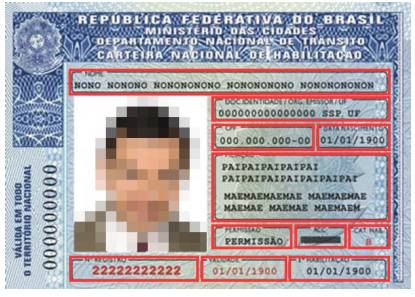
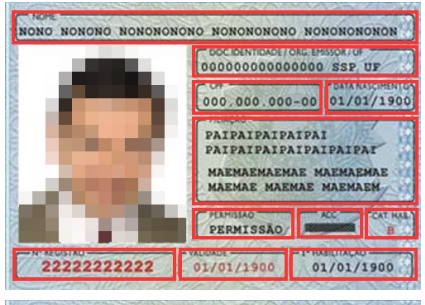
OPTICAL CHARACTER RECOGNITION (OCR) IN PYTHON

COURSE CONTENT

- OCR with Tesseract
- Techniques for pre-processing images
- OCR with EAST
- Training an OCR from scratch
  - Artificial Neural Networks and Convolutional Neural Networks
- EasyOCR for natural scenarios
- OCR in videos
- Projects
  - Searching for specific terms
  - Scanner + OCR
  - License plate reading

TEXT DETECTION – CONTROLLED SCENARIOS





Artificial intelligence and computer vision

Source: https://www.pyimagesearch.com/2017/07/17/credit-card-ocr-with-opencv-and-python/

EXPERT ACADEMY

TEXT DETECTION – CONTROLLED SCENARIOS

TEXT DETECTION – NATURAL SCENARIOS

Source: Mancas-Thillou e Gosselin

- Lighting conditions
- Blur
- Resolution
- Viewing angle
- Non-planar objects
- Objects that are not paper
- Noise in the image
- Unknown layout
- Slanted text
- Different letters

What to do?

Source: https://www.rsipvision.com/real-time-ocr/

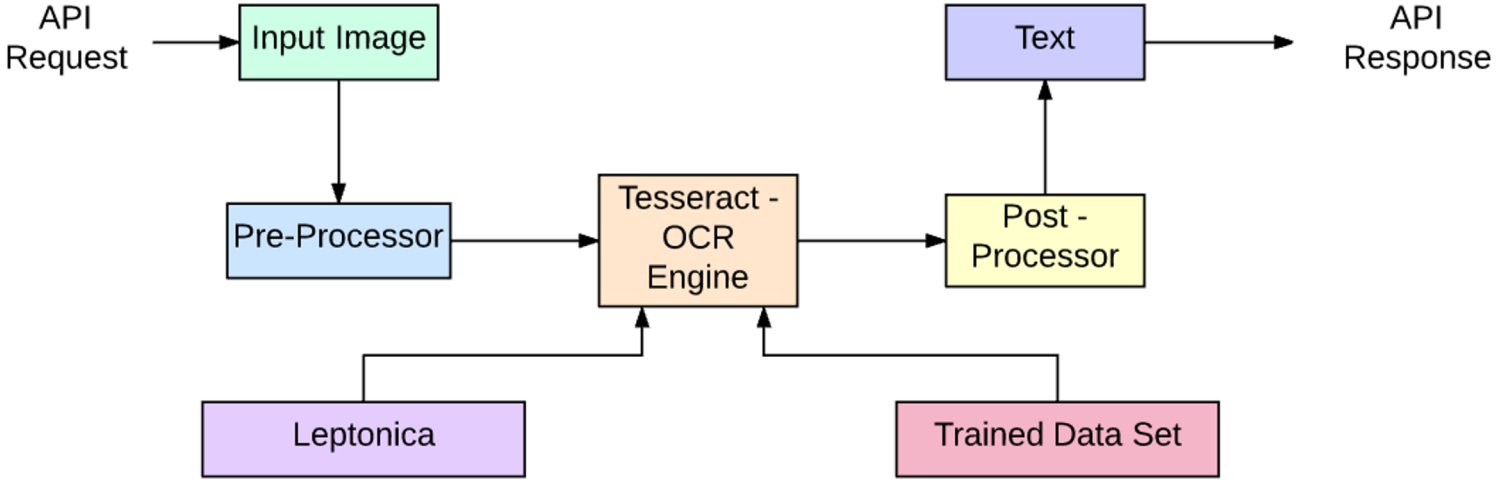
TESSERACT

- It is an OCR engine (*Optical Character Recognition*)
- Originally started as a PhD project at Hewlett-Packard (HP) laboratories – scanner
- After gaining popularity, it was developed by HP in 1984. The team worked until 1994
- It was ported to Windows in 1996
- In 2005 it was released to the community as an open source project
- In 2006 its development began to be sponsored by Google
- Since 2006, it is considered to be one of the best and most popular OCR tools
- Oficial repository: github.com/tesseract-ocr/tesseract

TESSERACT

- The **first** version of Tesseract only supported English
- The **second** version supported Brazilian Portuguese, French, Italian, German and Dutch
- The **third** version dramatically expanded support to include ideographic (symbolic) languages such as Japanese and Chinese, as well as right-to-left writing languages such as Arabic and Hebrew.
- The **fourth** (June 2021) supports over 100 languages for characters and symbols
- Pytesseract: https://pypi.org/project/pytesseract/

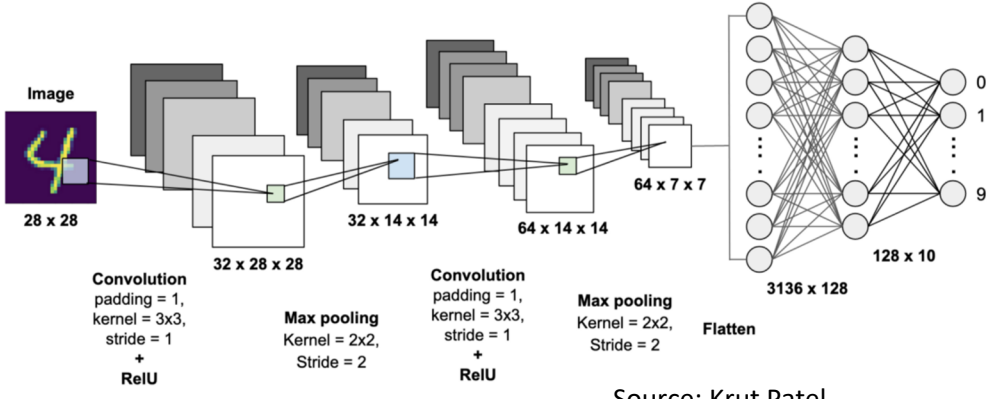
TESSERACT



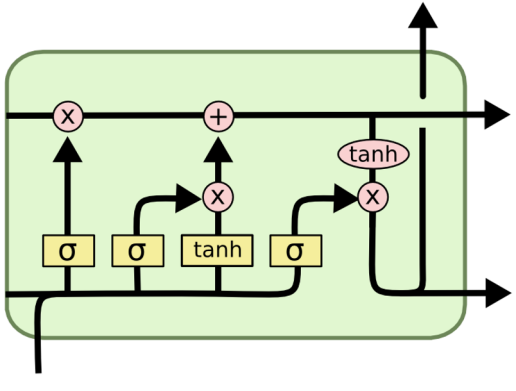
Source: Balaaji Parthasarathy

TESSERACT

To recognize an image containing a single character, we normally use a Convolutional Neural Network (CNN)



Source: Krut Patel

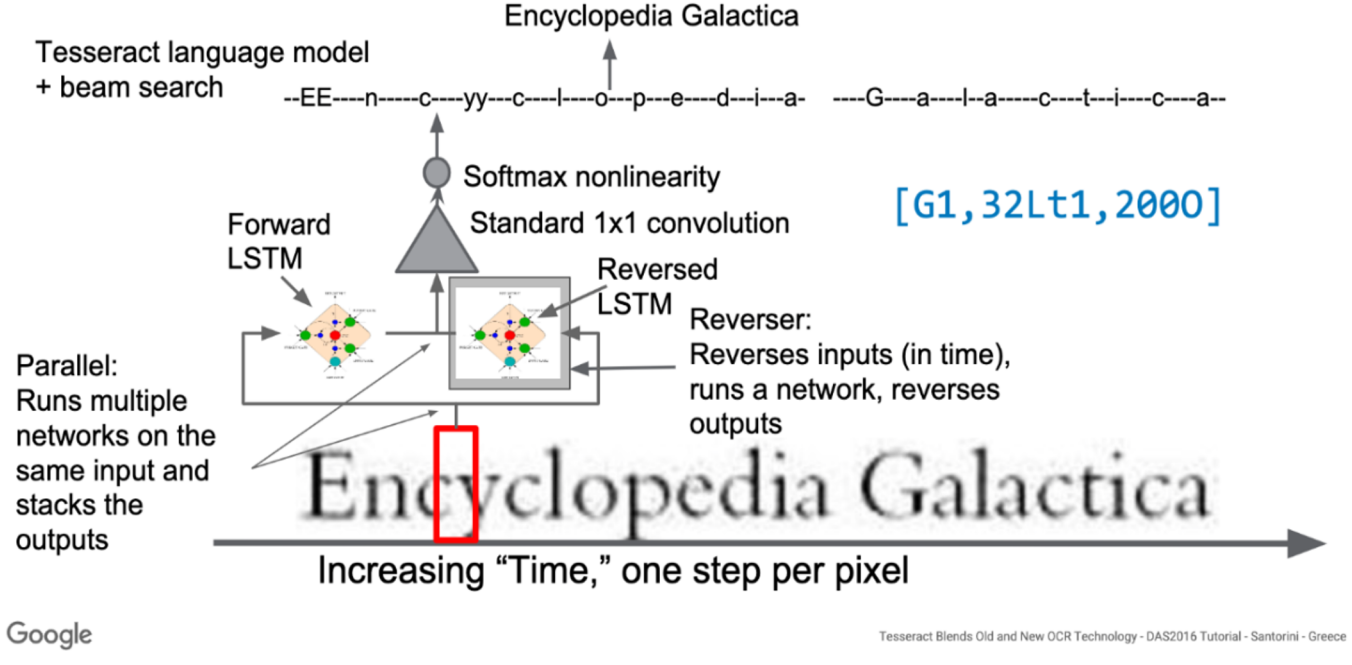


LSTM architecture

Source: Colah's blog

A text of arbitrary length is a sequence of characters and it is more interesting to use RNNs (Recurrent Neural Networks). LSTM (Long short-term memory) is a popular form of RNN.

TESSERACT



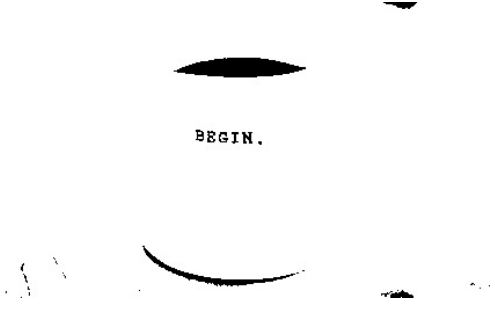
Source: Tesseract 3 OCR process paper

TESSERACT

- Tesseract has a function to detect the orientation of the text in the image, as well as the language it is written in
- This option is called OSD - Orientation and script detection
- Detect if the text in the image is rotated
- If it's rotated, we can apply some kind of preprocessing

THRESHOLDING

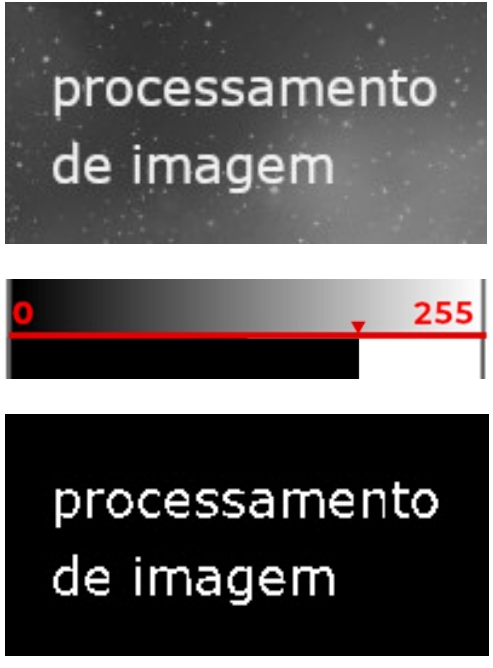
- Thresholding (binarization) is the simplest method of image segmentation
- It consists of separating an image into regions of interest and non-interest by choosing a cut-off point (called threshold)



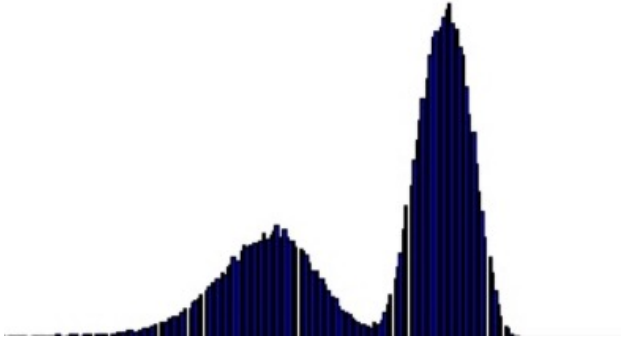
SIMPLE THRESHOLDING

The value of the new color that the pixel will have is calculated according to the cutoff point (threshold)

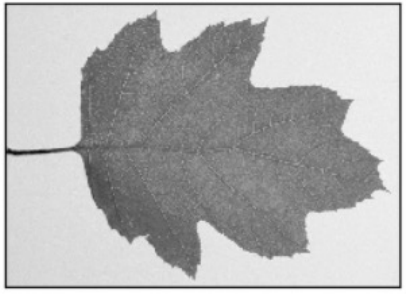
Any pixel with intensity less than or equal to the threshold becomes black. If the pixel has an intensity greater than the threshold, it becomes white.

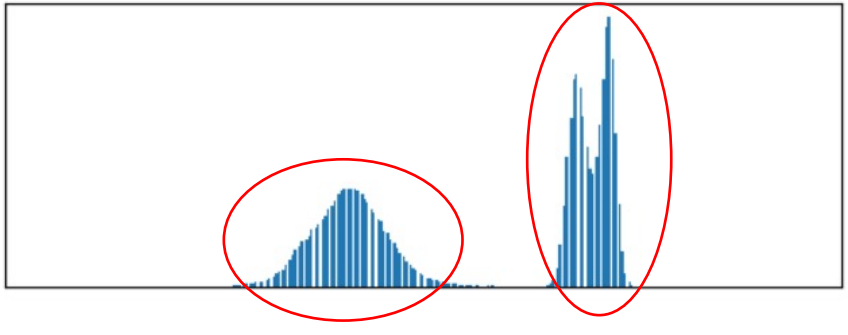


OTSU METHOD



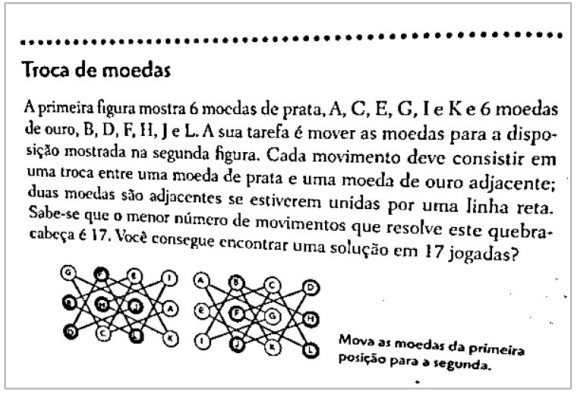
Example of a histogram of a bimodal image





ADAPTIVE THRESHOLDING (GAUSSIAN)

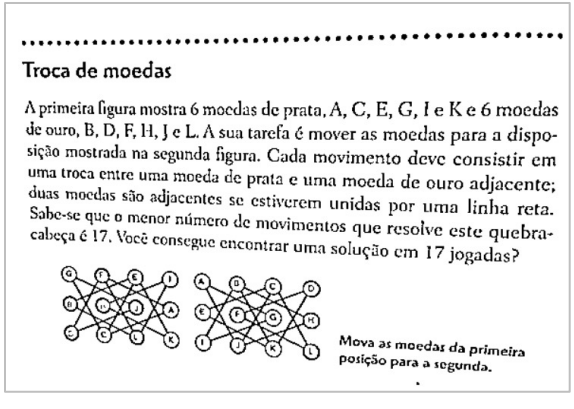
- The threshold is calculated for small regions of the image, different thresholds are obtained for small regions of the image (using the average)
- Gaussian: also uses standard deviation (considering the variation in the pixels) – uses convolution



Troca de moedas

A primeira figura mostra 6 moedas de prata, A, C, E, G, I e K e 6 moedas de ouro, B, D, F, H, J e L. A sua tarefa é mover as moedas para a disposição mostrada na segunda figura. Cada movimento deve consistir em uma troca entre uma moeda de prata e uma moeda de ouro adjacente; duas moedas são adjacentes se estiverem unidas por uma linha reta. Sabe-se que o menor número de movimentos que resolve este quebra-cabeça é 17. Você consegue encontrar uma solução em 17 jogadas?

Mova as moedas da primeira posição para a segunda.



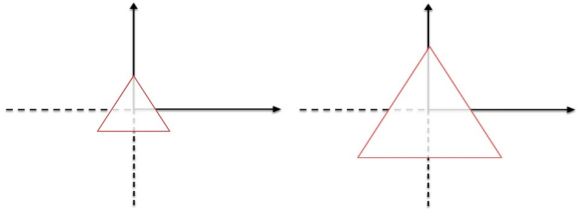
Troca de moedas

A primeira figura mostra 6 moedas de prata, A, C, E, G, I e K e 6 moedas de ouro, B, D, F, H, J e L. A sua tarefa é mover as moedas para a disposição mostrada na segunda figura. Cada movimento deve consistir em uma troca entre uma moeda de prata e uma moeda de ouro adjacente; duas moedas são adjacentes se estiverem unidas por uma linha reta. Sabe-se que o menor número de movimentos que resolve este quebra-cabeça é 17. Você consegue encontrar uma solução em 17 jogadas?

Mova as moedas da primeira posição para a segunda.

RESIZING

$$\begin{bmatrix} x_T \\ y_T \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

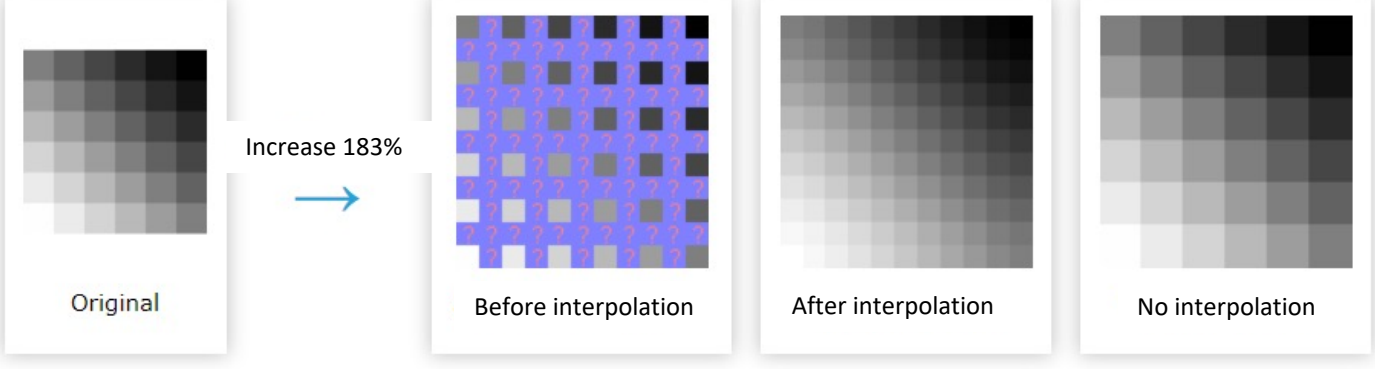


Scale factor:

>1 increase

<1 decrease

sx = sy uniform scale factor (no distortion)



RESIZING

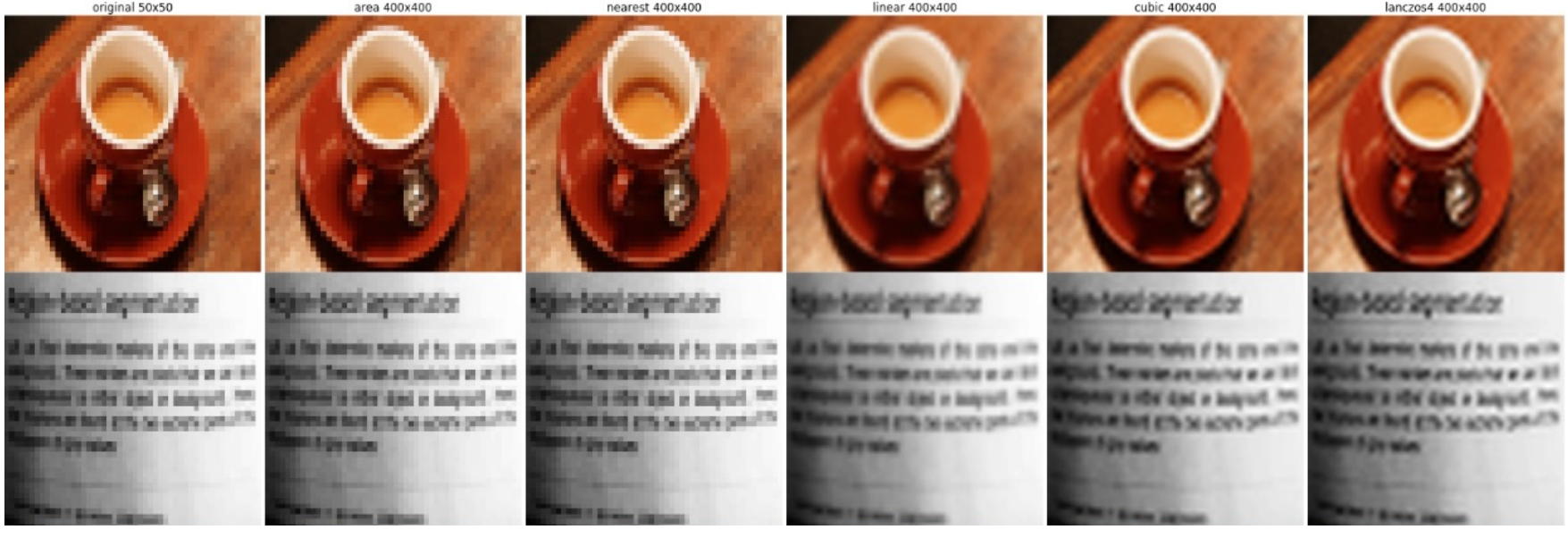
OpenCV options

- **INTER_NEAREST** - a nearest neighbor interpolation. It is widely used because it is the fastest
- **INTER_LINEAR** - a bilinear interpolation (it's used by default), generally good for zooming in and out of images
- **INTER_AREA** - uses the pixel area ratio. May be a preferred method for image reduction as it provides good results
- **INTER_CUBIC** - bicubic (4x4 neighboring pixels). It has better results
- **INTER_LANCZOS4** - Lanczos interpolation (8x8 neighboring pixels). Among these algorithms, it is the one with the best quality results

RESIZING

Comparison between interpolation algorithms – increasing the size

original (50x50) area (400x400) nearest (400x400) linear (400x400) cubic (400x400) lanczos4 (400x400)

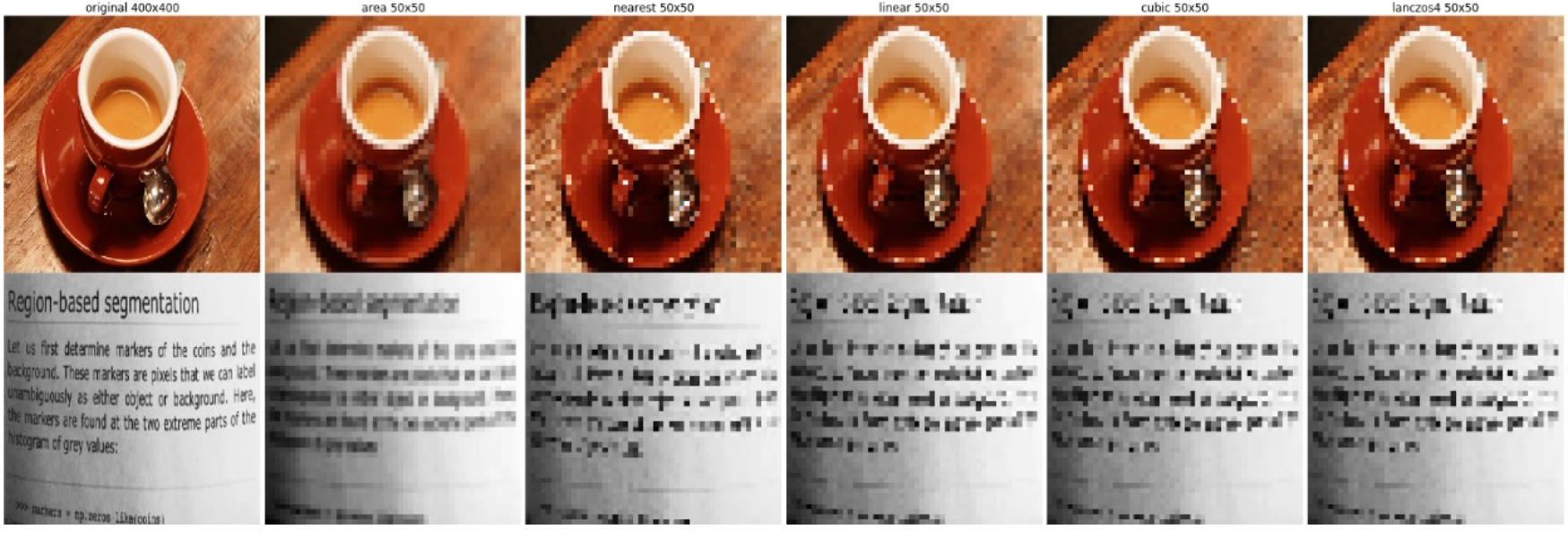


http://tanbakuchi.com/posts/comparison-of-openv-interpolation-algorithms/

RESIZING

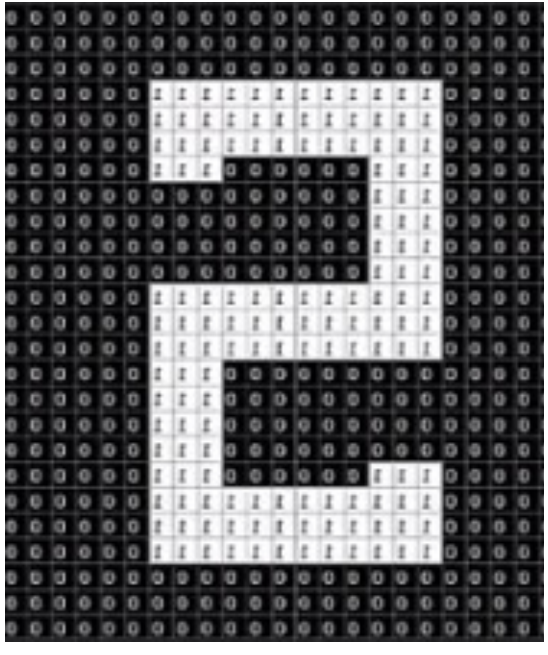
Comparison between interpolation algorithms – decreasing the size

original (400x400) area (50x50) nearest (50x50) linear (50x50) cubic (50x50) lanczos4 (50x50)



http://tanbakuchi.com/posts/comparison-of-openv-interpolation-algorithms/

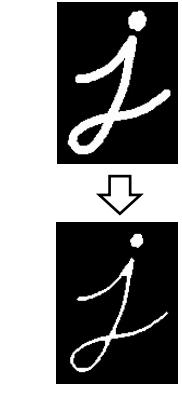
MORPHOLOGICAL OPERATIONS – EROSION AND DILATION



0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	0	1	1	1	0	0
0	1	1	1	1	0	0
0	1	1	1	1	0	0
0	1	1	1	1	0	0
0	0	0	0	0	0	0

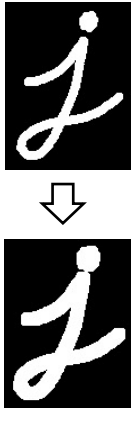
EROSION

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

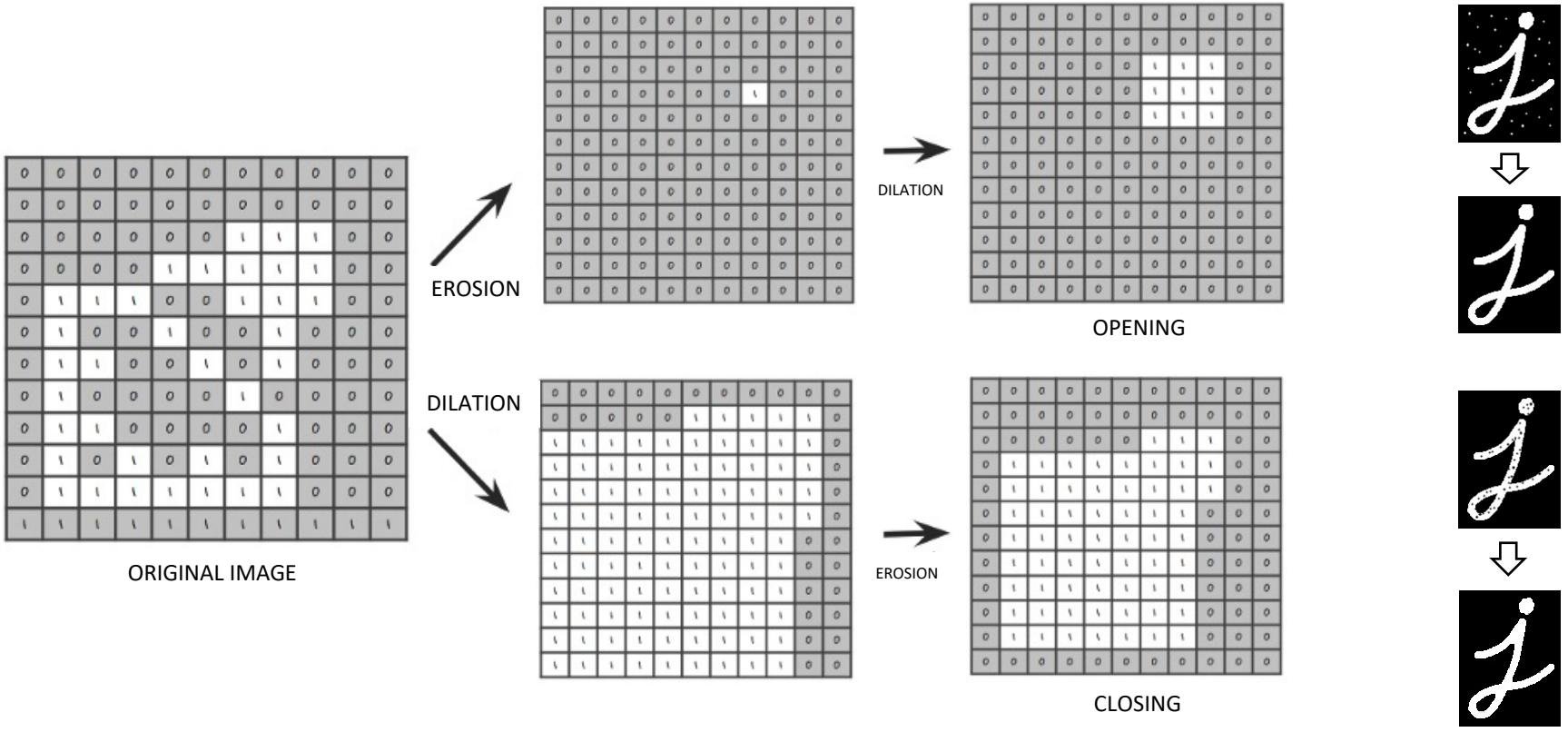


DILATION

0	0	0	0	0	0	0
0	1	1	1	1	1	0
1	1	1	1	1	1	0
1	1	1	1	1	1	0
1	1	1	1	1	1	0
1	1	1	1	1	1	0
1	1	1	1	1	1	0



MORPHOLOGICAL OPERATIONS – OPENING AND CLOSING



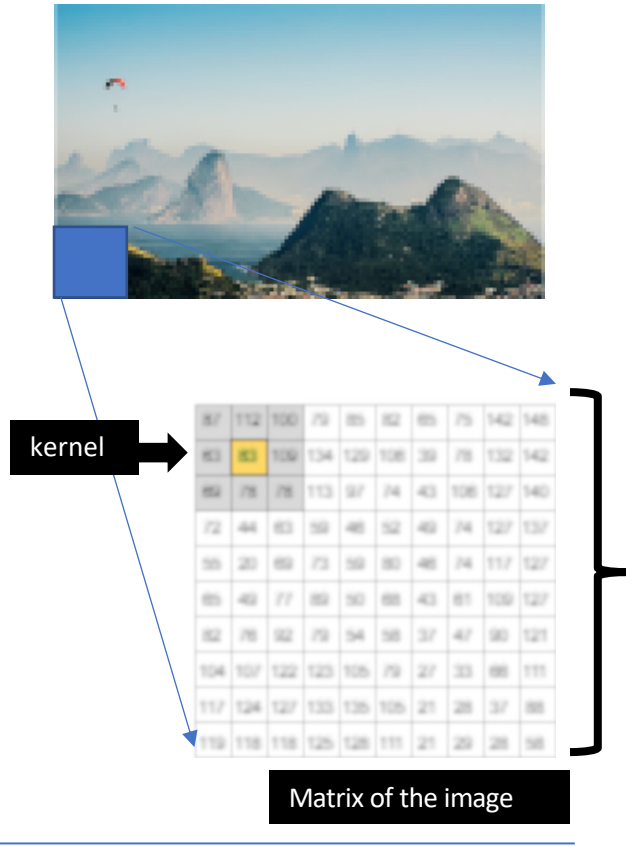
NOISE REMOVAL WITH BLUR

Spatial filters are implemented through kernels (masks/arrays), with odd dimensions. Filters can be::

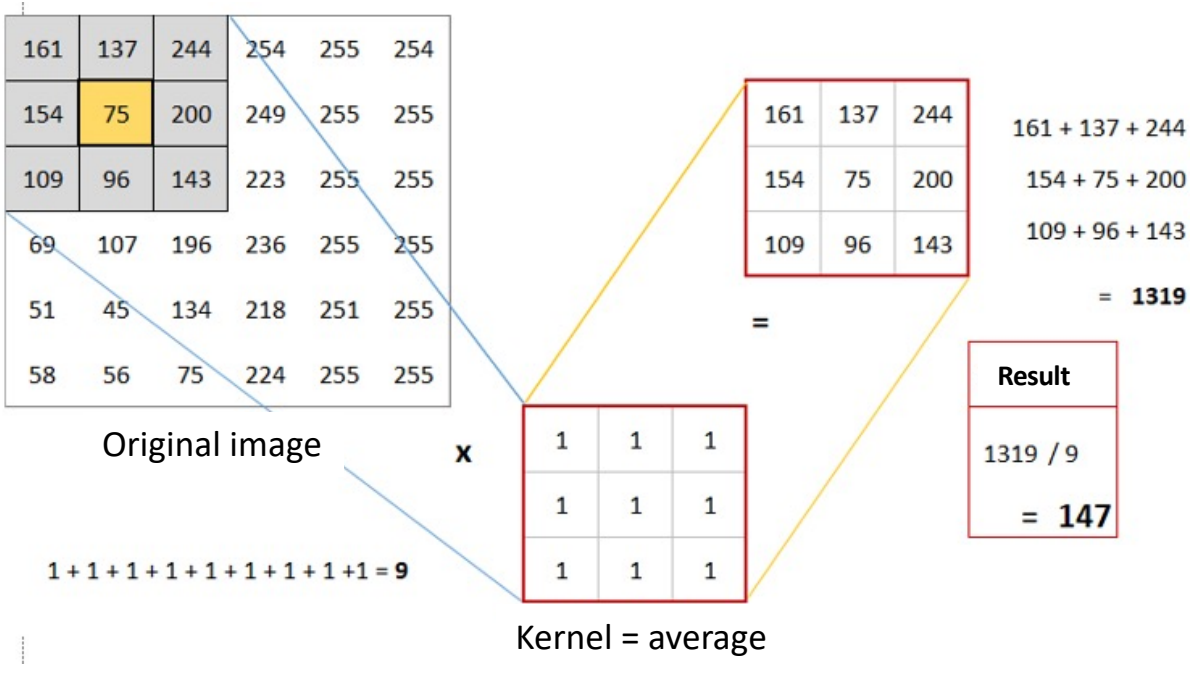
- Low-pass – used for blur
- High-pass – used to sharpen

CONVOLUTION

- Convolution is a mathematical operation performed on two matrices, which produces a third matrix that is the result of the operation
- The primary matrix is the image to be treated, and the treatment of this matrix of the original image is done by another matrix called “kernel”, or mask
- Depending on the kernel values it is possible to obtain filters of different types, such as blur and sharpen

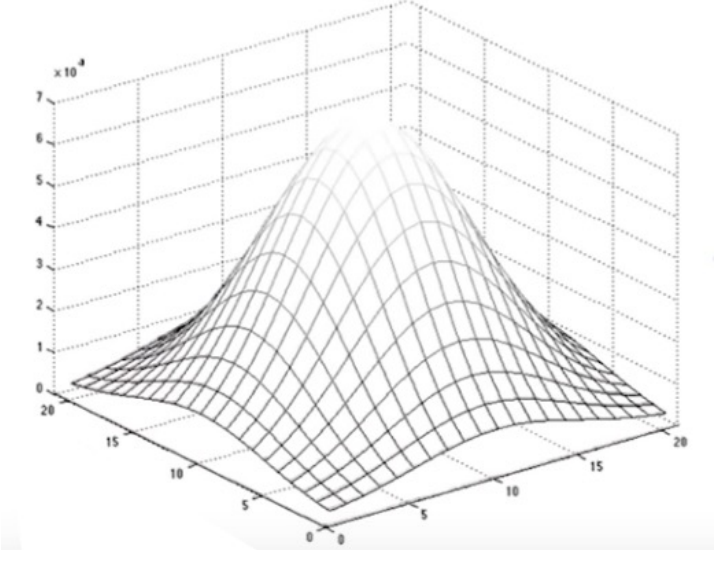


BLUR WITH AVERAGE

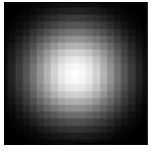


$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

GAUSSIAN BLUR

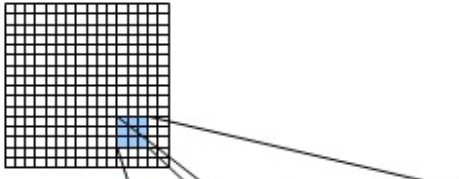


$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



Graphic representation of a 21x21 Gaussian filter

BLUR WITH MEDIAN



101	69	0
56	255	87
123	96	157

The pixels below the kernel are sorted and the middle value is chosen to be the new value of the central (main) pixel

0	56	69	87	96	101	123	157	255

2, 2, 3, **7**, 8, 9, 9

Median = 7

1, 4, 4, **5**, **6**, 7, 7, 7

Median = (5+6) ÷ 2

= 5.5

BILATERAL FILTER

TEXT DETECTION

Text **detection** before **recognition** is an essential step


```
'conf': ['-1', '-1', '-1', '-1', 90, '-1', 74, 66],
'height': [400, 236, 236, 92, 92, 87, 76, 87],
'left': [0, 38, 38, 38, 38, 102, 102, 307],
'text': ['', '', '', '', 'TESTANDO', '', 'O', 'OCR...'],
'top': [0, 79, 79, 79, 79, 228, 233, 228],
'width': [700, 607, 607, 607, 607, 532, 77, 327],
```

Example of images where there would be no need for prior detection

EAST – TEXT DETECTOR

EAST (*Efficient Accurate Scene Text detector*) is a deep learning model, officially published in 2017 by Zhou et al.

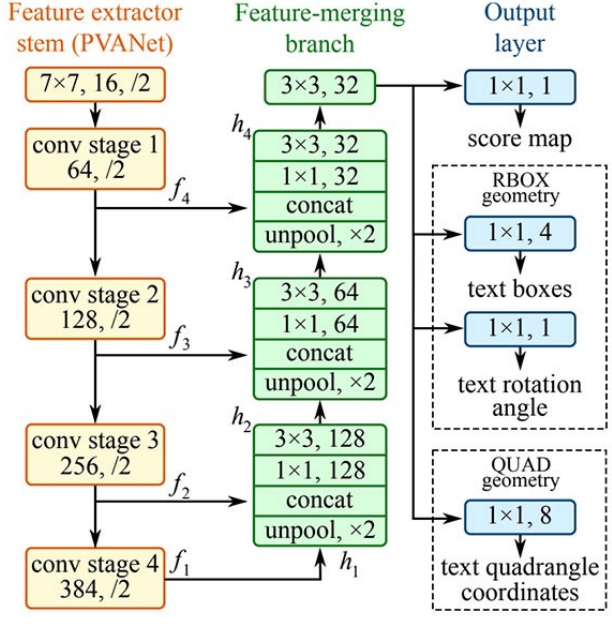
- Uses convolutional layers to extract features from images and thus detect the existence of texts
- It only identifies the **location of the text**. It is not an OCR that will convert it to characters
- It is one of the most accurate text detection techniques
- It is also one of the best known and gained popularity after the release of version 3.4.2 of OpenCV, which made it possible to implement it more easily through the DNN module

EAST – TEXT DETECTOR

Features at different processing levels are treated in a matrix, leading to the identification of bounding boxes where the text appears in the image

- It has only two stages: FCN (Fully Convolutional Network) and NMS (non-maximum suppression) to suppress weak and overlapping bounding boxes

EAST architecture
(Fully-Convolutional Network)



Source: [Zhou et al.]

ADITIONAL READING

EAST Paper
https://arxiv.org/abs/1704.03155v2

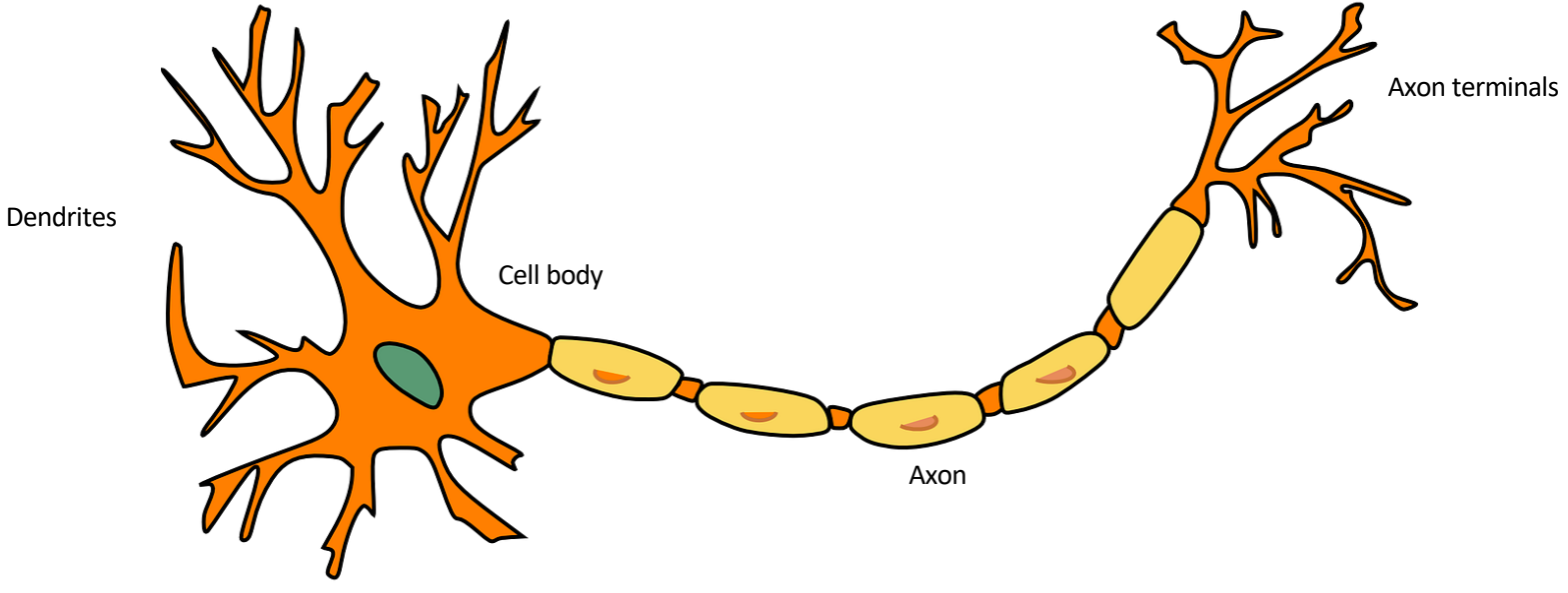
Mancas-Thillou e Gosselin
https://www.tcts.fpms.ac.be/publications/regpapers/2007/VS_cmtbg2007.pdf

Tesseract OCR
https://github.com/tesseract-ocr/tesseract

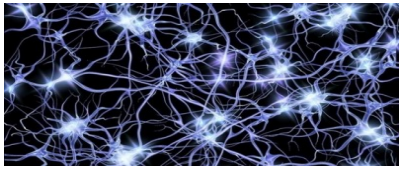
EAST geometry
https://stackoverflow.com/questions/55583306/decoding-geometry-output-of-east-text-detection

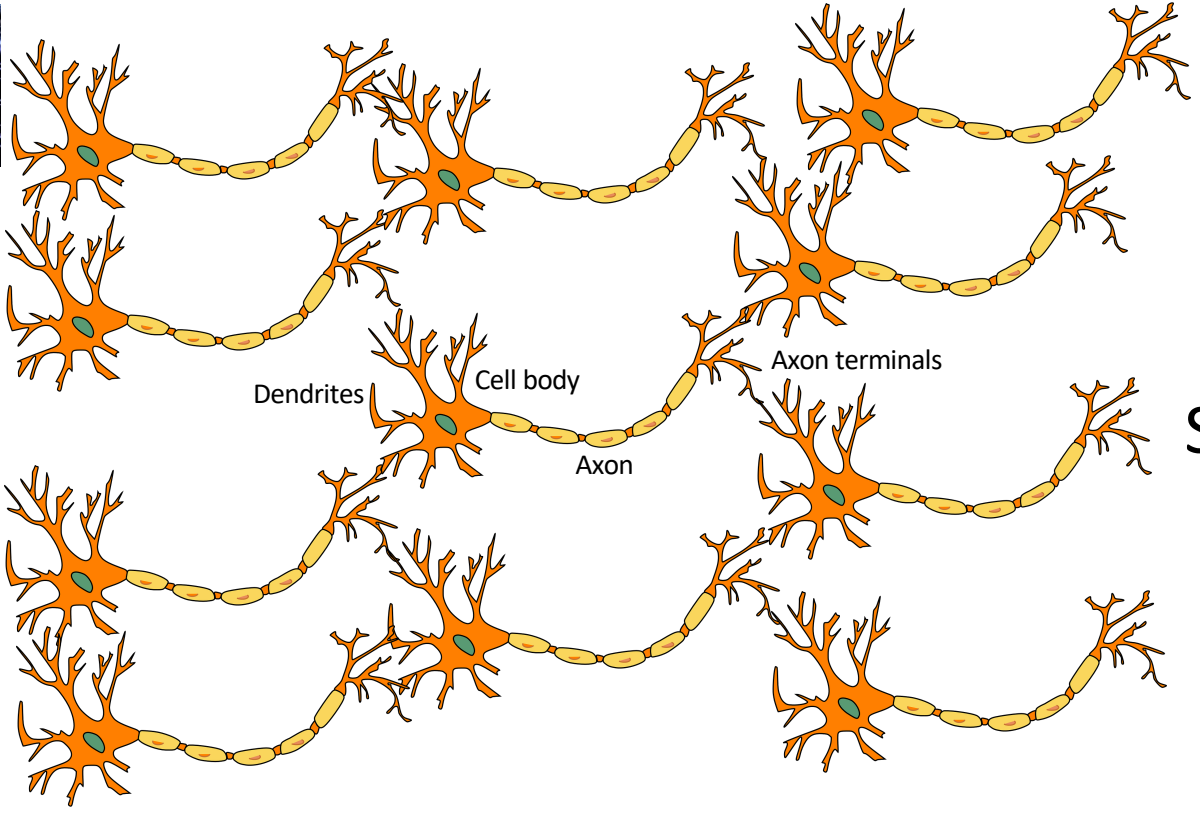
ARTIFICIAL NEURAL NETWORKS

ARTIFICIAL NEURAL NETWORKS



ARTIFICIAL NEURAL NETWORKS

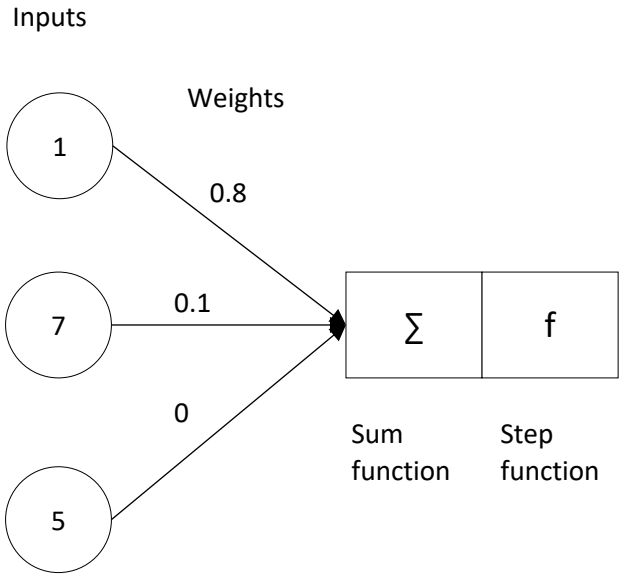




Synapse

ARTIFICIAL NEURON

$$sum = \sum_{i=1}^{n} xi * wi$$



$sum = (1 * 0.8) + (7 * 0.1) + (5 * 0)$

$sum = 0.8 + 0.7 + 0$

$sum = 1.5$

Output = 1

Step function

Greater or equal to 1 = 1
Otherwise = 0

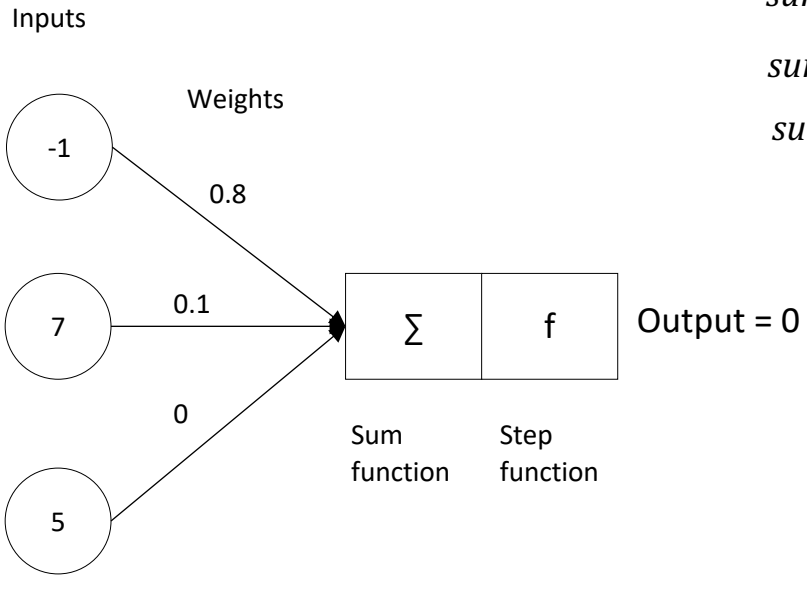
ARTIFICIAL NEURON

$$sum = \sum_{i=1}^{n} xi * wi$$

sum = (-1 * 0.8) + (7 * 0.1) + (5 * 0)

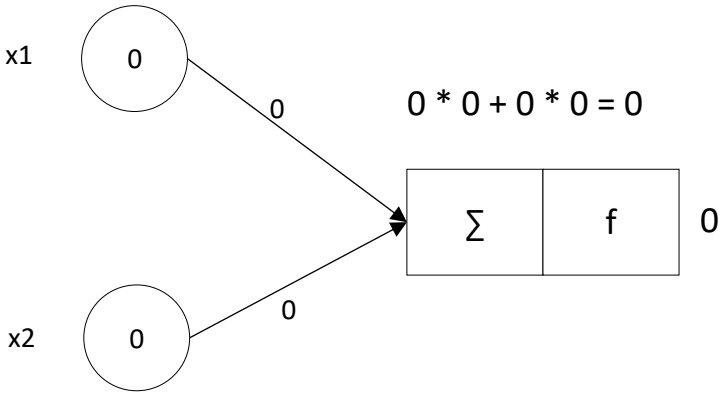
sum = -0.8 + 0.7 + 0

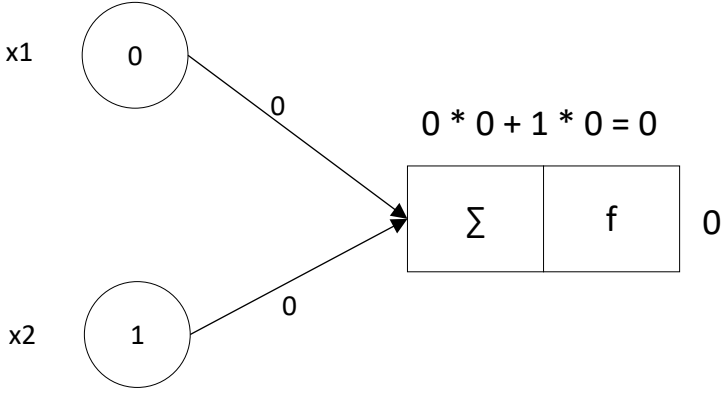
sum = -0.1



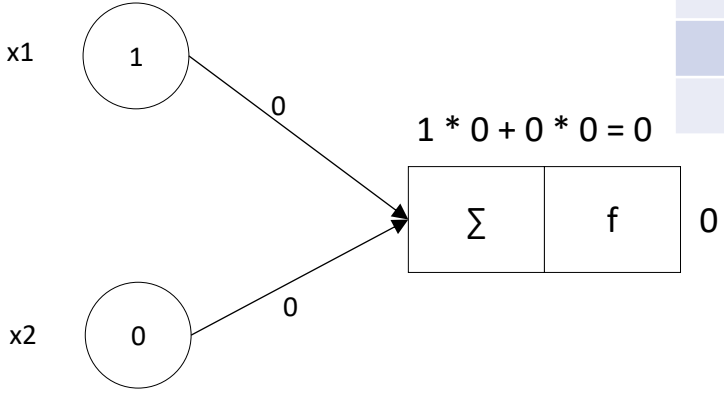
“AND” OPERATOR

X1	X2	Class
0	0	0
0	1	0
1	0	0
1	1	1

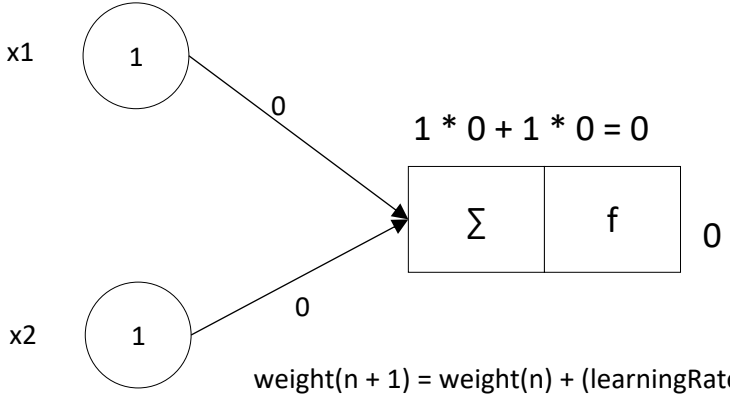




x1	x2	Class
0	0	0
0	1	0
1	0	0
1	1	1



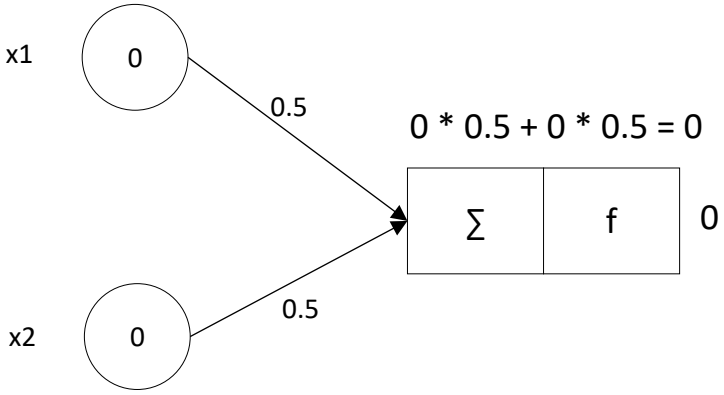
0 – 0 = 0
0 – 0 = 0
0 – 0 = 0
1 – 0 = 1

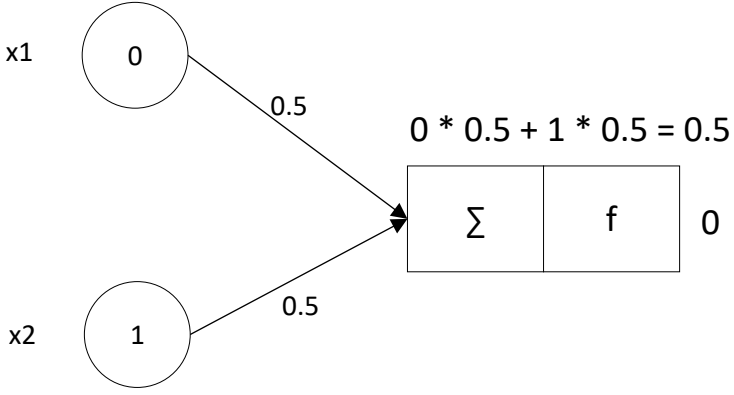


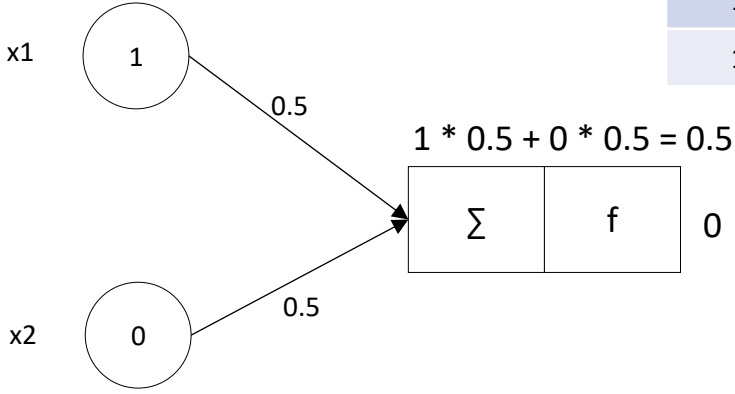
weight(n + 1) = weight(n) + (learningRate * input * error)
weight(n + 1) = 0 + (0.1 * 1 * 1) = 0.1

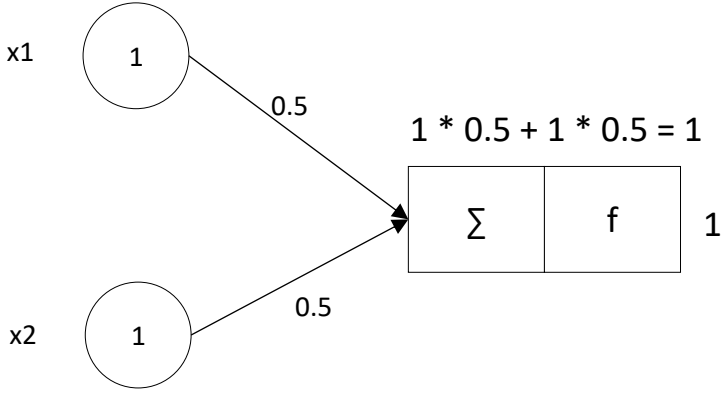
EXPERT ACADEMY

X1	X2	Class
0	0	0
0	1	0
1	0	0
1	1	1



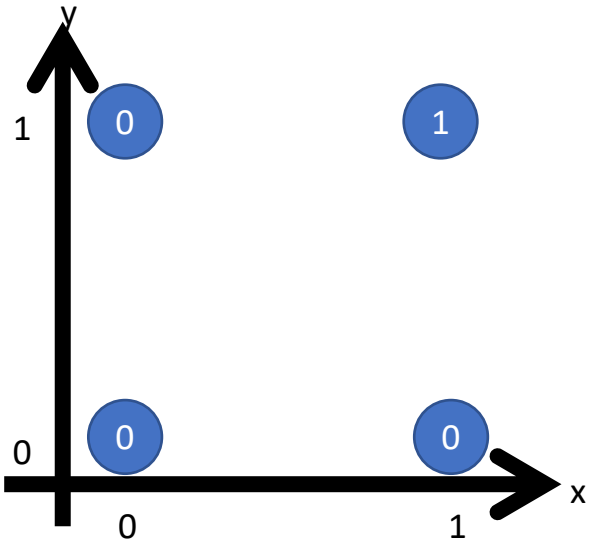


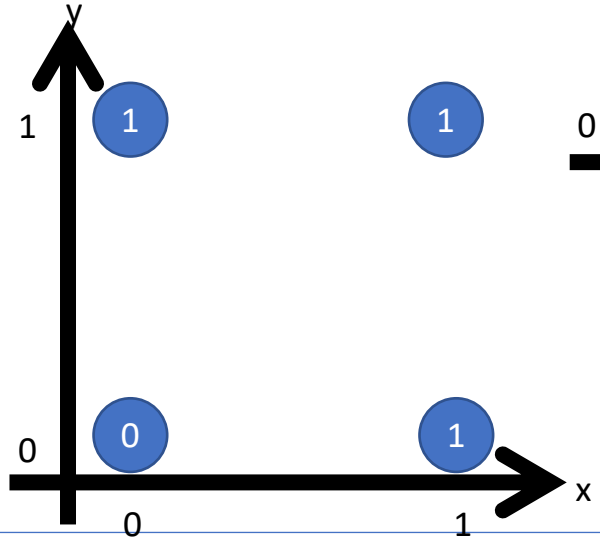


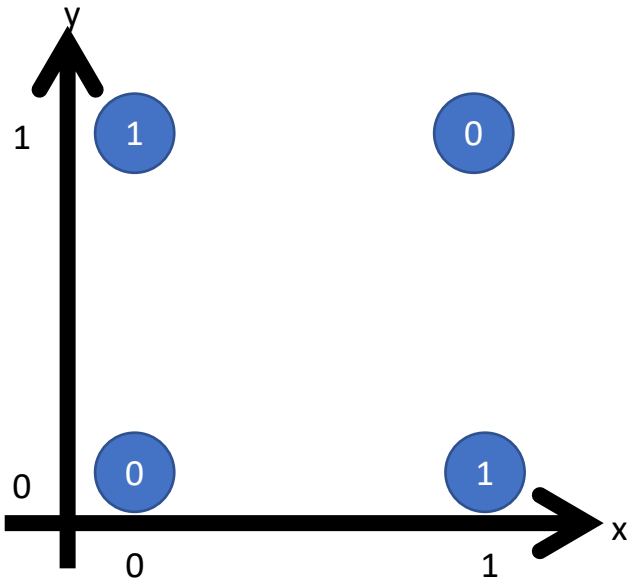


0 – 0 = 0
0 – 0 = 0
0 – 0 = 0
1 – 1 = 0

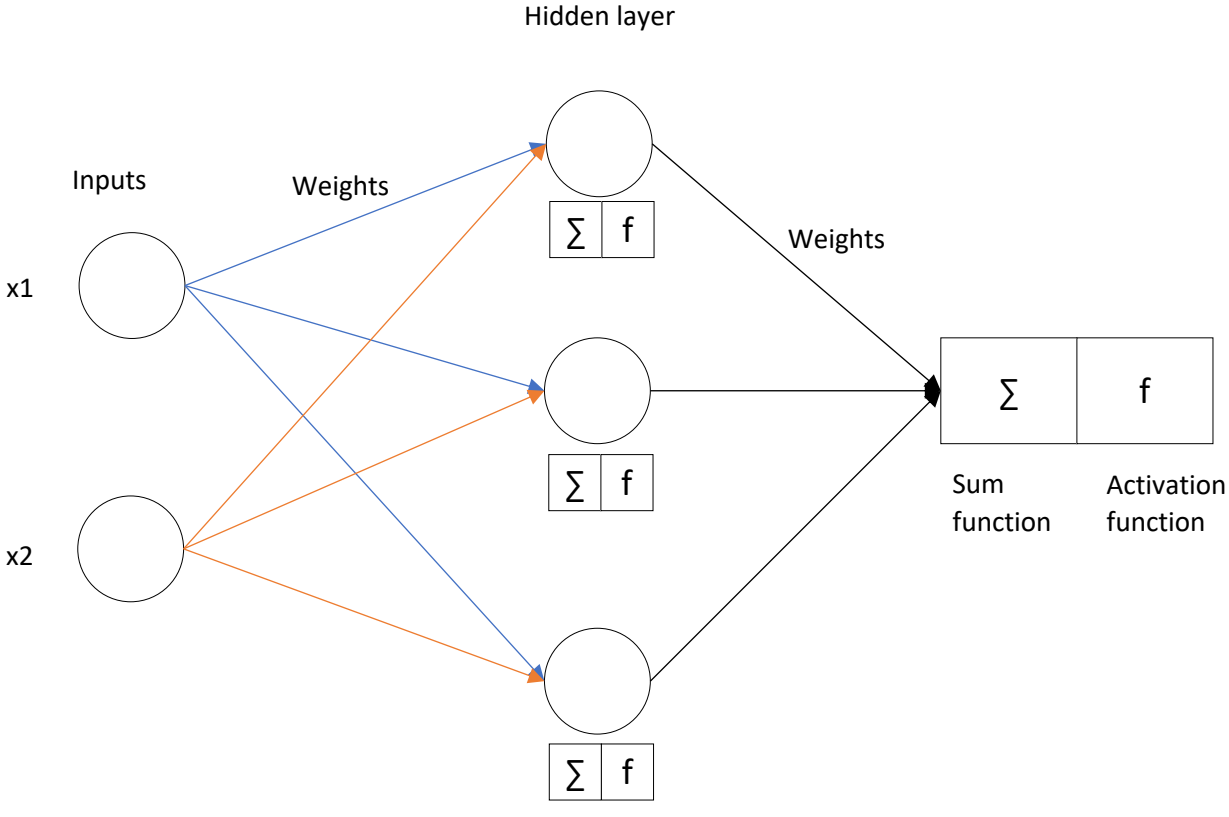
SINGLE LAYER PERCEPTRON





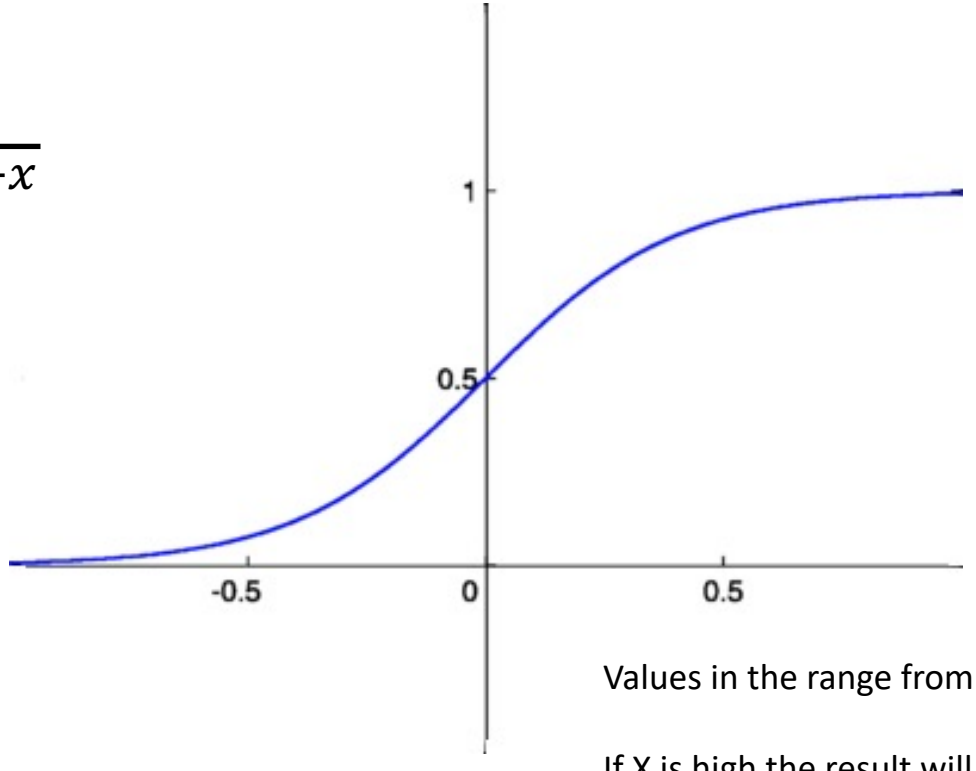


MULTILAYER PERCEPTRON



SIGMOID FUNCTION

$$y = \frac{1}{1 + e^{-x}}$$



Values in the range from 0 to 1

If X is high the result will be approximately 1
If X is low the result will be approximately 0
No negative number is returned

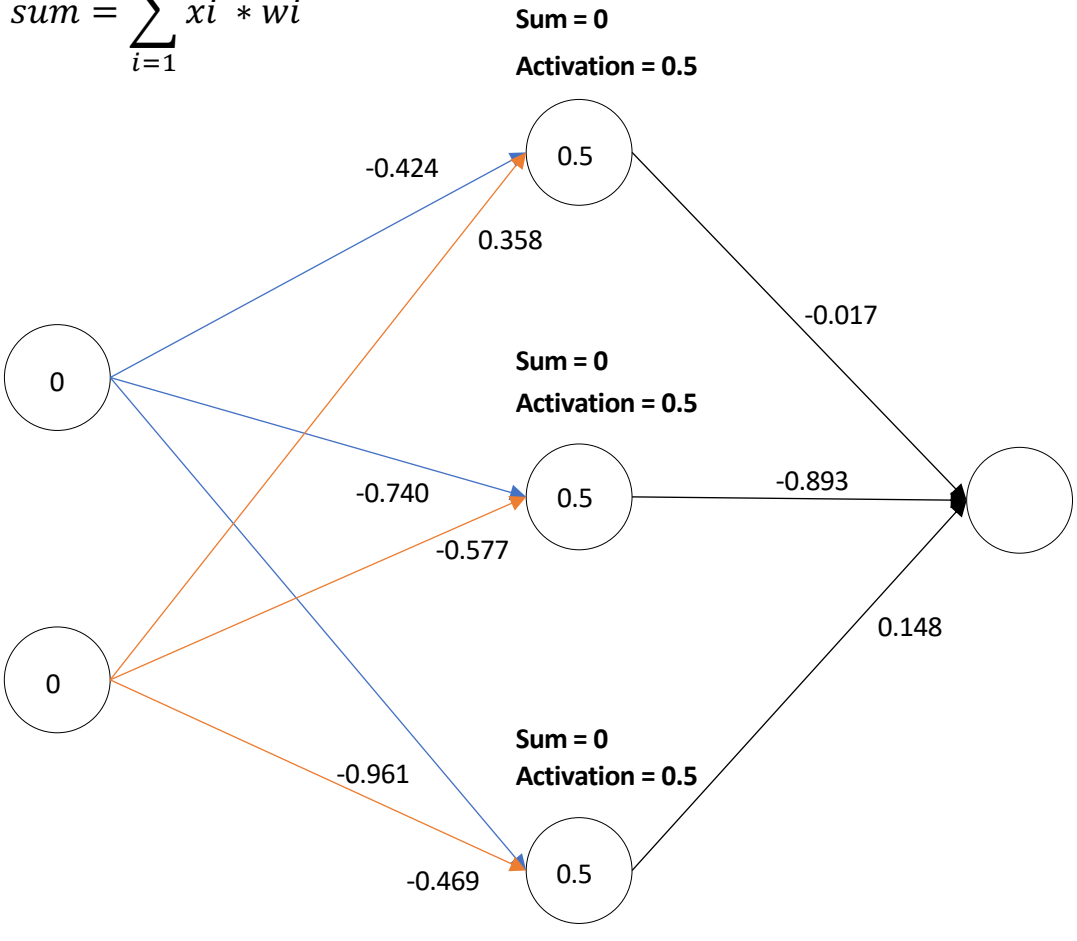
“XOR” OPERATOR

X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0

$$sum = \sum_{i=1}^{n} xi * wi$$

$$y = \frac{1}{1 + e^{-x}}$$

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0



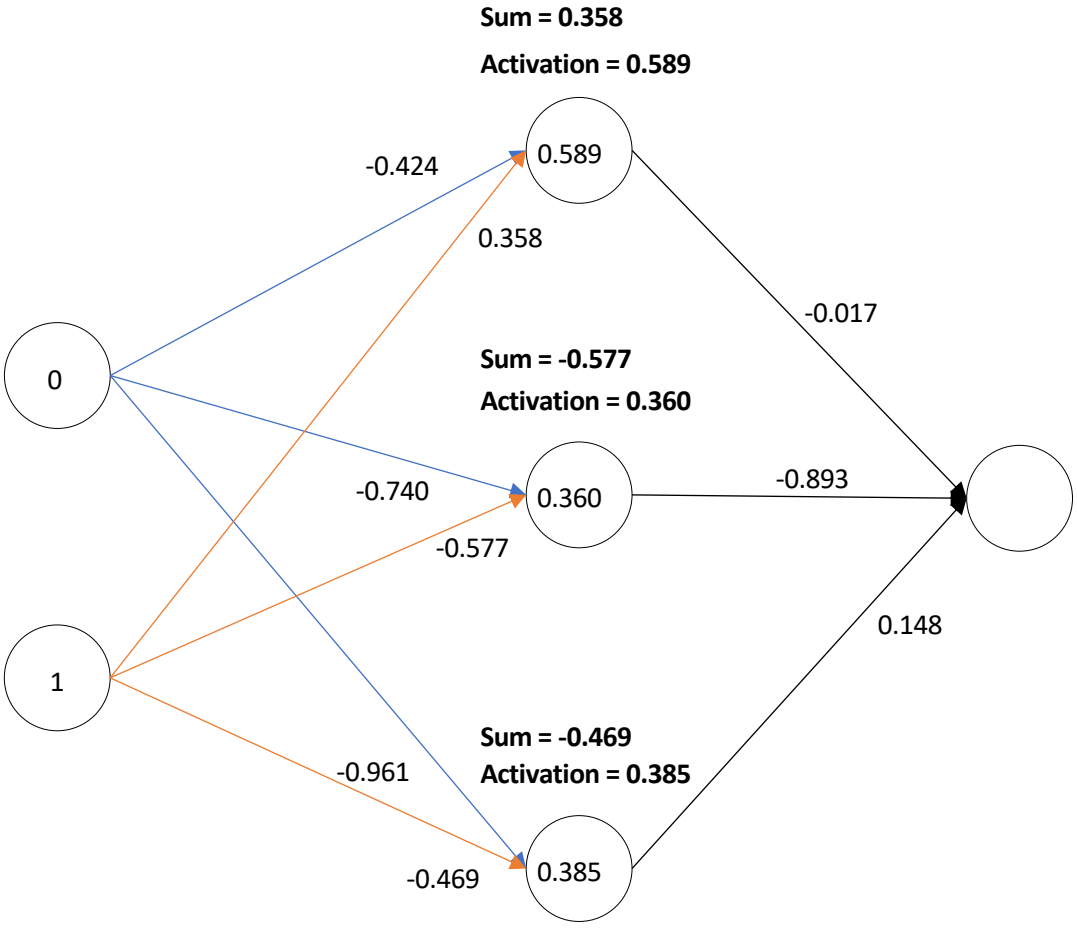
0 * (-0.424) + 0 * 0.358 = 0

0 * (-0.740) + 0 * (-0.577) = 0

0 * (-0.961) + 0 * (-0.469) = 0

EXPERT ACADEMY

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0

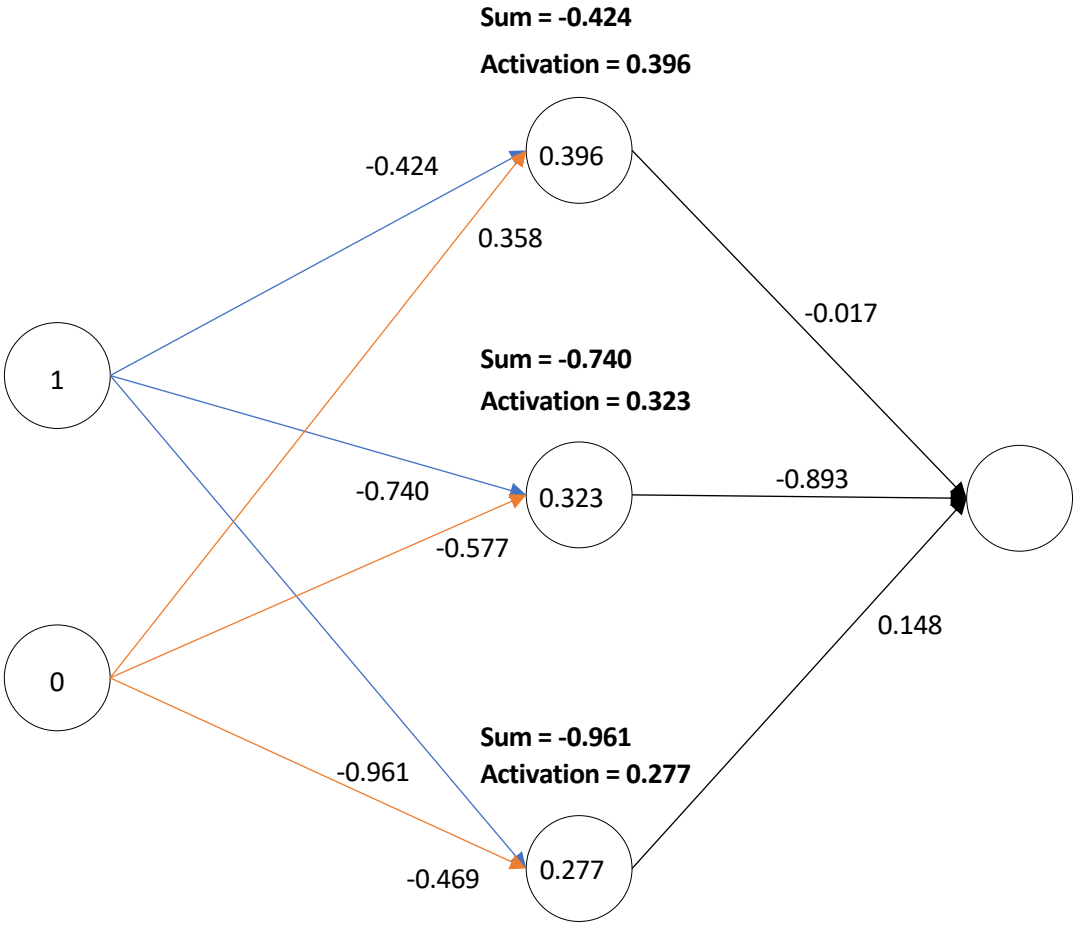


$0 * (-0.424) + 1 * 0.358 = 0.358$

$0 * (-0.740) + 1 * (-0.577) = -0.577$

$0 * (-0.961) + 1 * (-0.469) = -0.469$

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0

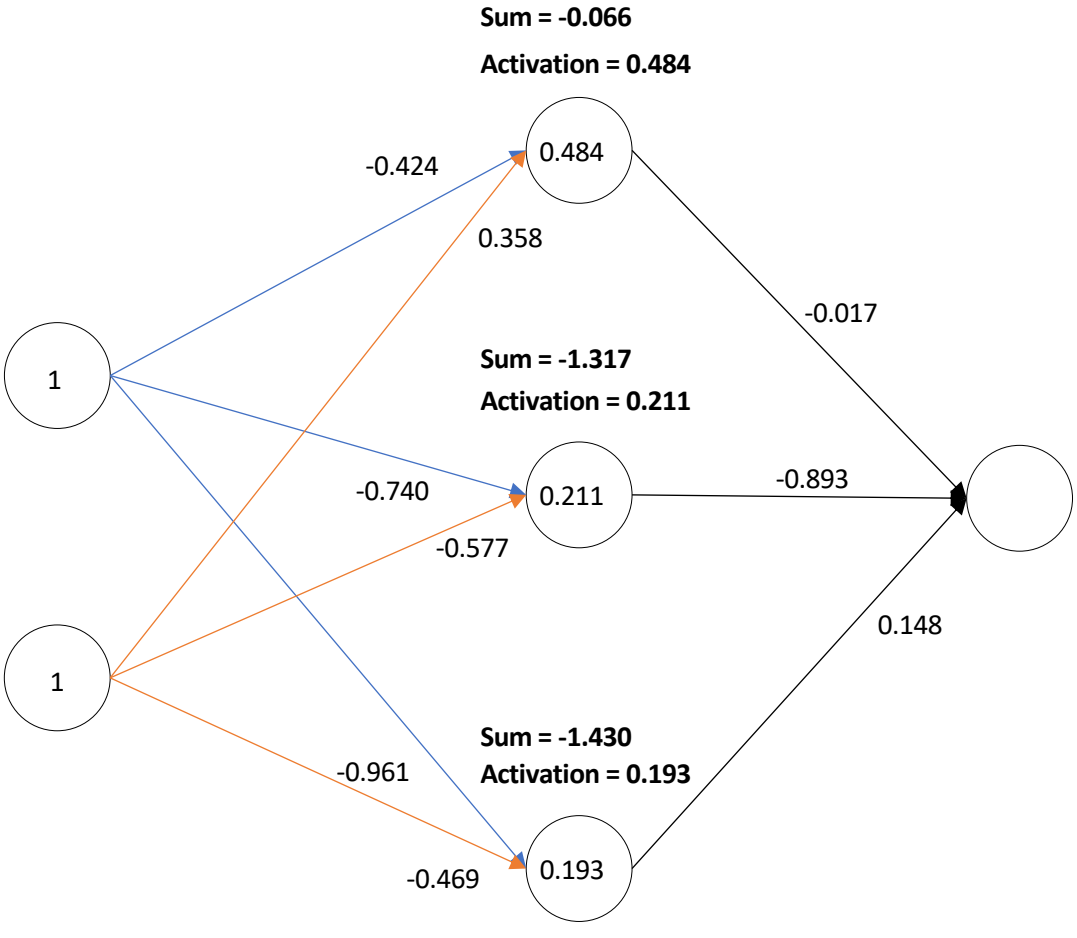


1 * (-0.424) + 0 * 0.358 = -0.424

1 * (-0.740) + 0 * (-0.577) = -0.740

1 * (-0.961) + 0 * (-0.469) = -0.961

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0



$1 * (-0.424) + 1 * 0.358 = -0.066$

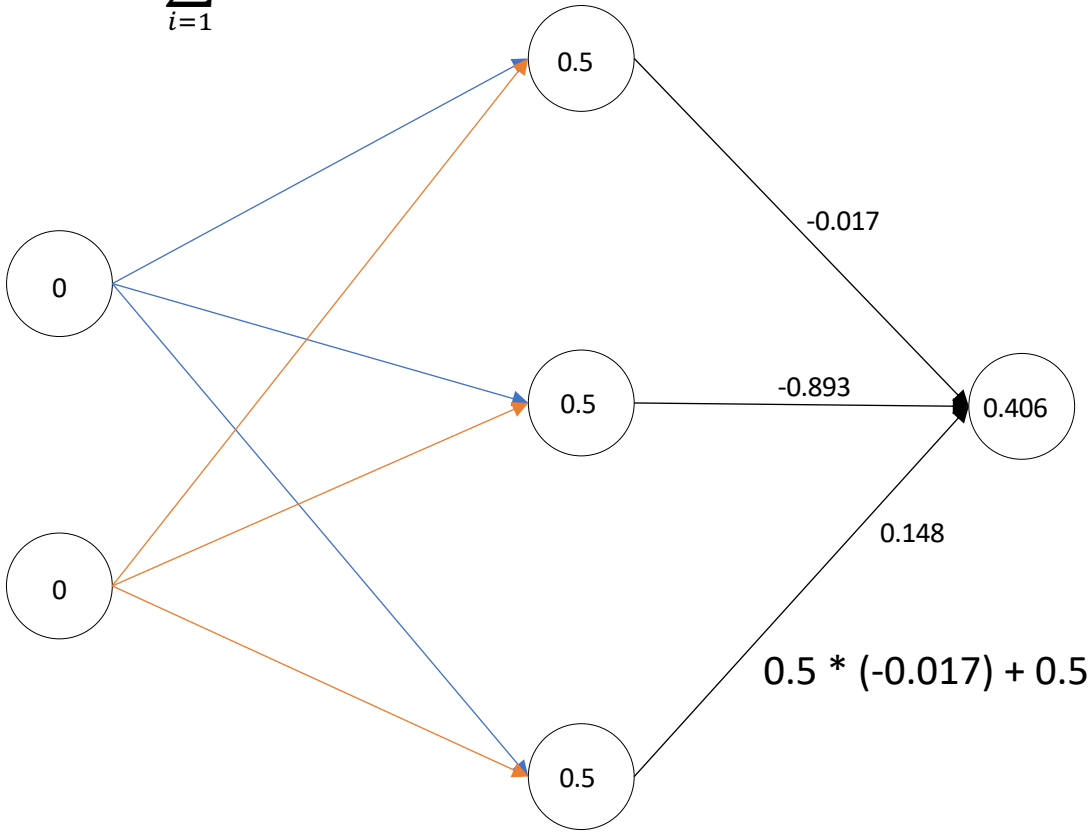
$1 * (-0.740) + 1 * (-0.577) = -1.317$

$1 * (-0.961) + 1 * (-0.469) = -1.430$

$$sum = \sum_{i=1}^{n} xi * wi$$

$$y = \frac{1}{1 + e^{-x}}$$

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0



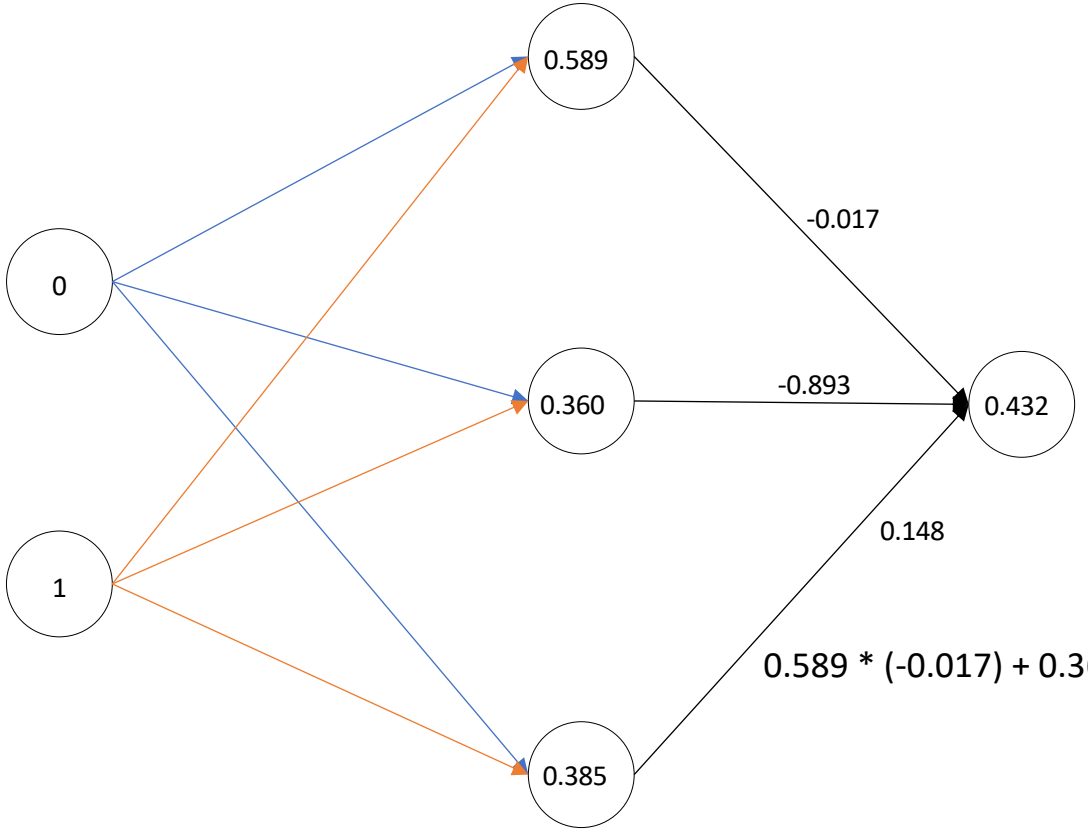
Sum = -0.381

Activation = 0.406

0.5 * (-0.017) + 0.5 * (-0.893) + 0.5 * 0.148 = -0.381

EXPERT ACADEMY

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0

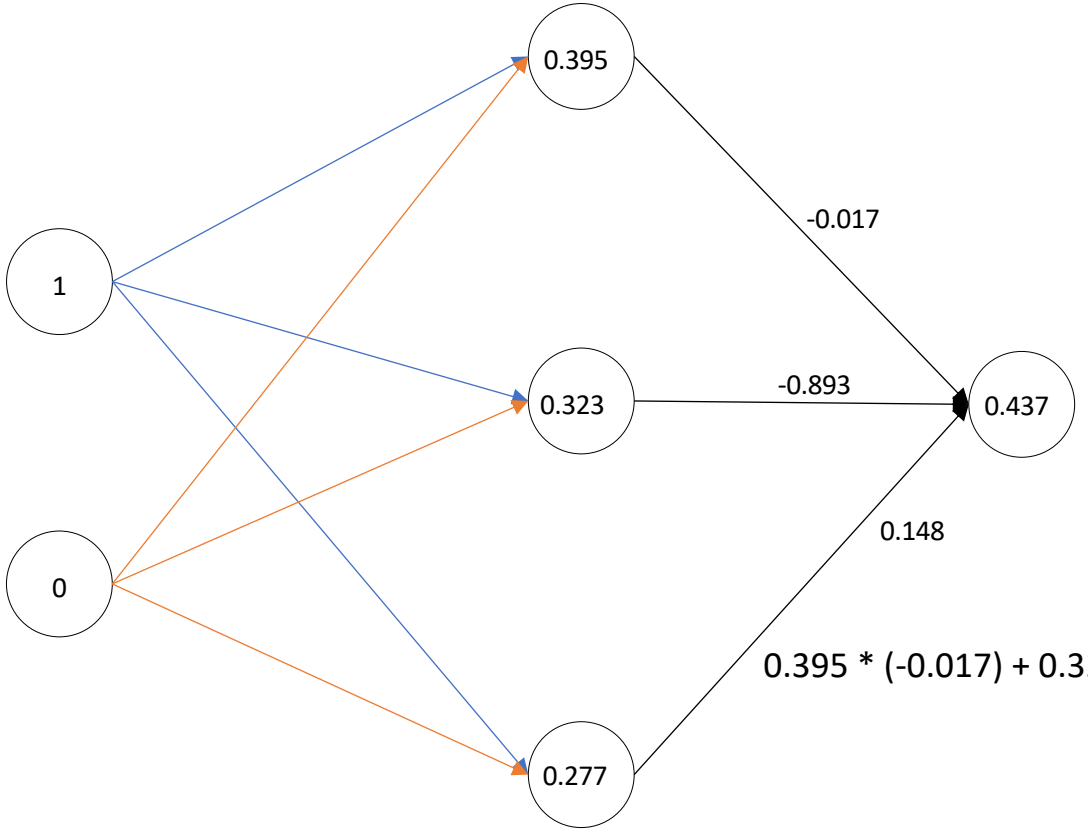


Sum = -0.274

Activation = 0.432

0.589 * (-0.017) + 0.360 * (-0.893) + 0.385 * 0.148 = -0.274

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0

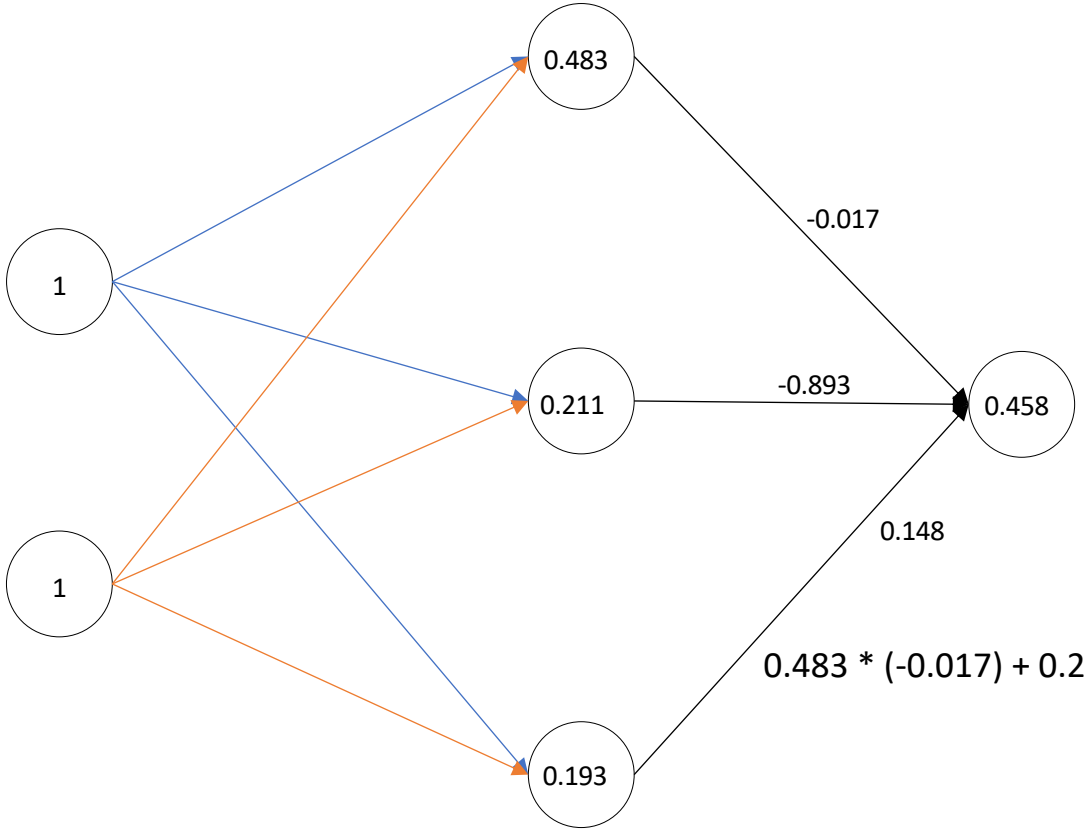


Sum = -0.254

Activation = 0.437

0.395 * (-0.017) + 0.323 * (-0.893) + 0.277 * 0.148 = -0.254

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0



Sum = -0.168

Activation = 0.458

0.483 * (-0.017) + 0.211 * (-0.893) + 0.193 * 0.148 = -0.168

ERROR (LOSS)

- Simplest algorithm
  - error = expectedOuput – prediction

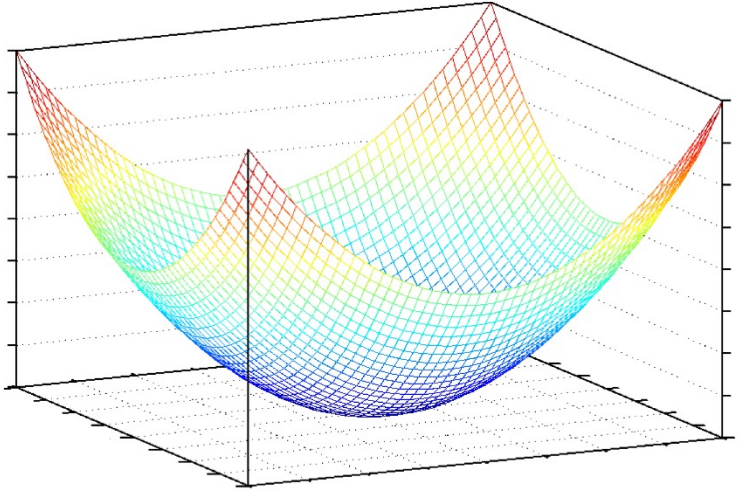
x1	x2	Class	Prediction	Error
0	0	0	0.406	-0.406
0	1	1	0.432	0.568
1	0	1	0.437	0.563
1	1	0	0.458	-0.458

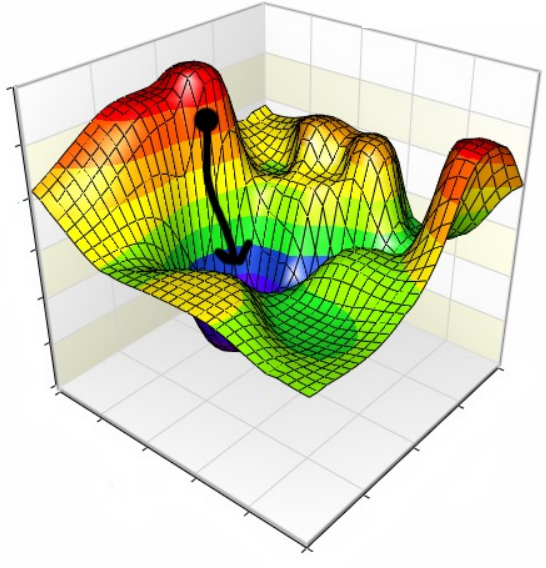
Absolute average = 0.49

GRADIENT

min $C(w_1, w_2 \ldots w_n)$

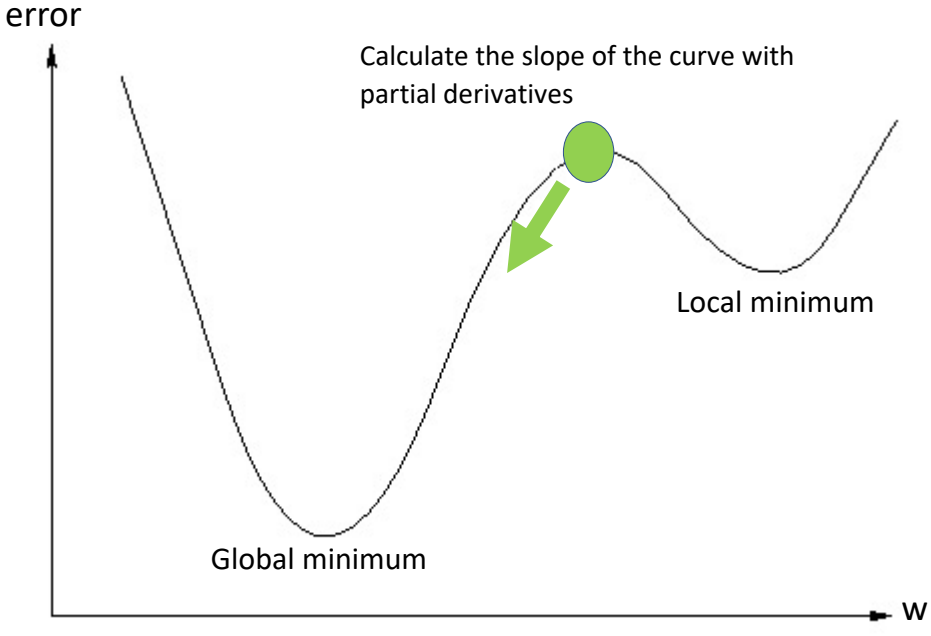
Calculate partial derivative to move
to the gradient direction





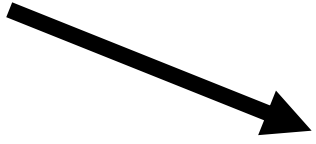
GRADIENT

- Find the combination of weights where the error is as small as possible
- Gradient is calculated to know how much to adjust the weights



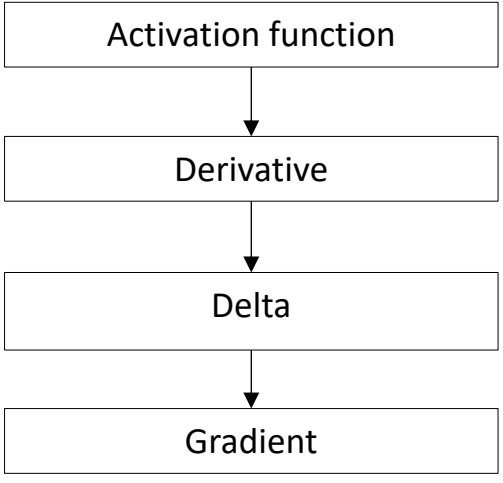
GRADIENT (DERIVATIVE)

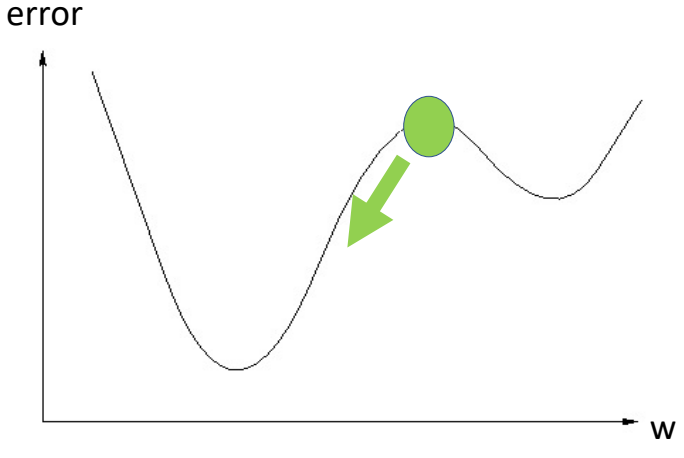
$$y = \frac{1}{1 + e^{-x}}$$



$$d = y * (1 - y)$$

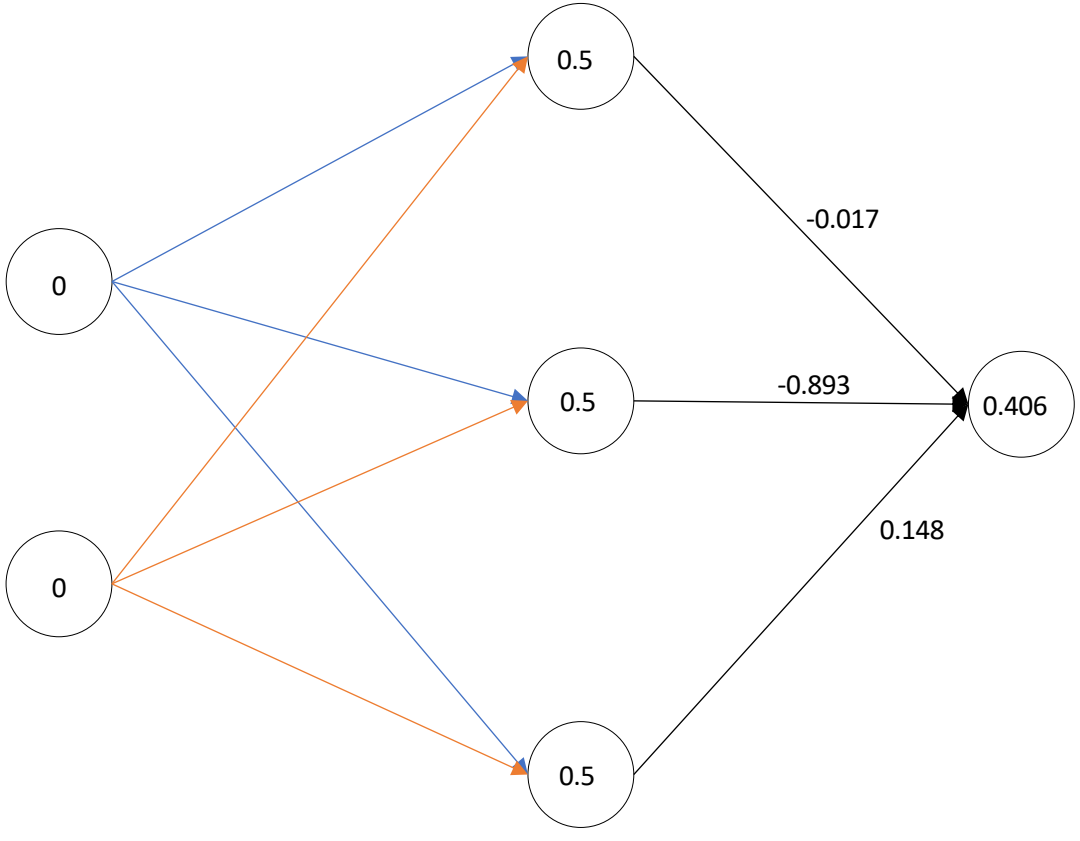
DELTA PARAMETER





$$DeltaOutput = Error * SigmoidDerivative$$

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0



Sum = -0.381

Activation = 0.406

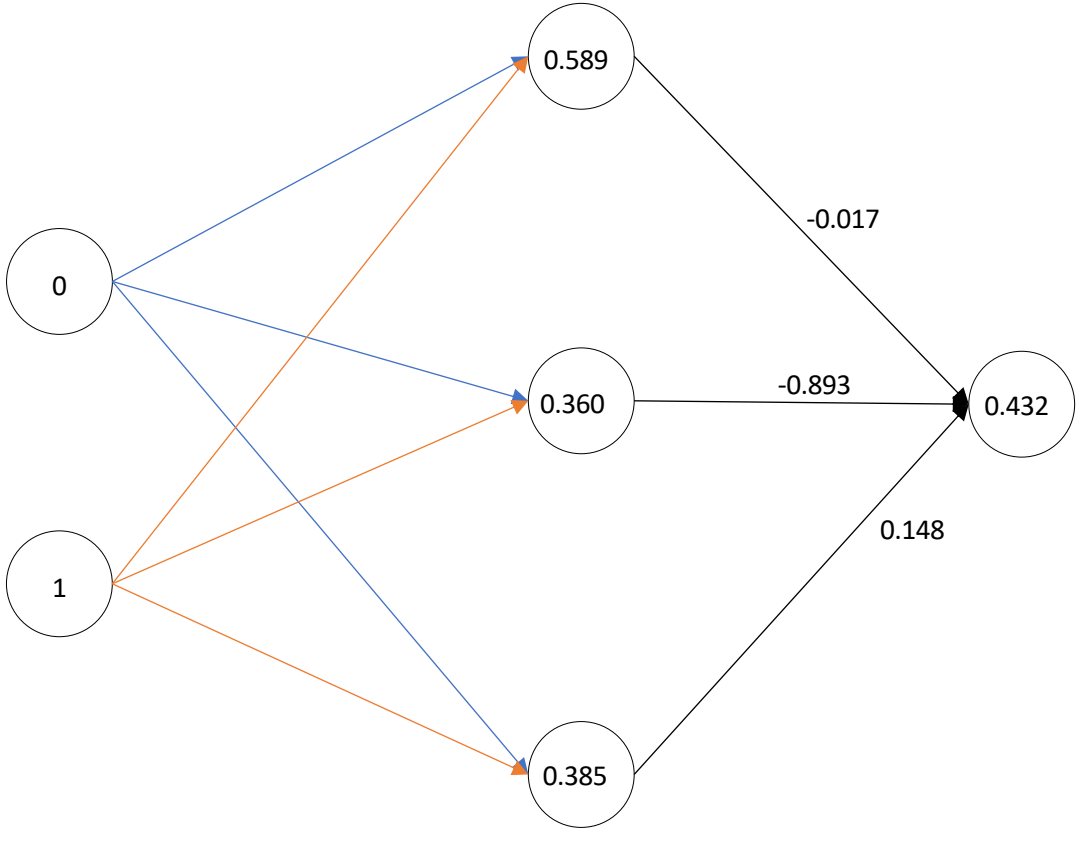
Error = 0 – 0.406 = -0.406

Derivative (sigmoid) = 0.241

DeltaOutput = -0.406 * 0.241 = -0.098

$$DeltaOutput = Error * SigmoidDerivative$$

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0



Sum = -0.274

Activation = 0.432

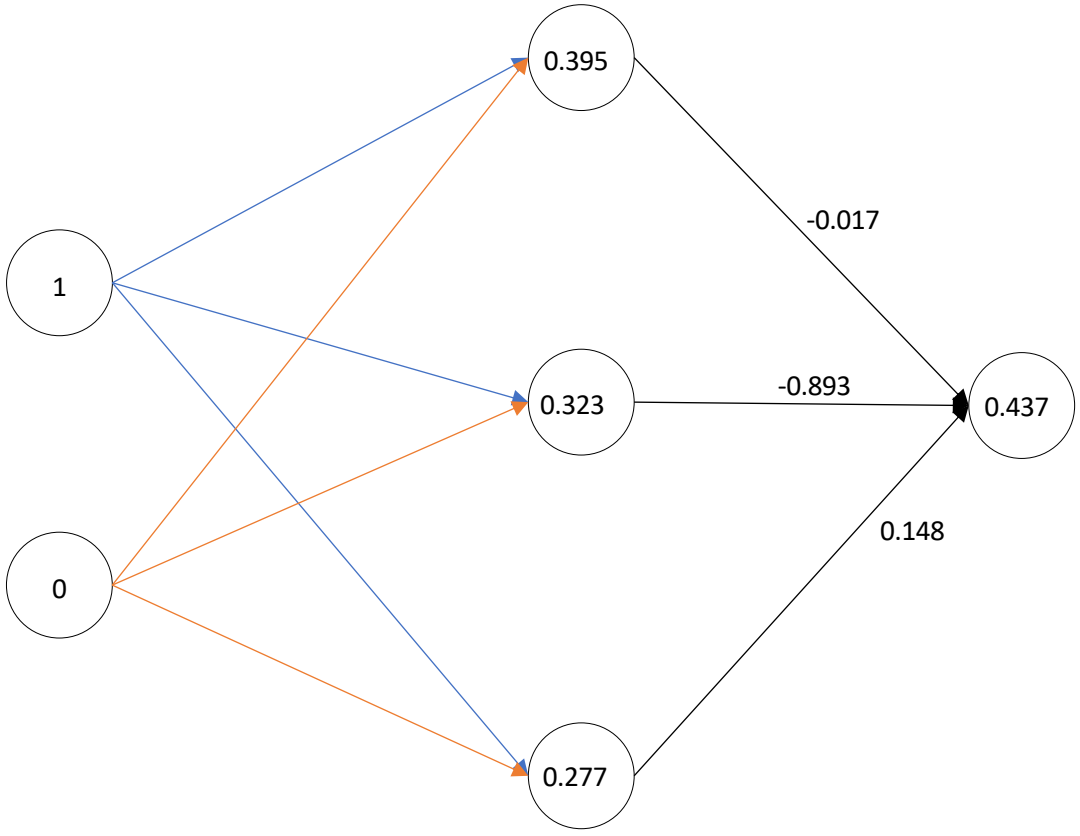
Error = 1 – 0.432 = 0.568

Derivative (sigmoid) = 0.245

DeltaOutput = 0.568 * 0.245 = 0.139

$$DeltaOutput = Error * SigmoidDerivative$$

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0



Sum = -0.254

Activation = 0.437

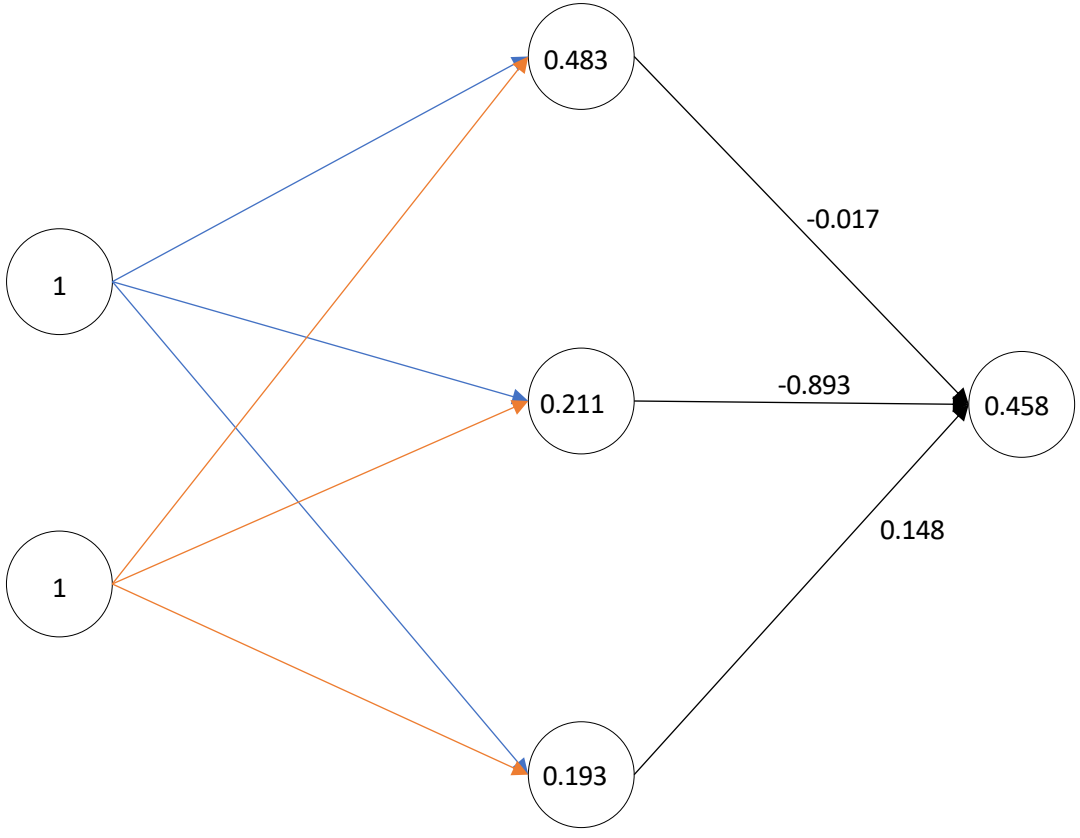
Error = 1 – 0.437 = 0.563

Derivative (sigmoid) = 0.246

DeltaOutput = 0.563 * 0.246 = 0.139

$$DeltaOutput = Error * SigmoidDerivative$$

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0



Sum = -0.168

Activation = 0.458

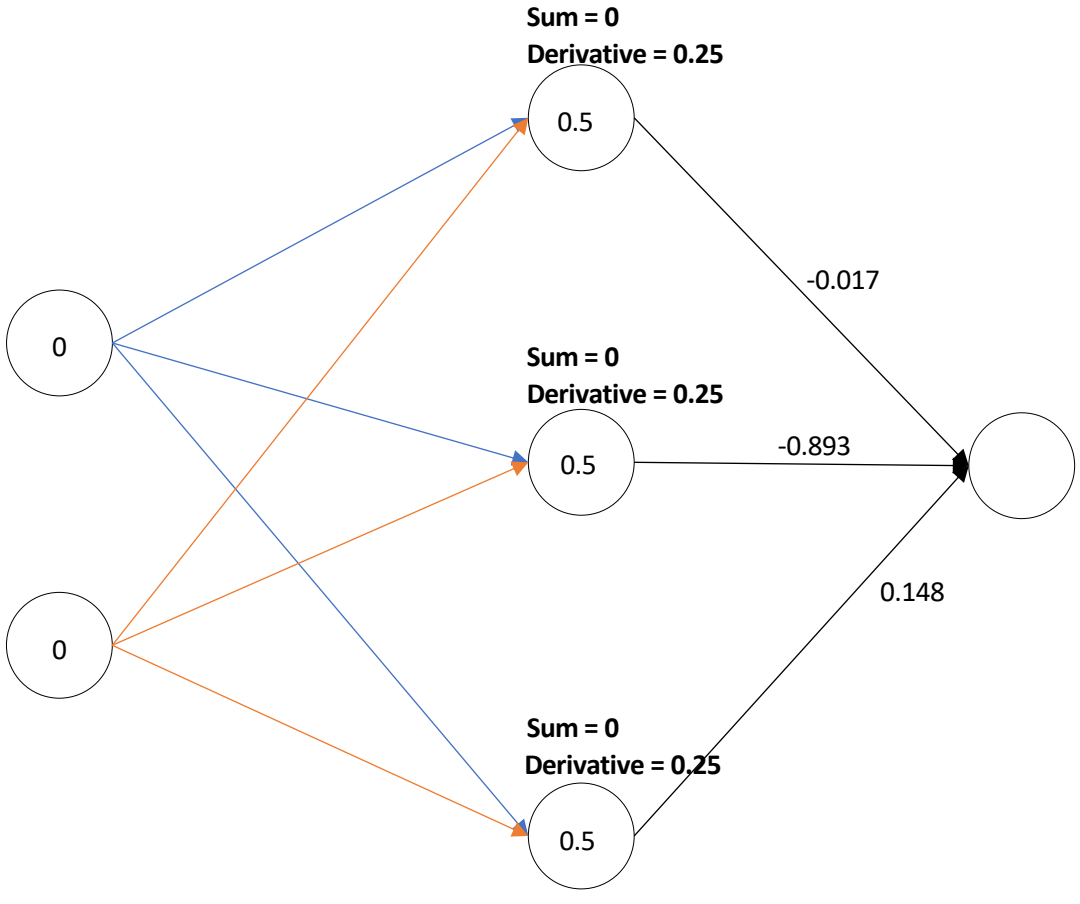
Error = 0 – 0.458 = -0.458

Derivative (sigmoid) = 0.248

DeltaOutput = -0.458 * 0.248 = -0.114

$$DeltaHidden = DerivativeSigmoid * weight * DeltaOutput$$

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0



DeltaOutput = -0.098

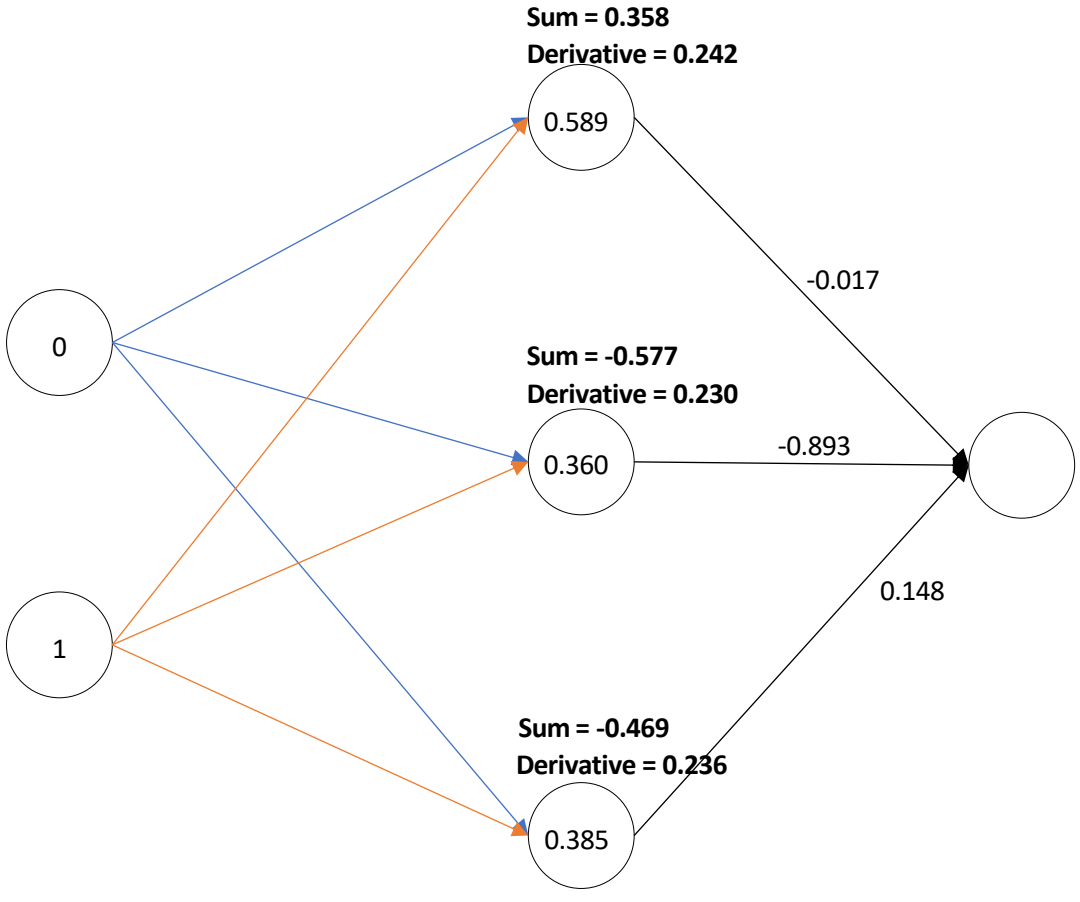
0.25 * (-0.017) * (-0.098) = 0.000

0.25 * (-0.893) * (-0.098) = 0.022

0.25 * 0.148 * (-0.098) = -0.004

$$DeltaHidden = DerivativeSigmoid * weight * DeltaOutput$$

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0



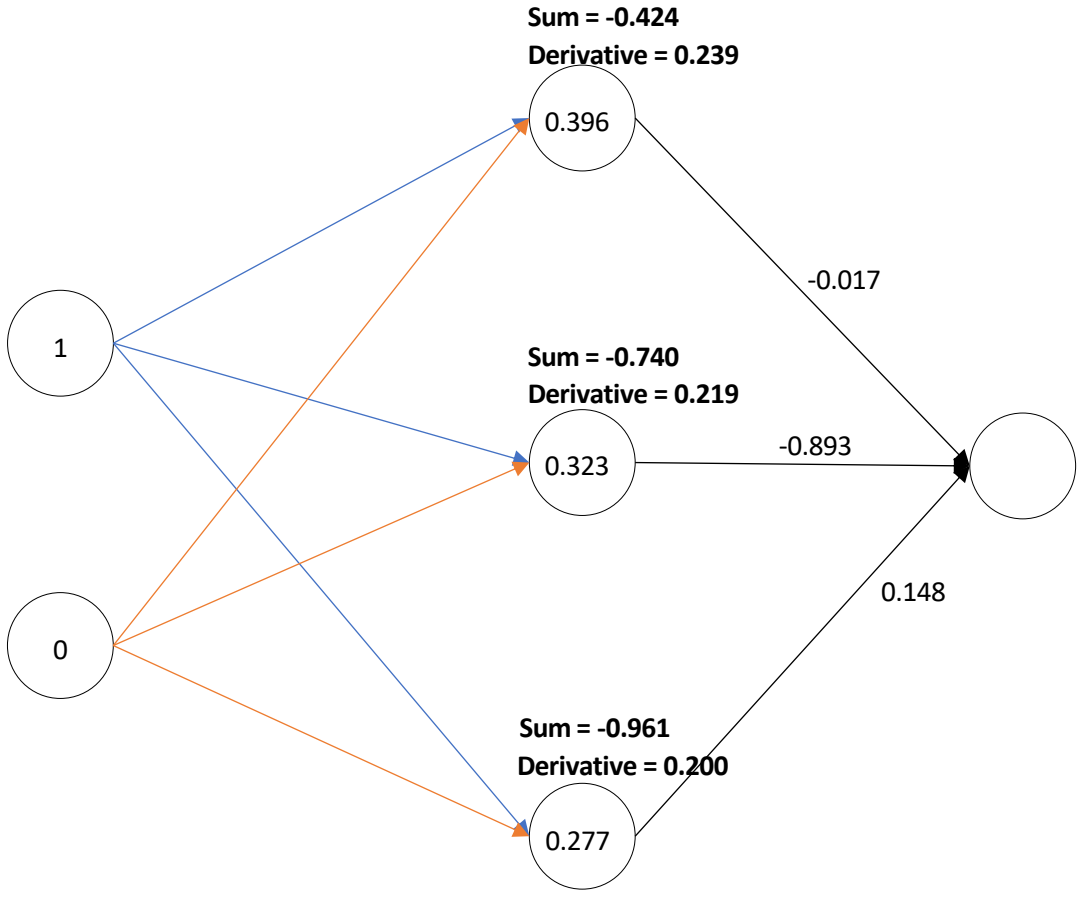
0.242 * (-0.017) * 0.139 = -0.001

0.230 * (-0.893) * 0.139 = -0.029

0.236 * 0.148 * 0.139 = 0.005

$$DeltaHidden = DerivativeSigmoid * weight * DeltaOutput$$

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0



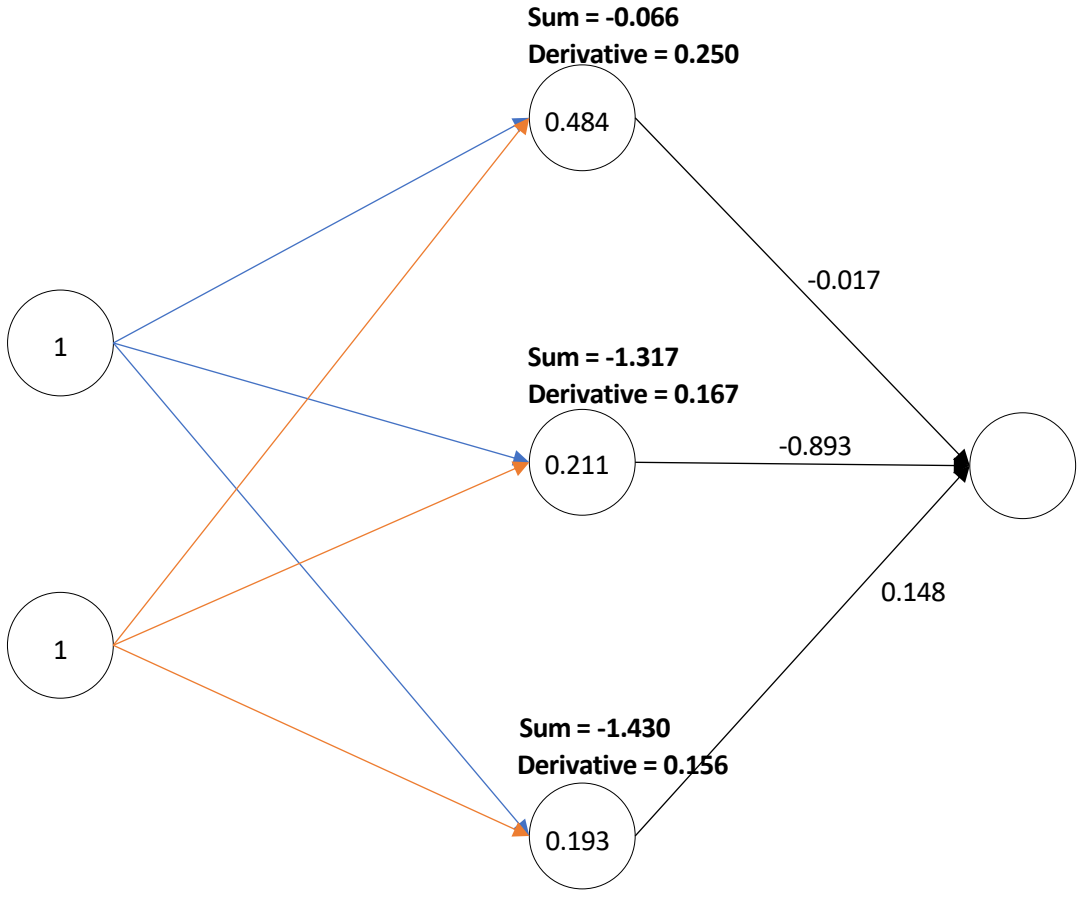
0.239 * (-0.017) * 0.139 = -0.001

0.219 * (-0.893) * 0.139 = -0.027

0.200 * 0.148 * 0.139 = 0.004

$$DeltaHidden = DerivativeSigmoid * weight * DeltaOutput$$

x1	x2	Class
0	0	0
0	1	1
1	0	1
1	1	0

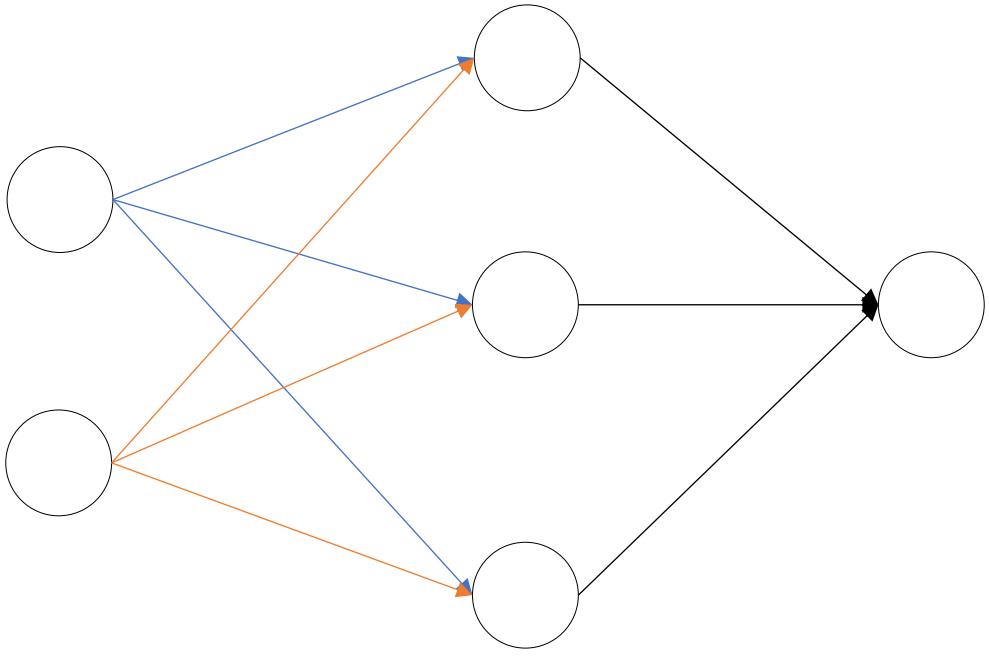


0.250 * (-0.017) * (-0.114) = 0.000

0.167 * (-0.893) * (-0.114) = 0.017

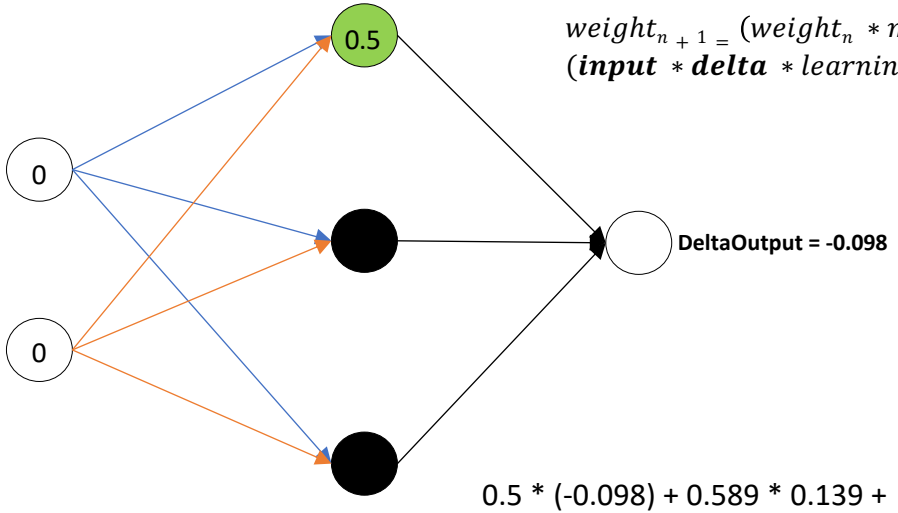
0.156 * 0.148 * (-0.114) = -0.003

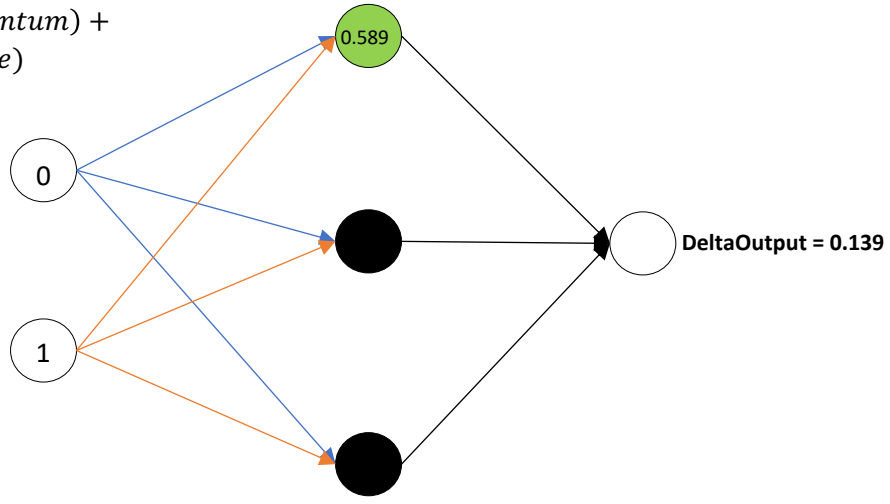
BACKPROPAGATION



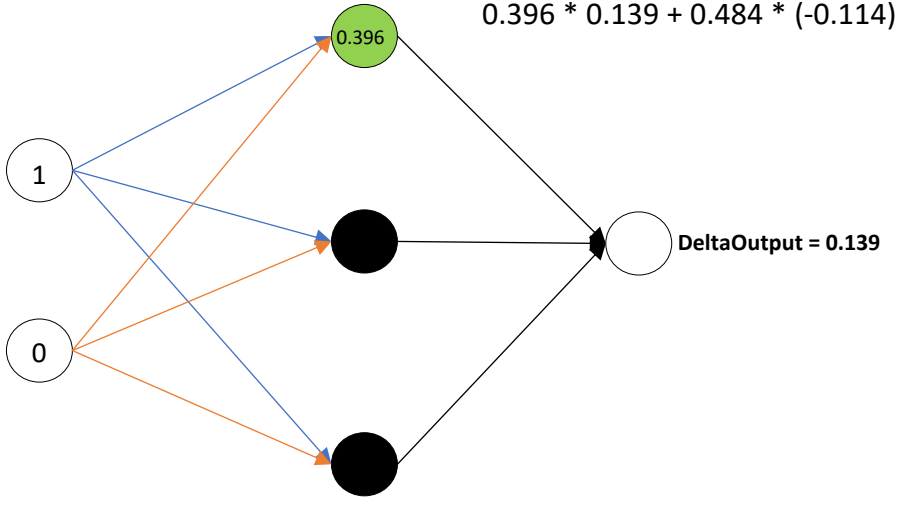
$$weight_{n+1} = (weight_n * momentum) + (input * delta * learning\ rate)$$

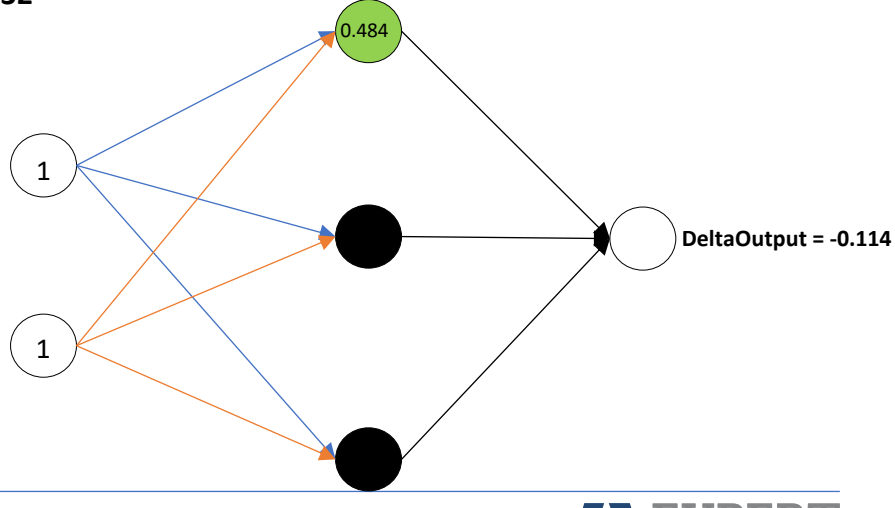
$$weight_{n+1} = (weight_n * momentum) + (\boldsymbol{input} * \boldsymbol{delta} * learning\ rate)$$



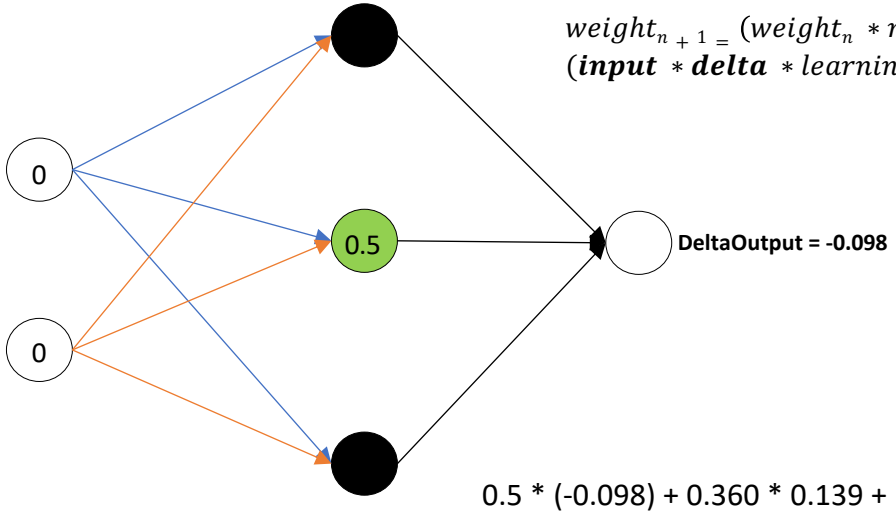


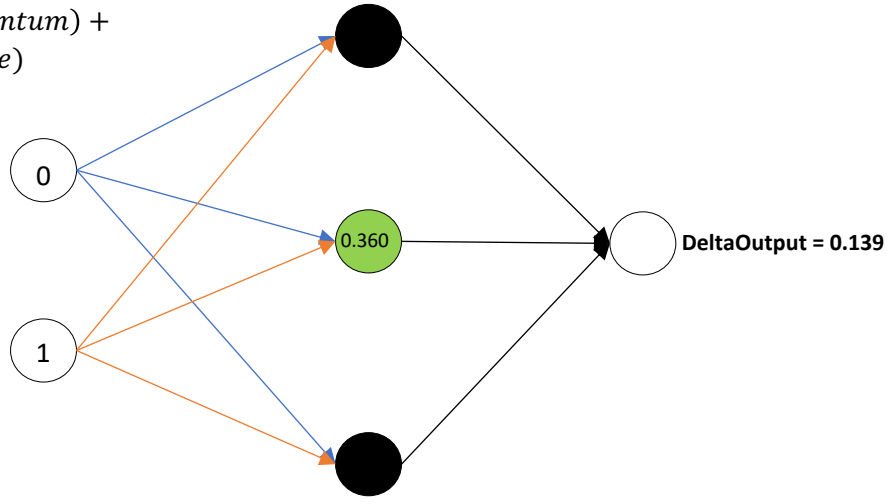
0.5 * (-0.098) + 0.589 * 0.139 +
0.396 * 0.139 + 0.484 * (-0.114) = **0.032**



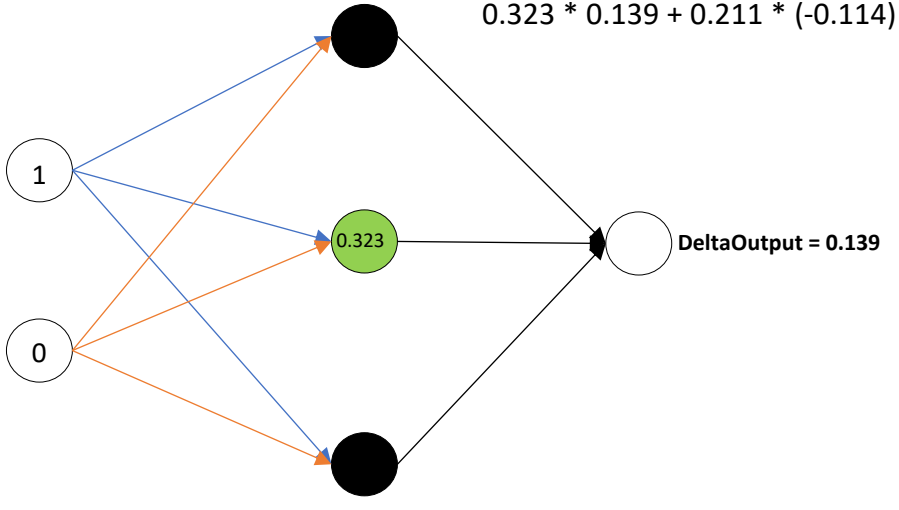


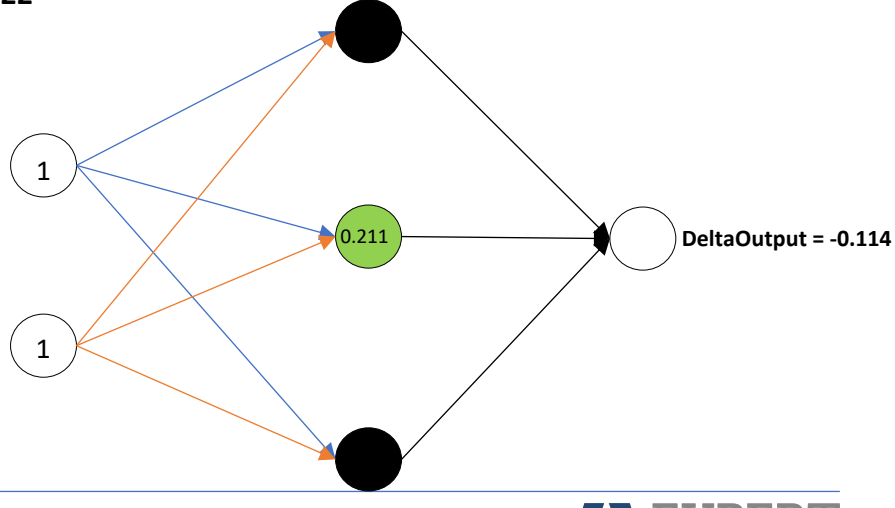
$$weight_{n+1} = (weight_n * momentum) + (\boldsymbol{input} * \boldsymbol{delta} * learning\ rate)$$



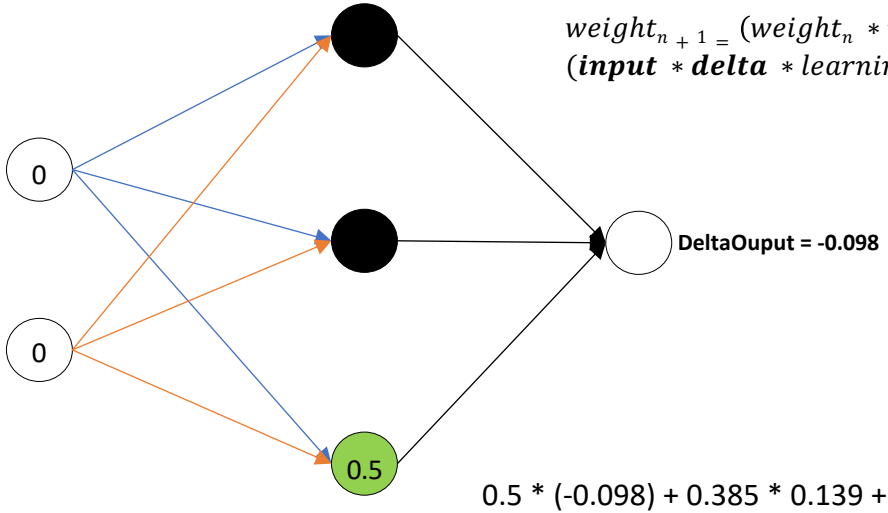


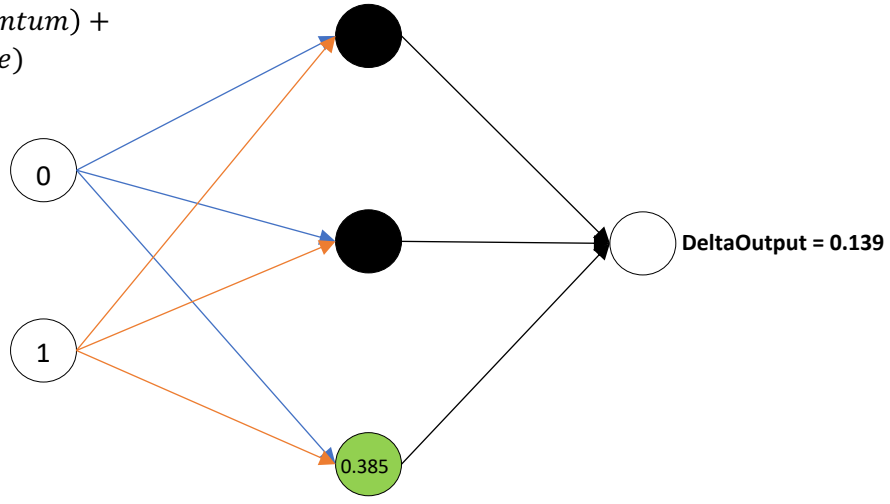
0.5 * (-0.098) + 0.360 * 0.139 +
0.323 * 0.139 + 0.211 * (-0.114) = **0.022**



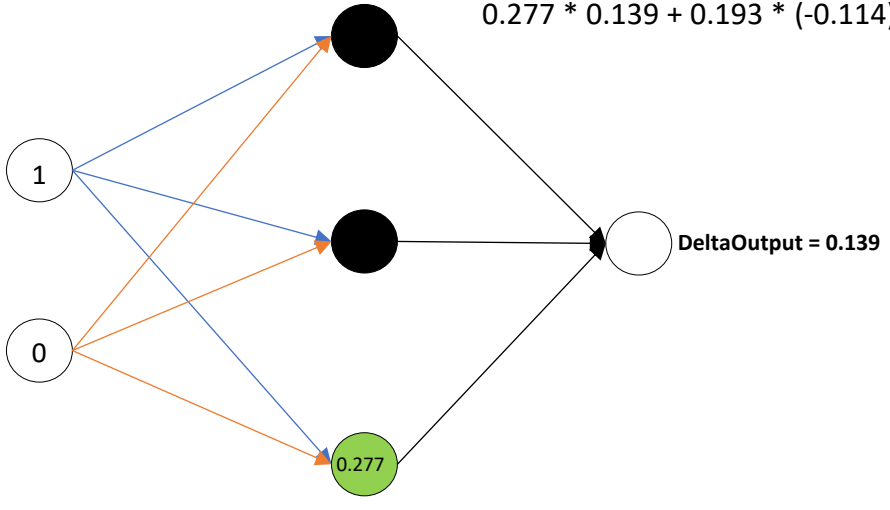


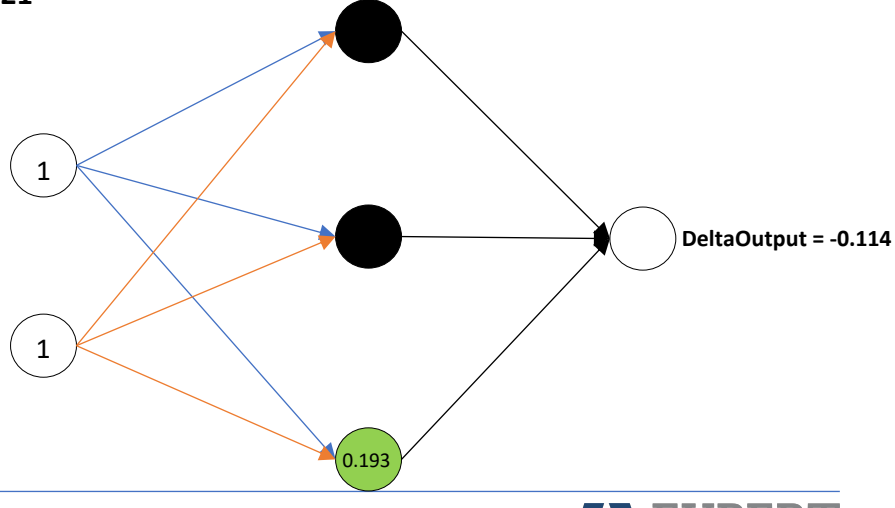
$$weight_{n+1} = (weight_n * momentum) + (\boldsymbol{input} * \boldsymbol{delta} * learning\ rate)$$





0.5 * (-0.098) + 0.385 * 0.139 +
0.277 * 0.139 + 0.193 * (-0.114) = **0.021**





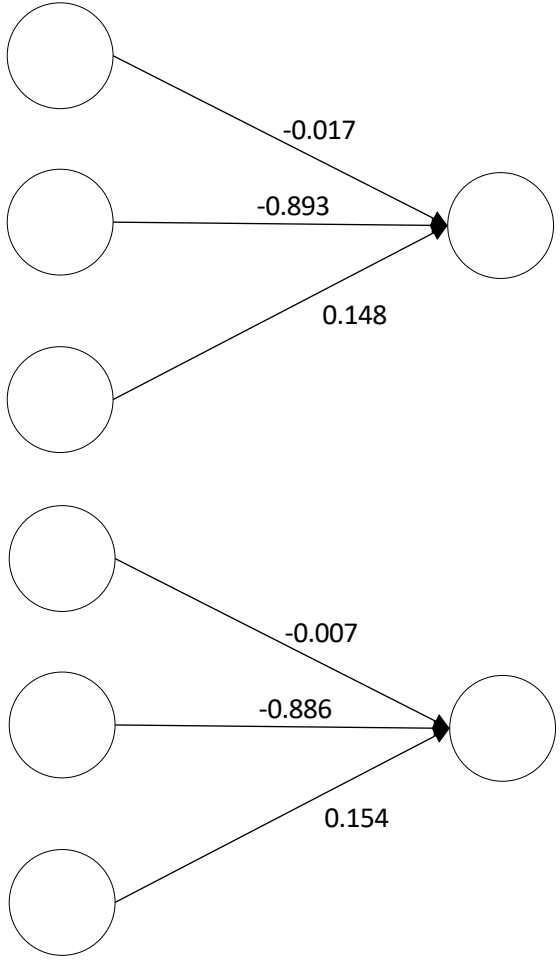
Learning rate = 0.3
Momentum = 1
Input x delta
0.032
0.022
0.021

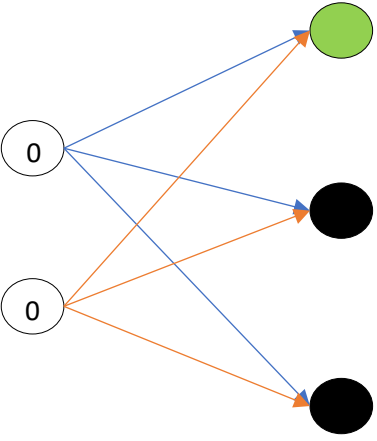
$$Weight_{n+1} = (weight_n * momentum) + (input * delta * learning\ rate)$$

(-0.017 * 1) + 0.032 * 0.3 = **-0.007**

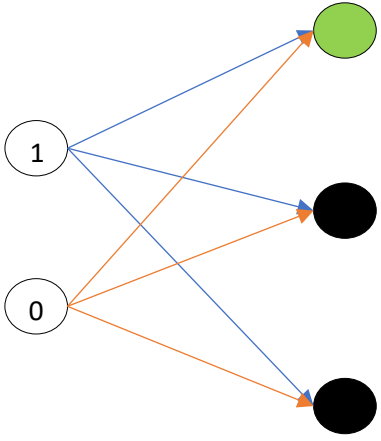
(-0.893 * 1) + 0.022 * 0.3 = **-0.886**

(0.148 * 1) + 0.021 * 0.3 = **0.154**



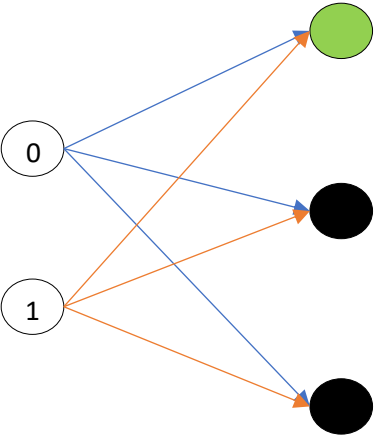


Delta = 0.000

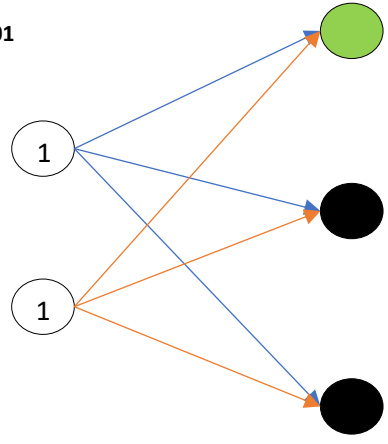


Delta = -0.001

$$weight_{n+1} = (weight_n * momentum) + (\boldsymbol{input} * \boldsymbol{delta} * learning\ rate)$$



Delta = -0.001

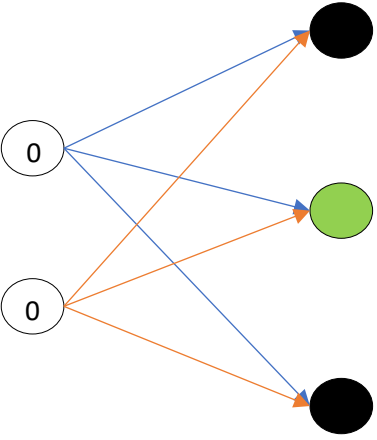


Delta = 0.000

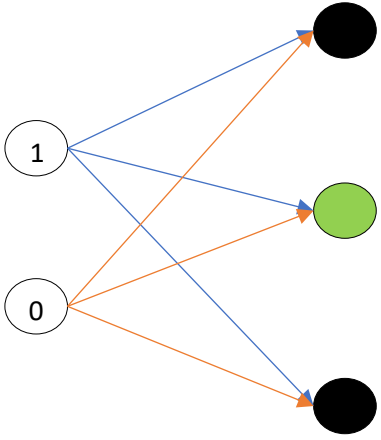
0 * 0.000 + 0 * (-0.001) + 1 * (-0.001) + 1 * 0.000 = -0.000

0 * 0.000 + 1 * (-0.001) + 0 * (-0.001) + 1 * 0.000 = -0.000

Rounded!

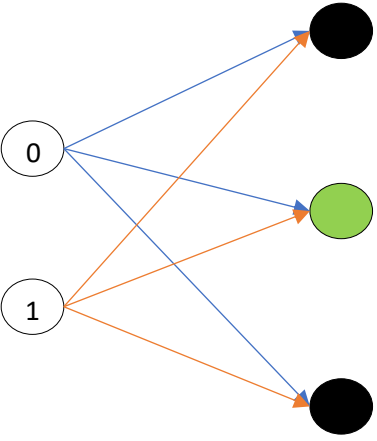


Delta = 0.022

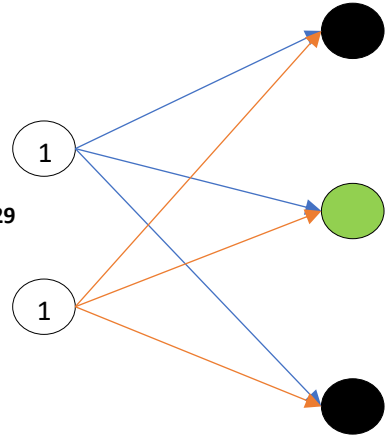


Delta = -0.027

$$weight_{n+1} = (weight_n * momentum) + (\boldsymbol{input} * \boldsymbol{delta} * learning\ rate)$$



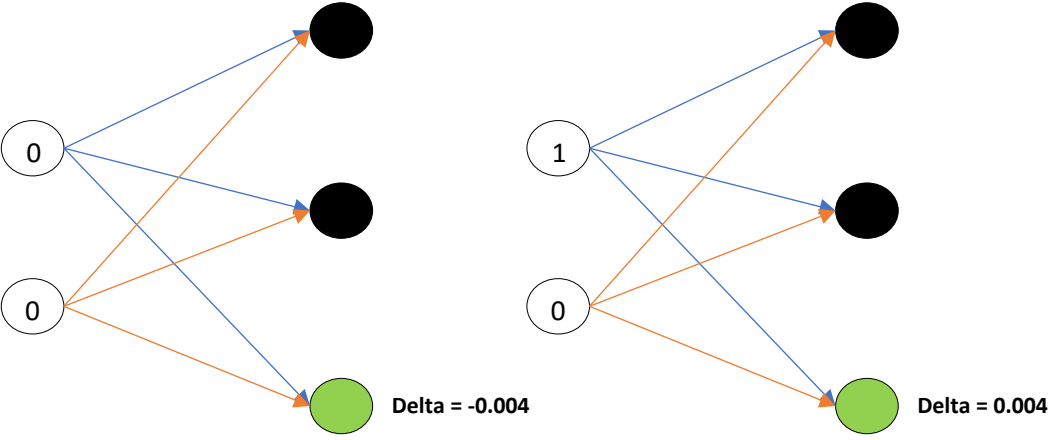
Delta = -0.029



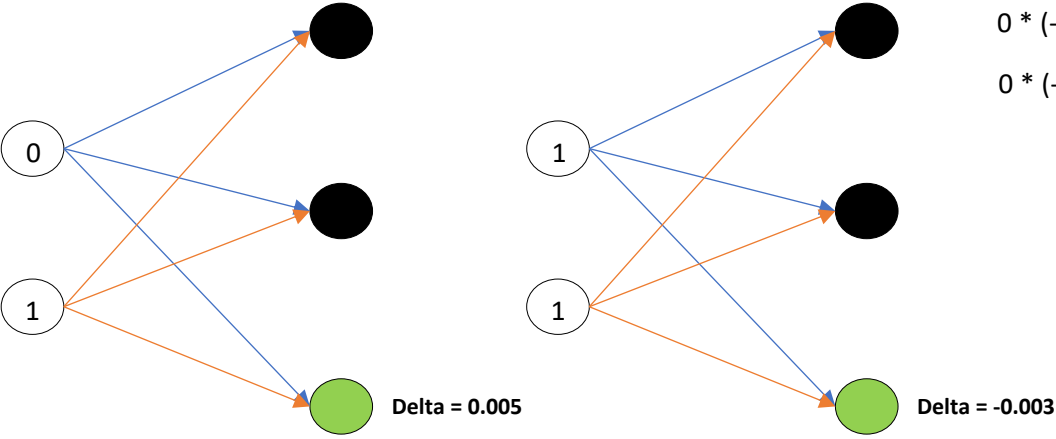
Delta = 0.017

0 * 0.022 + 0 * (-0.029) + 1 * (-0.027) + 1 * 0.017 = -0.010

0 * 0.022 + 1 * (-0.029) + 0 * (-0.027) + 1 * 0.017 = -0.012



$$weight_{n+1} = (weight_n * momentum) + (\boldsymbol{input} * \boldsymbol{delta} * learning\ rate)$$



0 * (-0.004) + 0 * 0.005 + 1 * 0.004 + 1 * (-0.003) = 0.001

0 * (-0.004) + 1 * 0.005 + 0 * 0.004 + 1 * (-0.003) = 0.002

Learning rate = 0.3
Momentum = 1
Input x delta
-0.000 -0.010 0.001
-0.000 -0.012 0.002

$$Weight_{n+1} = (weight_n * momentum) + (input * delta * learning\ rate)$$

(-0.424 * 1) + (-0.000) * 0.3 = **-0.424**

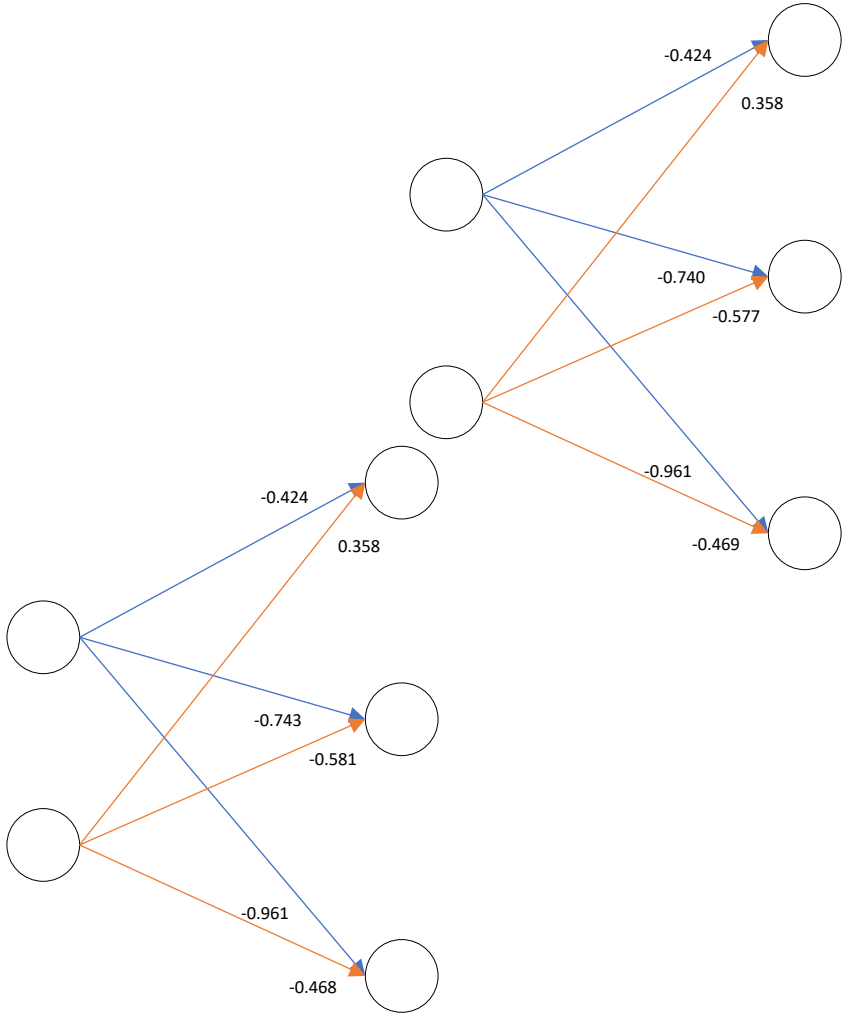
(0.358 * 1) + (-0.000) * 0.3 = **0.358**

(-0.740 * 1) + (-0.010) * 0.3 = **-0.743**

(-0.577 * 1) + (-0.012) * 0.3 = **-0.581**

(-0.961 * 1) + 0.001 * 0.3 = **-0.961**

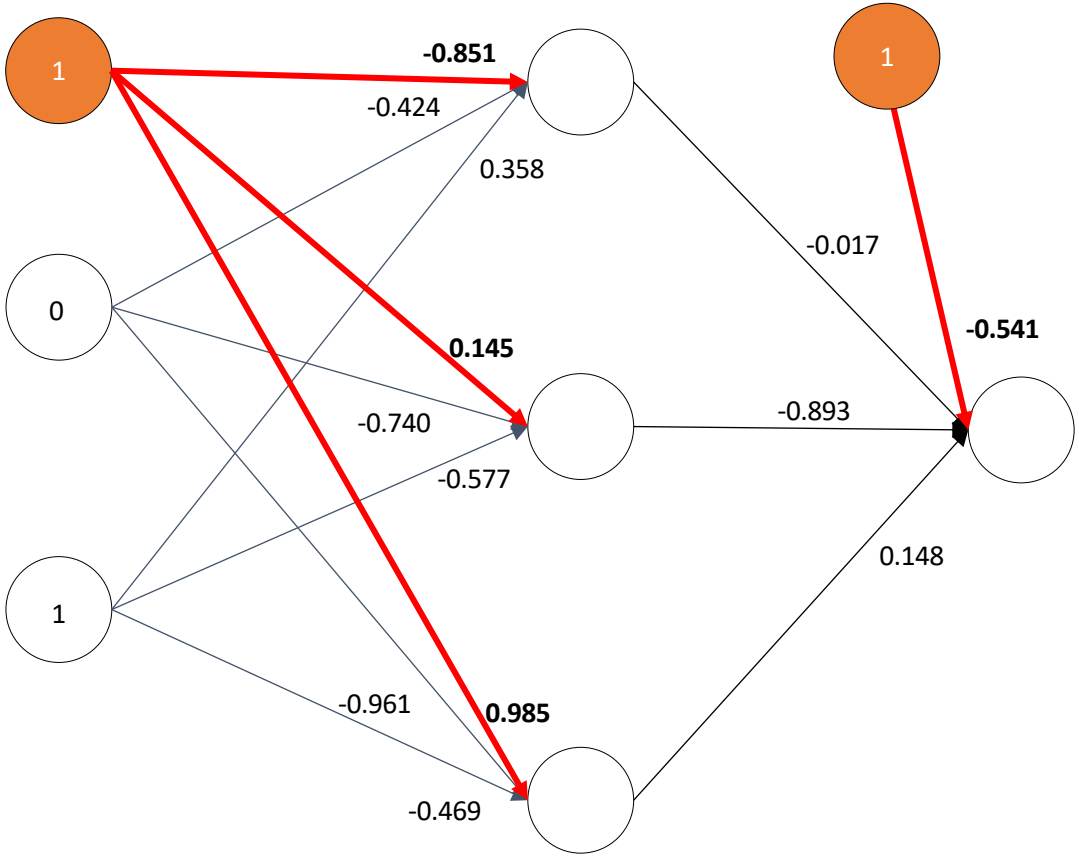
(-0.469 * 1) + 0.002 * 0.3 = **-0.468**



BIAS

Different values even if all entries are zero

Change the output



EXPERT ACADEMY

ERROR (LOOS)

- Simplest algorithm
  - error = expected output – prediction

x1	x2	Class	Prediction	Error
0	0	0	0.406	-0.406
0	1	1	0.432	0.568
1	0	1	0.437	0.563
1	1	0	0.458	-0.458

Absolute average = 0.49

MEAN SQUARED ERROR (MSE) AND ROOT MEAN SQUARED ERROR (RMSE)

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(f_i - o_i)^2}$$

x1	x2	Class	Prediction	Error
0	0	0	0.406	$(0 - 0.406)^2 = 0.164$
0	1	1	0.432	$(1 - 0.432)^2 = 0.322$
1	0	1	0.437	$(1 - 0.437)^2 = 0.316$
1	1	0	0.458	$(0 - 0.458)^2 = 0.209$

Sum = 1.011

MSE = 1.011 / 4 = 0.252

RMSE = 0.501

Credit history	Debts	Properties	Income	Risk
3	1	1	1	1 0 0
2	1	1	2	1 0 0
2	2	1	2	0 1 0
2	2	1	3	1 0 0
2	2	1	3	0 0 1
2	2	2	3	0 0 1
3	2	1	1	1 0 0
3	2	2	3	0 1 0
1	2	1	3	0 0 1
1	1	2	3	0 0 1
1	1	1	1	1 0 0
1	1	1	2	0 1 0
1	1	1	3	0 0 1
3	1	1	2	1 0 0

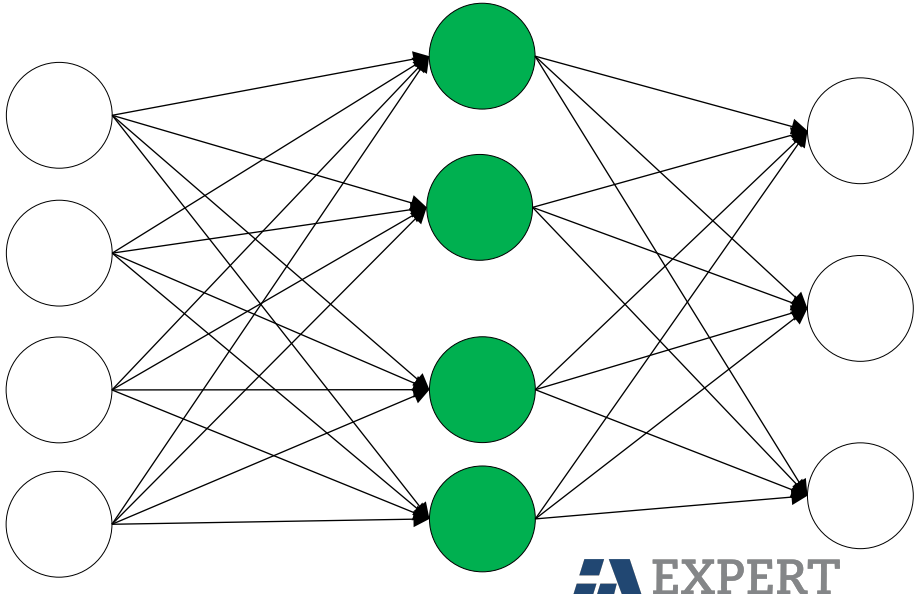
Calculates the error for all rows and updates the weights

Batch gradient descent

Credit history	Debts	Properties	Income	Risk
3	1	1	1	1 0 0
2	1	1	2	1 0 0
2	2	1	2	0 1 0
2	2	1	3	1 0 0
2	2	1	3	0 0 1
2	2	2	3	0 0 1
3	2	1	1	1 0 0
3	2	2	3	0 1 0
1	2	1	3	0 0 1
1	1	2	3	0 0 1
1	1	1	1	1 0 0
1	1	1	2	0 1 0
1	1	1	3	0 0 1
3	1	1	2	1 0 0

Calculates the error for each row and updates the weights

Stochastic gradient descent

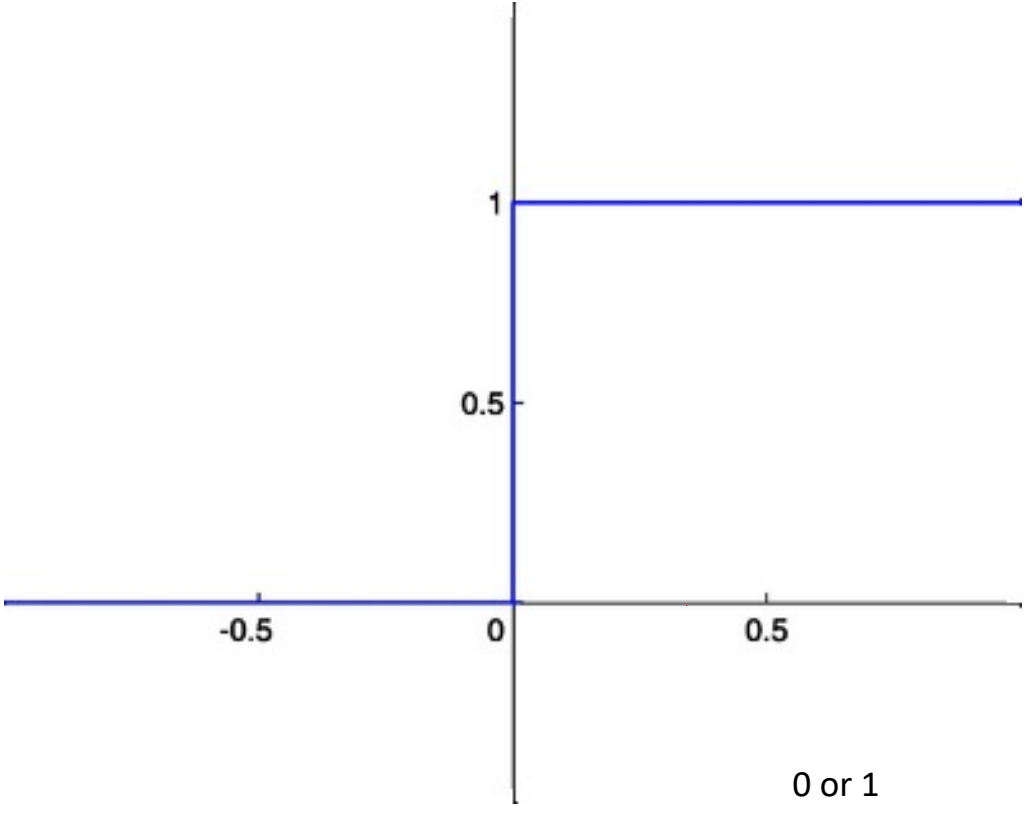


EXPERT ACADEMY

GRADIENT DESCENT

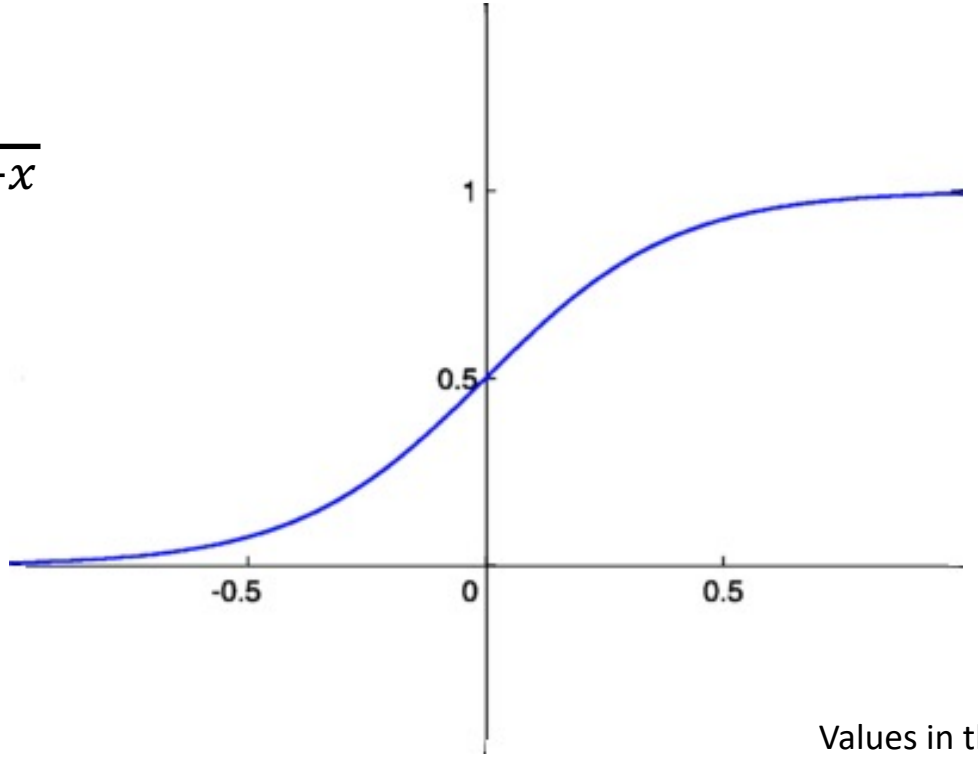
- Stochastic
  - Helps prevent local minima
  - Faster (no need to load all rows into memory)
- Mini batch gradient descent
  - Choose a number of rows to calculate the error and update the weights

STEP FUNCTION



SIGMOID FUNCTION

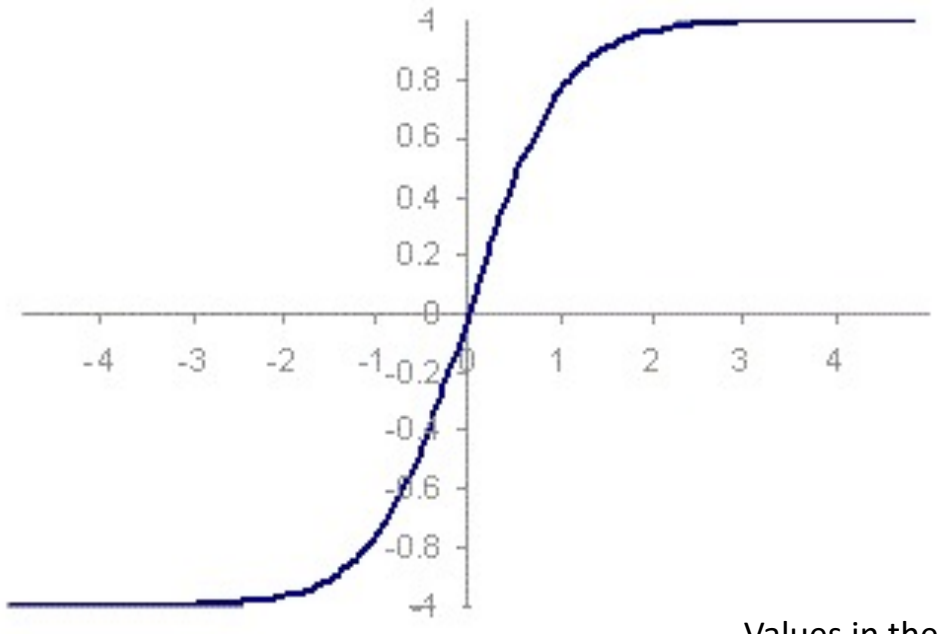
$$y = \frac{1}{1 + e^{-x}}$$



Values in the range from 0 to 1

HYPERBOLIC TANGENT FUNCTION

$$Y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



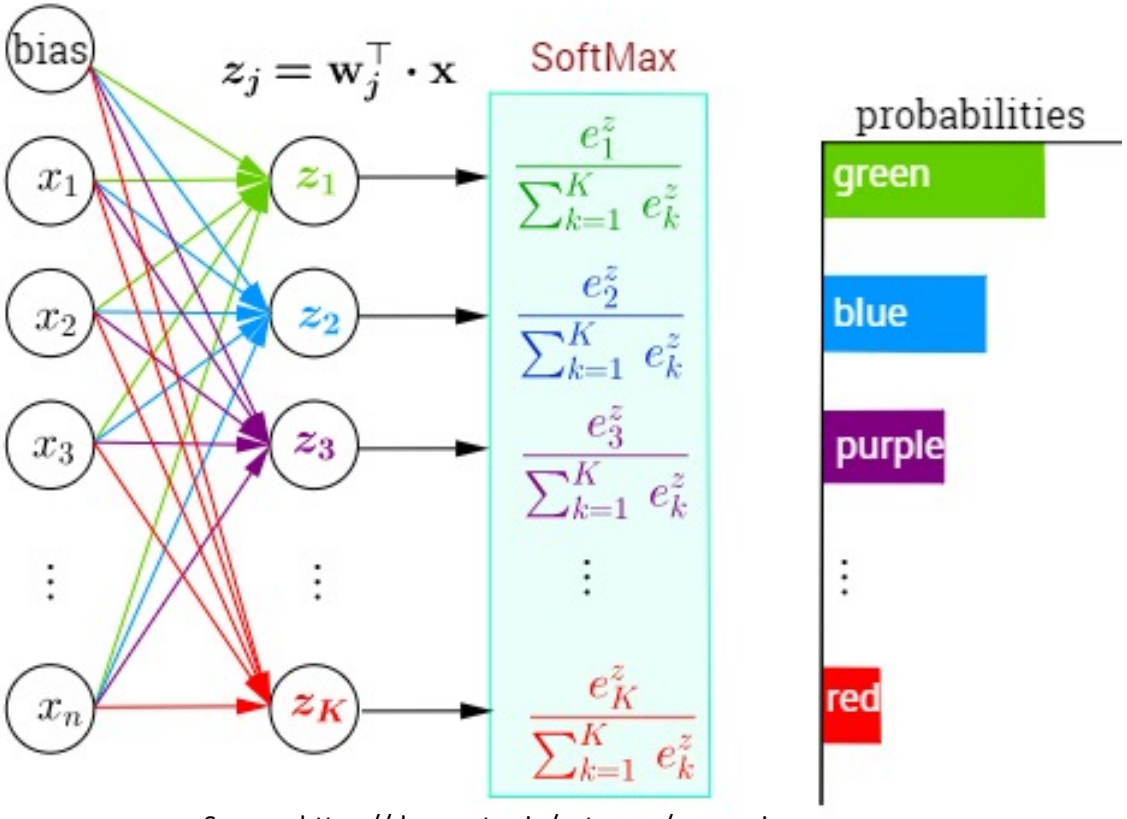
Values in the range from -1 to 1

RELU FUNCTION (RECTIFIED LINEAR UNITS)

$$Y = \max(0, x)$$

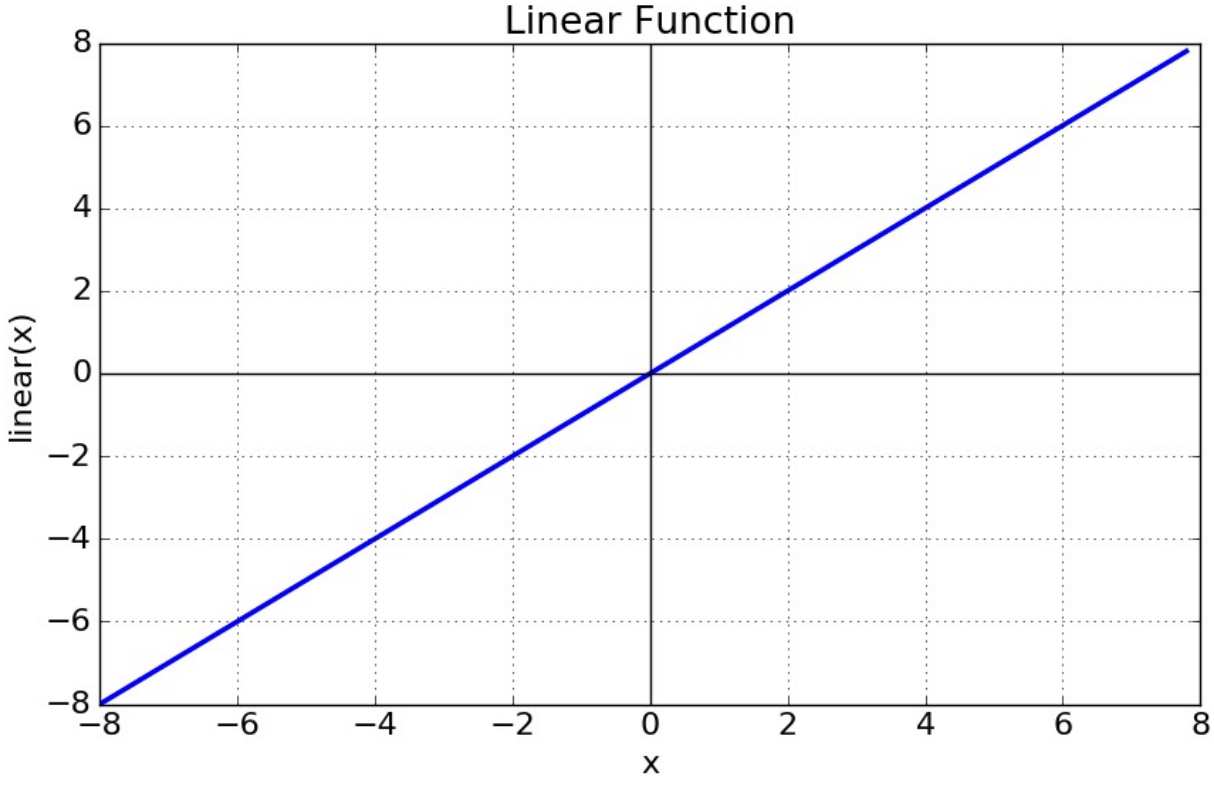
SOFTMAX FUNCTION

$$Y = \frac{e(x)}{\sum \quad e(x)}$$



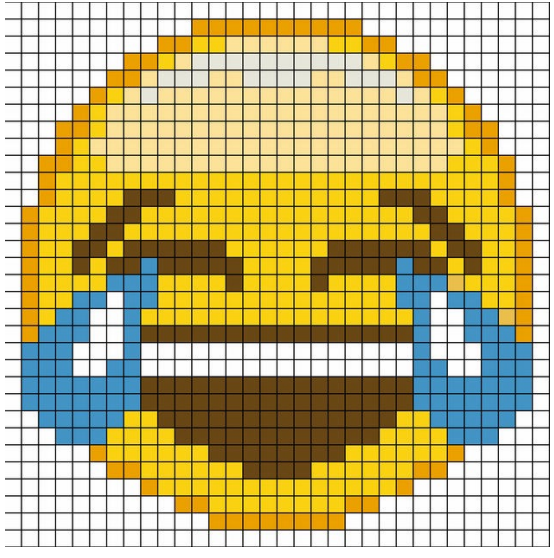
Source: https://deepnotes.io/category/cnn-series

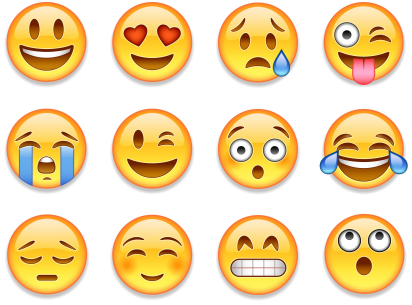
LINEAR FUNCTION

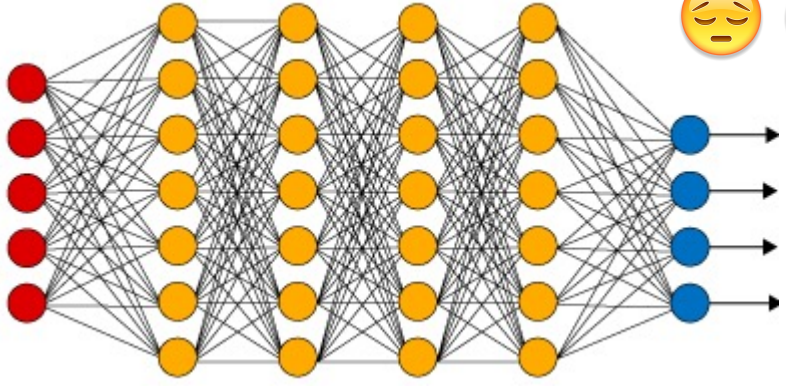


PIXELS

32 x 32 = 1.024 x 3 = **3.072 inputs**





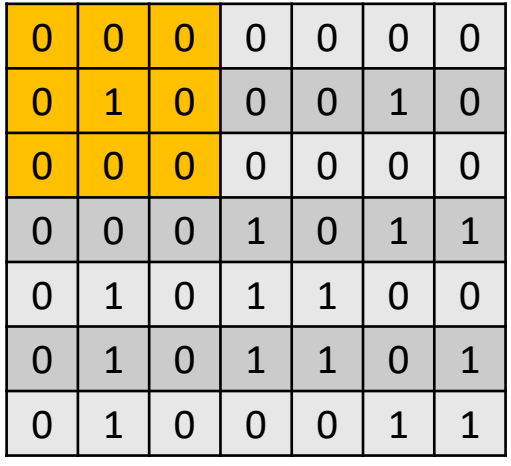


- It does not use all pixels
- It applies a dense neural network, but at the beginning it transforms the data
- What are the most important features?

CONVOLUTIONAL NEURAL NETWORK STEPS

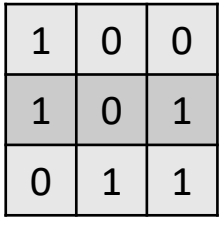
1. Convolution operation
2. Pooling
3. Flattening
4. Dense neural network

STEP 1: CONVOLUTION OPERATION



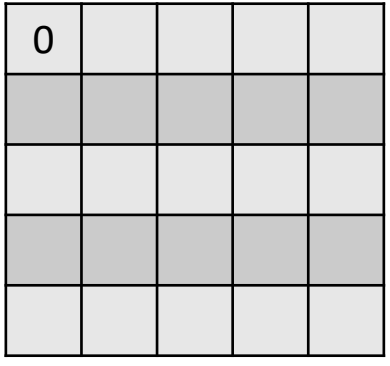
Image

X



Feature detector

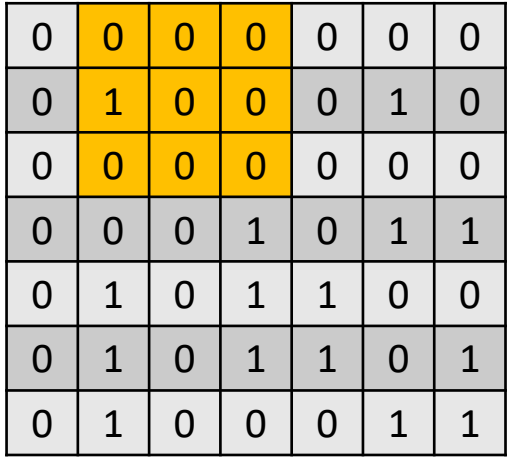
=



Feature map

0 * 1 + 0 * 0 + 0 * 0 + 0 * 1 + 1 * 0 + 0 * 1 + 0 * 0 + 0 * 1 + 0 * 1 = 0

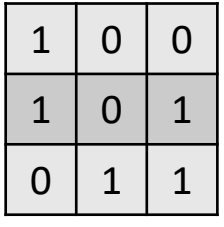
STEP 1: CONVOLUTION OPERATION



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	1	0	1	1	0	0
0	1	0	1	1	0	1
0	1	0	0	0	1	1

Image

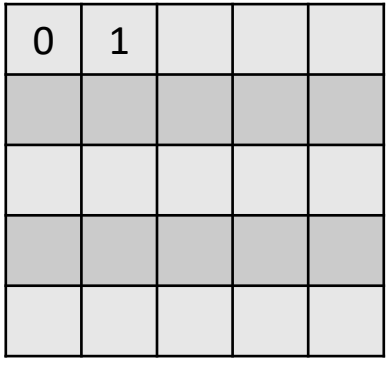
X



1	0	0
1	0	1
0	1	1

Feature detector

=

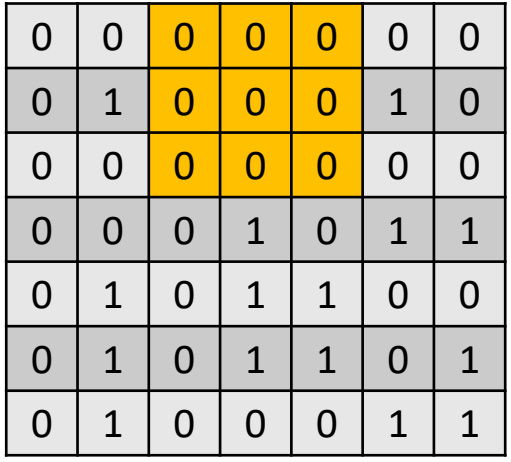


0	1			

Feature map

0 * 1 + 0 * 0 + 0 * 0 + 1 * 1 + 0 * 0 + 0 * 1 + 0 * 0 + 0 * 1 + 0 * 1 = 1

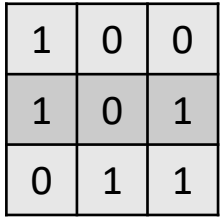
STEP 1: CONVOLUTION OPERATION



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	1	0	1	1	0	0
0	1	0	1	1	0	1
0	1	0	0	0	1	1

Image

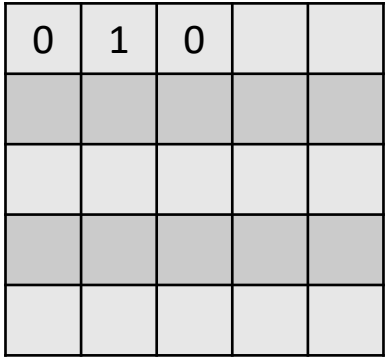
X



1	0	0
1	0	1
0	1	1

Feature detector

=

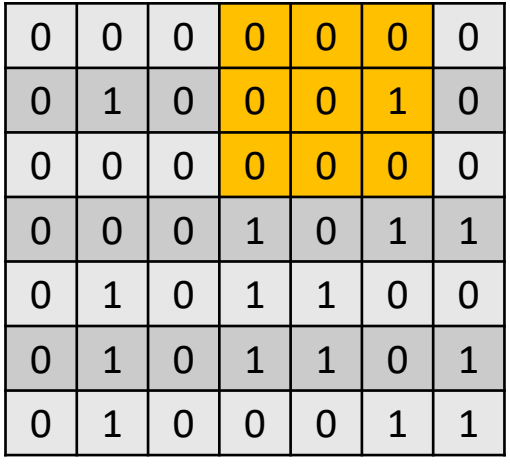


0	1	0		

Feature map

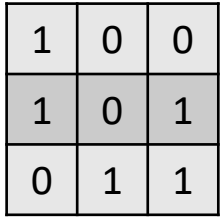
0 * 1 + 0 * 0 + 0 * 0 + 0 * 1 + 0 * 0 + 0 * 1 + 0 * 0 + 0 * 1 + 0 * 1 = 0

STEP 1: CONVOLUTION OPERATION



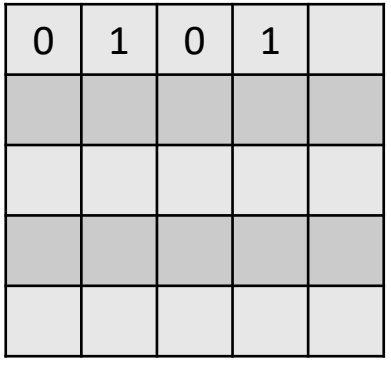
Image

X



Feature detector

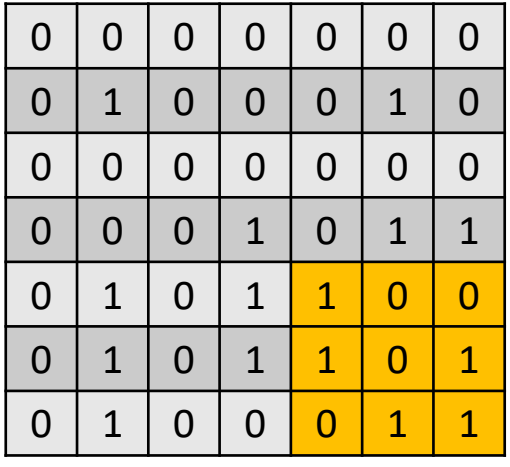
=



Feature map

0 * 1 + 0 * 0 + 0 * 0 + 0 * 1 + 0 * 0 + 1 * 1 + 0 * 0 + 0 * 1 + 0 * 1 = 1

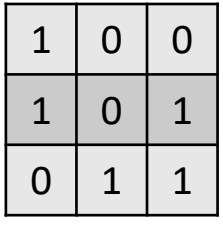
STEP 1: CONVOLUTION OPERATION



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	1	0	1	1	0	0
0	1	0	1	1	0	1
0	1	0	0	0	1	1

Image

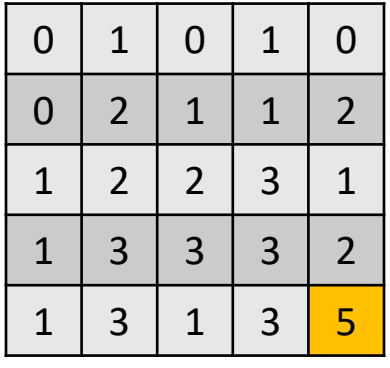
X



1	0	0
1	0	1
0	1	1

Feature detector

=



0	1	0	1	0
0	2	1	1	2
1	2	2	3	1
1	3	3	3	2
1	3	1	3	5

Feature map

1 * 1 + 0 * 0 + 0 * 0 + 1 * 1 + 0 * 0 + 1 * 1 + 0 * 0 + 1 * 1 + 1 * 1 = 5

STEP 1: CONVOLUTION OPERATION – RELU

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	1	0	1	1	0	0
0	1	0	1	1	0	1
0	1	0	0	0	1	1

Image

X

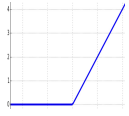
1	0	0
1	0	1
0	1	1

Feature detector

=

0	1	0	1	0
0	2	1	1	2
1	2	2	3	1
1	3	3	3	2
1	3	1	3	5

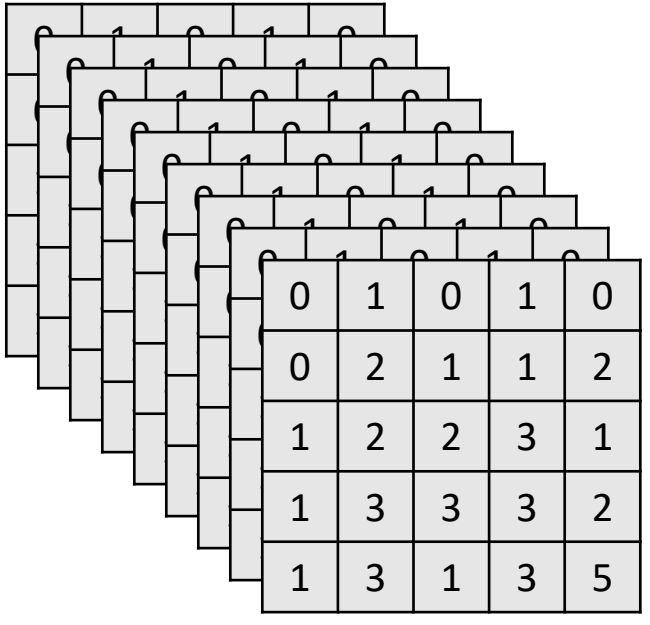
Feature map



STEP 1: CONVOLUTIONAL LAYER

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	1	0	1	1	0	0
0	1	0	1	1	0	1
0	1	0	0	0	1	1

Image

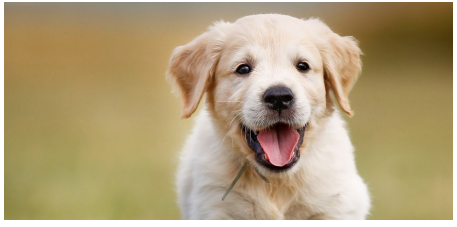


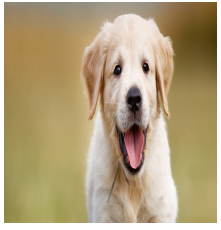
Feature maps

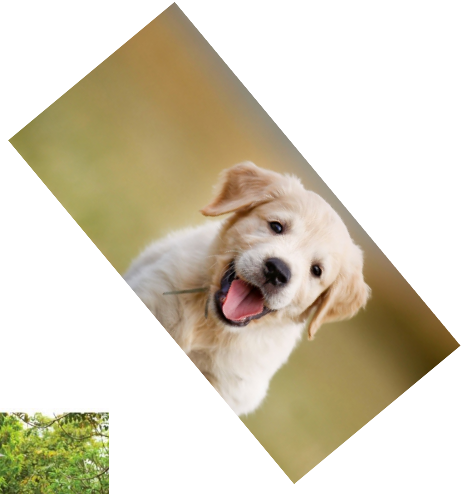
The network will decide which feature detector to use

The convolutional layer is the set of feature maps

STEP 2: POOLING







STEP 2: POOLING

0	1	0	1	0
0	2	1	1	2
1	2	2	3	1
1	3	3	3	2
1	3	1	3	5

Feature map

2		

STEP 2: POOLING

0	1	0	1	0
0	2	1	1	2
1	2	2	3	1
1	3	3	3	2
1	3	1	3	5

Feature map

2	1	

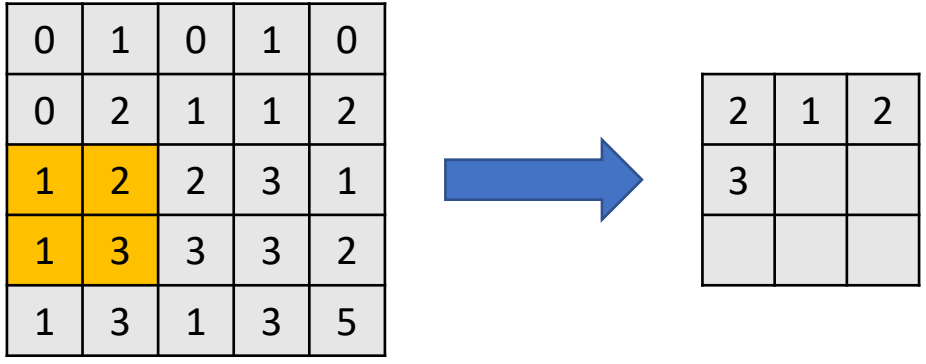
STEP 2: POOLING

0	1	0	1	0
0	2	1	1	2
1	2	2	3	1
1	3	3	3	2
1	3	1	3	5

Feature map

2	1	2

STEP 2: POOLING



Feature map

STEP 2: POOLING

0	1	0	1	0
0	2	1	1	2
1	2	2	3	1
1	3	3	3	2
1	3	1	3	5

Feature map

2	1	2
3	3	

STEP 2: POOLING

0	1	0	1	0
0	2	1	1	2
1	2	2	3	1
1	3	3	3	2
1	3	1	3	5

Feature map

2	1	2
3	3	2

STEP 2: POOLING

0	1	0	1	0
0	2	1	1	2
1	2	2	3	1
1	3	3	3	2
1	3	1	3	5

Feature map

2	1	2
3	3	2
3		

STEP 2: POOLING

0	1	0	1	0
0	2	1	1	2
1	2	2	3	1
1	3	3	3	2
1	3	1	3	5

Feature map

2	1	2
3	3	2
3	3	

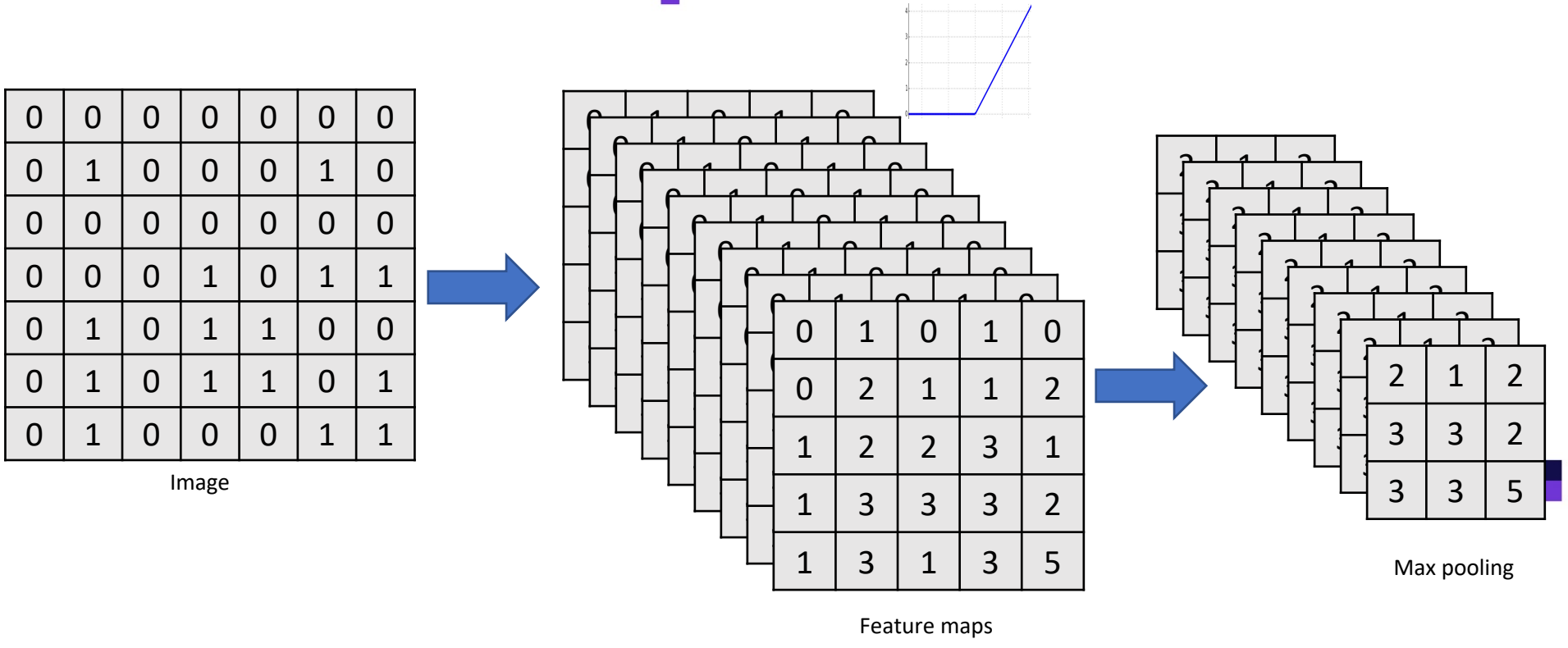
STEP 2: POOLING

0	1	0	1	0
0	2	1	1	2
1	2	2	3	1
1	3	3	3	2
1	3	1	3	5

Feature map

2	1	2
3	3	2
3	3	5

CONVOLUTIONAL NEURAL NETWORK – POOLING

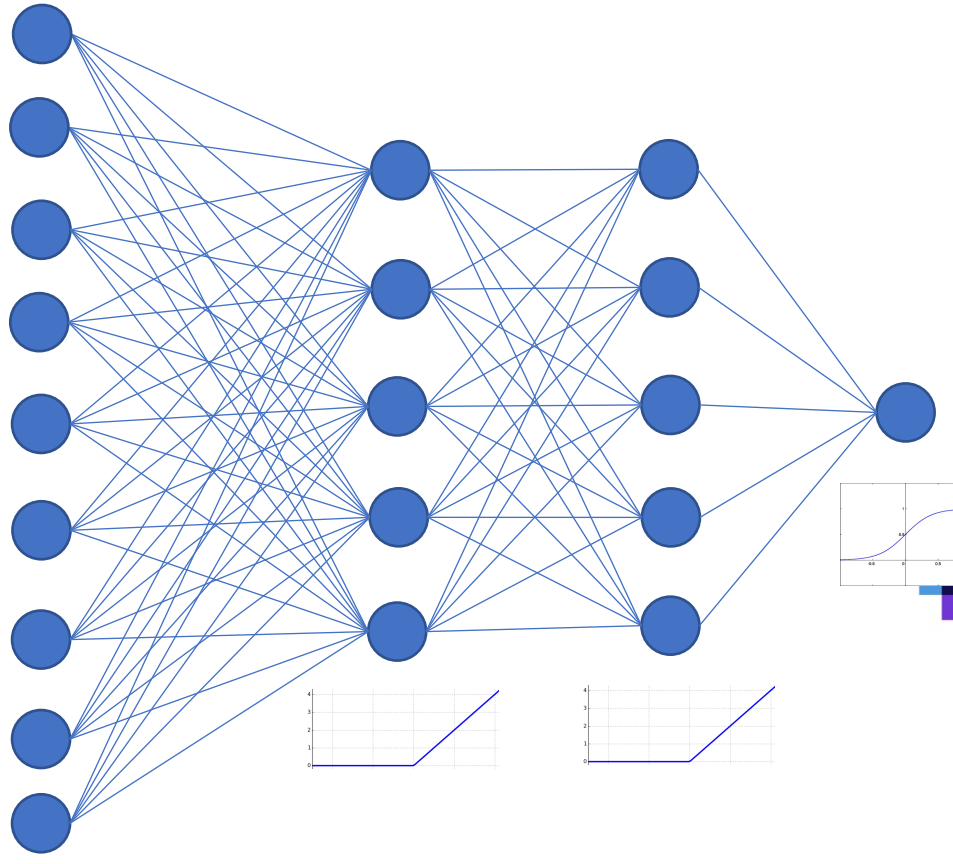


STEP 3: FLATTENING

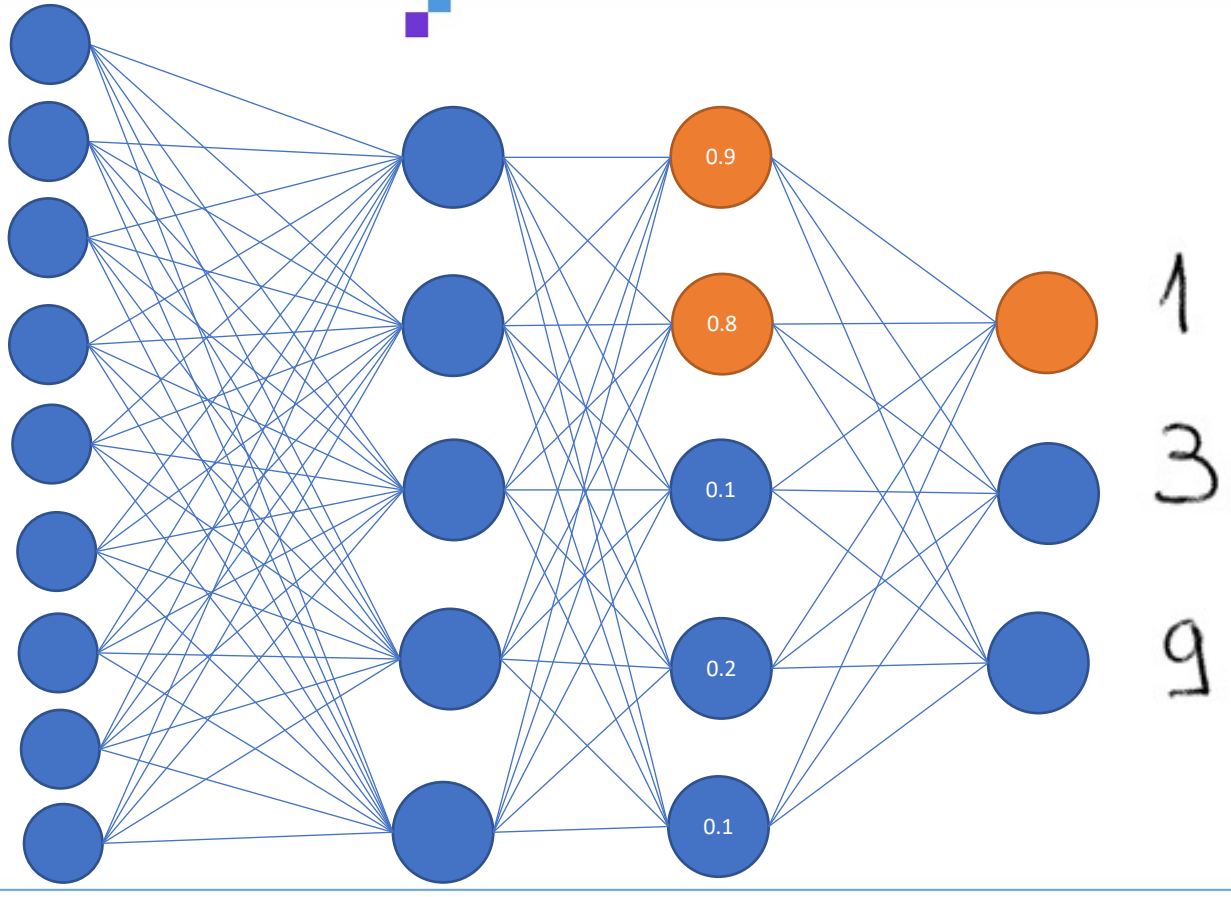
2	1	2
3	3	2
3	3	5

Pooled feature map

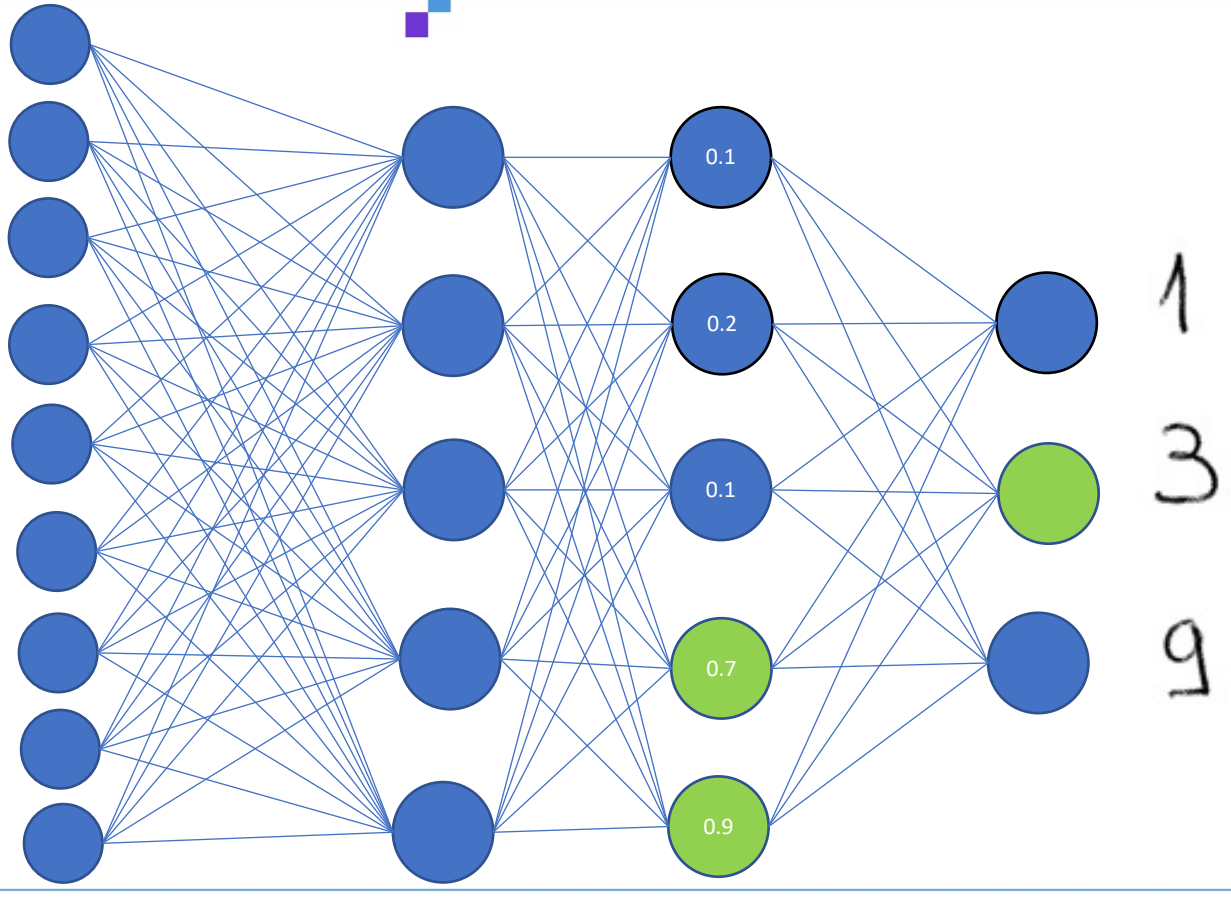
2
1
2
3
3
2
3
3
5



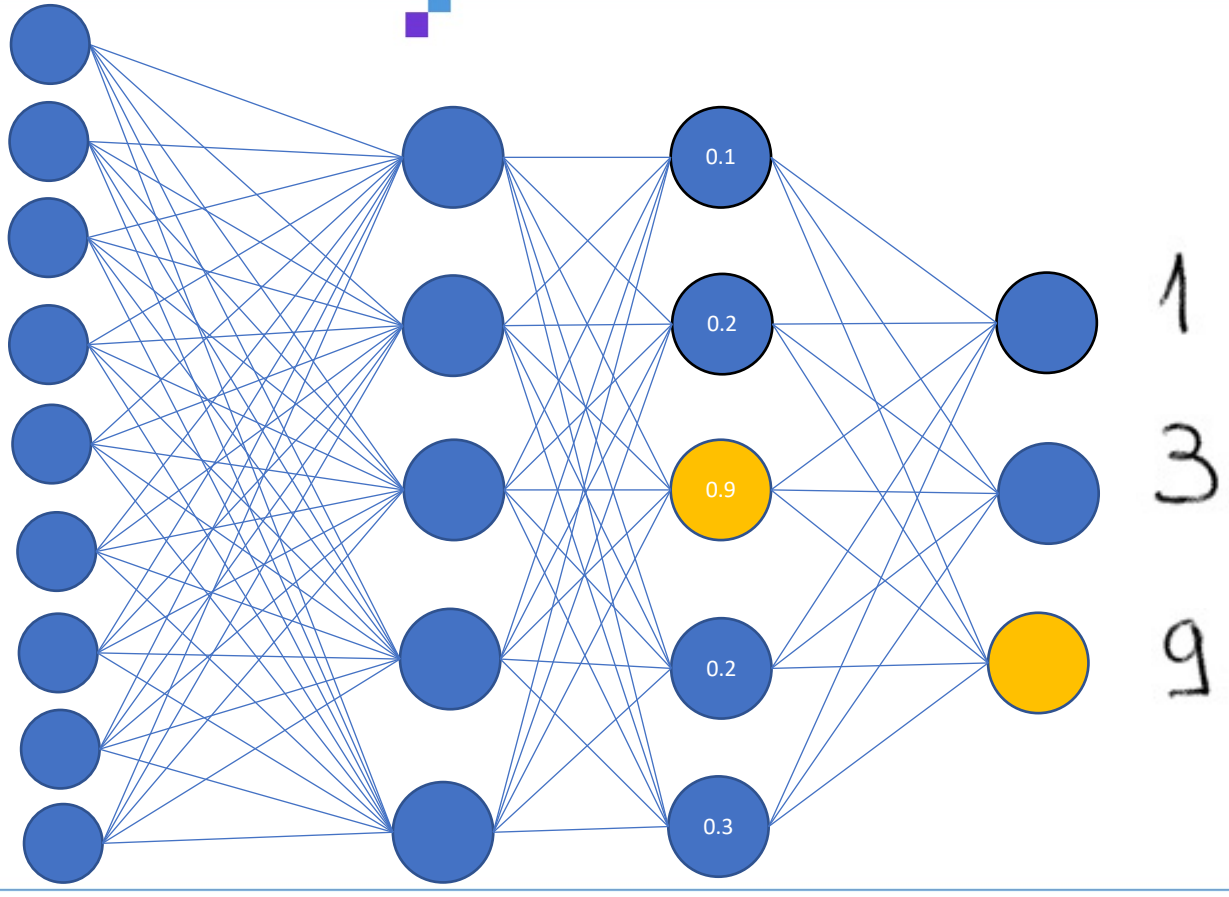
STEP 3: DENSE NEURAL NETWORK



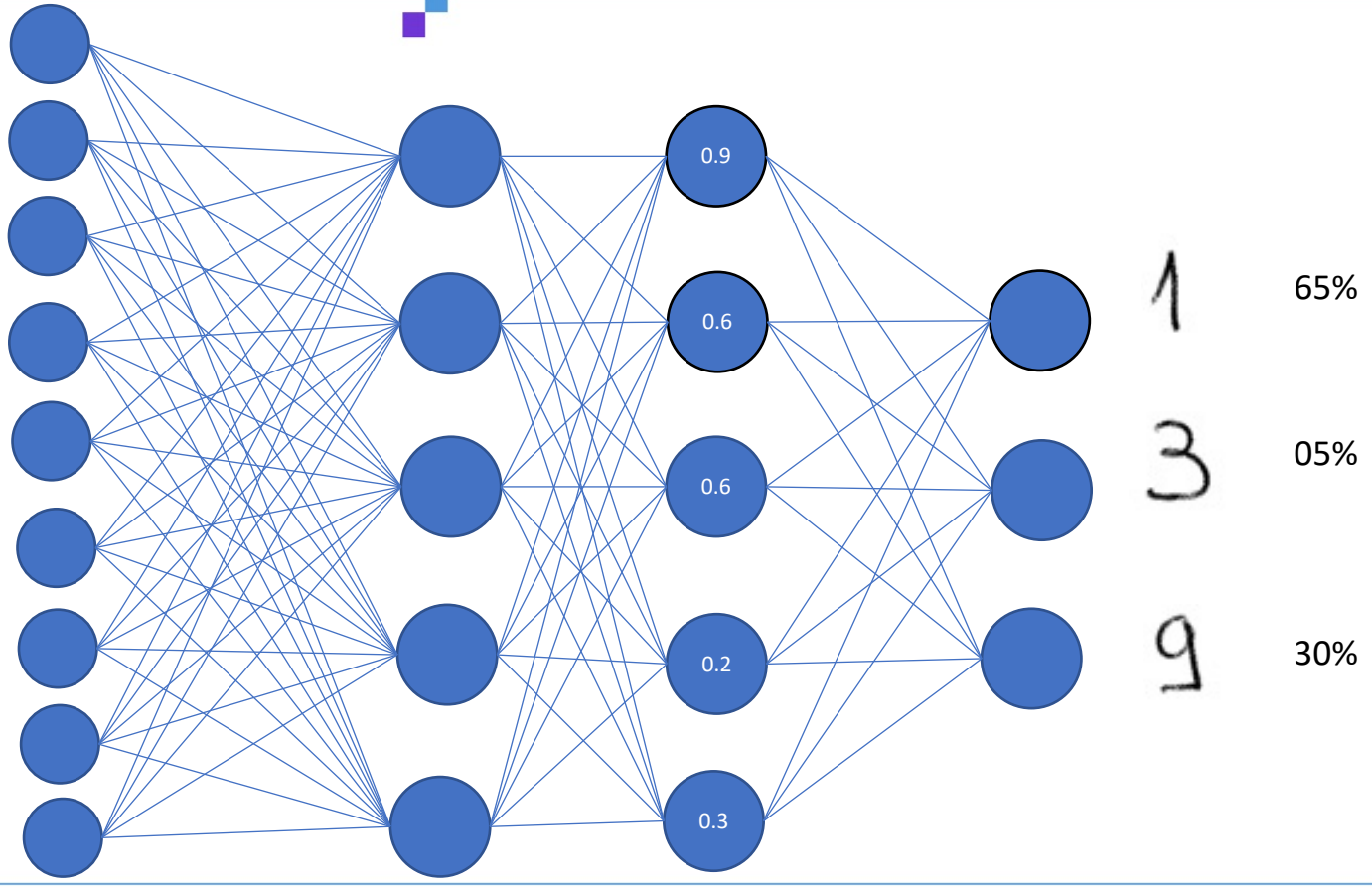
STEP 3: DENSE NEURAL NETWORK



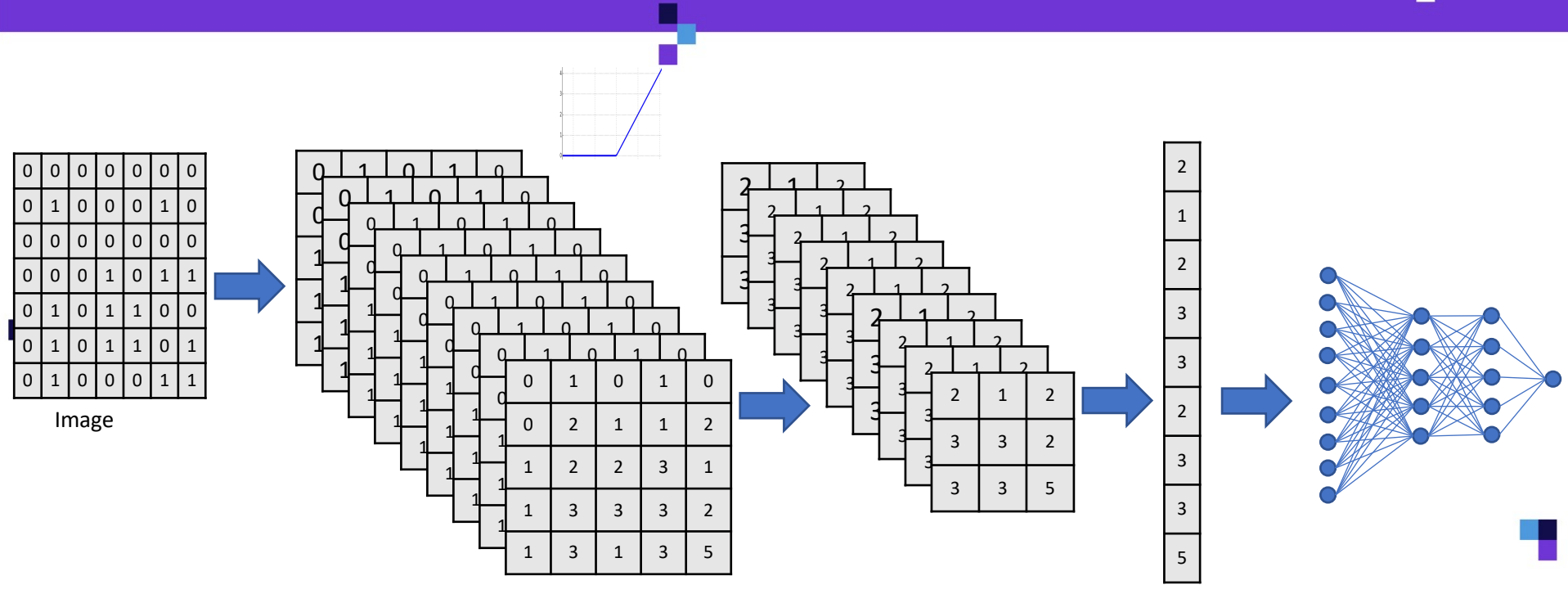
STEP 3: DENSE NEURAL NETWORK



STEP 3: DENSE NEURAL NETWORK



CONVOLUTIONAL NEURAL NETWORK



Training using gradient descent

In addition to adjusting the weights, the feature detector is also changed