Author of PYTHON, MATLAB®, MAPLE™ AND MATHEMATICA® BOOKS

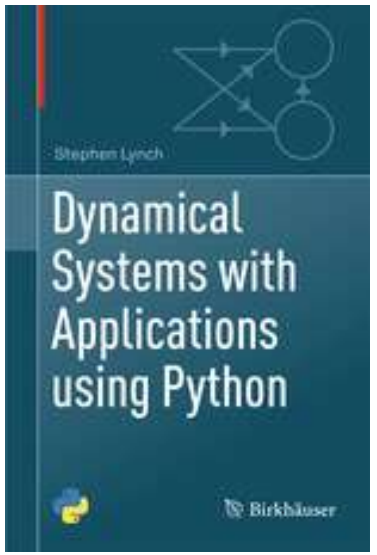
STEM Ambassador and Speaker for Schools

s.lynch@mmu.ac.uk

https://www2.mmu.ac.uk/scmdt/staff/profile/index.php?id=2443

Schedule (Day 4): Start Session 1

Day 4			
Topics	**Hours**	**Topics**	**Hours**
AI: Introduction to Image Processing	10am-11am	AI: Artificial Intelligence	1pm-2pm
AI: Binary Oscillator Computing	11am-12pm	AI: The Backpropagation Algorithm	2pm-3pm



Chapter 18: Image Processing

scikit-image
image processing in python

https://scikit-image.org

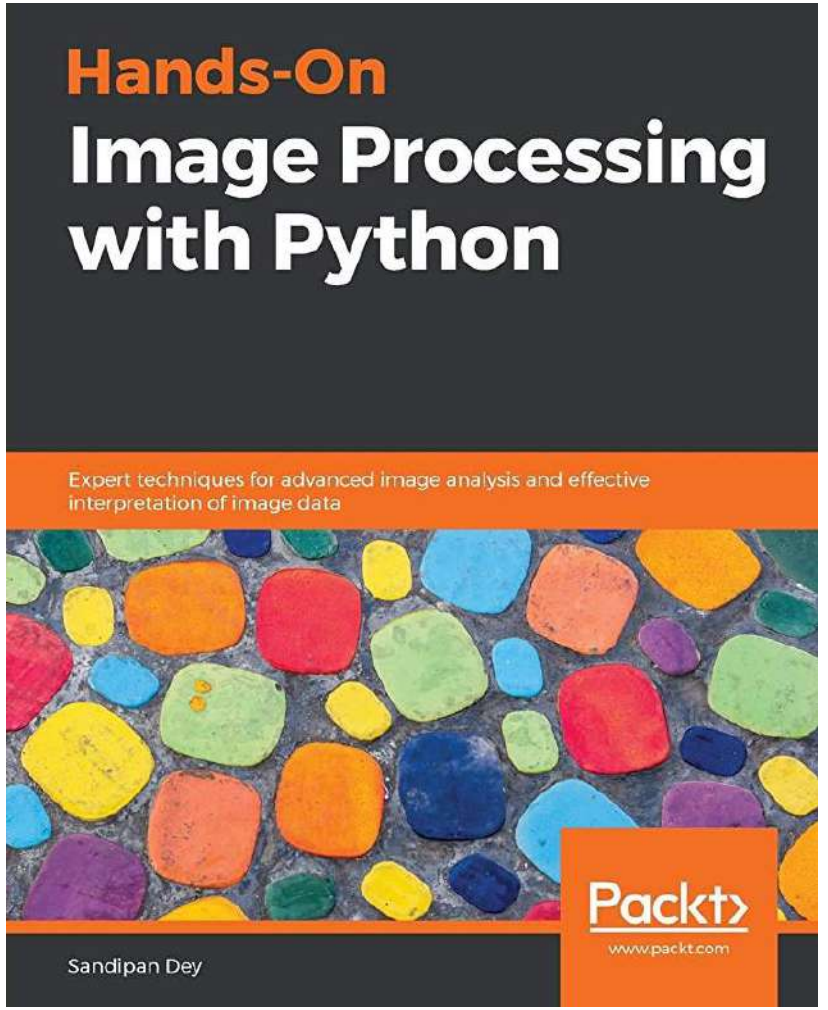


Image Processing in Python: scikit-image

Installation Gallery Documentation Community Source

Search documentation ...

Docs for 0.19.0
All versions

User Guide

- Getting started
- A crash course on NumPy for images
  - NumPy indexing
  - Color images
  - Coordinate conventions
  - Notes on the order of array dimensions
  - A note on the time dimension
- Image data types and what they mean
  - Input types
  - Output types
  - Working with OpenCV
  - Image processing pipeline
  - Rescaling intensity values
  - Note about negative values
  - References
- I/O Plugin Infrastructure
- Handling Video Files
  - A Workaround: Convert the Video to an Image Sequence
  - PyAV
  - Adding Random Access to PyAV
  - MoviePy
  - Imageio
  - OpenCV
- Data visualization
  - Matplotlib
  - Plotly
  - Mayavi
  - Napari
- Image adjustment: transforming image content
  - Color manipulation
  - Contrast and exposure
- Geometrical transformations of images
  - Cropping, resizing and rescaling images
  - Projective transforms (homographies)
- Tutorials
  - Image Segmentation
  - How to parallelize loops
- Getting help on using `skimage`
  - Examples gallery
  - Search field
  - API Discovery
  - Docstrings
  - Mailing-list
- Image Viewer
  - Quick Start

Image Processing: Colour Image

```
Console 4/A

Python 3.9.7 (default, Sep 16 2021, 08:50:36)
Type "copyright", "credits" or "license" for more
information.

IPython 7.29.0 -- An enhanced Interactive Python.

In [1]: from skimage import data, io

In [2]: retina = data.retina()

In [3]: io.imshow(retina)
Out[3]: <matplotlib.image.AxesImage at 0x12ec9a6d0>

In [4]: retina.shape
Out[4]: (1411, 1411, 3)

In [5]: retina.dtype
Out[5]: dtype('uint8')

In [6]: retina[900 , 900]
Out[6]: array([216,  86,  60], dtype=uint8)

IPython console  History
```

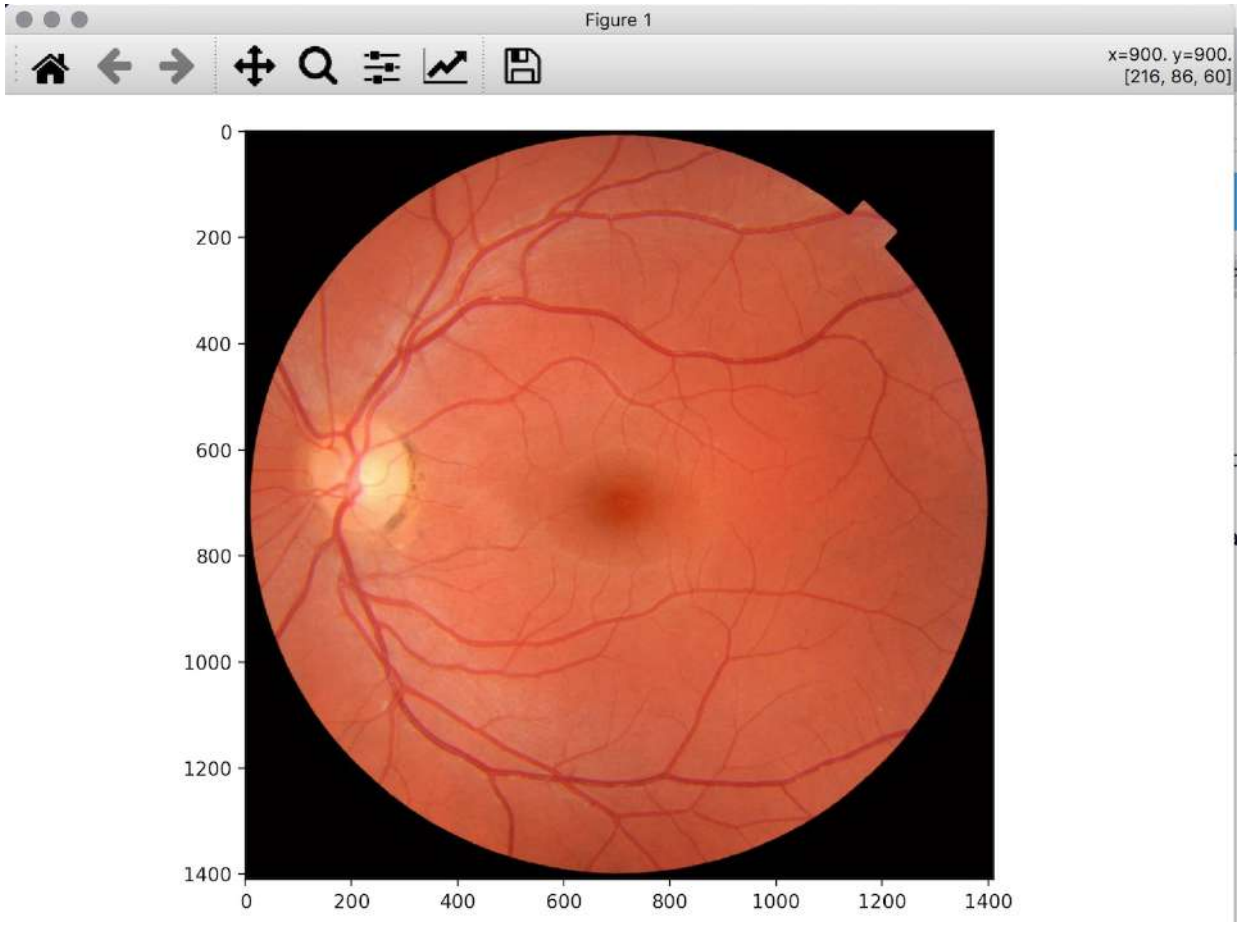


Image Processing: Vascular Architecture using Ridge Filters

```
# New Book: Vascular architecture tracing using ridge filters.
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from skimage import color, data, filters
from skimage import morphology
retina_source = data.retina()
_, ax = plt.subplots()
ax.imshow(retina_source)
retina = color.rgb2gray(retina_source)
t0, t1 = filters.threshold_multiotsu(retina, classes=3)
mask = (retina > t0)
vessels = filters.sato(retina, sigmas=range(1, 10)) * mask
img_gray = rgb2gray(vessels)
t = 0.015
binary_mask = img_gray > t
fig, ax = plt.subplots()
binary_mask = morphology.remove_small_objects(binary_mask, 600)
plt.imshow(binary_mask, cmap="gray")
plt.show()
```

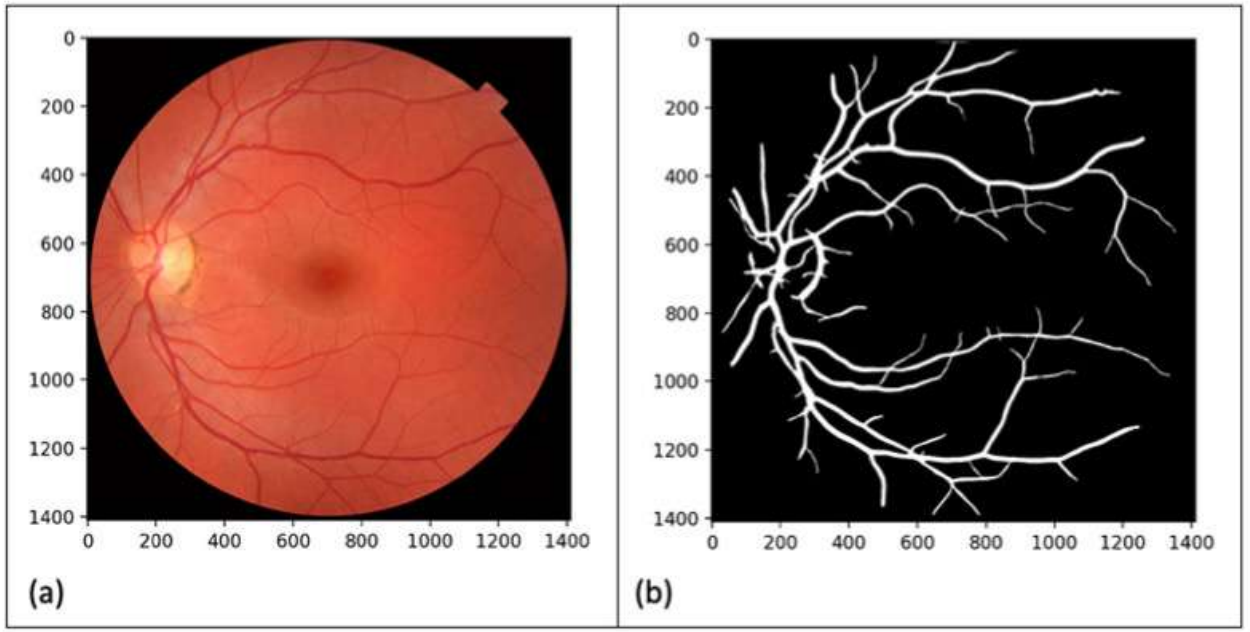
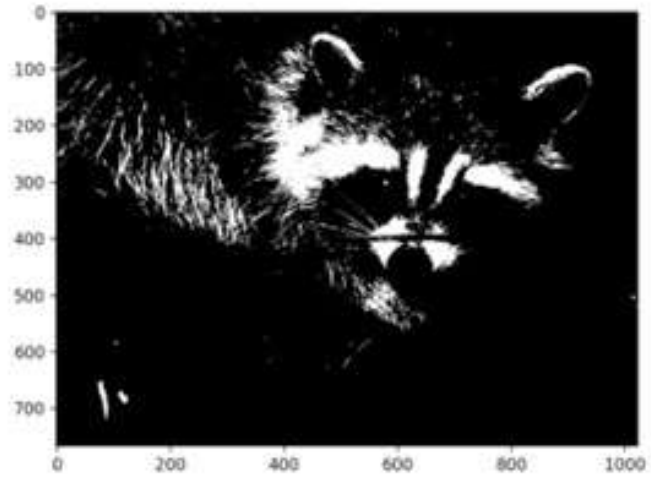


Image Processing: Binarize a Colour Image

```
# Programs_18b: Color picture of a raccoon.
# See Figure 18.2.

from scipy import misc
import matplotlib.pyplot as plt
import numpy as np
face = misc.face()

fig1 = plt.figure()
plt.imshow(face)
width, height, _ = face.shape

face[100, 100]                          # RGB values and data type.
print('RGB value=',face[100, 100])  # RGB values of pixel.
print('Image dimensions: {}x{}'.format(width, height))
WhitePixels = np.zeros((width, height))

def white_pixel(pixel, threshold):
    return 1 if all(value > threshold for value in pixel) else 0
for i, row in enumerate(face):
    for j, pixel in enumerate(row):
        WhitePixels[i, j] = white_pixel(pixel, threshold=180)
fig2 = plt.figure()
plt.imshow(WhitePixels,cmap='gray')
print('There are {:,} white pixels'.format(int(np.sum(WhitePixels))))
plt.show()
```



Problem: How many pixels are green?

Loading Files into Google Colab from Google Drive

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.
Enter your authorization code:
```

Google

Sign in

Please copy this code, switch to your application and paste it there:

```
4/1AY0e-g79sI_Bzwfd6FdPuUxQAt-
j780TkL_X9JPJmJCmVNWgkFBkX3PZ2LU
```

Google Drive for desktop wants to access your Google Account

lynch.drs@googlemail.com

Make sure that you trust Google Drive for desktop

You may be sharing sensitive info with this site or app. Find out how Google Drive for desktop will handle your data by reviewing its **terms of service** and **privacy policies**. You can always see or remove access in your **Google Account**.

Find out about the risks

Cancel Allow

Loading Files into Google Colab from Google Drive

```
Enter your authorization code:
4/1AY0e-g79sl_Bzwfd6FdP
```

```
[2] from google.colab import drive
    drive.mount('/content/gdrive')
```

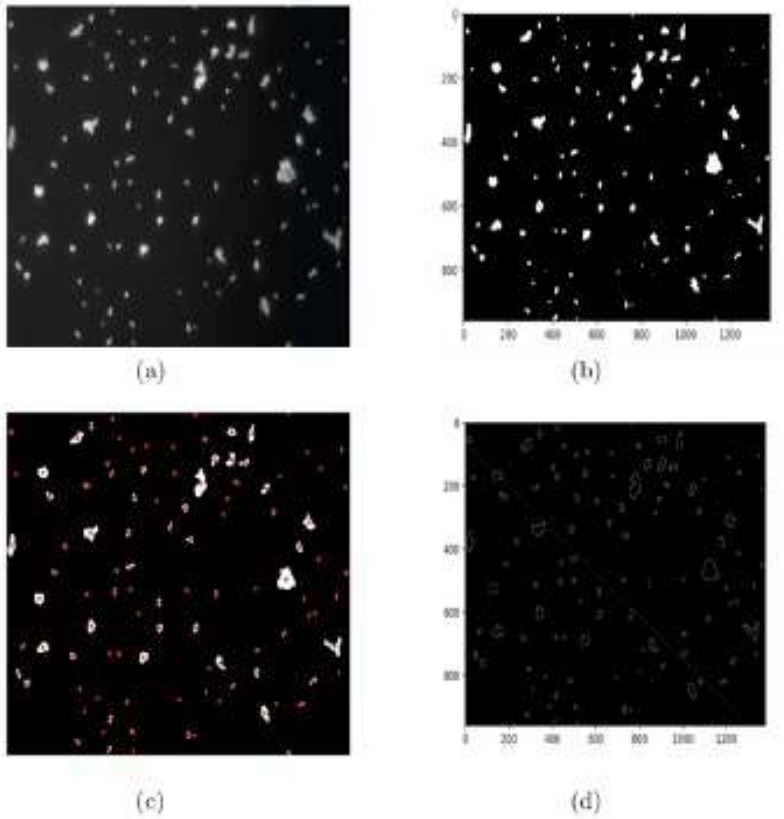
```
Mounted at /content/gdrive
```

```
!ls '/content/gdrive/MyDrive/'
```

```
'Colab Notebooks'   housing.txt   'Personal Photos'
 Figure_1.png       Microbes.png
```

Image Processing in Google Colab: Microbes Data

```
# Program 18c: Statistical Analysis on Microbes.png.
# See Figures 18.3 and 18.4.

import matplotlib.pyplot as plt
from skimage import io
import numpy as np
from skimage.measure import regionprops
from scipy import ndimage
from skimage import feature

microbes_img = io.imread('/content/gdrive/MyDrive/Microbes.png')
fig1 = plt.figure()
plt.imshow(microbes_img,cmap='gray', interpolation='nearest')
width, height, _ = microbes_img.shape
binary = np.zeros((width, height))

for i, row in enumerate(microbes_img):
    for j, pixel in enumerate(row):
        if pixel[0] > 80:
            binary[i, j] = 1
```



(a) Microbes.png; (b) binarized;
(c) centroids; (d) edges.

Problem: Can you find the centroids of the clusters?

Image Processing in Google Colab: Microbes Data

```
blobs = np.where(binary>0.5, 1, 0)
labels, no_objects = ndimage.label(blobs)
props = regionprops(blobs)
print('There are {:,} clusters of cells:'.format(no_objects))

fig3 = plt.figure()
edges=feature.canny(binary,sigma=2,low_threshold=0.5)
plt.imshow(edges,cmap=plt.cm.gray)

fig4 = plt.figure()
labeled_areas = np.bincount(labels.ravel())[1:]
print(labeled_areas)
plt.hist(labeled_areas,bins=no_objects)
plt.xlabel('Area',fontsize=15)
plt.ylabel('Number of clusters',fontsize=15)
plt.tick_params(labelsize=15)
plt.show()
```

Histogram of data.

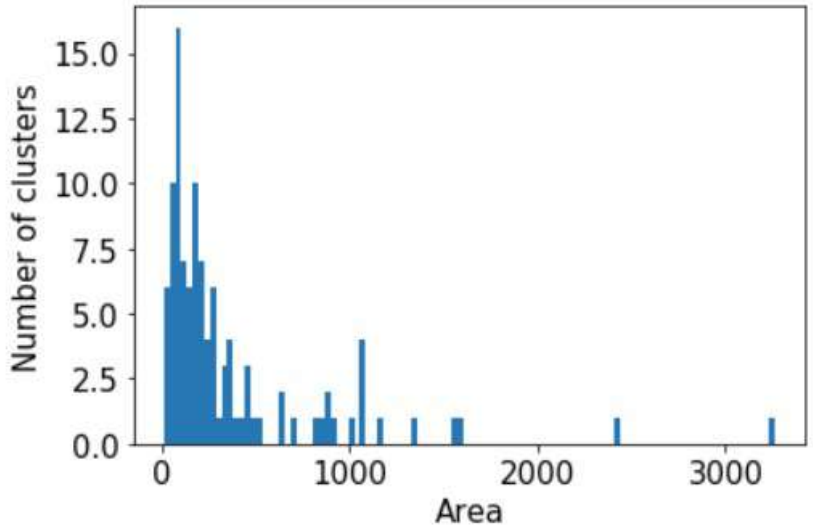
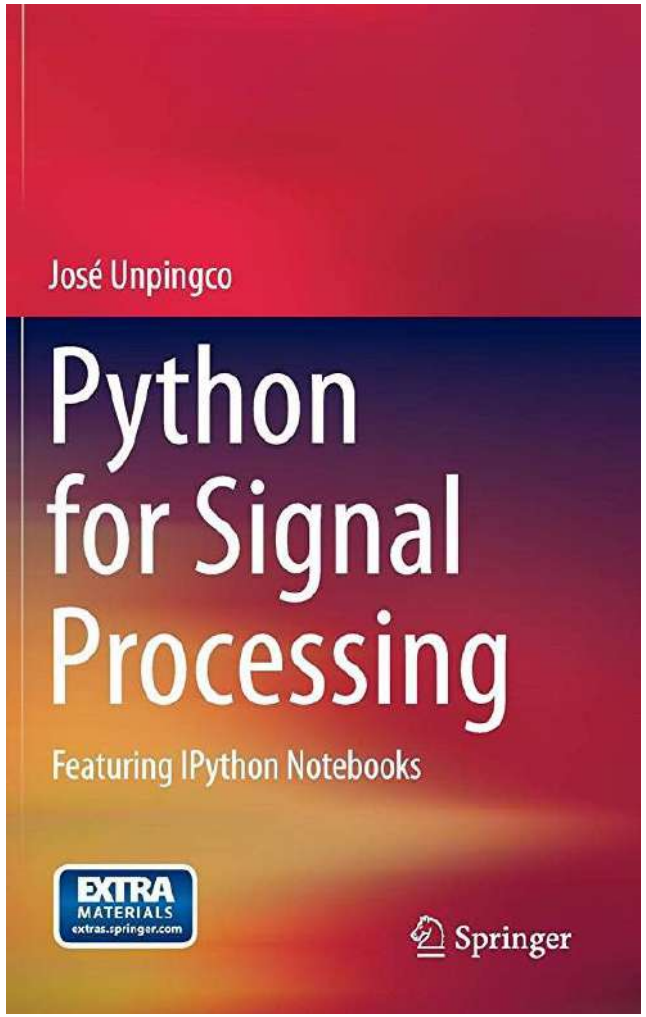
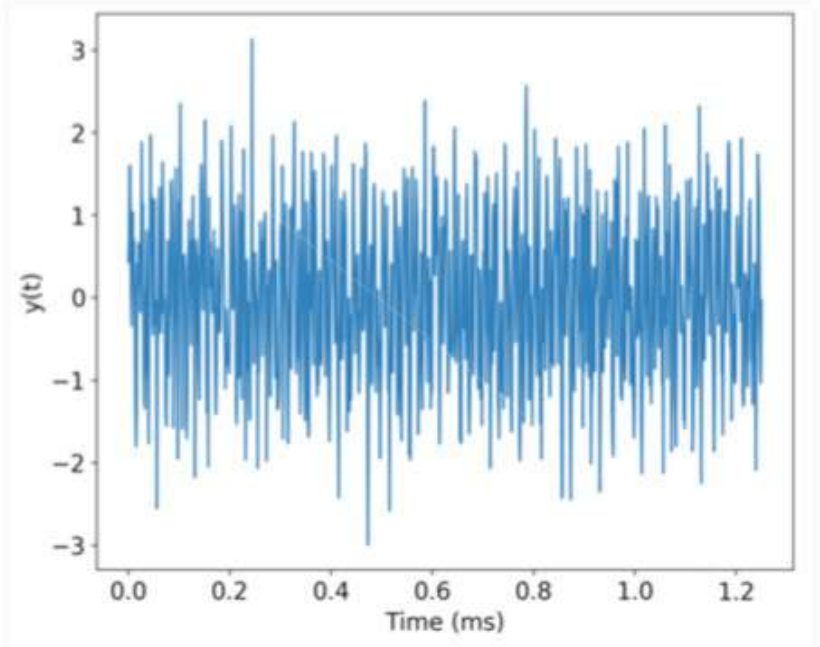


Image Processing: Signal Processing

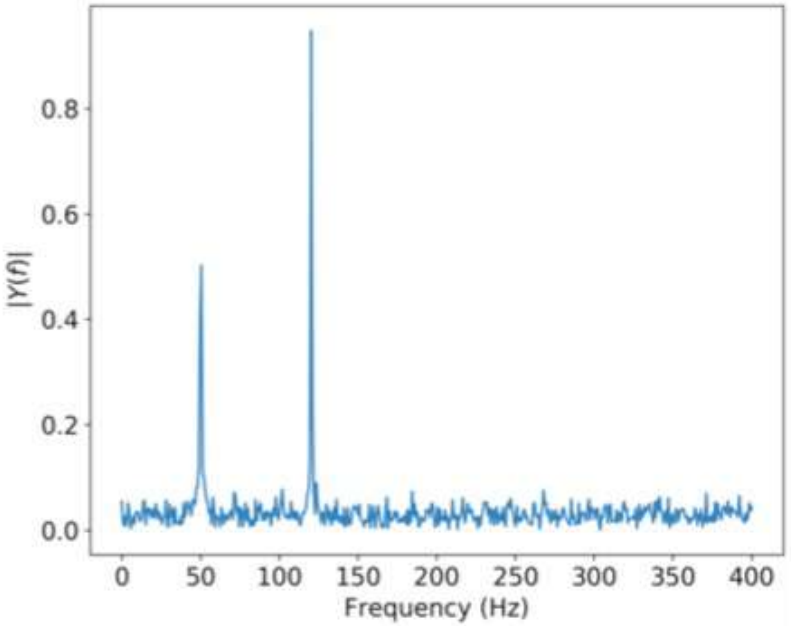



```python
# Program 18d: Fast Fourier transform of a noisy signal.
# See Figure 18.5.

import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

Ns = 1000  # Number of sampling points
Fs = 800 # Sampling frequency
T = 1/Fs  # Sample time
t = np.linspace(0, Ns*T, Ns)
amp1, amp2 = 0.7, 1
freq1, freq2 = 50, 120

# Sum a 50Hz and 120 Hz sinusoid
x = amp1 * np.sin(2*np.pi * freq1*t) + amp2*np.sin(2*np.pi * freq2*t)
y = x + 0.5*np.random.randn(Ns)
fig1 = plt.figure()
plt.plot(t, y)
plt.xlabel('Time (ms)', fontsize=15)
plt.ylabel('y(t)', fontsize=15)
plt.tick_params(labelsize=15)

fig2 = plt.figure()
yf = fft(y)
xf = np.linspace(0, 1/(2*T), Ns//2)
plt.plot(xf, 2/Ns * np.abs(yf[0:Ns//2]))
plt.xlabel('Frequency (Hz)', fontsize=15)
plt.ylabel('$|Y(f)|$', fontsize=15)
plt.tick_params(labelsize=15)
plt.show()
```

Image Processing: Fast Fourier Transform (FFT) of a Noisy Signal



(a)



(b)

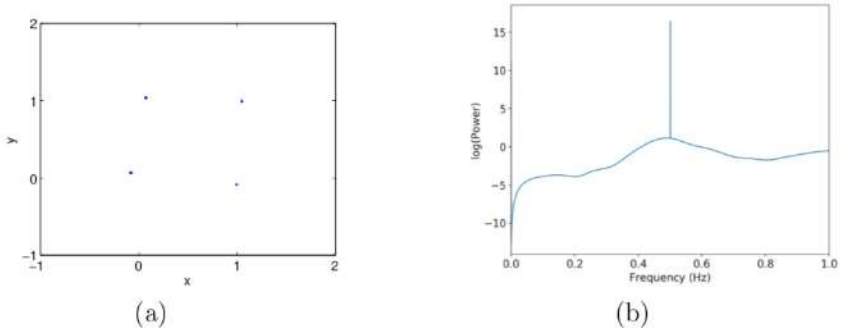
Figure 18.5: [Python] (a) Signal corrupted with zero-mean random noise. (b) The amplitude spectrum of y(t). You can read off the amplitude and frequencies.

Image Processing: FFT as a Chaos Detector

Example 5. Consider the 2-dimensional discrete map defined by

$$x_{n+1} = 1 + \beta x_n - \alpha y_n^2$$
$$y_{n+1} = x_n,$$

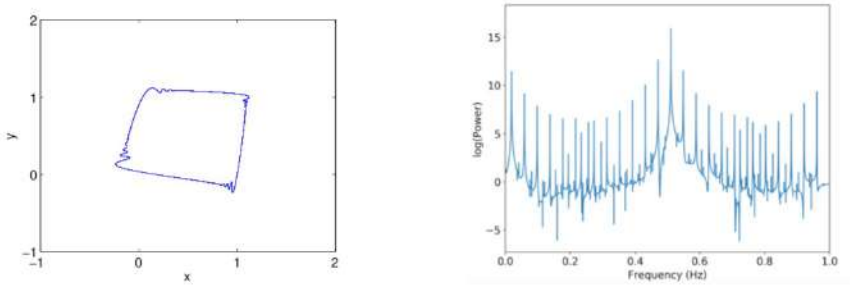
$\alpha = 1\,, \beta = 0.05$



(a) (b)

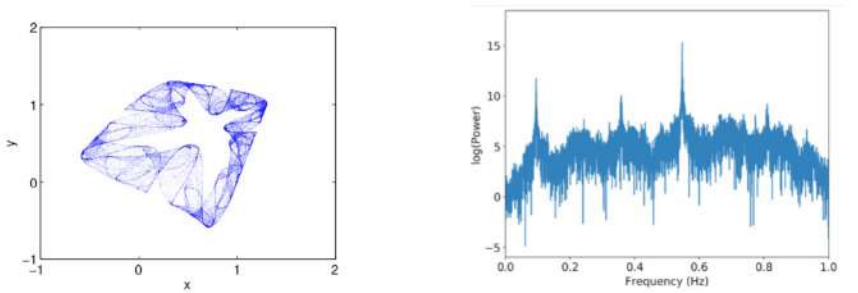
See Program 18e.

$\alpha = 1\,, \beta = 0.12$



(c) (d)

$\alpha = 1\,, \beta = 0.3$



(e) (f)

Image Processing: Edge Detection (Program 18g) End Session 1

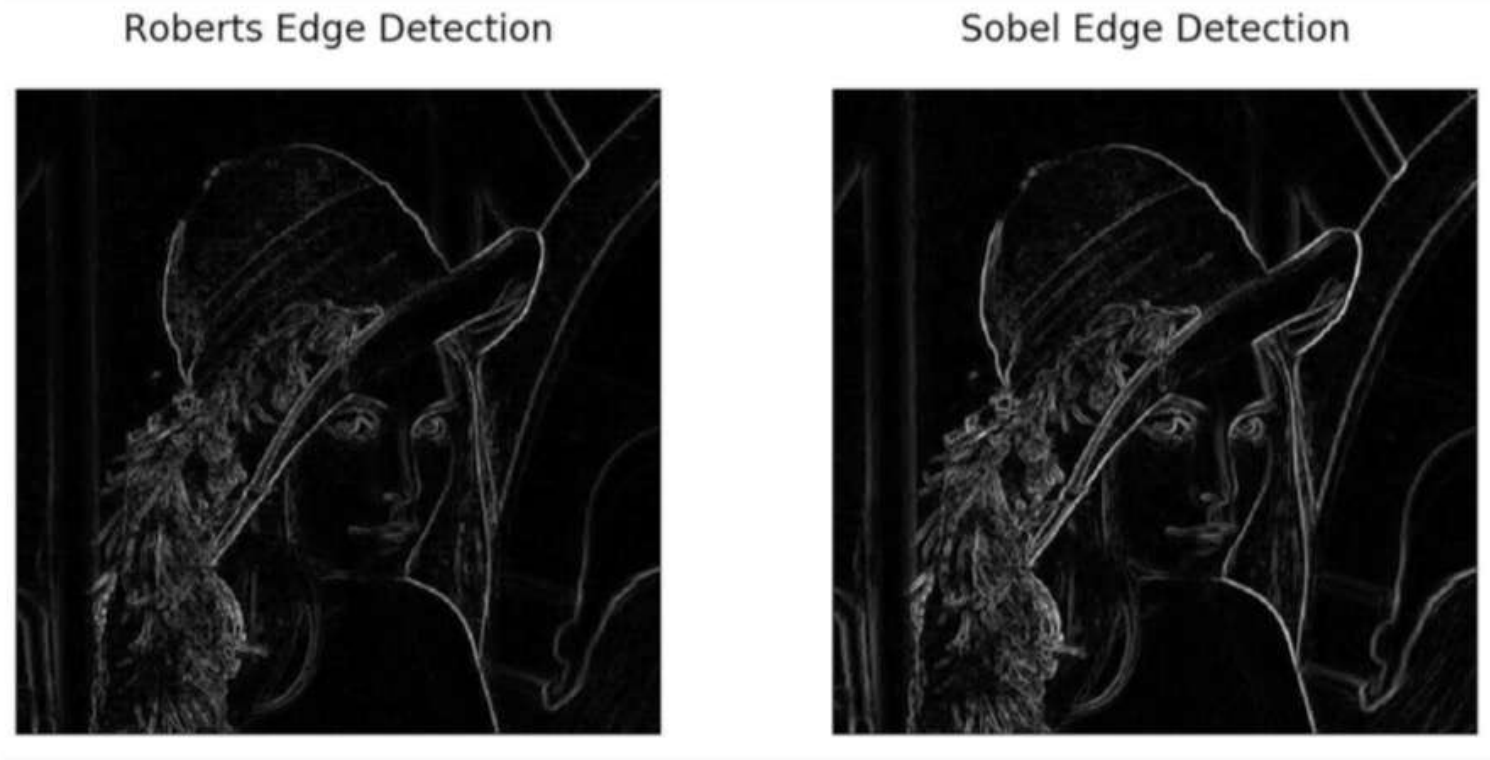
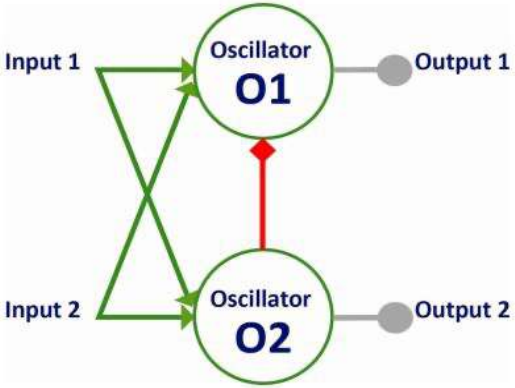


Figure 18.8: [Python] Edge detection in the Lena image using both Roberts and Sobel edge detection.

Binary Oscillator Computing

Dr Jon Borresen and Dr Stephen Lynch FIMA SFHEA

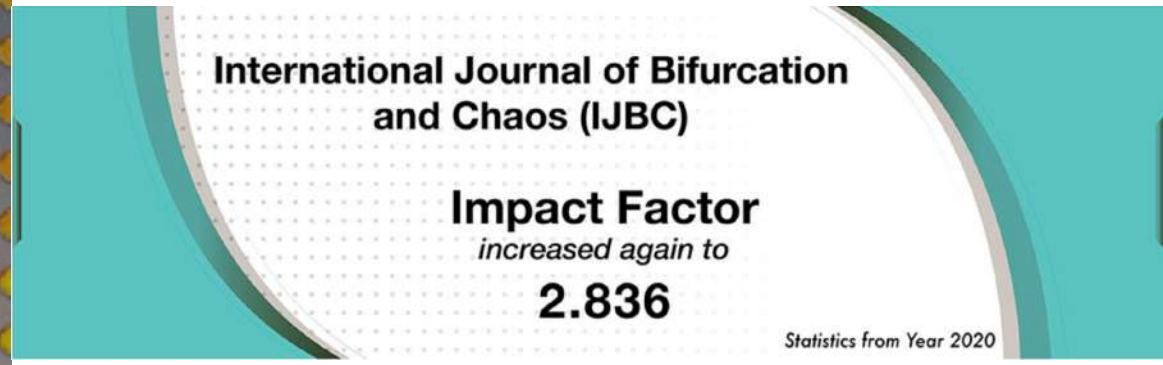


International patents.

AI and Binary Oscillator Computing

My co-inventor, Jon Borresen was my final year project student in 2001.

In 2002, we had a journal paper published.



International Journal of Bifurcation and Chaos | Vol. 12, No. 01, pp. 129-134 (2002) | Letters

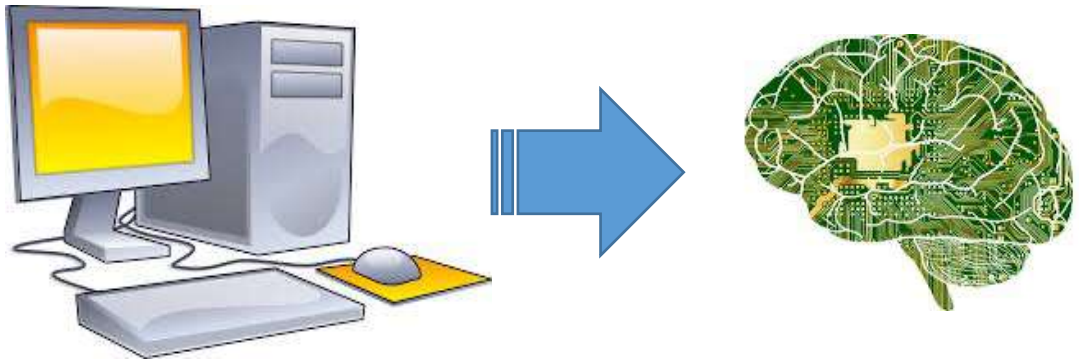
FURTHER INVESTIGATION OF HYSTERESIS IN CHUA'S CIRCUIT

J. BORRESEN and S. LYNCH

https://doi.org/10.1142/S021812740200422X | **Cited by:** 18

AI and Binary Oscillator Computing

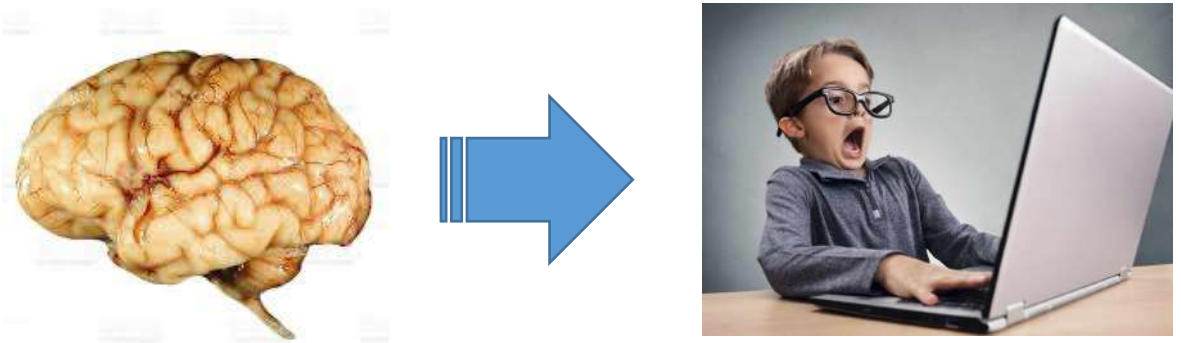
Artificial Intelligence



Artificial Intelligence, machine learning and deep learning.

Using computers to act like the human brain.

Brain Inspired Computing



Binary oscillator computing.

Using biological brain dynamics to create a powerful conventional supercomputer.

Outline

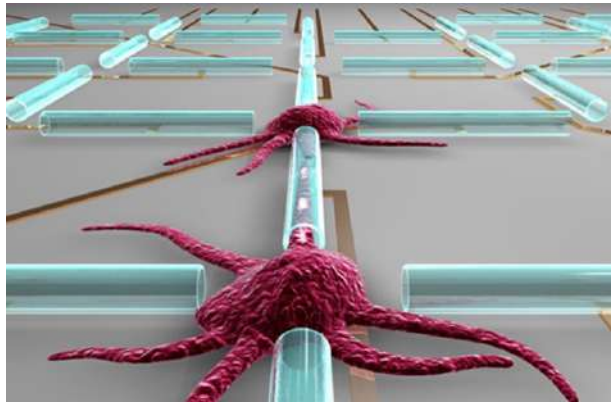
Neurons and the brain

Computing with threshold oscillators:

- Superconducting devices
- Biological neurons
- Transistor-based oscillators
- All-optical oscillators
- Memristor oscillators

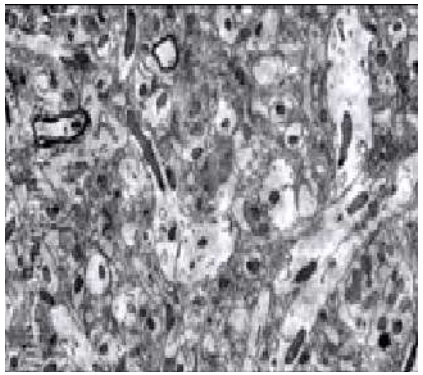
Possible applications:

- Exascale (or beyond) supercomputing
- An assay for neuronal degradation

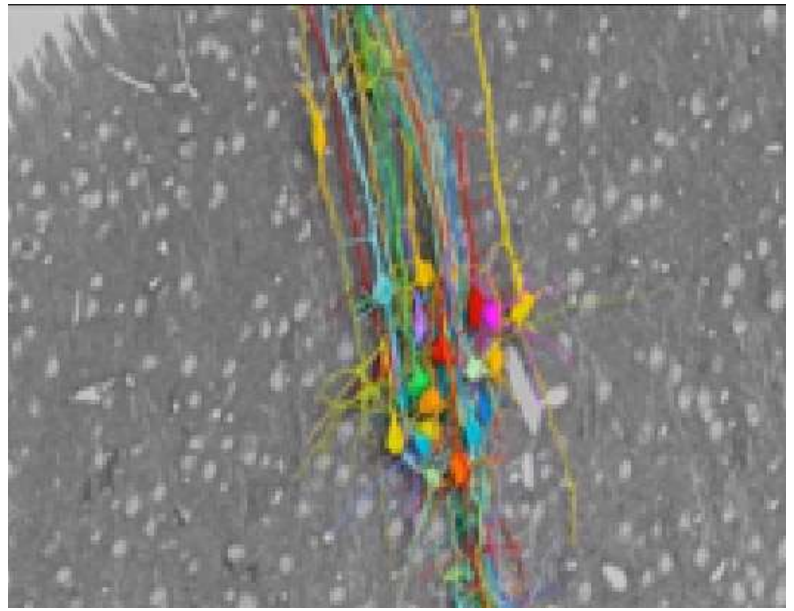


Neurons in the Mouse Brain

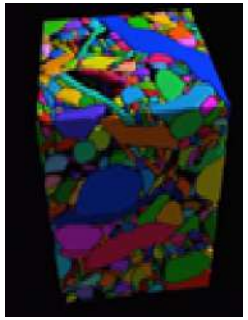
Slices of a mouse brain.



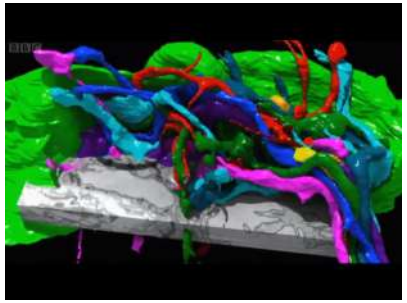
One slice of a mouse brain.



Mouse neurons.



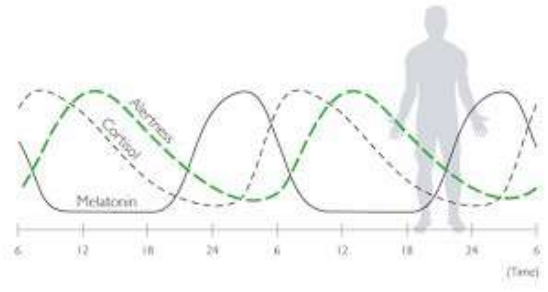
A cube of synaptic connections.



A few connections.

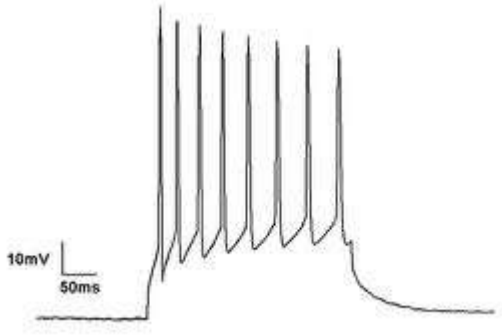
Using current technology it would take 4 million years to produce a map like this for one human brain!
Research conducted by Jeff Lichtman and his group (Harvard University, USA).

Oscillators in the Human Body



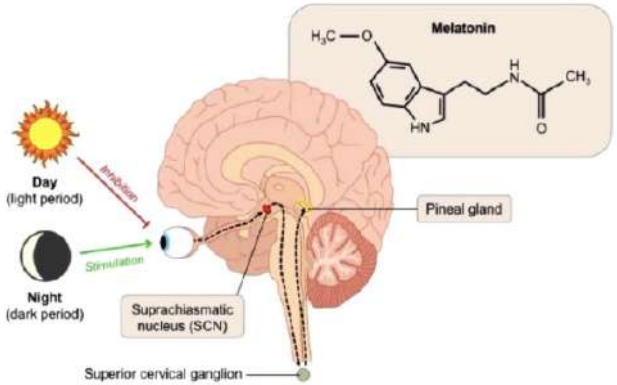
Circadian oscillations – wake/sleep.

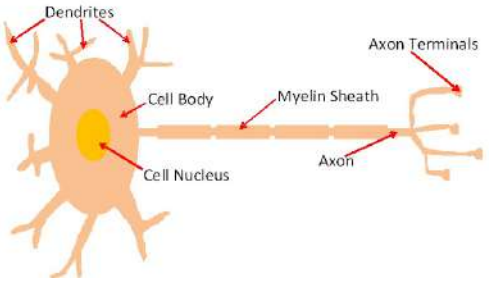
The human heart: an average 60-80 beats per minute.

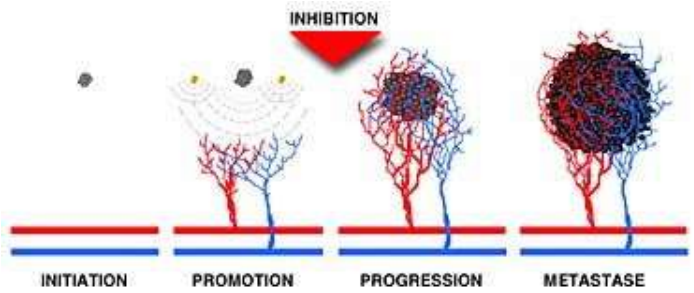


A neuron spike train: beats up to a thousand times faster than the heart.

Angiogenesis: new blood vessels form from pre-existing vessels. N is tumour size, P is quantity of growth factors, E is vessel density.

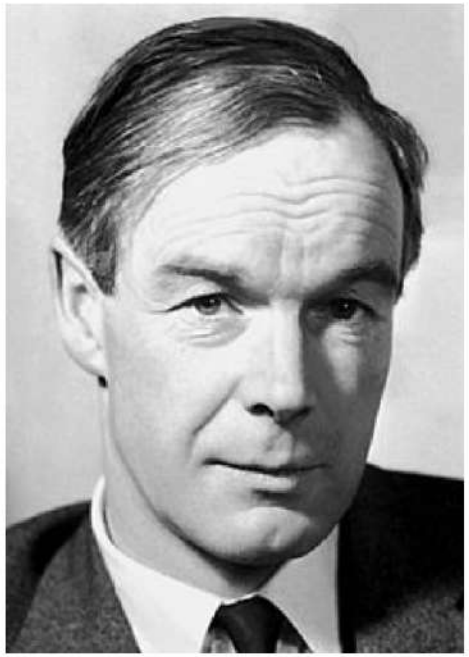




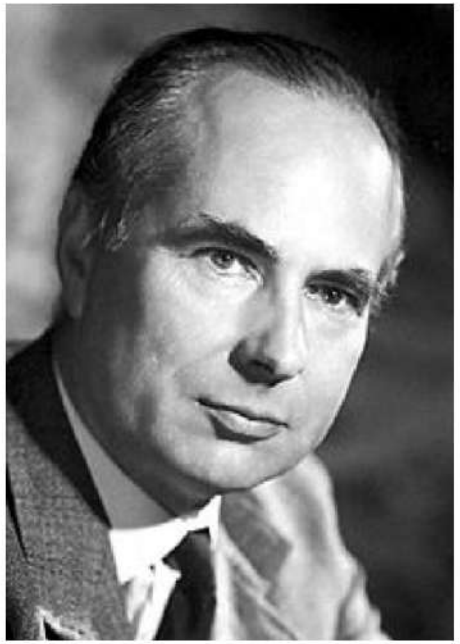


Mathematical Modelling of Neurons

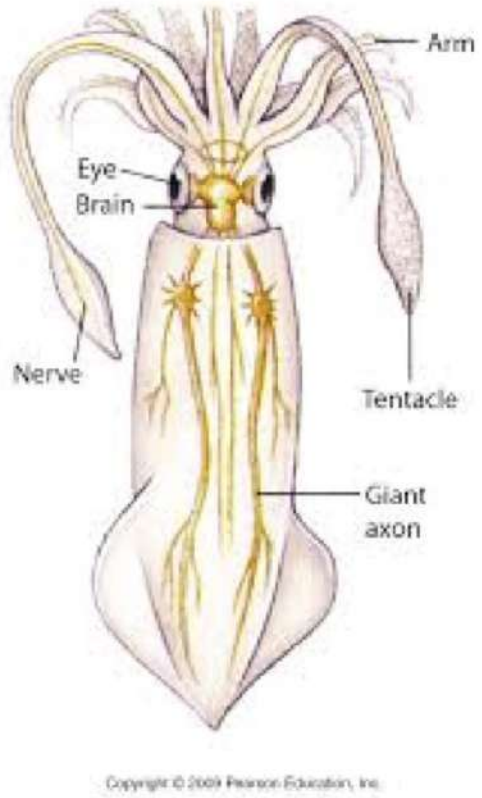
In 1952, Alan Lloyd Hodgkin and Andrew Huxley developed a mathematical model to describe how action potentials in neurons are initiated and propagated.



Sir Alan Lloyd Hodgkin

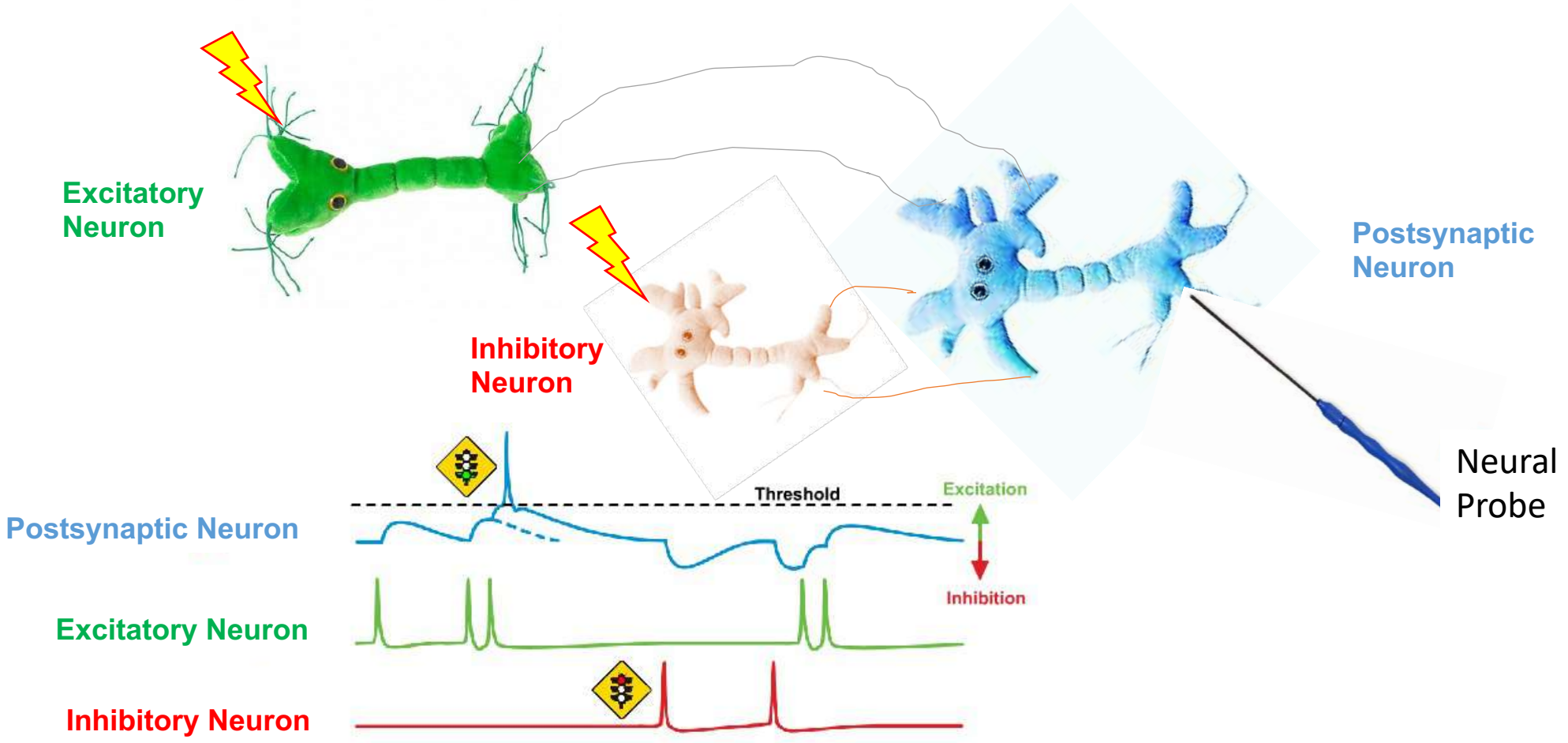


Sir Andrew Huxley



Hodgkin-Huxley studied the axon of a giant squid.

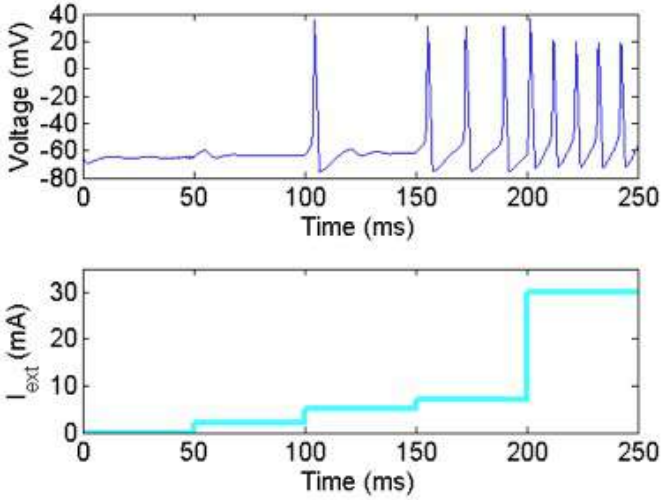
Neuron Model (Excitation and Inhibition)



Neuron Model (Mathematical Models: Program 21a)

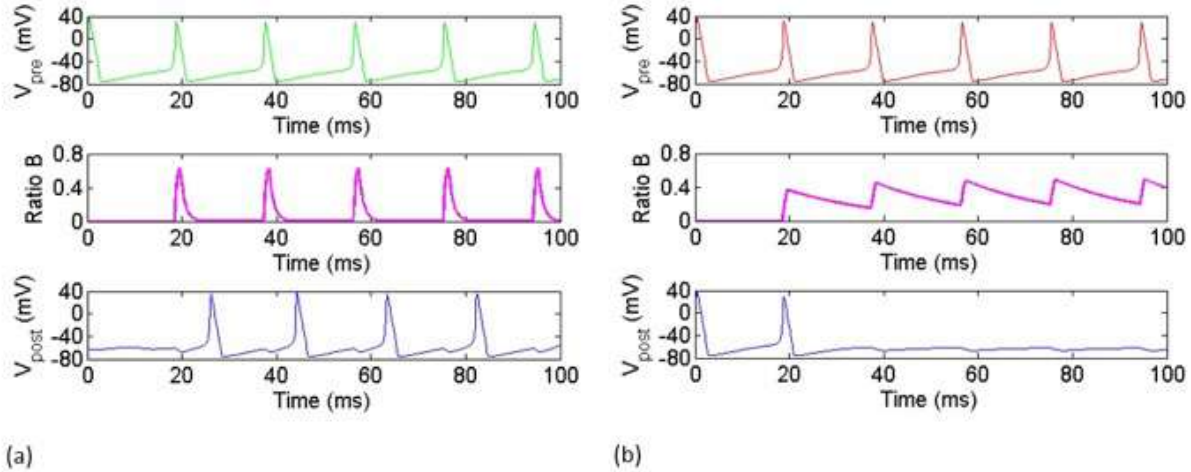
Mathematical Modelling (Hodgkin-Huxley)

Electric Input



Threshold oscillations. Lower curve input current, upper curve voltage in neuron.

Chemical Kinetics



Connecting neurons: [T], [R] and [B]=[RT], are concentrations of neurotransmitter, unbound receptor and bound receptor, respectively: (a) excitation and (b) inhibition.

Mathematical Modelling of Neurons

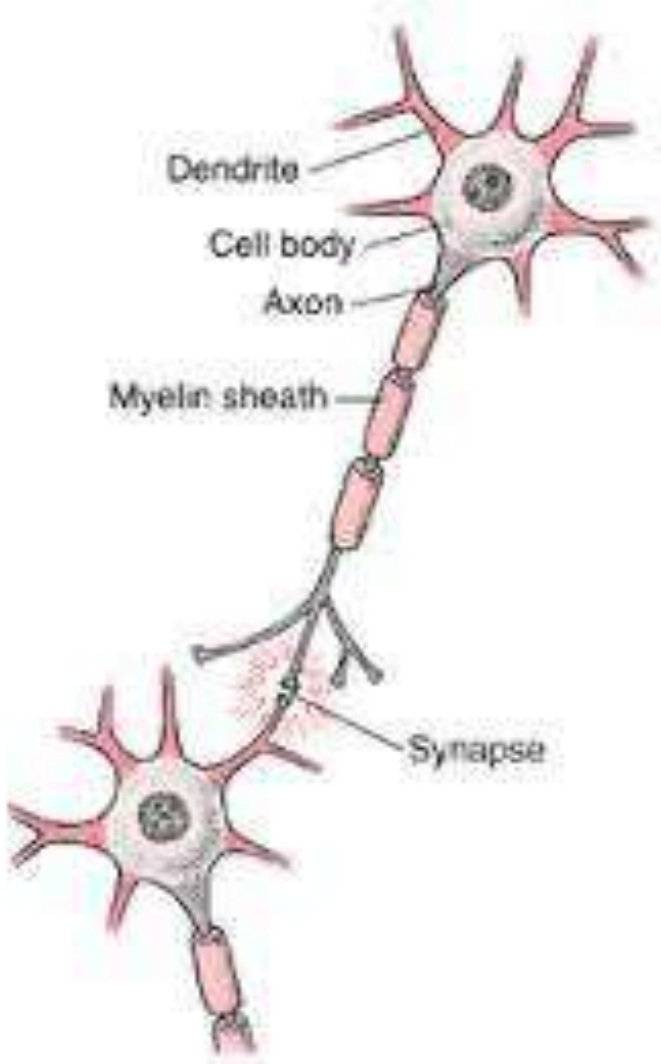
1952: The Hodgkin-Huxley Model
(Biophysically meaningful)

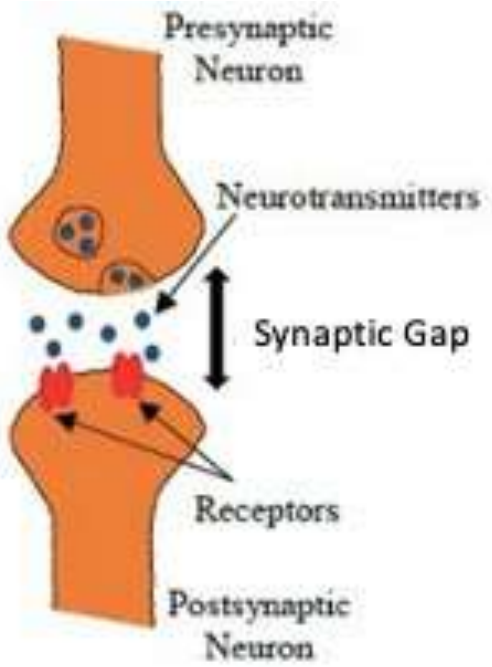
1961: Fitzhugh-Nagumo Models

1981: Morris-Lecar Model

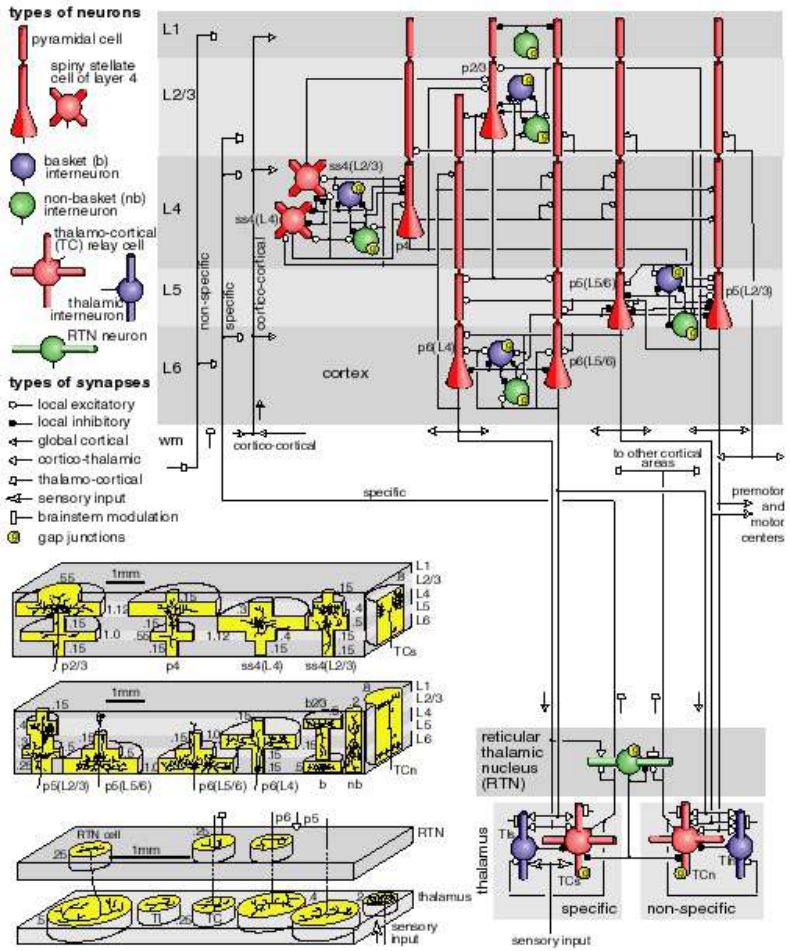
1984: Hindmarsh-Rose Model

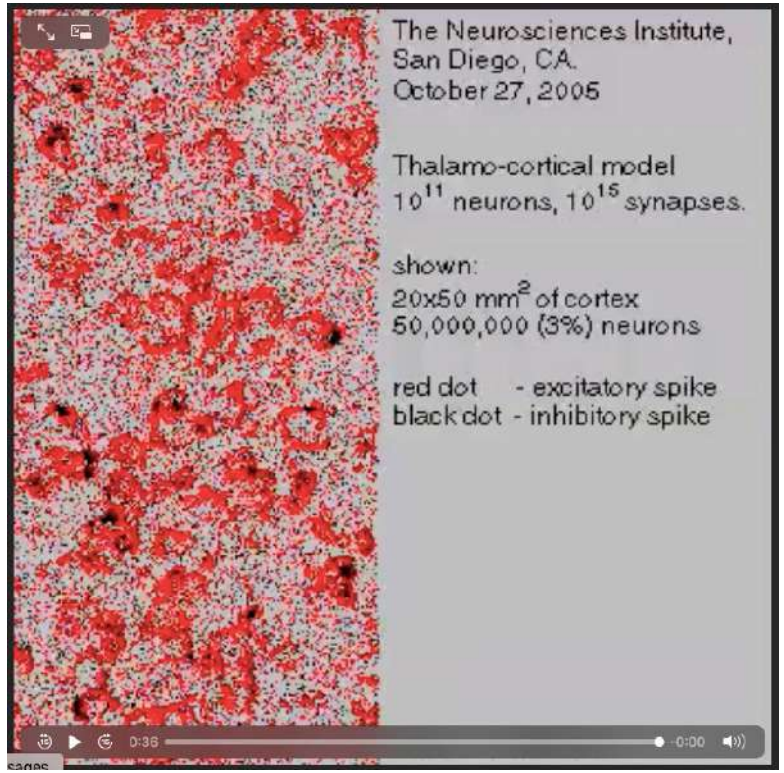
2005: Izhikevich Model
(Biophysically meaningless)





Izhikevich Mathematical Model of the Human Brain

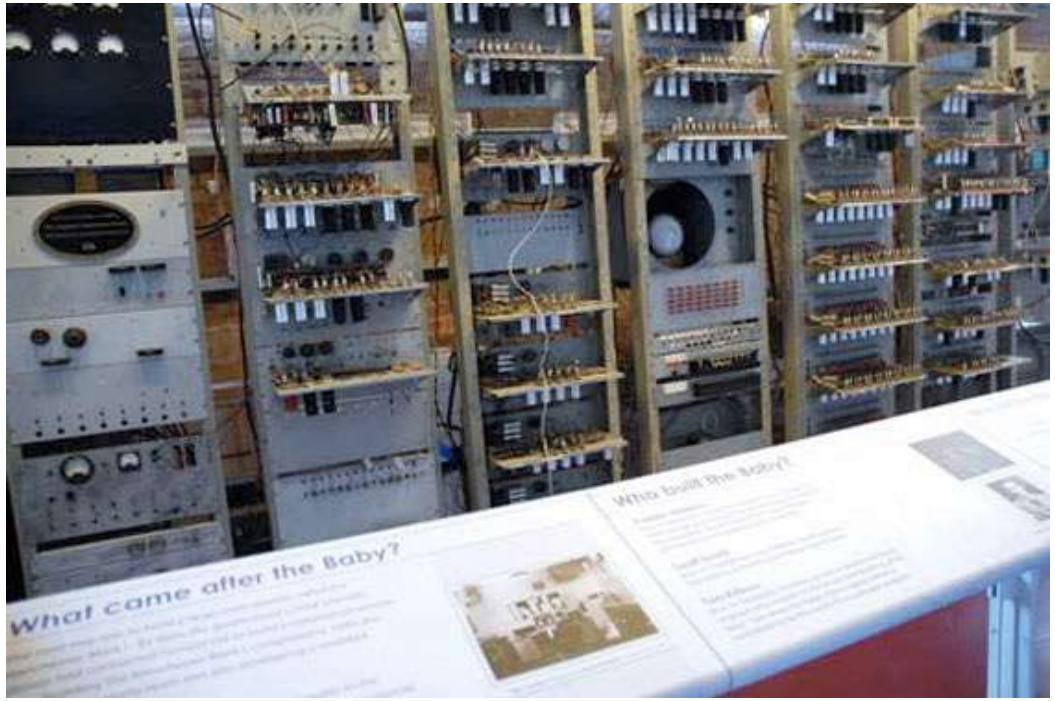




http://www.izhikevich.org/human_brain_simulation/Blue_Brain.htm

The Baby Computer

In 1948, the world's first program was run on Manchester University's small-scale experimental machine the "Baby". One of the principal components used was the vacuum tube oscillator. The Manchester Museum of Science and Industry (MOSI) built a working replica in 1998.

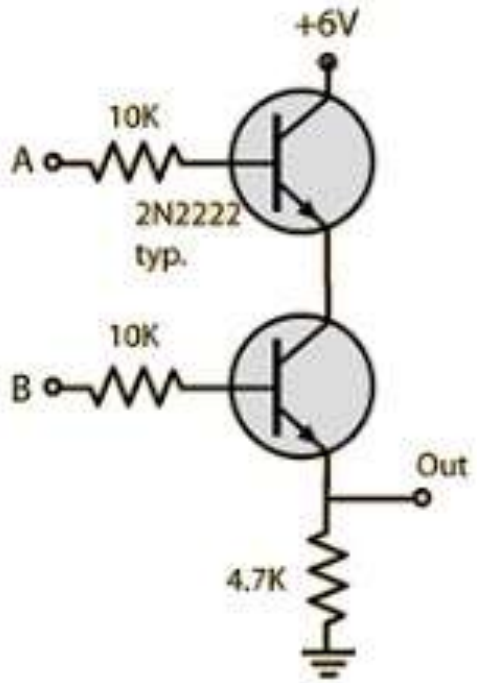


The Baby computer.

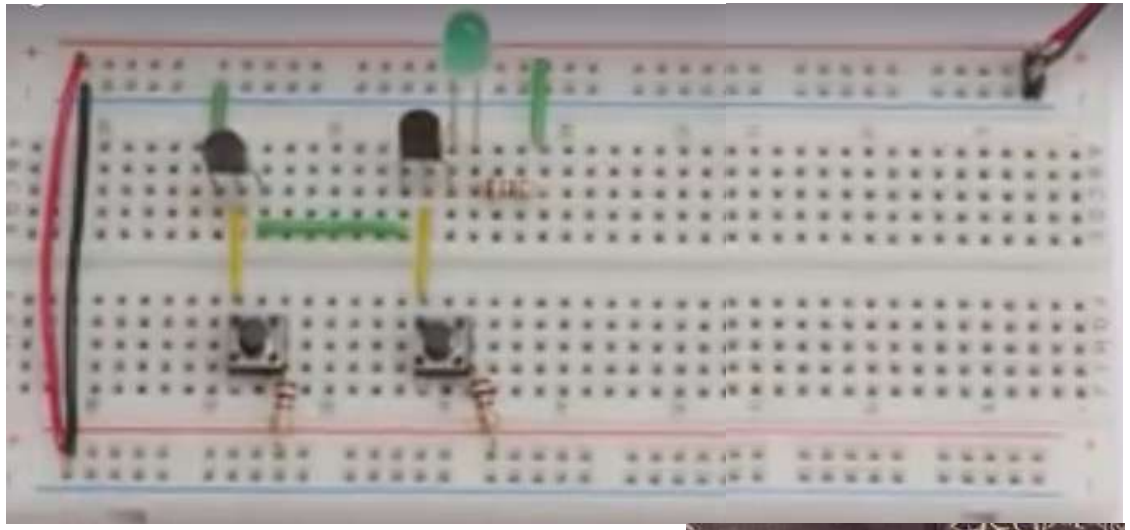
A vacuum tube.

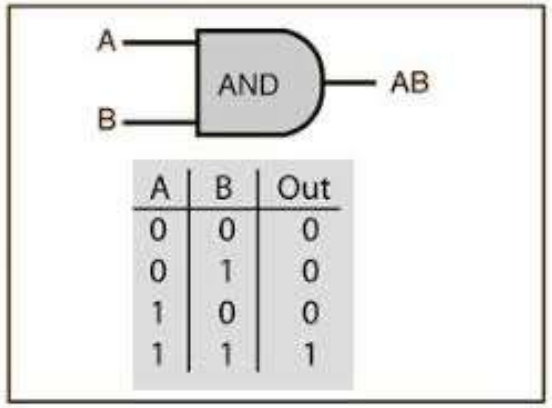
Electronics: AND Gate with Transistors

Ben Eater on YouTube: AND GATE

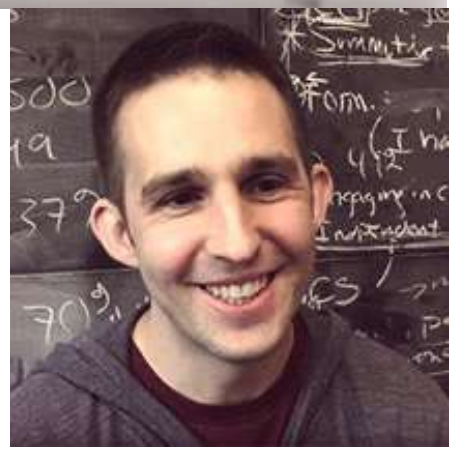


Two-transistor AND gate.



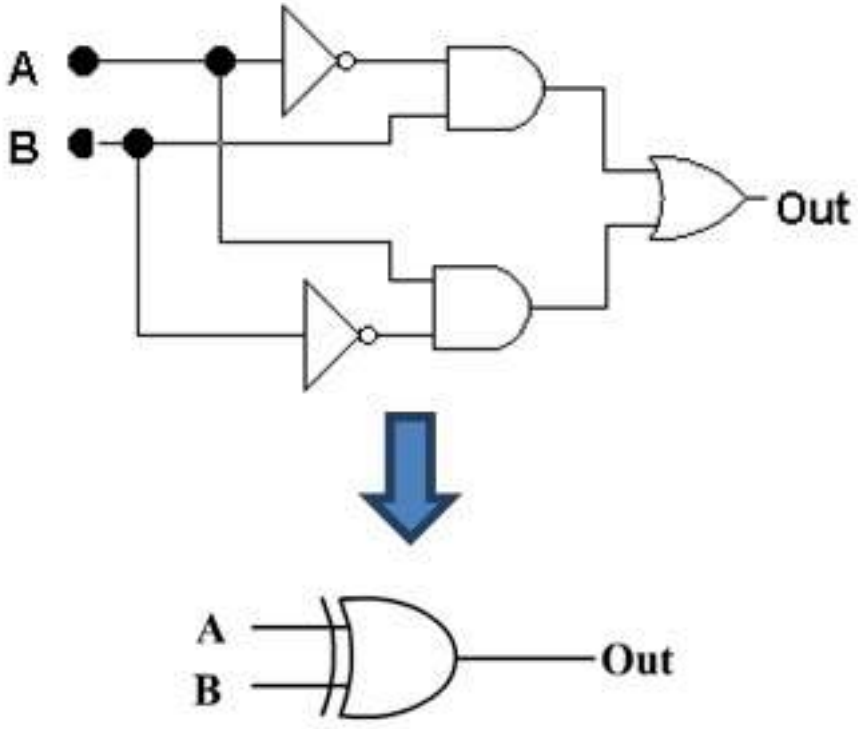


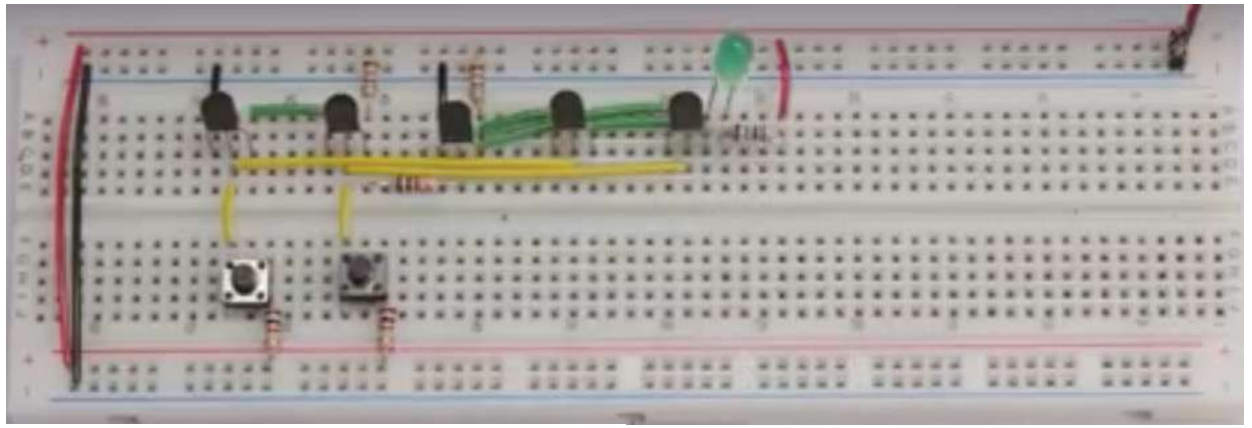
A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

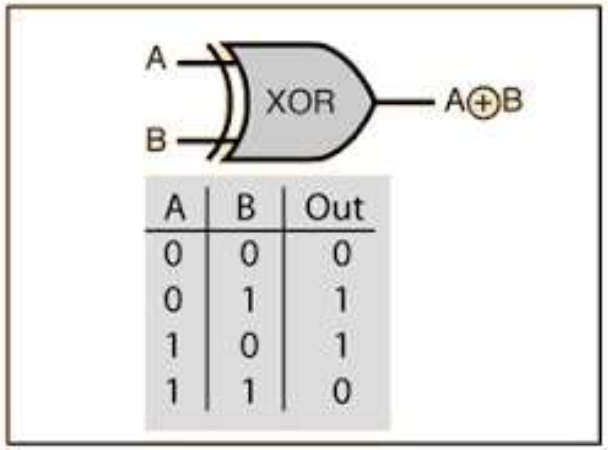


Electronics: XOR Gate with Transistors

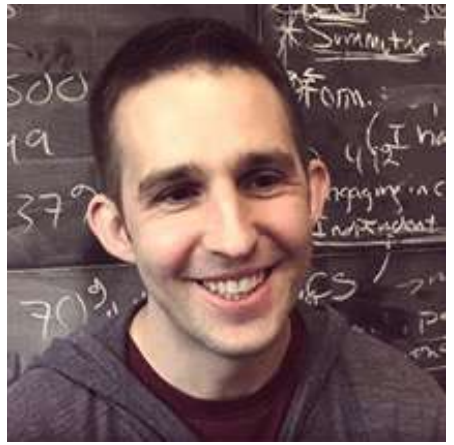
Ben Eater on YouTube: XOR GATE







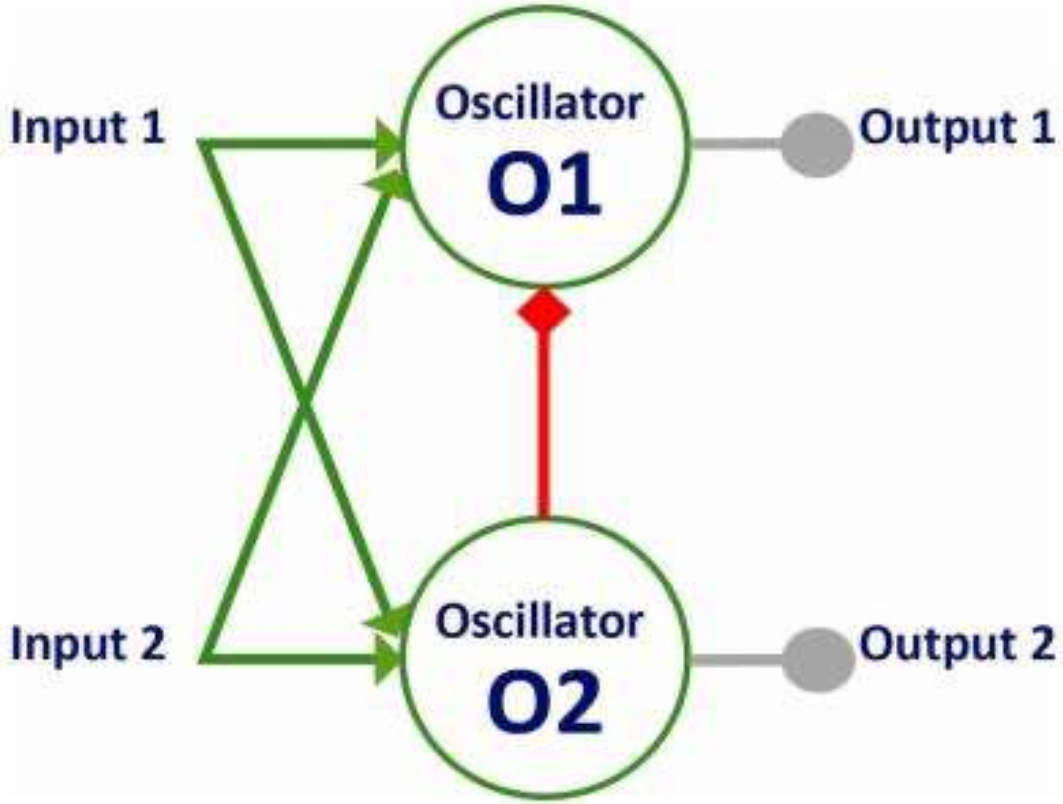
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0



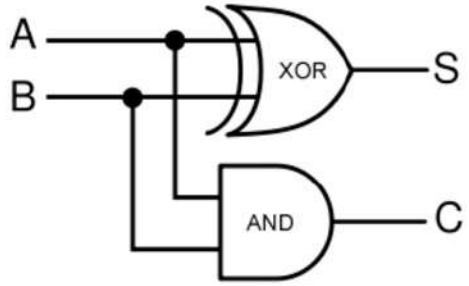
Binary Oscillator Half Adder

Simple design:

Two oscillators
Four input connections
One inhibitory connection
One cross connection
Two outputs
Oscillator O1 has a low threshold
Oscillator O2 has a high threshold



Binary Oscillator Half Adder: Arithmetic Logic

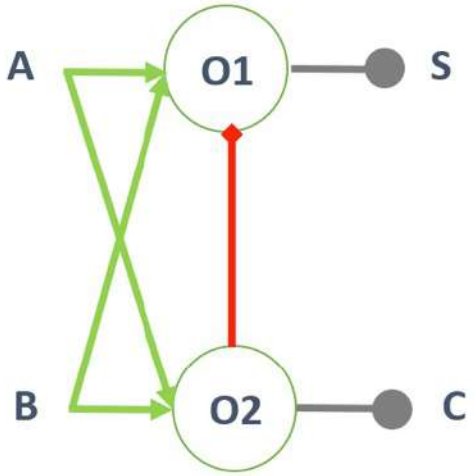


(a)

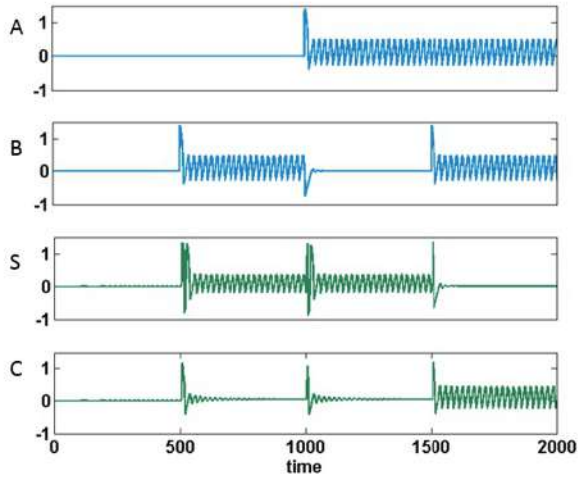
A	0	0	1	1
B	0	1	0	1
S	0	1	1	0
C	0	0	0	1

(b)

Using transistors: Two transistors in the AND gate and five in the XOR gate.



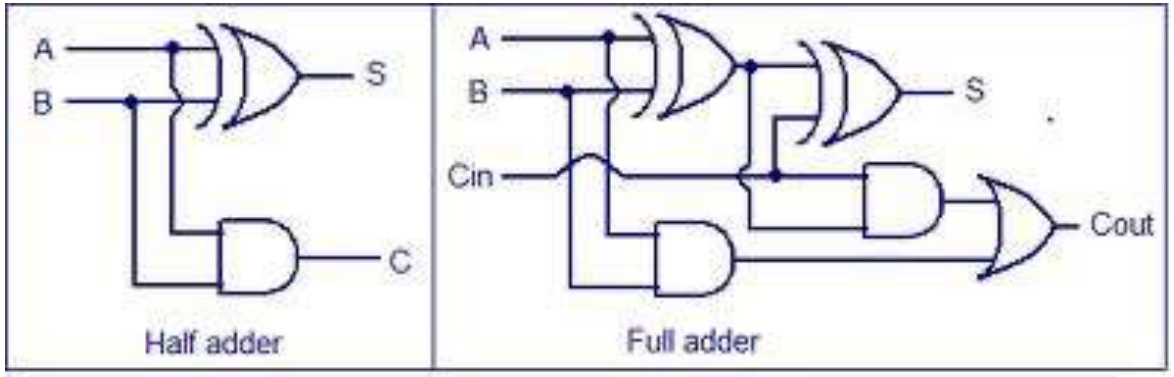
(c)

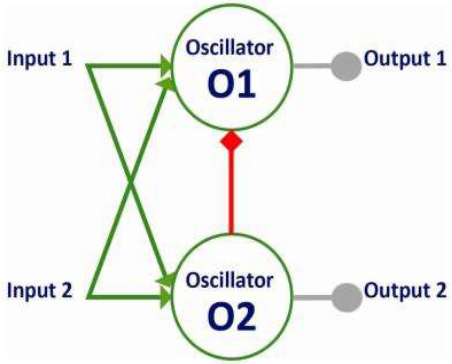


(d)

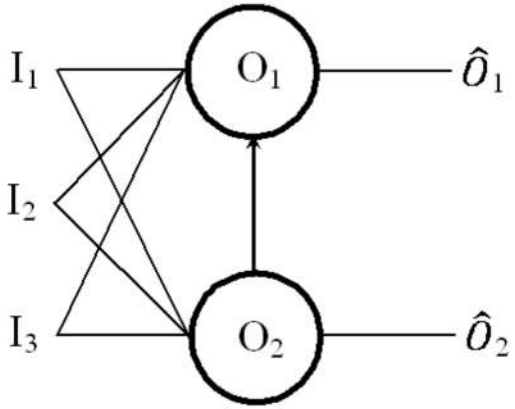
Using threshold oscillators: Four excitatory connections and one inhibitory connection.

Binary Logic Circuitry





Half-adder schematic.

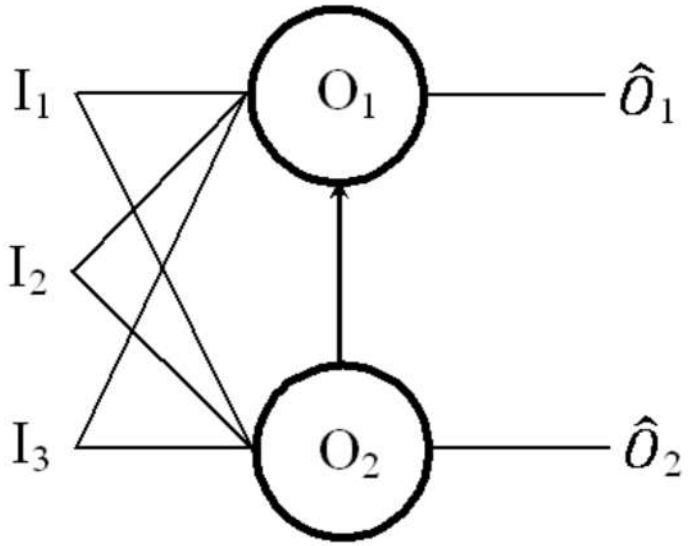


Full-adder schematic.

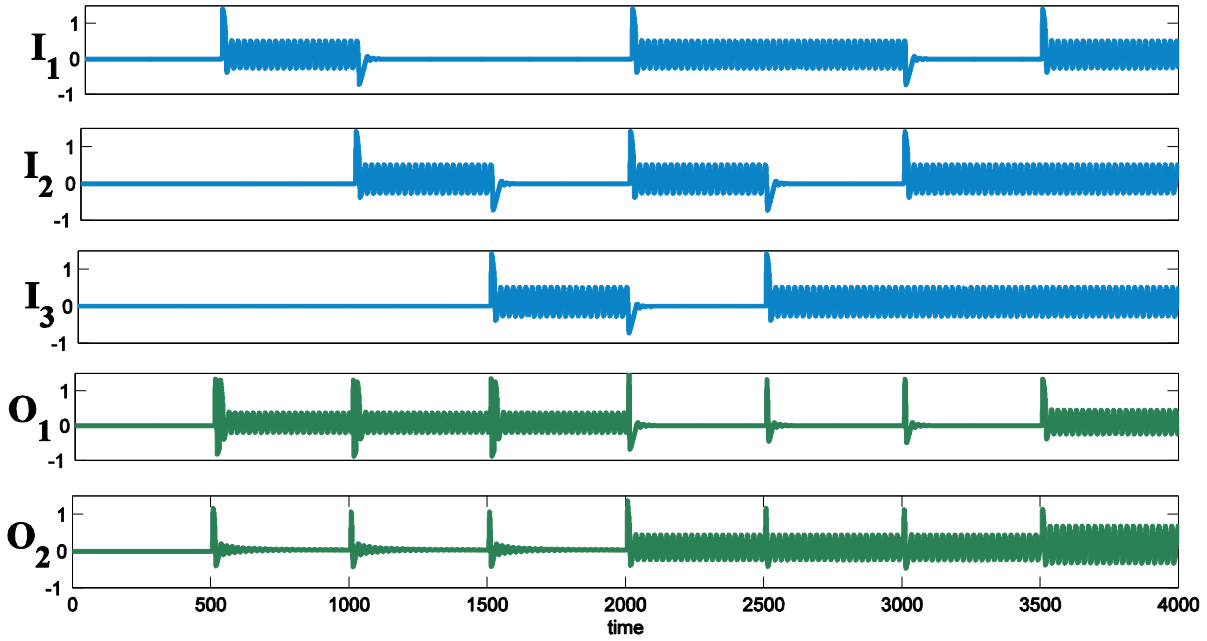
Using transistors: For standard designs, to double processing power it is necessary to at least double the number of components.

Using oscillators: For this design, it is possible to double processing power with a linear increase in components!

Binary Oscillator Full Adder

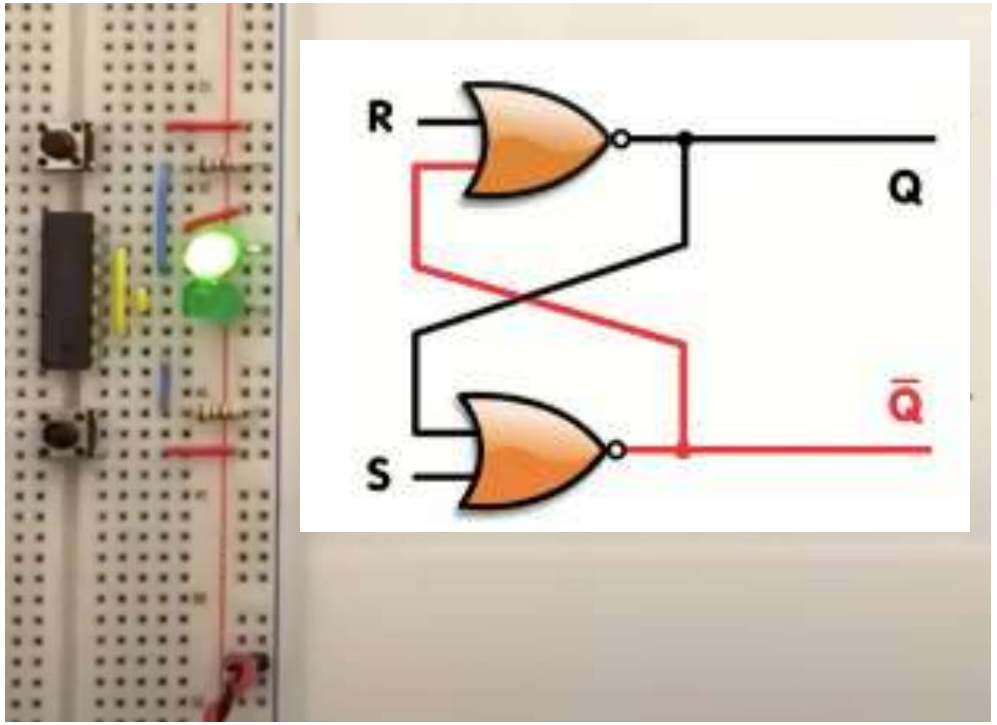


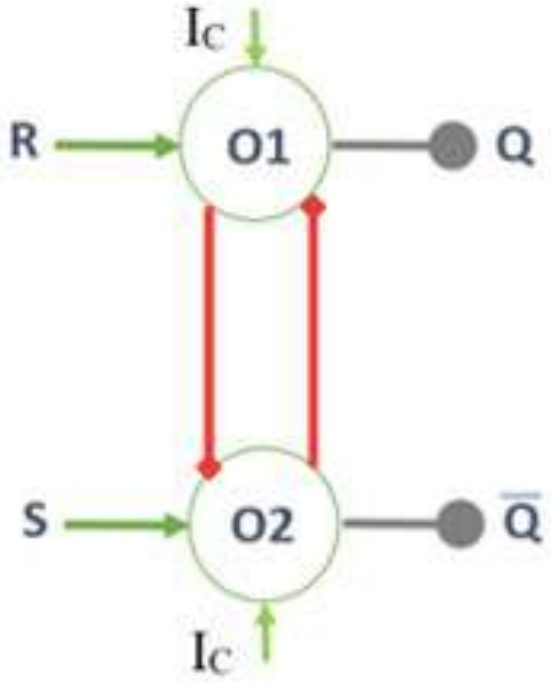
Full-adder schematic.

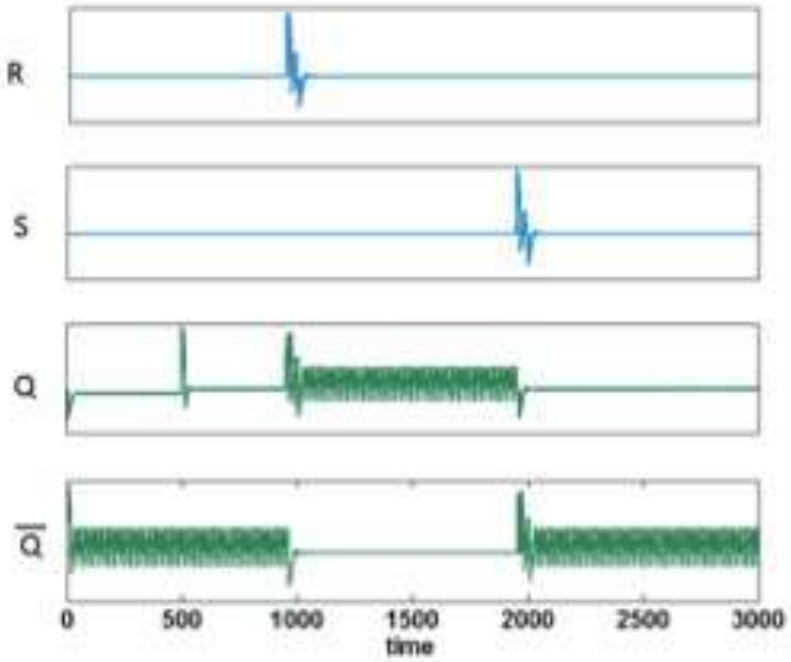


Input and output of a Fitzhugh-Nagumo full-adder.

Set Reset (SR) Flip-Flop: How Computers Store Memory





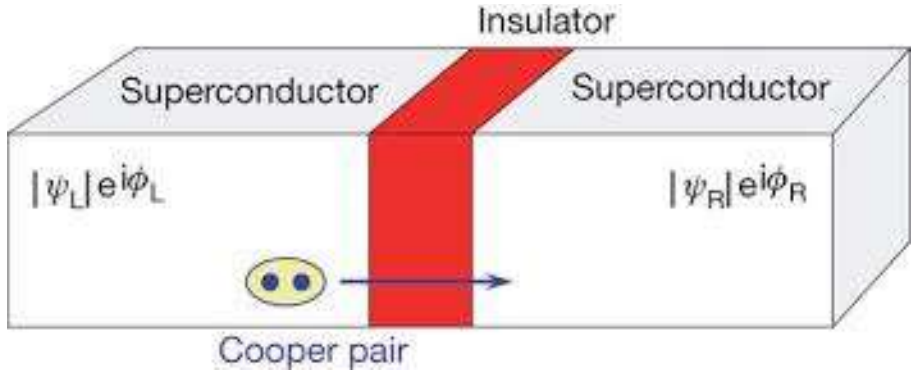


Josephson Junctions (JJs)

- 1st Avenue of Research: Josephson Junctions

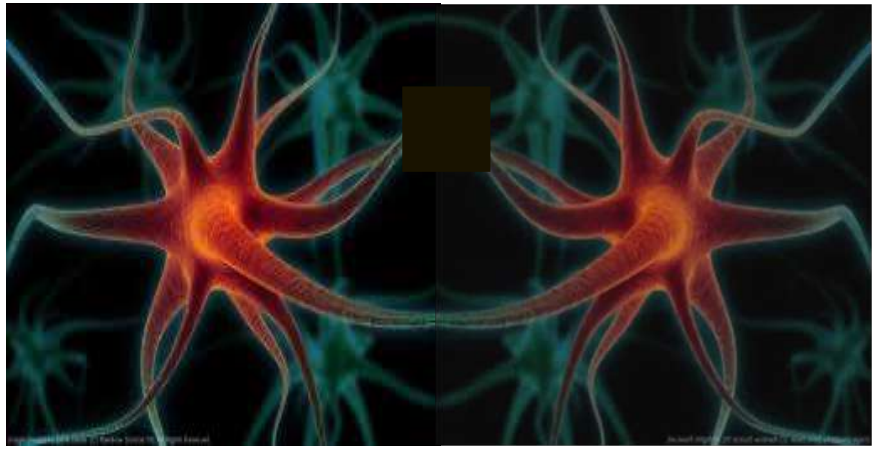
In 1962, Brian David Josephson predicted the Josephson effect. He was the first to predict the tunneling of superconducting Cooper pairs.

Brian David Josephson

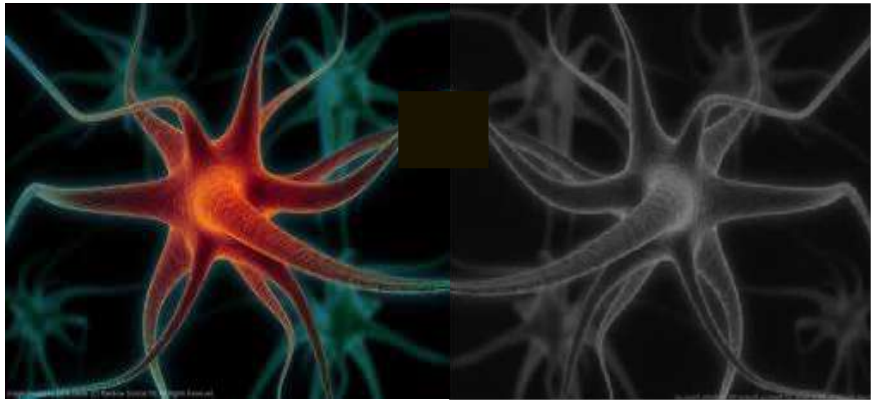


Josephson junctions are natural threshold super-cooled (4 Kelvin), superconducting oscillators that oscillate up to **100 million** times faster than neurons!

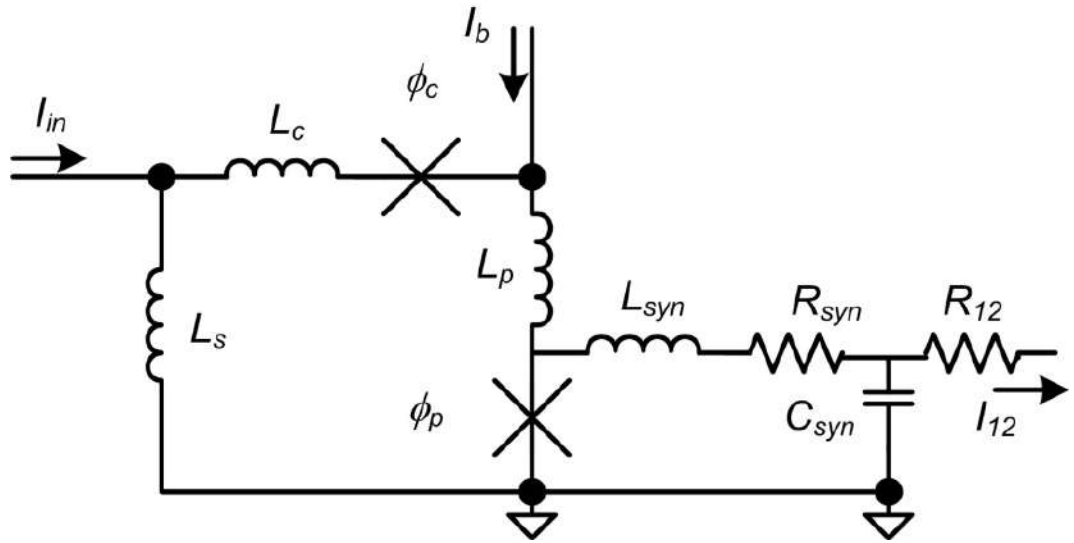
Using JJs to Model Neurons



Excitatory connection.



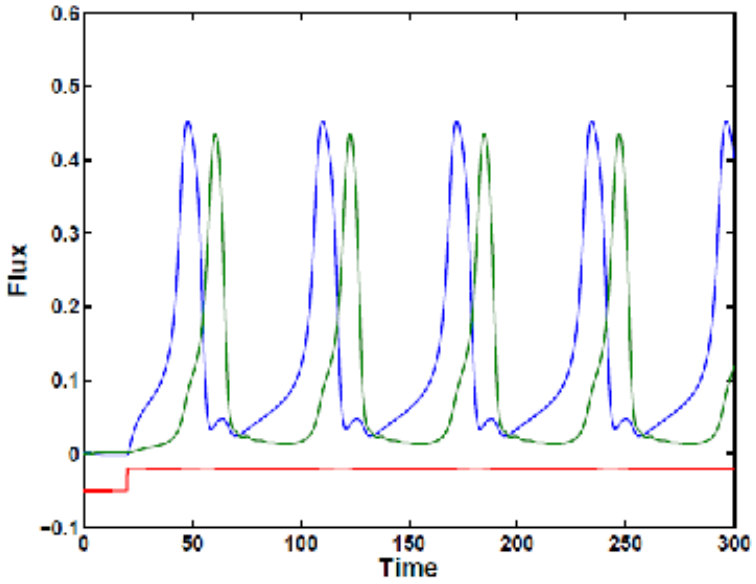
Inhibitory connection.



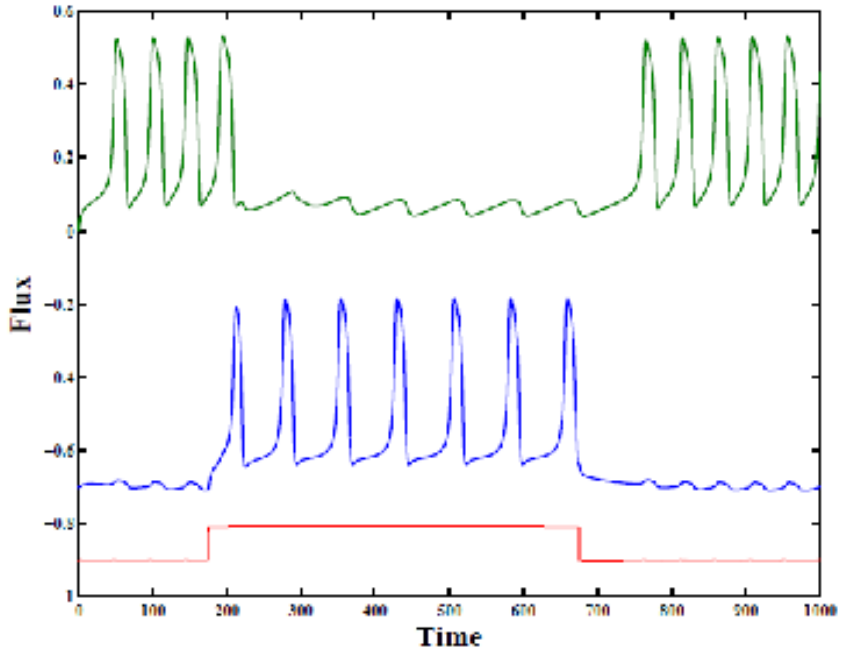
The loop on the left represents a neuron, the loop on the right is a synapse. If the bias current applied to the JJ neuron is positive (negative) with respect to ground, then the synapse is excitatory (inhibitory).

P. Crotty, D. Schult and K. Segall, Josephson junction simulation of neurons, *Phys. Rev. E* **82** (1), 011914, (2010).

Results from Phys. Rev. E Paper



JJ excitatory synaptic coupling.



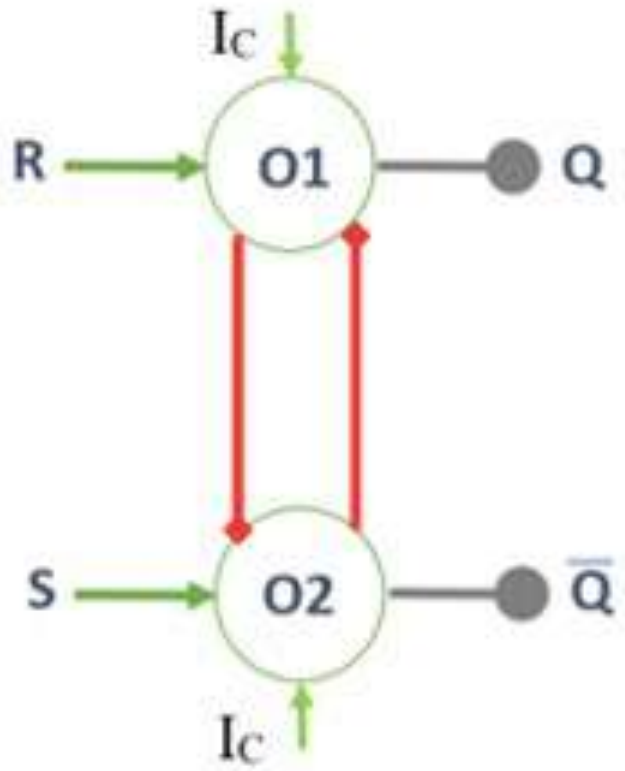
JJ inhibitory synaptic coupling.

Ken Segall.

Ken Segall et al, Synchronization dynamics on the picosecond timescale in coupled Josephson junction neurons, Phys. Rev. E 95.032220 (2017). COLGATE UNIVERSITY, NEW YORK

Toomey E, Segall K and Berggren KK, Design of a Power Efficient Artificial Neuron using Superconducting Nanowires. Front. Neurosci. 13:933 (2019). MIT, BOSTON

Exciting New Results

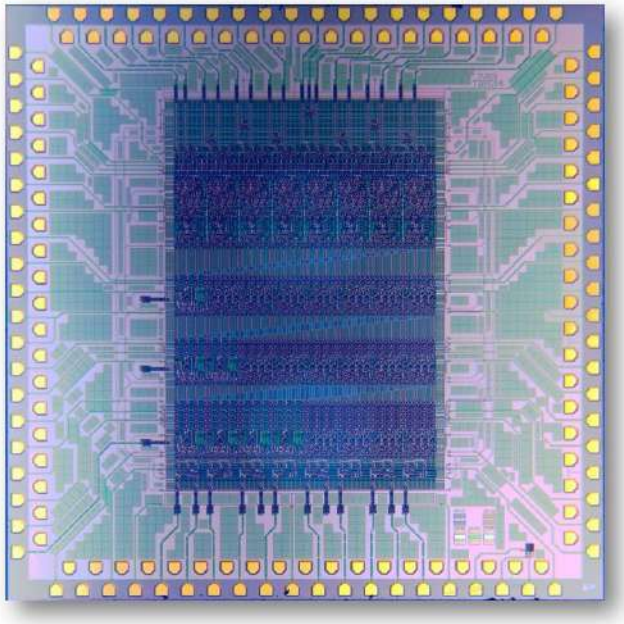


SR flip-flop for memory.

Ken Segall's Laboratory.

If Our Invention Works …

Double Processing Power with a Linear Increase in Components



The world's fastest ALU chip has about 8000 JJs (Ref: HYPRES).

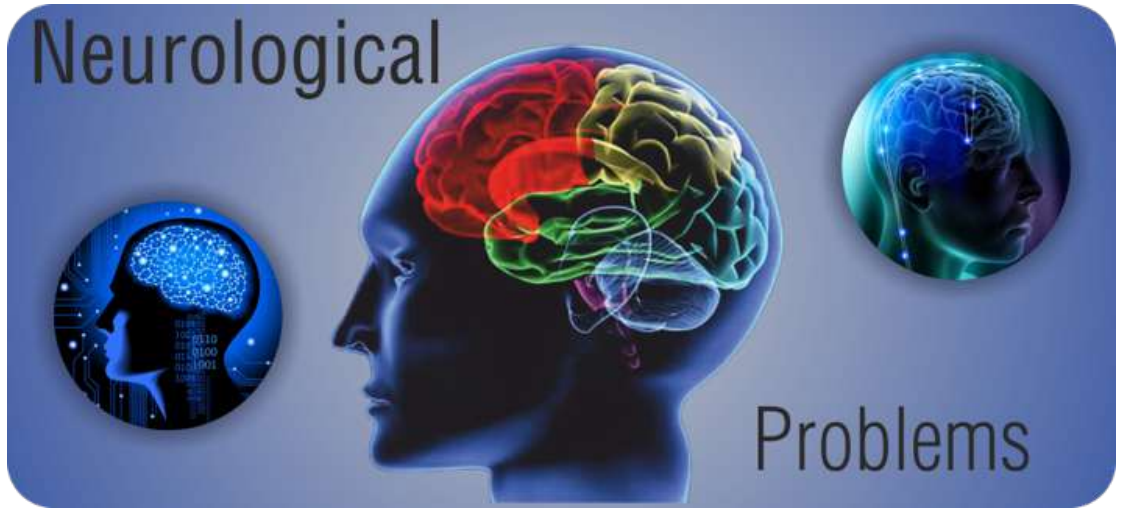
Powerful supercomputers are made up of thousands of trillions of transistors. The Japanese Fugaku supercomputer operates at 537 petaFLOPS. Power consumption 30-40 Mega Watts. June 2021.

Which means that the JJ chip on the left would be more powerful than the computer on the right!

2nd Avenue of Research: An Assay for Neuronal Degradation

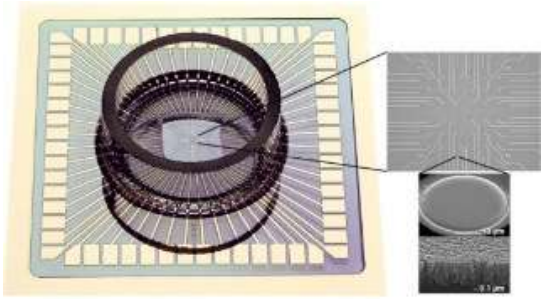
Examples of neurological conditions and disorders include:

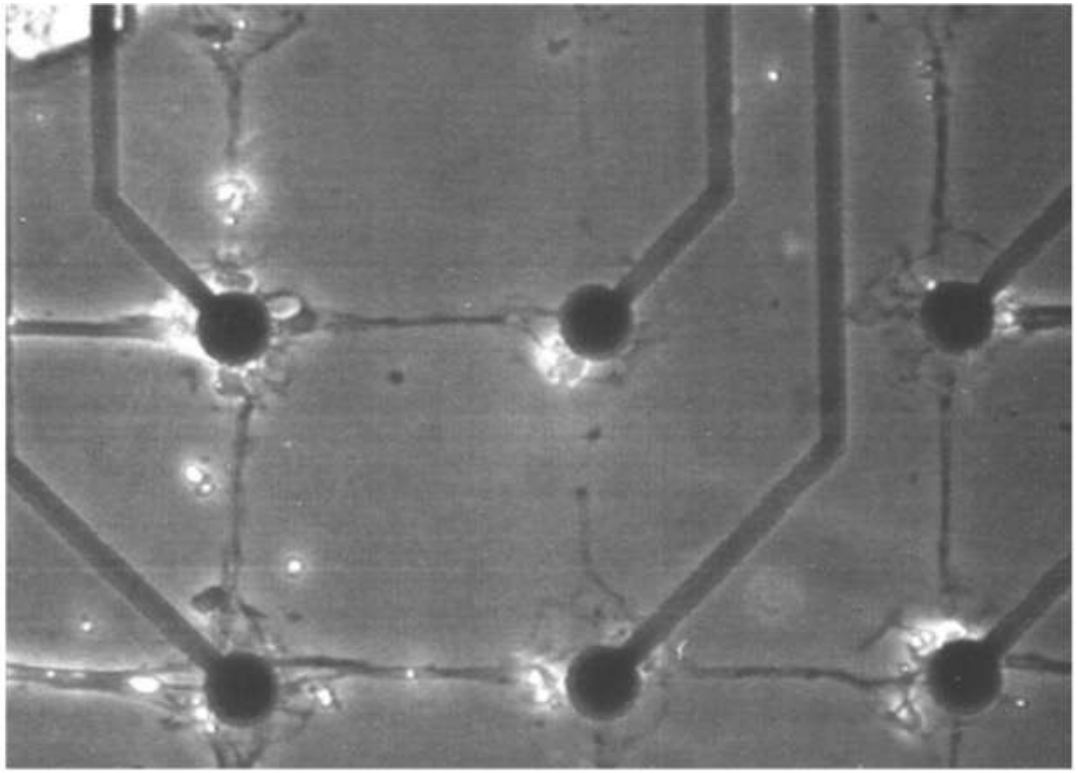
- Alzheimer's disease
- Autism
- Parkinson's disease
- Epilepsy
- Stroke and tetanus
- Brain damage
- Cerebral palsy



Neurons on a Chip

Multi-Electrode Array (MEA)





Magnification: Neurons sitting on electrodes of an MEA connected with axons.

An Assay for Neuronal Degradation

Collaborative work with:

Dr Mark Cotter: Cambridge University

STEM Cells and Cellular Reprogramming

Dr Eric Hill: Aston University

Tissue Engineering Strategies

Dr Paul Roach: Loughborough University

Biomaterials and Interface Science

Professor Mark Slevin: MMU, Romania, China

Cell Pathology and Alzheimer's Disease

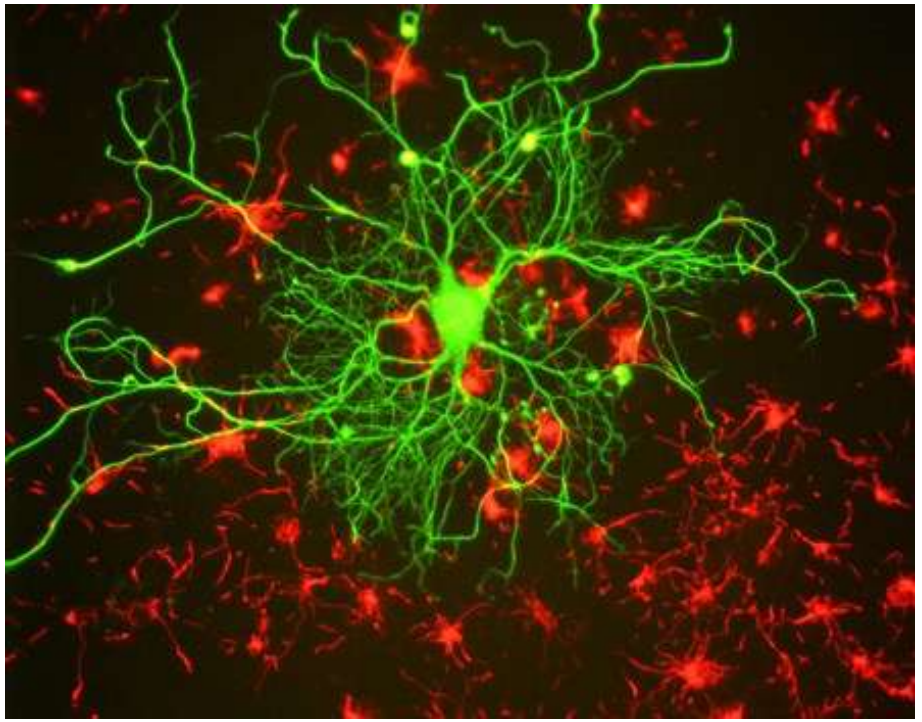
Lynch S, Borresen J, Roach P, Kotter M and Slevin MA, Mathematical modelling of neuronal logic, memory and clocking circuits. International Journal of Bifurcation and Chaos, 30, 2050003, 1-16 (2020).

In the Future … No Need for Electronics: End Session 2

Biological Neuron Circuits

In the future we can control our circuits with light.

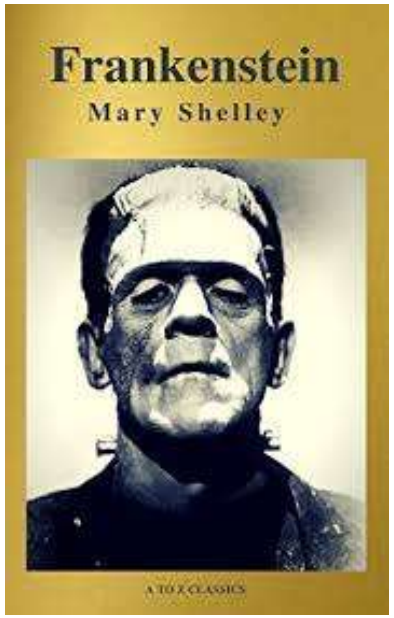
Caenorhabditis elegans has just 302 neurons. Genetically modified worm. Neurons can be switched on and off using light.



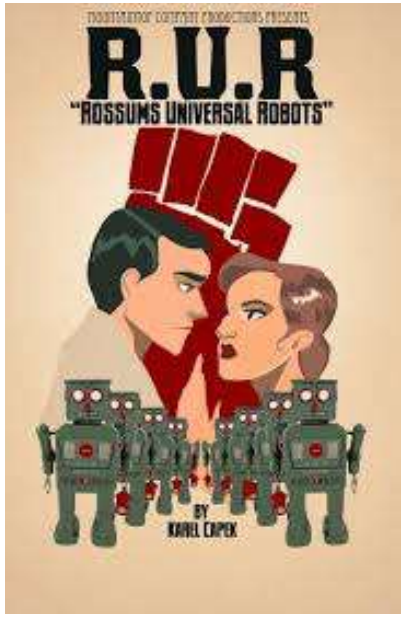
Optogenetics: biological neurons can be controlled (switched) with light and fluorescent dyes can be used to indicate whether or not a neuron is firing.

Artificial Intelligence: Start Session 3

Talos: A giant automotan made from bronze.



Novel: First published in 1818. Mary Shelley.



Play: First published in 1920. Karel Čapek.

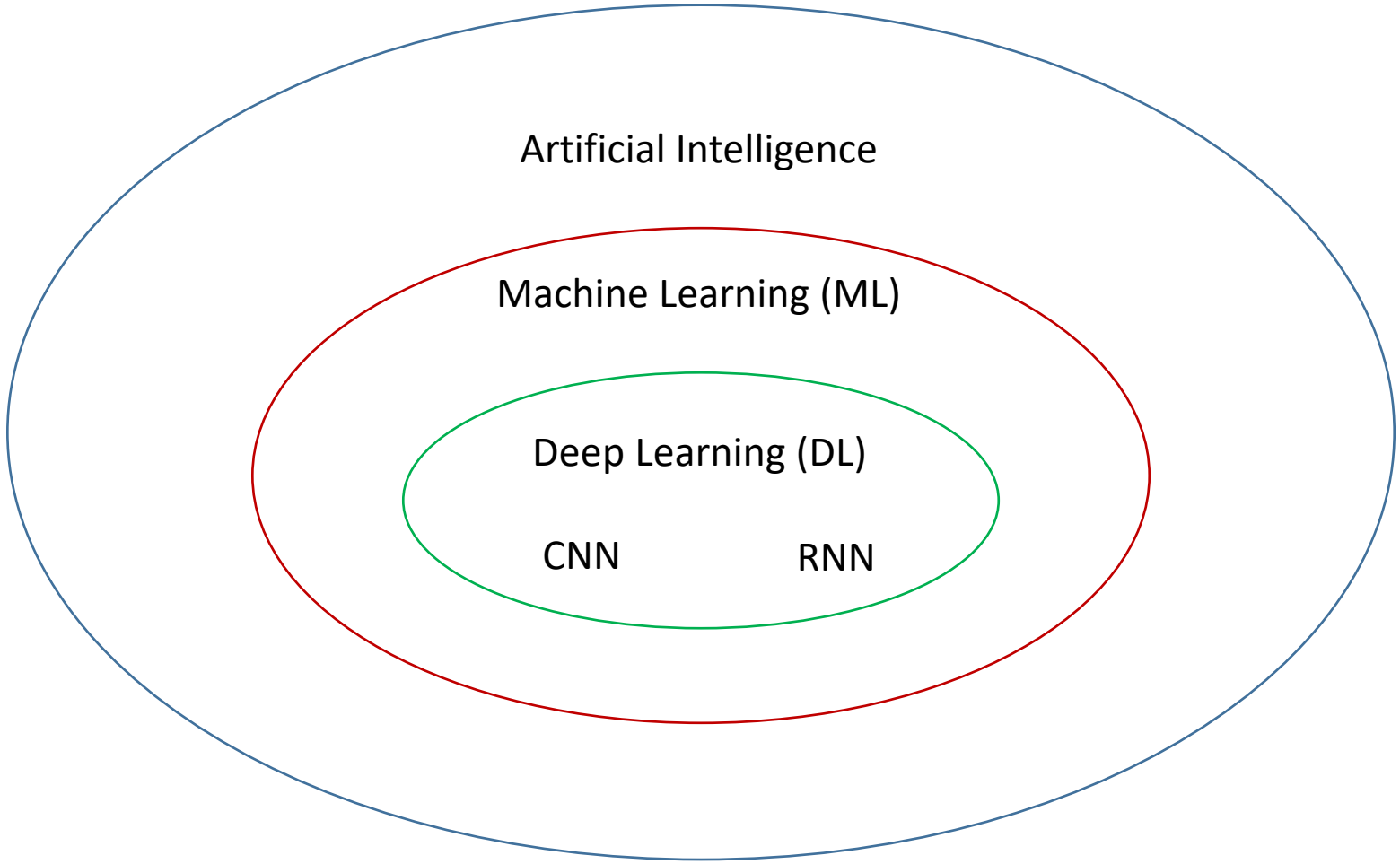
Movie: First screened in 1968. Stanley Kubrick.

Artificial Intelligence (AI)

AI: The science and engineering of making intelligent machines.

ML: A subset of AI involved with the creation of algorithms which can modify itself without human intervention to produce desired output - by feeding itself through structured data.

DL: A subset of ML where algorithms are created and function similar to those in ML, but there are numerous layers of these algorithms - each providing a different interpretation to the data it feeds on.

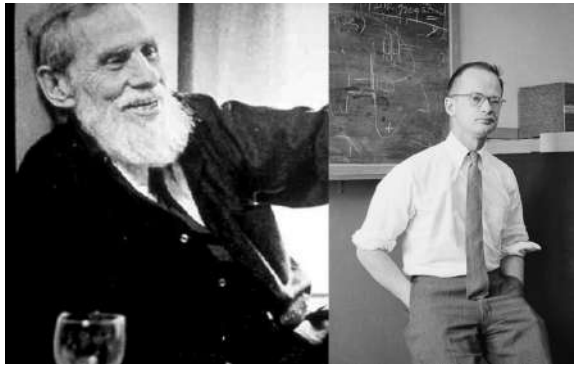


CNN: Convolutional Neural Network
RNN: Recurrent Neural Network

History of AI

Alan Turing (1930s):

The father of theoretical Computer Science and AI. The development of modern Computer Science. A.M. Turing (1950) Computing Machinery and Intelligence. *Mind 49:* 433-460.

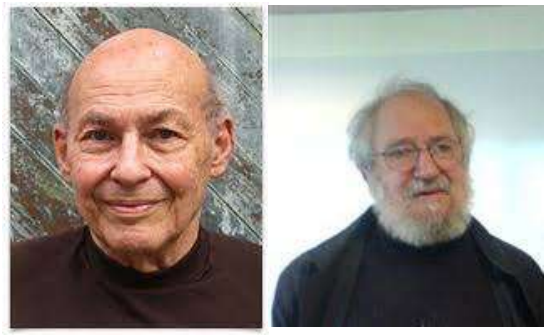


Warren McCullough and Walter Pitts (1943):

A logical calculus and the ideas immanent in nervous activity. The development of modern neural networks.

Frank Rosenblatt (1957):

The perceptron – a perceiving and recognizing automaton. The perceptron learning rule.



Marvin Minsky & Seymour Papert (1969):

Limitations of the perceptron learning algorithm and the XOR gate.

History of AI



D.E. Rumelhart, G.E. Hinton & R.J. Williams (1986): *Learning representation by back-propagating*.

The backpropagation algorithm.

1997: IBMs Deep Blue v Gary Kasparov. First computer program to defeat a world champion in a *match* under tournament regulations. The Man vs. The Machine: Documentary film.

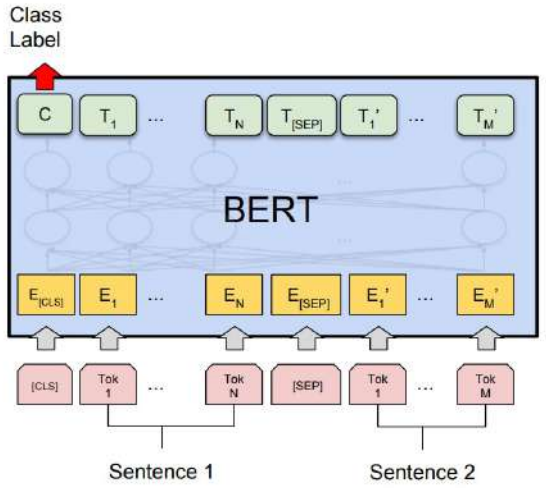
1990s: Work on Machine learning shifts from a knowledge-driven approach to a data-driven approach. Support Vector Machines and Recurrent Neural Networks.

2012: Geoff Hinton: Deep Learning, Microsoft Research, Google, Toronto. DNN translated his English talk into Chinese. Image Net Competition: An error rate of just 16%.

2014: Google-Backed DeepMind Technologies learnt and successfully played 49 classic Atari games by itself using Deep Reinforcement Leaning.

2016: AlphaGo beat Lee Sedol, the first computer Go program to beat a 9-dan professional introducing the Monte Carlo Tree Search (MCTS) algorithm.

History of AI



Amazon Alexa (2014): Natural Language Processing (NLP).

BERT (2018): The best NLP model ever.

The BERT model's architecture is a bidirectional transformer encoder.

TensorFlow 2.0 (2019): An easy to use framework.

TensorFlow 2.0 provides a comprehensive ecosystem of tools for developers, enterprises, and researchers who want to push the state-of-the-art machine learning and build scalable ML-powered applications.

Discovery of new exoplanets (2021)

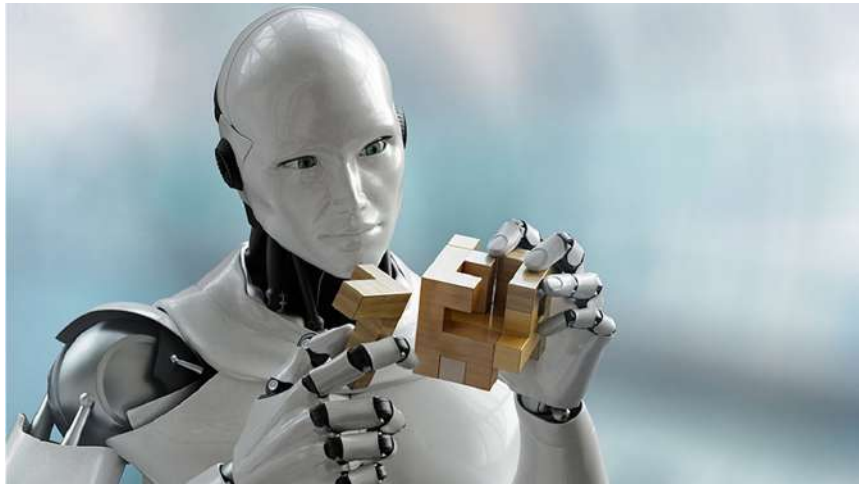
Self replicating robots - Xenobots (2021)

The Future of AI?

Driverless Cars (????): Autonomous vehicles.

The Internet of Things (IoT)

Muthana MSA, Muthana A, Rafiq A,Khakimov A, Albelaly S, Elgendy, Hammoudeh M, **Lynch S** and Elboseny M (2022) Deep reinforcement learning based transmission policy enforcement and multi-hop routing in Quality of Service aware Long Range IoT networks. *Computer Communications* 183(1), 33-50.

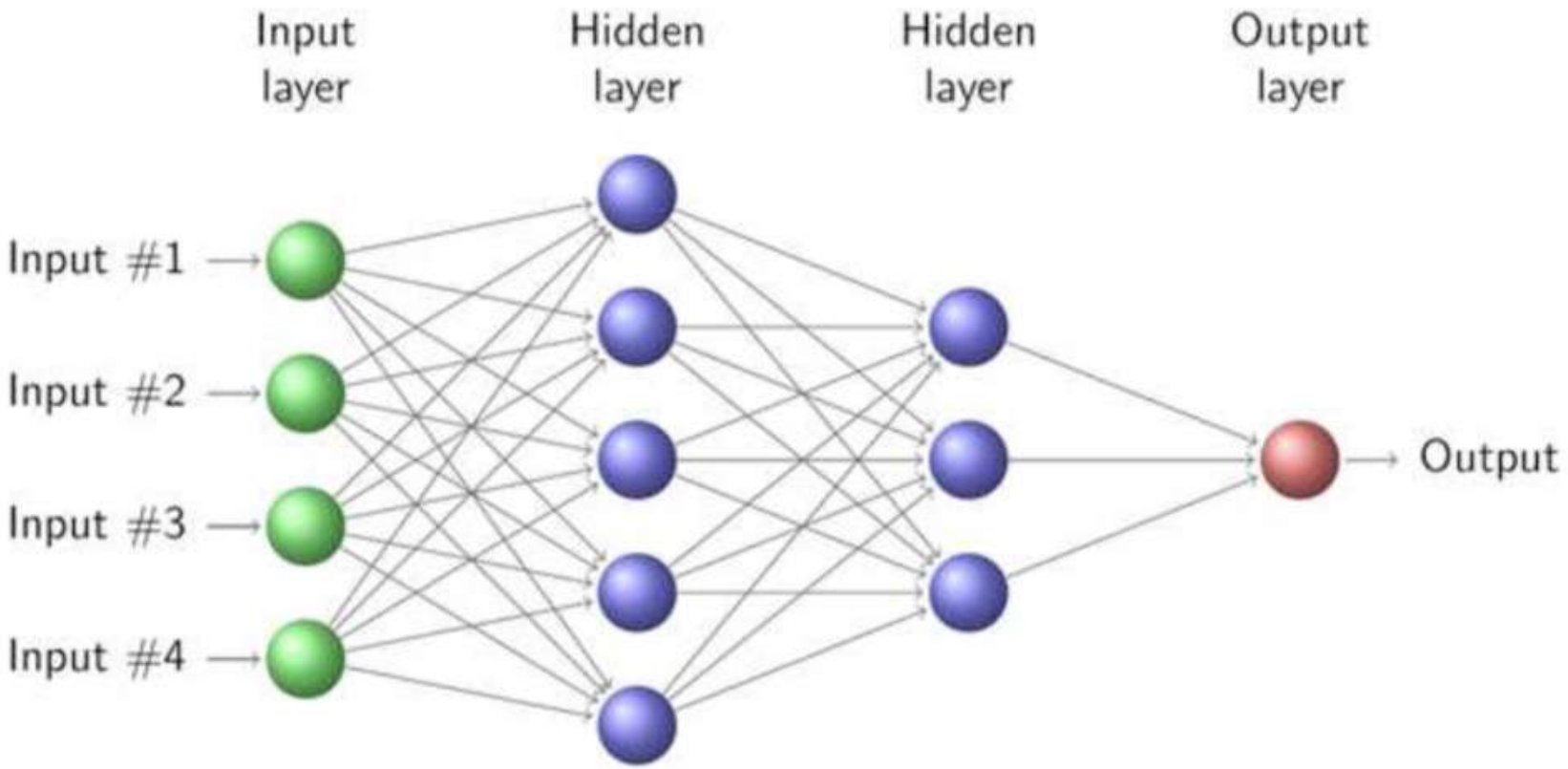


Humanoid Robots(????): Androids are humanoid robots built to aesthetically resemble humans.

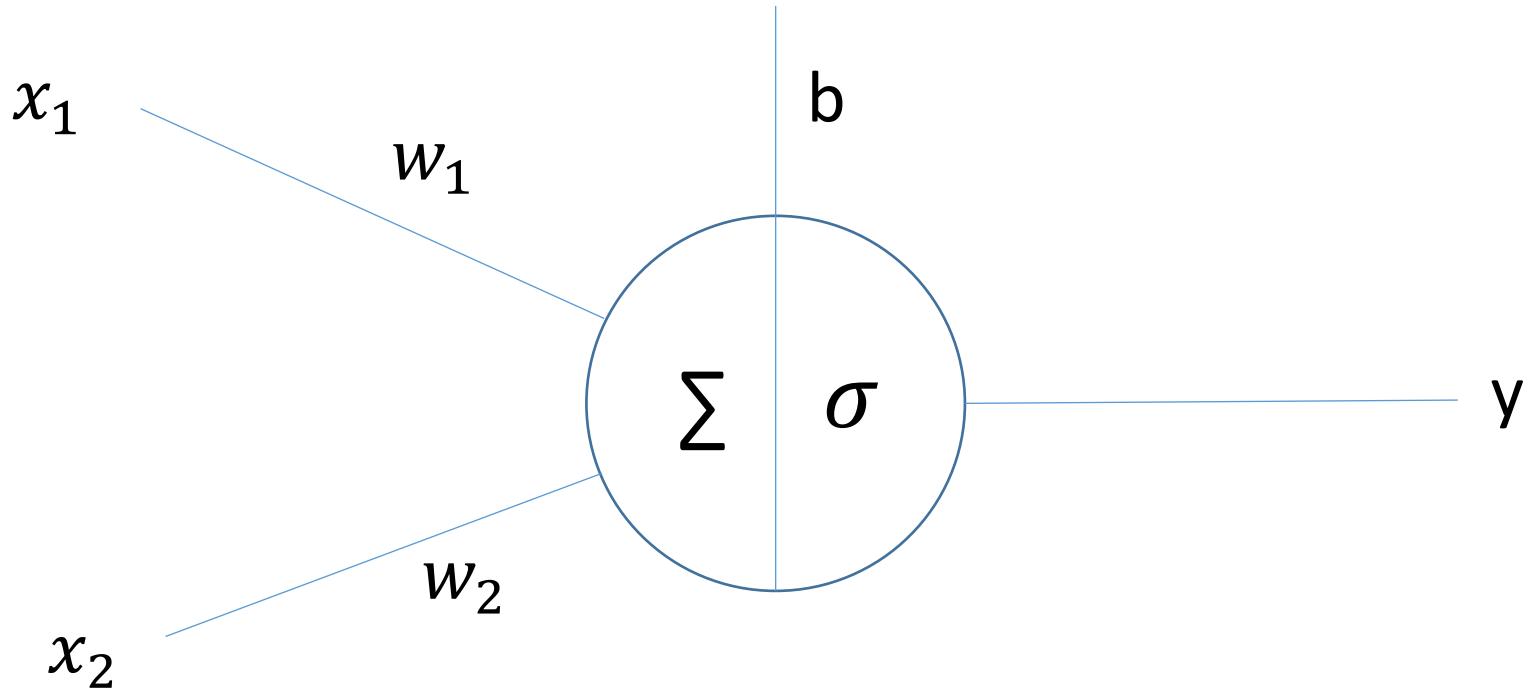
Avatars and AI (????). Frankenstein!

Art, music, poetry, books and mathematical proof!

Neural Networks



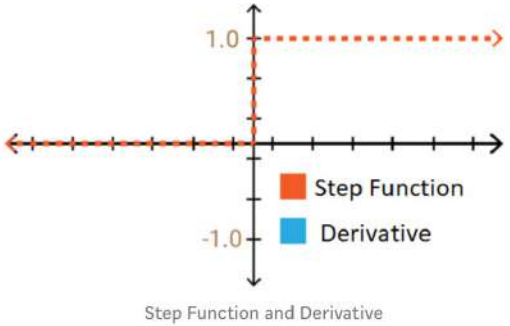
Perceptron: Output of an Artificial Neuron



$$y = \sigma(x_1 w_1 + x_2 w_2 + b)$$

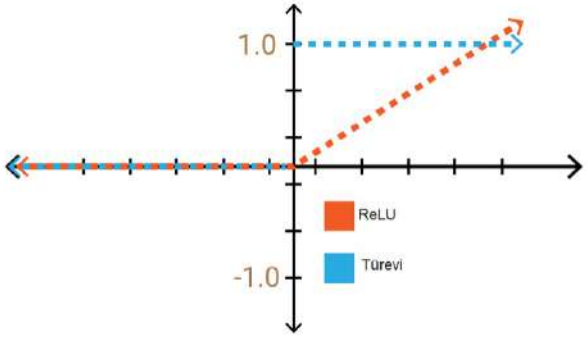
Activation Functions: $\phi(x)$ or $\sigma(x)$

Step function



Step Function and Derivative

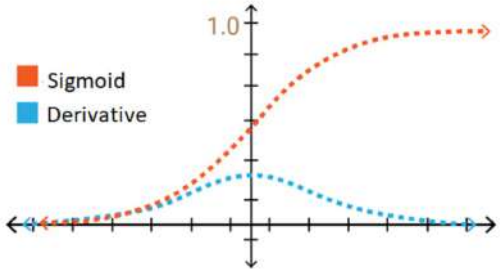
ReLU function



ReLU Function and Derivative

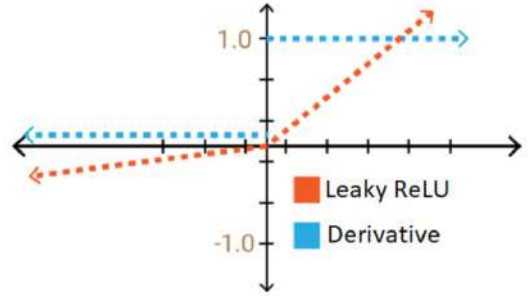
Type equation here.

Sigmoid function



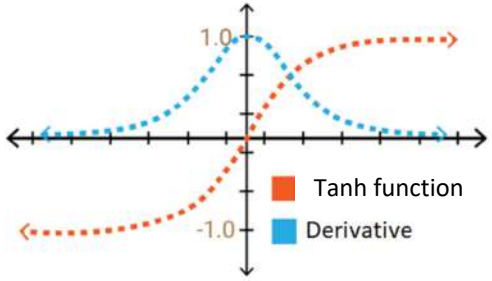
Sigmoid Function and Derivative

Leaky ReLU



Leaky ReLU Function and Derivative

Tanh function

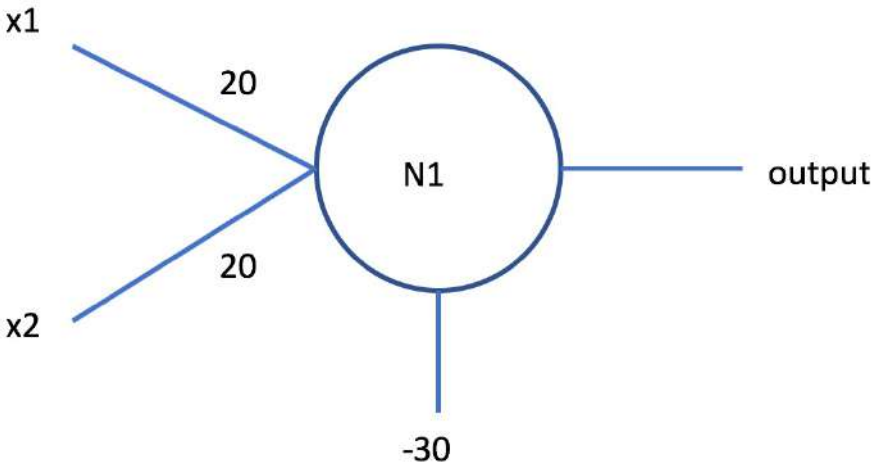


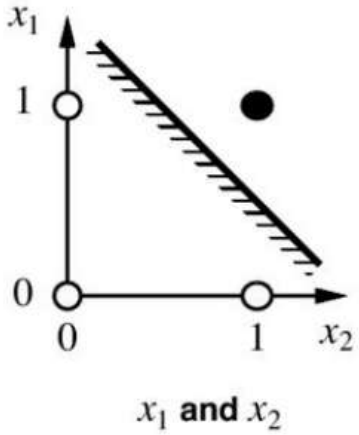
Hyperbolic Tangent and Derivative

Softmax function $S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$

Simple Neural Network: The AND Gate

Take $\sigma(x) = \frac{1}{1+e^{-x}}$, sigmoid.





x_1 and x_2

$$\sigma(20\times0 + 20\times0 - 30) \approx 0$$
$$\sigma(20\times1 + 20\times0 - 30) \approx 0$$
$$\sigma(20\times0 + 20\times1 - 30) \approx 0$$
$$\sigma(20\times1 + 20\times1 - 30) \approx 1$$

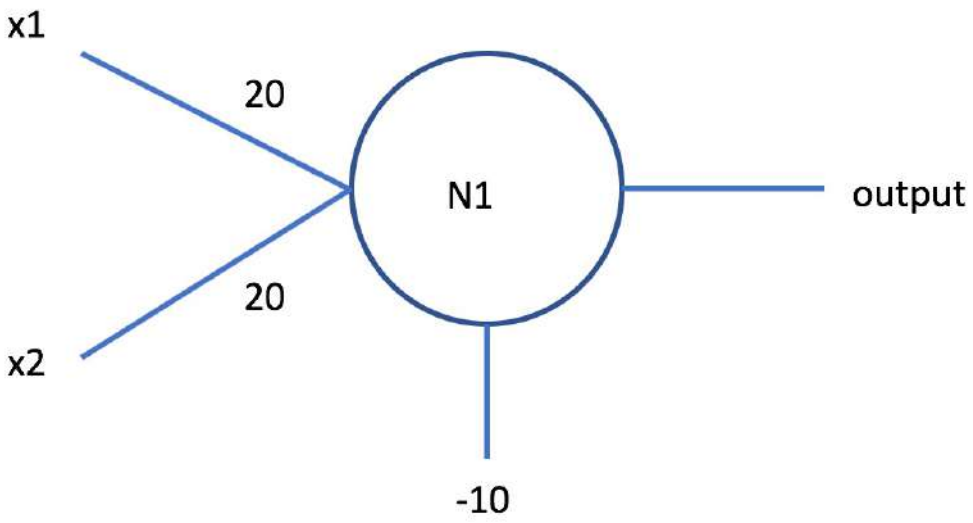
AND gate

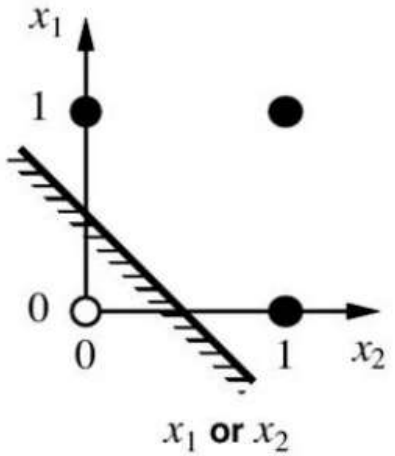
x1	x2	Output
0	0	0
1	0	0
0	1	0
1	1	1

Problem: Write a Python program for this simple neural network.

Simple Neural Network: The OR Gate

Take $\sigma(x) = \frac{1}{1+e^{-x}}$





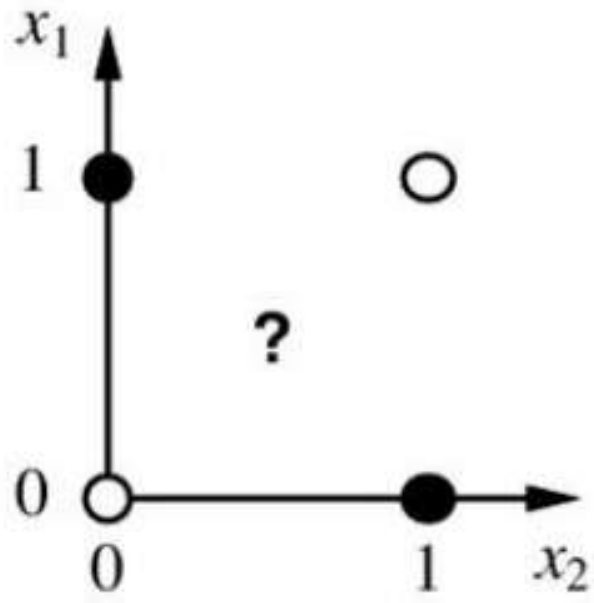
$$\sigma(20\times0 + 20\times0 - 10) \approx 0$$
$$\sigma(20\times1 + 20\times0 - 10) \approx 1$$
$$\sigma(20\times0 + 20\times1 - 10) \approx 1$$
$$\sigma(20\times1 + 20\times1 - 10) \approx 1$$

OR gate

x1	x2	Output
0	0	0
1	0	1
0	1	1
1	1	1

Problem: Write a Python program for this simple neural network.

Neural Networks: The XOR Gate

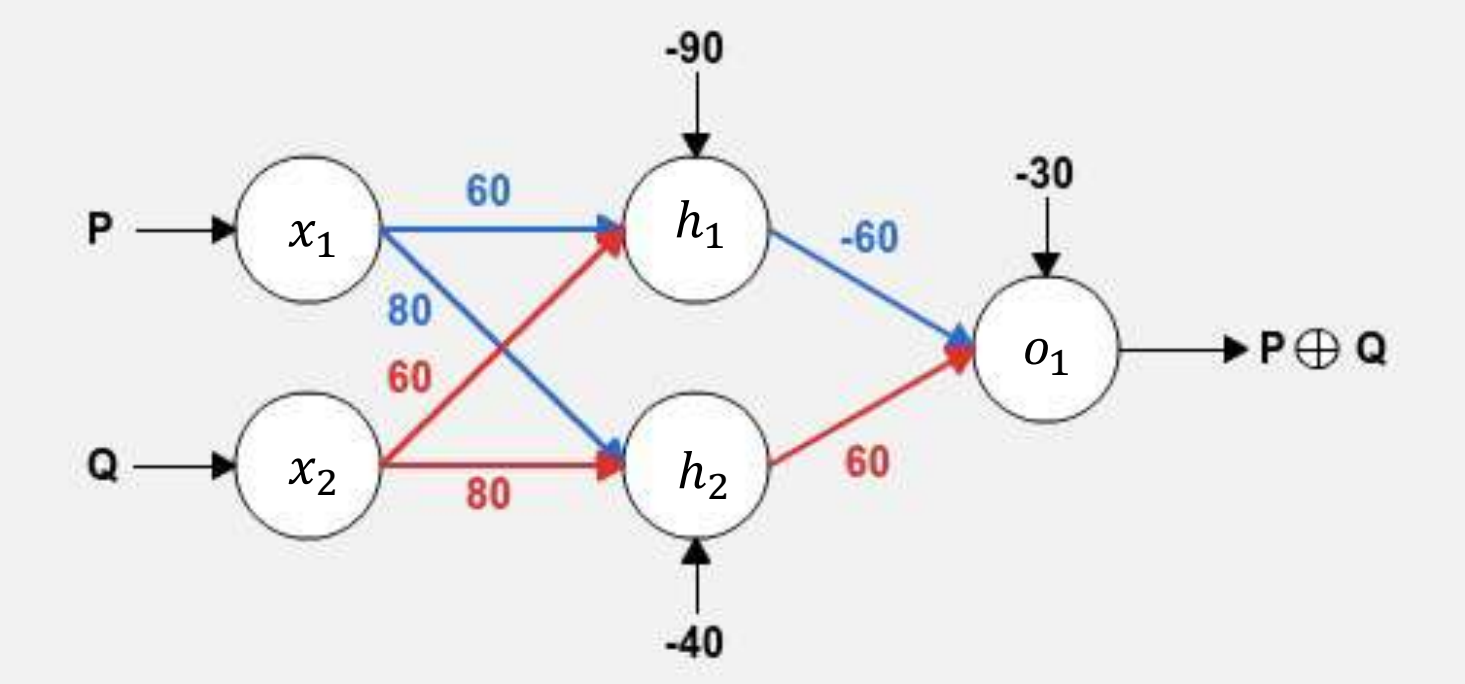


x_1 **xor** x_2

XOR is not linearly separable using a single neuron.

Neural Networks: The XOR Gate

Take $\sigma(x) = \frac{1}{1+e^{-x}}$



XOR is linearly separable when inserting a hidden layer.

Problem: Show that this neural network acts as an XOR gate.

XOR gate

x1	x2	Output
0	0	0
1	0	1
0	1	1
1	1	0

Boston Housing Data

Example 1. Write a Python program to apply the generalized delta learning rule to the Boston housing data for three attributes: columns six (average number of rooms), nine (index of accessibility to radial highways), and 13 (percentage lower status of population), using the target data presented in column 14 (median value of owner-occupied homes in thousands of dollars). Use the activation function $\phi(v) = \tanh(v)$ and show how the weights are adjusted as the number of iterations increases. This is a simple three-neuron feedforward network; there are no hidden layers and there is only one output (see Figure 20.1).

Python Program

Solution. The Python program file is listed in Section 20.5. A summary of the algorithm is listed below to aid in understanding the program:

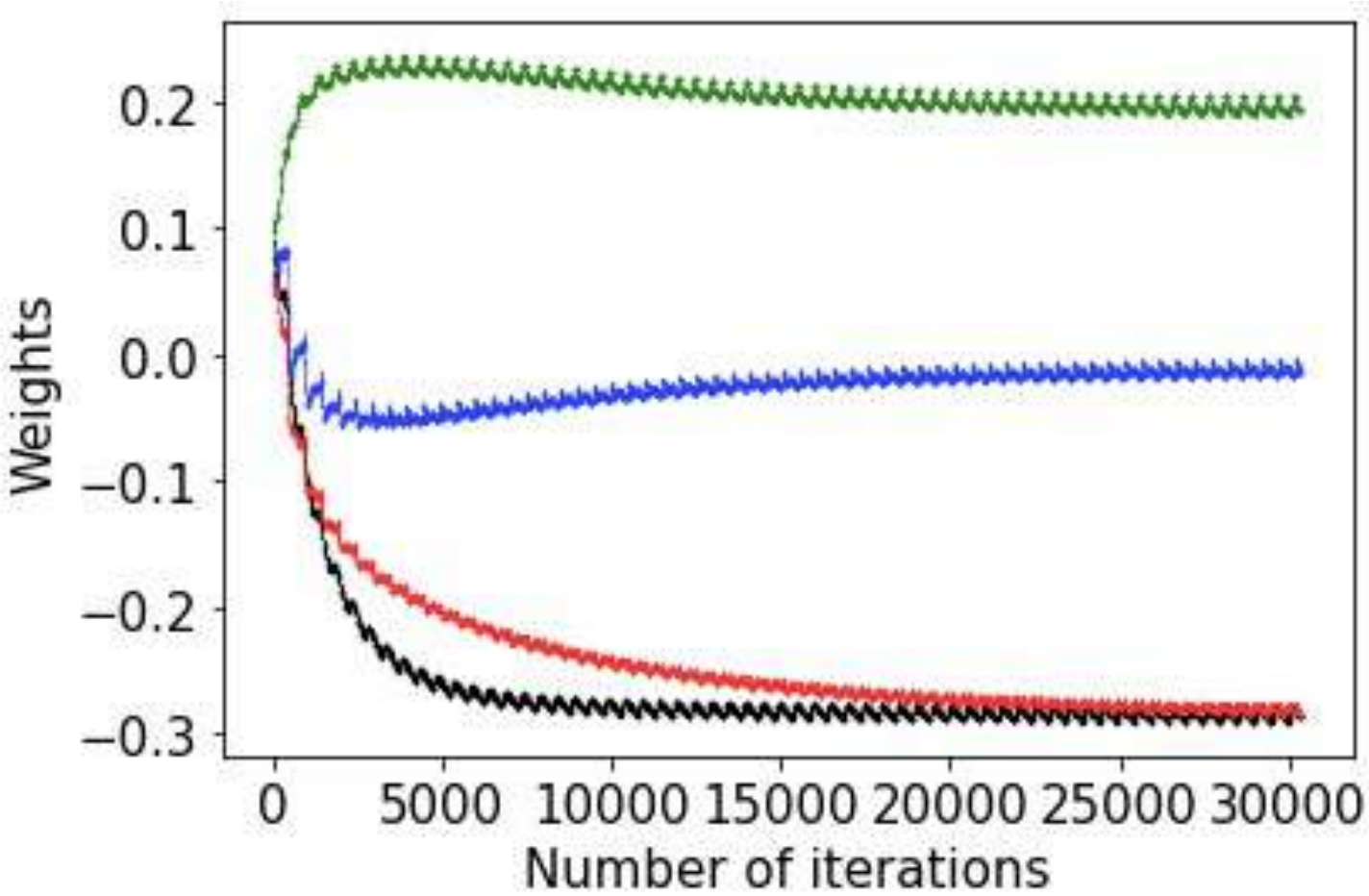
1. Scale the data to zero mean, unit variance, and introduce a bias on the input.
2. Set small random weights.
3. Set the learning rate, say, η, and the number of epochs.
4. Calculate model outputs y_k, the error $t_k - y_k$, the gradients g, and perform the gradient descent to evaluate $w_{kj}(n+1) = w_{kj}(n) - \eta g_{kj}$ for each weight, see (20.1).
5. Plot a graph of weight values versus number of iterations.

Python Program on Google Colab

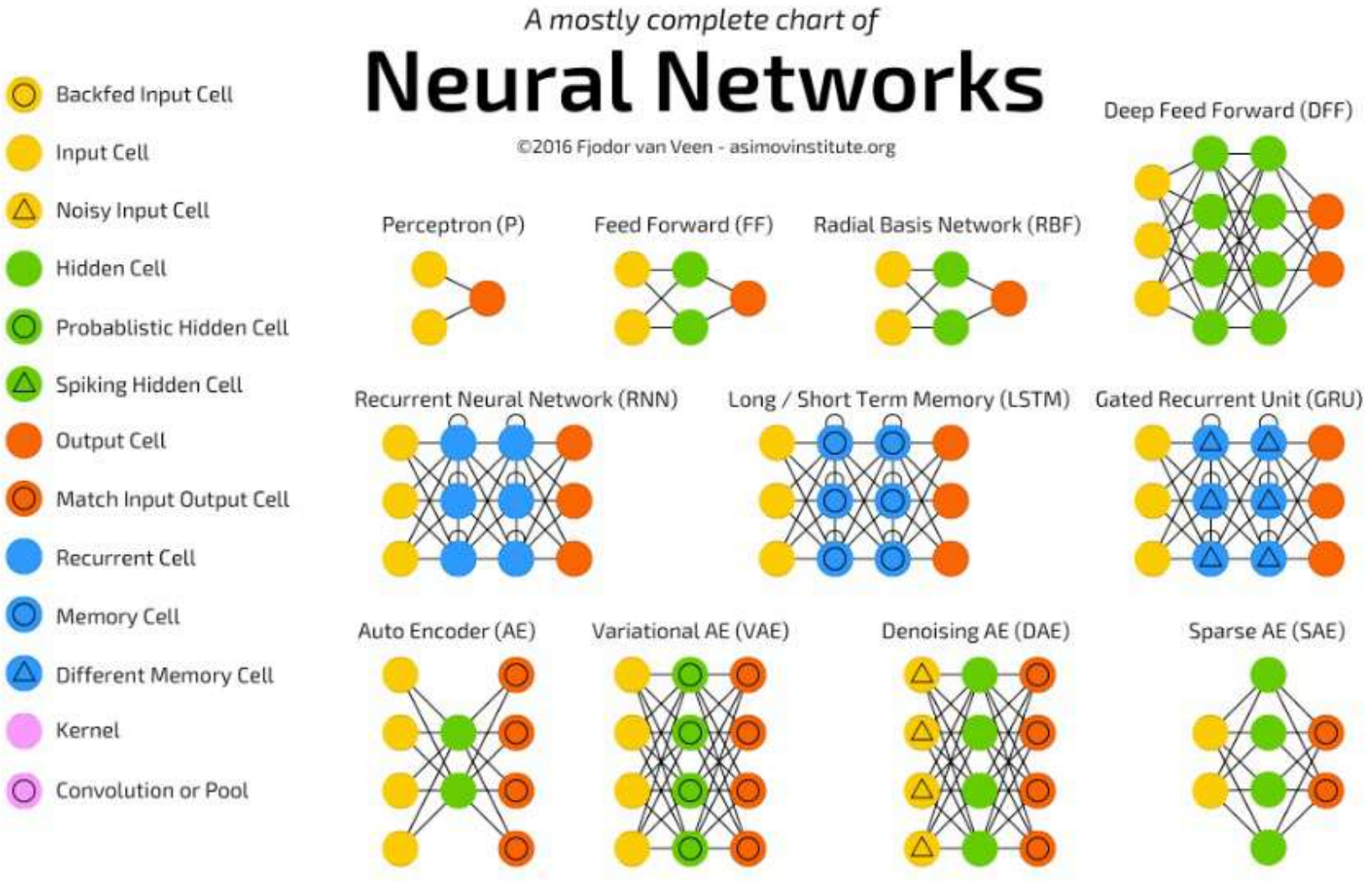
```
# Program 20a: The generalized delta learning rule.
# Backpropagation of errors - using the perceptron.
# Training Boston housing data (housing.txt).
# The target is the value of a house (column 13).
# In this case, use 3 attributes (columns 5, 8, and 12).
# See figure 20.7.

import matplotlib.pyplot as plt
import numpy as np
# Load data from Google Drive.
data = np.loadtxt('/content/gdrive/MyDrive/housing.txt')
rows, columns = data.shape
columns = 4  # Using 4 columns from data in this case
X = data[:, [5, 8, 12]]
t = data[:, 13]
ws1, ws2, ws3, ws4 = [], [], [], []
k = 0
# Normalize the data.
xmean = X.mean(axis=0)
xstd = X.std(axis=0)
ones = np.array([np.ones(rows)])
X = (X - xmean * ones.T) / (xstd * ones.T)
X = np.c_[np.ones(rows), X]
tmean = (max(t) + min(t)) / 2
tstd = (max(t) - min(t)) / 2
t = (t - tmean) / tstd
# Set random weights.
w = 0.1 * np.random.random(columns)
```

```
# Mean squared error.
y1 = np.tanh(X.dot(w))
e1 = t - y1
mse = np.var(e1)
num_epochs = 60  # Number of epochs
eta = 0.001
k = 1
# Backpropagagte.
for m in range(num_epochs):
    for n in range(rows):
        yk = np.tanh(X[n, :].dot(w))
        err = yk - t[n]
        g = X[n, :].T * ((1 - yk**2) * err)
        w = w - eta*g
        k += 1
        ws1.append([k, np.array(w[0]).tolist()])
        ws2.append([k, np.array(w[1]).tolist()])
        ws3.append([k, np.array(w[2]).tolist()])
        ws4.append([k, np.array(w[3]).tolist()])
ws1 = np.array(ws1)
ws2 = np.array(ws2)
ws3 = np.array(ws3)
ws4 = np.array(ws4)
plt.plot(ws1[:, 0], ws1[:, 1], 'k.', markersize=0.1)
plt.plot(ws2[:, 0], ws2[:, 1], 'g.', markersize=0.1)
plt.plot(ws3[:, 0], ws3[:, 1], 'b.', markersize=0.1)
plt.plot(ws4[:, 0], ws4[:, 1], 'r.', markersize=0.1)
plt.xlabel('Number of iterations', fontsize=15)
plt.ylabel('Weights', fontsize=15)
plt.tick_params(labelsize=15)
plt.show()
```

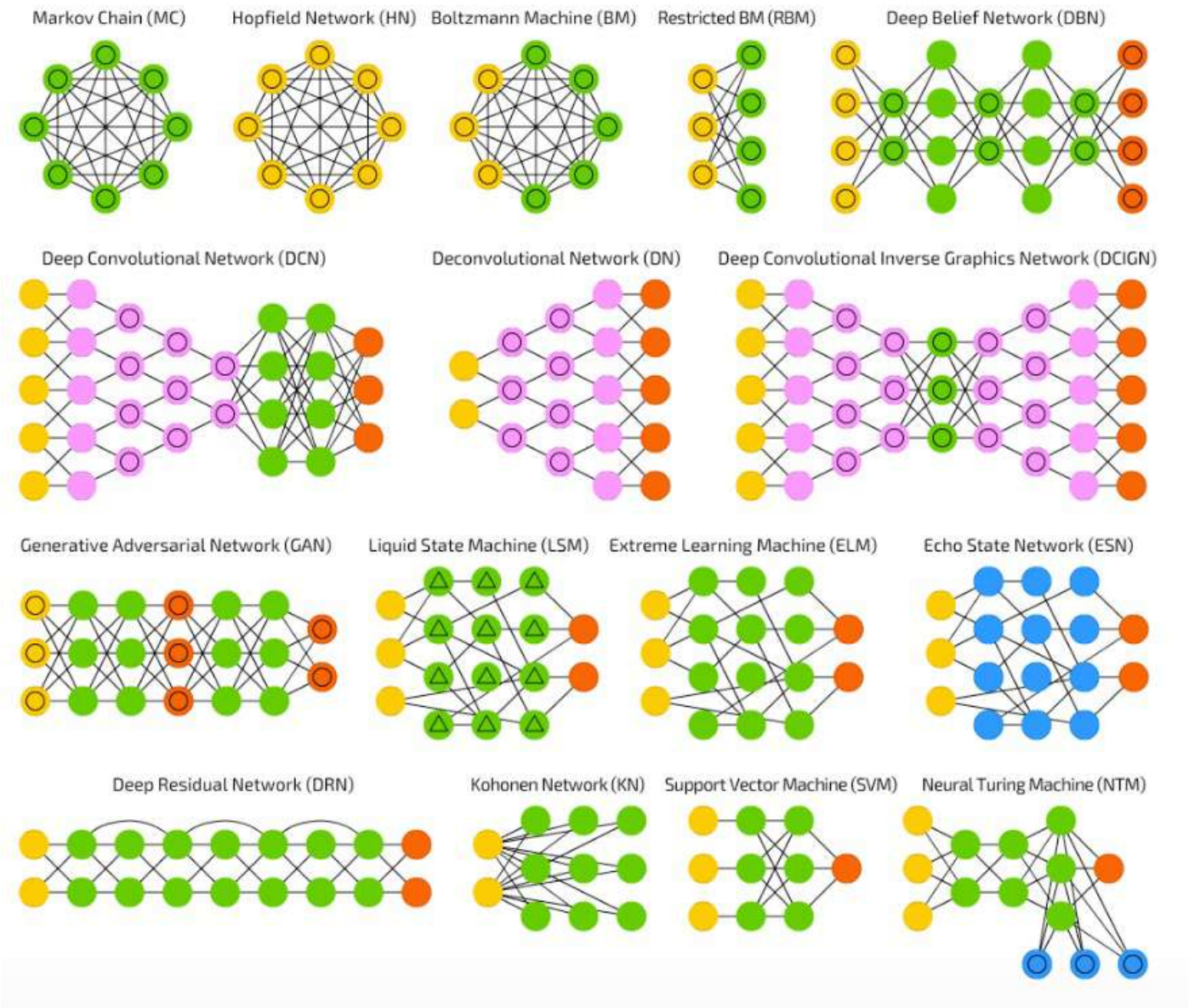
Boston Housing Data: Updates of the Weights



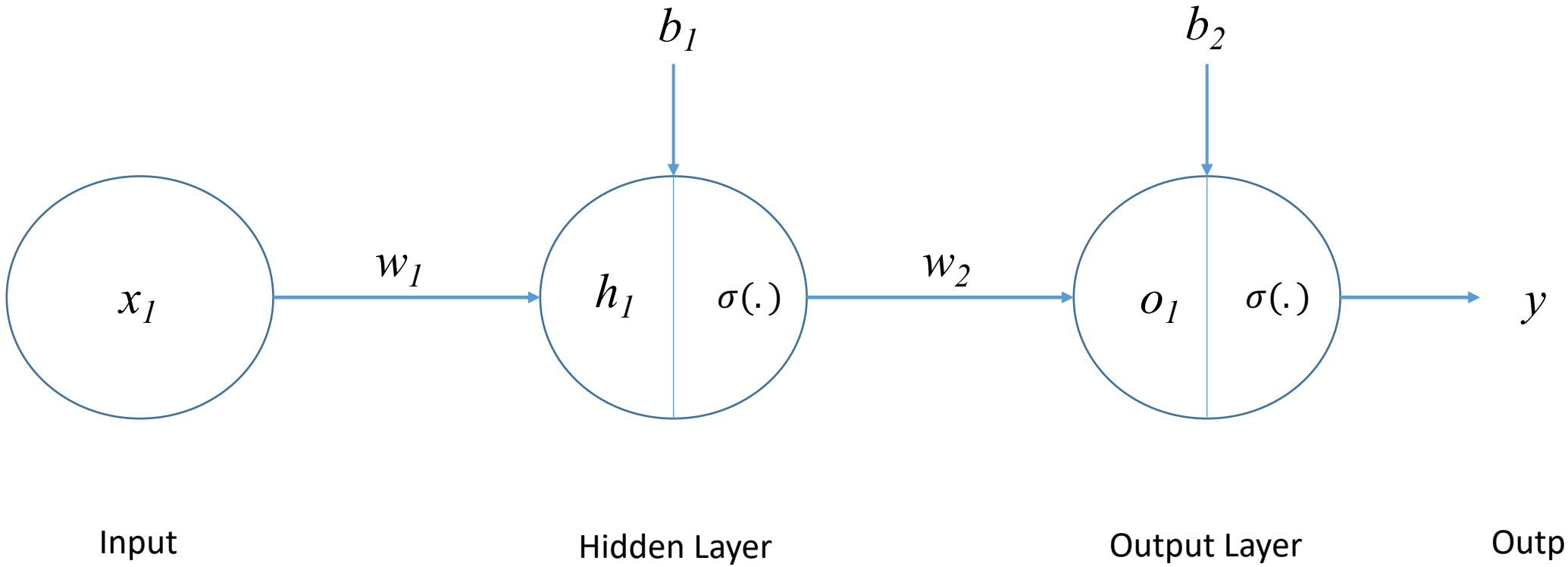
Neural Networks



Neural Networks: End Session 3

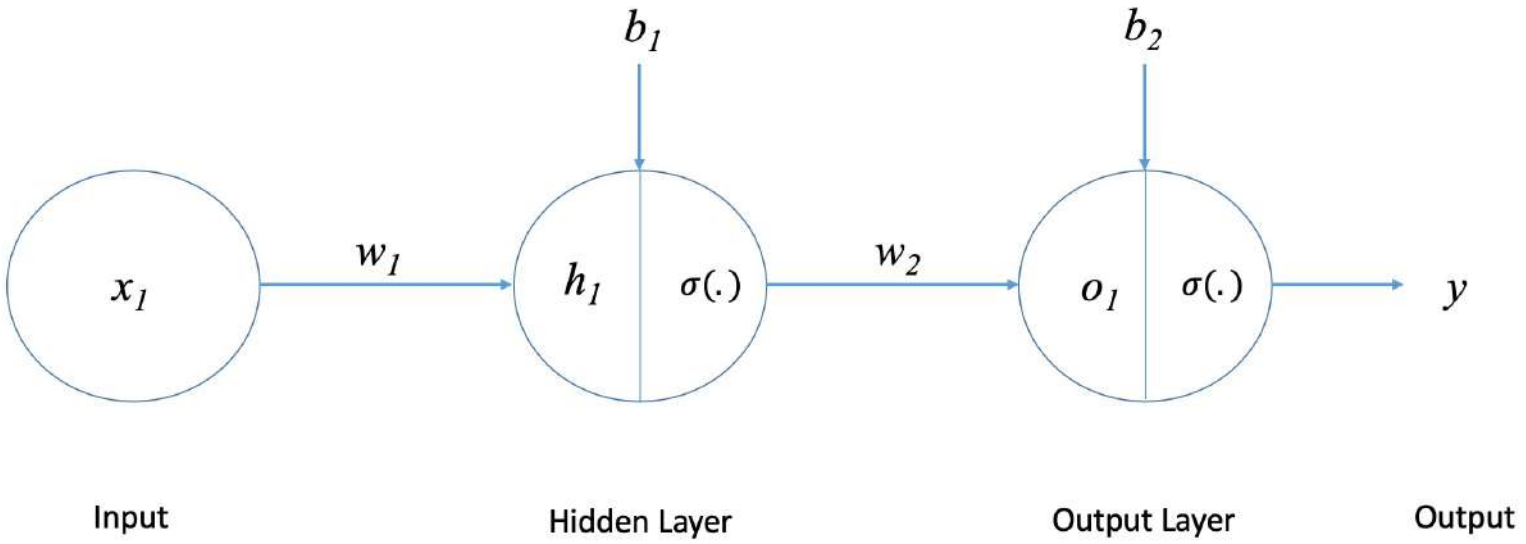


Backpropagation Algorithm: Simple Example: Start Session 4



Activation function: $\sigma(x) = \frac{1}{1+e^{-x}}$. Target is y_t.

Backpropagation Algorithm: Simple Example Feedforward



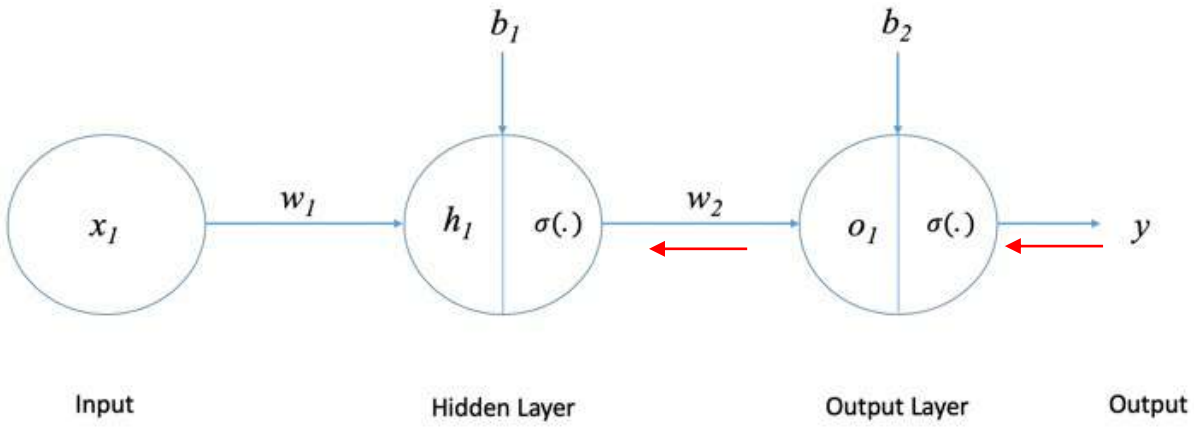
$$y = \sigma(o_1)$$

$$\text{Error} = \frac{1}{2}(y_t - y)^2$$

$$h_1 = x_1 w_1 + b_1 \qquad o_1 = w_2 \times \sigma(h_1) + b_2$$

Backpropagation Algorithm: Simple Example Backpropagate

Weight w_2:



$$h_1 = x_1 w_1 + b_1$$

$$\sigma(h_1) = \frac{1}{1 + e^{-h_1}}$$

$$o_1 = w_2 \times \sigma(h_1) + b_2$$

$$y = \sigma(o_1)$$

$$\text{Error} = \frac{1}{2}(y_t - y)^2$$

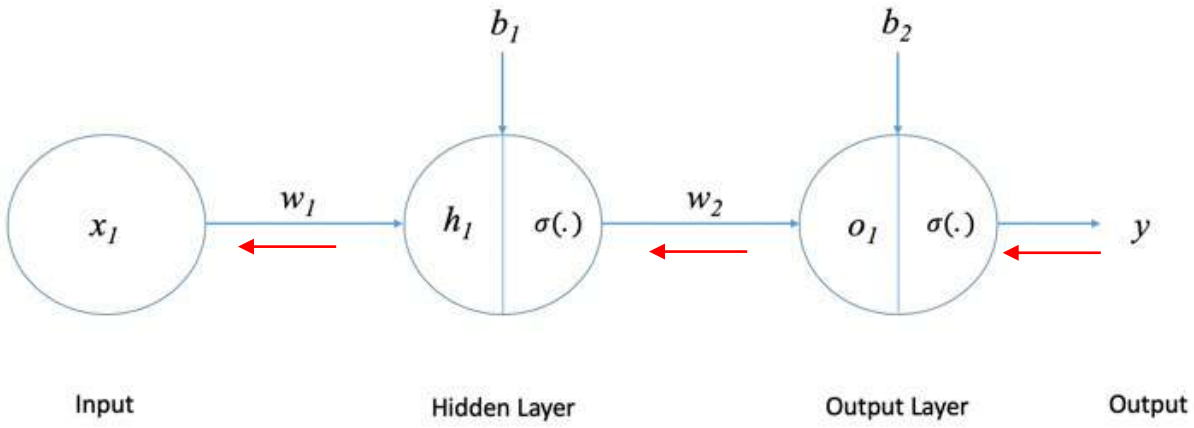
$$\frac{\partial\, Error}{\partial w_2} = \frac{\partial\, Error}{\partial y}\frac{\partial\, y}{\partial w_2} = \frac{\partial\, Error}{\partial y} \times \frac{\partial\, y}{\partial o_1} \times \frac{\partial o_1}{\partial w_2}$$

$$\frac{\partial\, Error}{\partial w_2} = (y_t - y) \times \sigma'(o_1) \times \sigma(h_1)$$

$$\frac{\partial\, Error}{\partial w_2} = (y_t - y) \times \sigma(o_1)\big(1 - \sigma(o_1)\big) \times \sigma(h_1)$$

Backpropagation Algorithm: Simple Example Backpropagate

Weight w_1:



$$h_1 = x_1 w_1 + b_1$$

$$\sigma(h_1) = \frac{1}{1 + e^{-h_1}}$$

$$o_1 = w_2 \times \sigma(h_1) + b_2$$

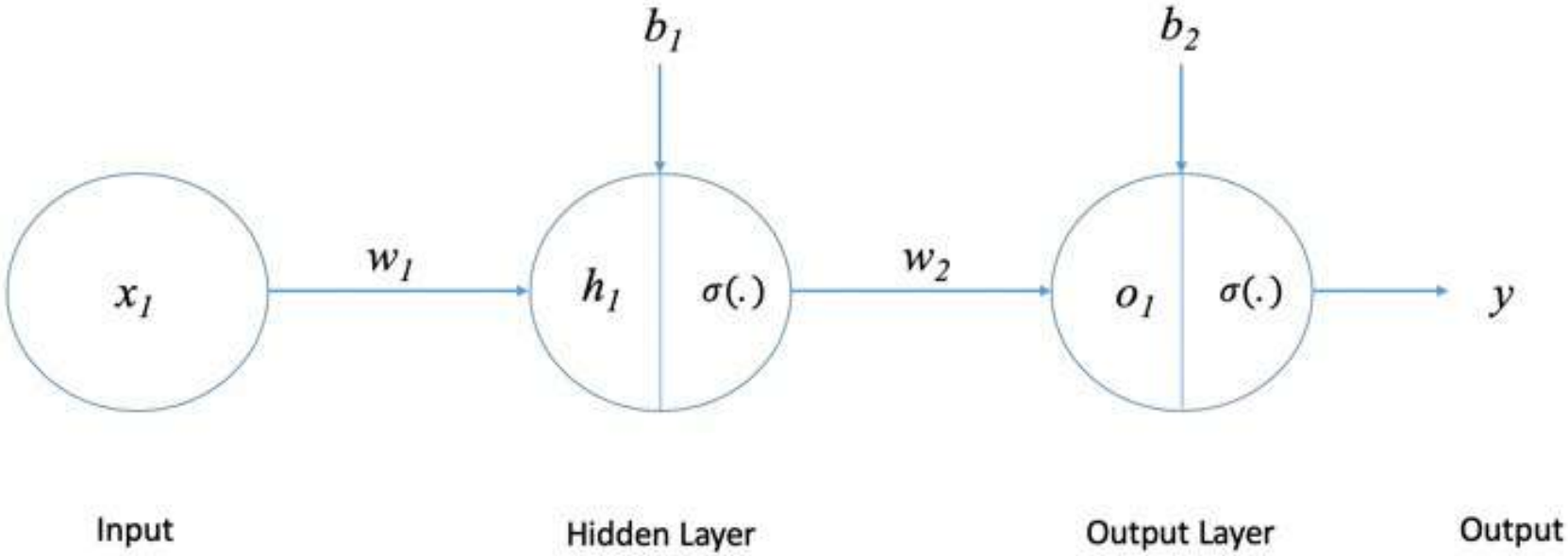
$$y = \sigma(o_1) = \frac{1}{1 + e^{-o_1}}$$

$$\text{Error} = \frac{1}{2}(y_t - y)^2$$

$$\frac{\partial\, Error}{\partial w_1} = \frac{\partial\, Error}{\partial y} \times \frac{\partial y}{\partial w_1} = \frac{\partial\, Error}{\partial y} \times \frac{\partial\, y}{\partial o_1} \times \frac{\partial o_1}{\partial \sigma(h_1)} \times \frac{\partial \sigma(h_1)}{\partial h_1} \times \frac{\partial h_1}{\partial w_1}$$

$$\frac{\partial\, Error}{\partial w_1} = (y_t - y) \times \sigma(o_1)\big(1 - \sigma(o_1)\big) \times w_2 \times \sigma(h_1)\big(1 - \sigma(h_1)\big) \times x_1$$

Simple Backpropagation: Update Weights



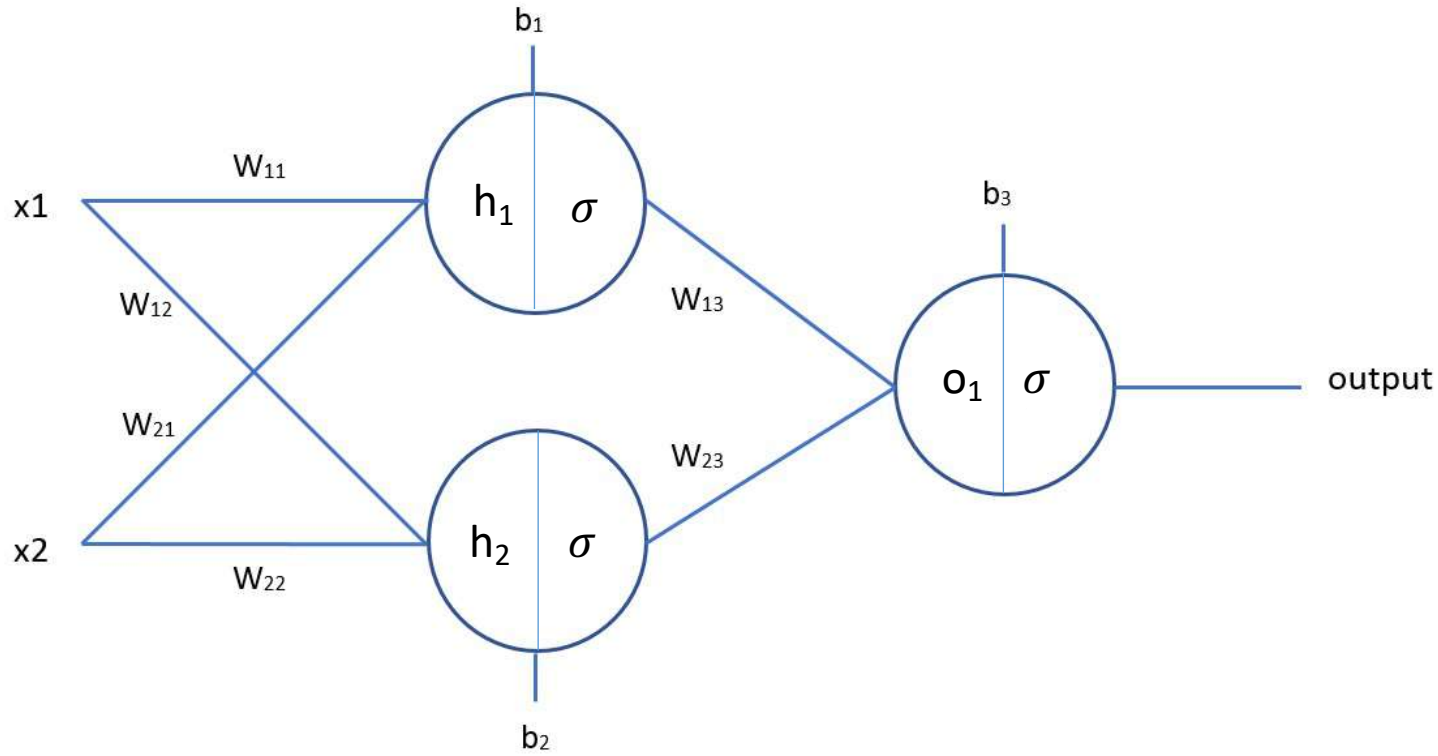
$$w_1 = w_1 - \eta \times \frac{\partial\, Error}{\partial w_1} \qquad w_2 = w_2 - \eta \times \frac{\partial\, Error}{\partial w_2}$$

η is the learning rate.

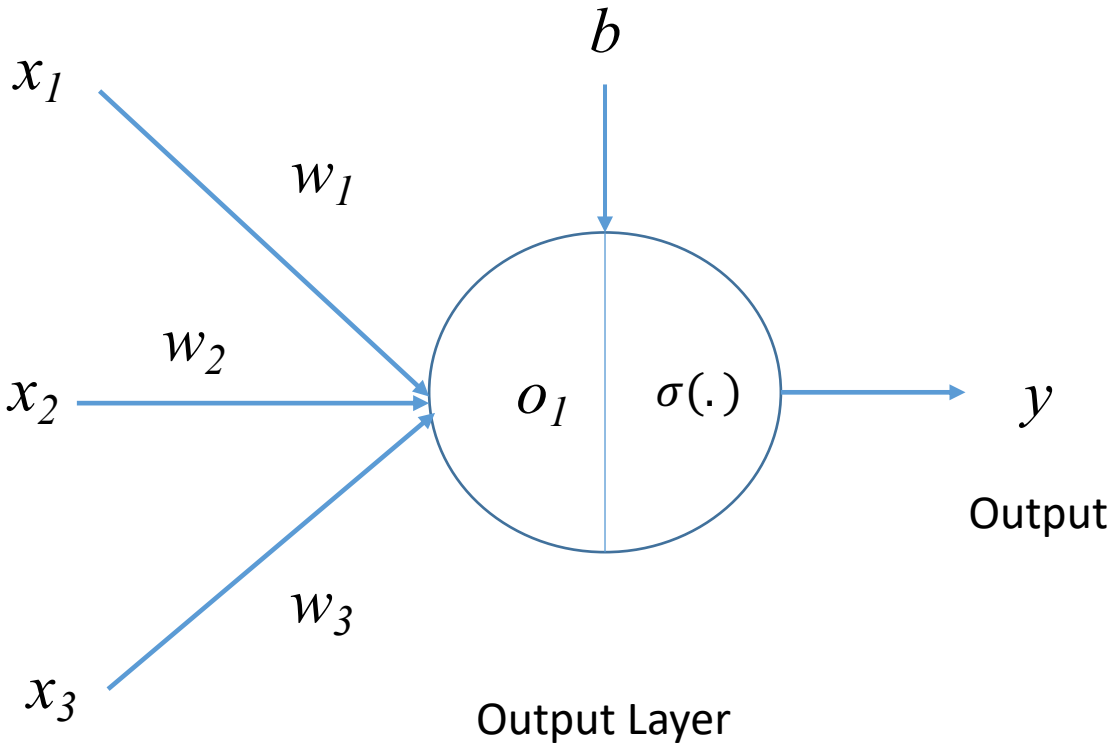
Backpropagation Algorithm: Python Exercise

Given w_{11}=0.11, w_{12}=0.12, w_{21}=0.21, w_{22}=0.08, w_{13}=0.14, w_{23}=0.15, b_1= b_2= b_3 = -1, x_1=0, x_2=1, and y_t=1,

Use backpropagation to update the weights after one forward and one reverse pass.



Backpropagation of Boston Housing Data: Program 20a



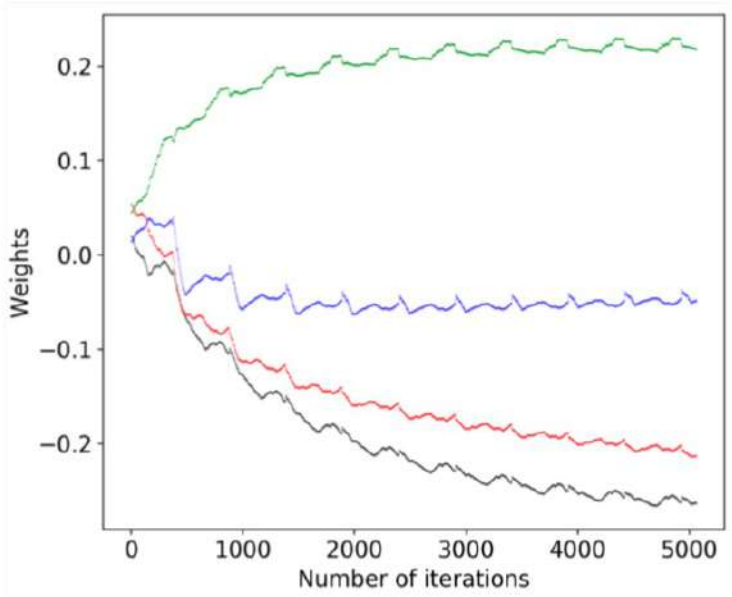


Figure 20.7: [Python] Updates of the four weights (including the bias) against the number of iterations.

QUESTION: How does η affect the results?

Boston Housing Data with a Hidden Layer

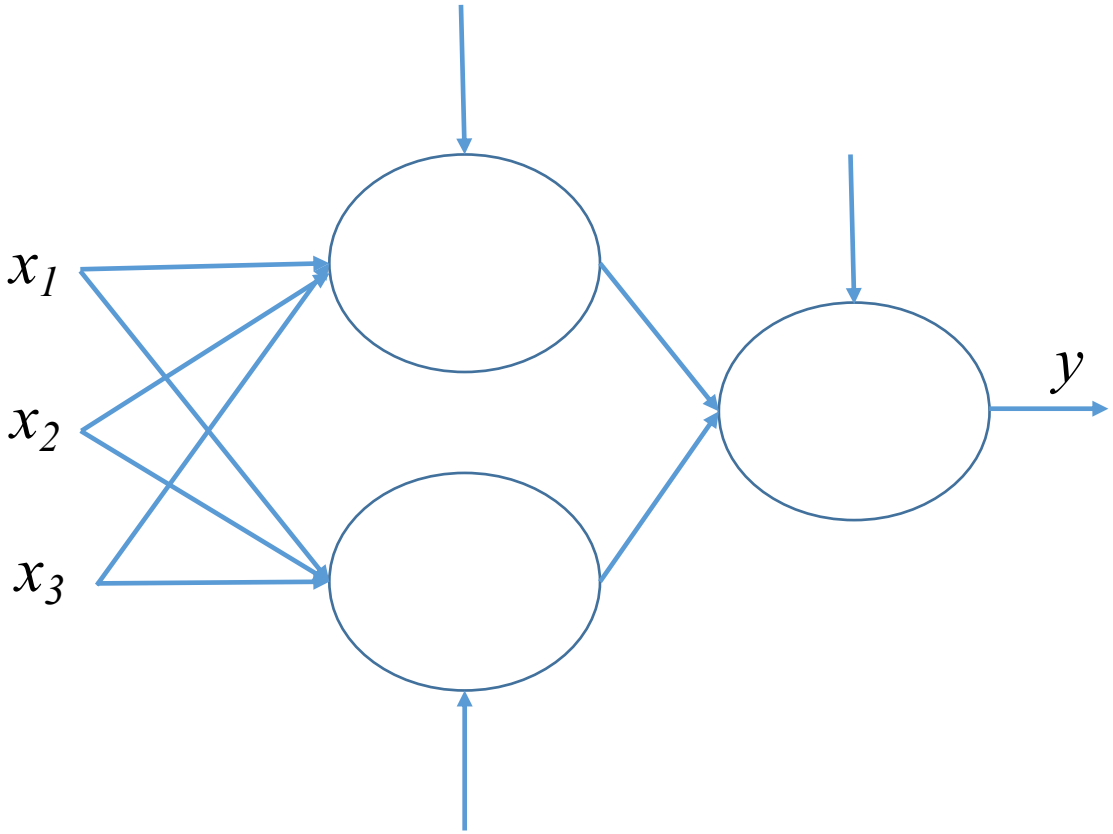
```
# Boston Housing with a Hidden Layer
# Imports
import numpy as np
import matplotlib.pyplot as plt
# Import keras so that we can access the Boston housing data
from tensorflow import keras

## Parameters
num_epochs = 5000
max_num_hidden = 2
# base eta value
eta = 0.05

## Functions
def UniformRandomMatrix (rows, cols):
    res = [[np.random.uniform () for c in range (cols)] for r in range (rows)]
    return np.matrix (res)

## Load the data
dataset = keras.datasets.boston_housing
(train_X, train_y), (_,_) = dataset.load_data (test_split = 0)
train_X = np.matrix (train_X)
train_y = np.matrix (train_y)
(num_samples, num_inputs) = train_X.shape
# Add bias
bias = np.ones (num_samples)
bias = np.matrix (bias).transpose ()
train_X = np.append (train_X, bias, axis=1)
```

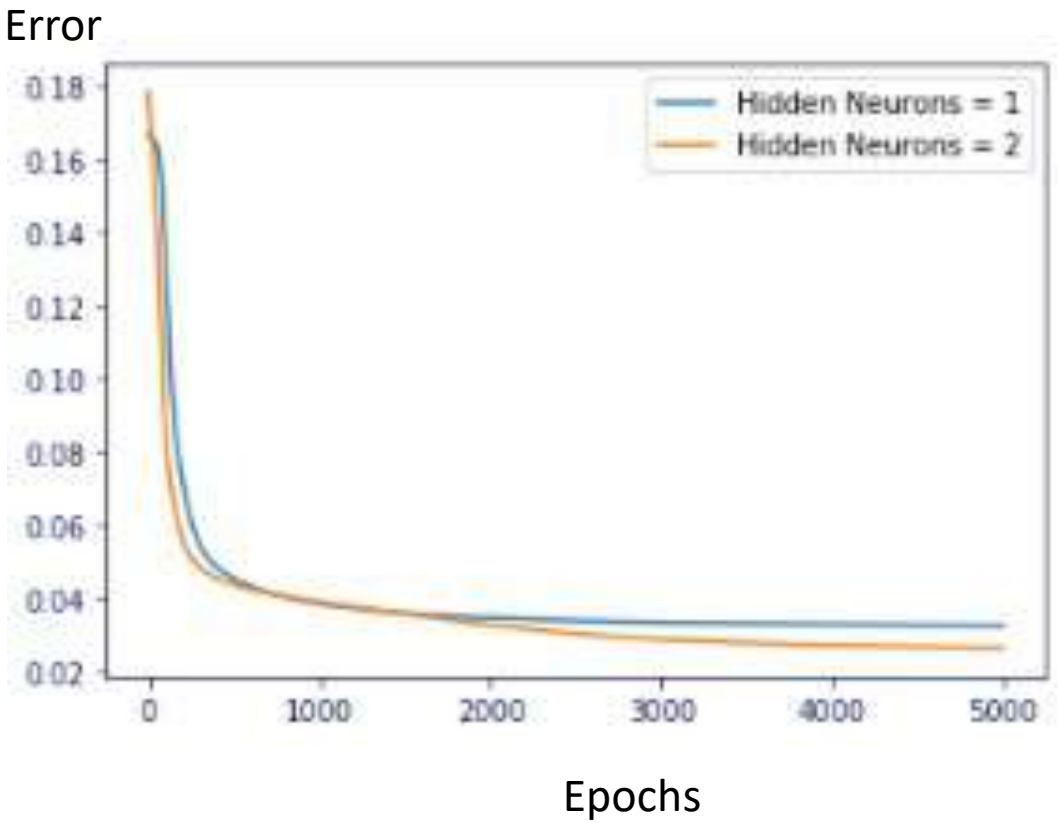
Boston Housing Data with a Hidden Layer

```
## Normalize data
for i in range (num_inputs):
    col = train_X[:,i]
    train_X[:,i] = (col - col.mean()) / col.std()
miny = train_y.min ()
maxy = train_y.max ()
mean = (maxy + miny)/2
std  = (maxy - miny)/2
train_y = (train_y - mean)/std
# Adjust for bias column
num_inputs += 1
# Adjust for sample size
eta /= num_samples

## Test various hidden node counts
for num_hidden in range (1, max_num_hidden + 1):

    ## Initialise weights
    np.random.seed (123456)
    w_hidden = 0.1*UniformRandomMatrix (num_inputs, num_hidden)
    w_output = 0.1*UniformRandomMatrix (num_hidden+1, 1)
```

```
    ## Iterate
    mse = []
    for _ in range (num_epochs):
        # Outputs
        phi = np.append (bias, np.tanh (train_X*w_hidden), axis=1)
        y = phi * w_output
        err = y - train_y.transpose ()
        # Gradients
        g_output = phi.transpose() * err
        phi_range = np.array (phi [:, range (1, num_hidden+1)])
        w_output_range = w_output [range (1, num_hidden+1), 0].transpose()
        err_term = np.array (err*w_output_range)
        g_hidden = train_X.transpose() * np.matrix((1 - phi_range**2)*err_term)
        # Update weights
        w_output -= eta * g_output
        w_hidden -= eta * g_hidden
        mse.append (err.var ())

    ## Plot
    plt.plot (range (num_epochs), mse, label="Hidden Neurons = {0}".format (num_hidden))

plt.legend ()
plt.show ()
```

Backpropagation: Two Neurons in Hidden Layer





End Day 4 Summary

Day 4			
Topics	**Hours**	**Topics**	**Hours**
AI: Introduction to Image Processing	10am-11am	AI: Artificial Intelligence	1pm-2pm
AI: Binary Oscillator Computing	11am-12pm	AI: The Backpropagation Algorithm	2pm-3pm