



SuperDataScience

COMPUTER VISION



Part 1 Face detection	Part 2 Face recognition	Part 3 Object tracking	Part 4 Neural networks for image classification	Part 5 Convolutional neural networks for image classification
Part 6 Transfer learning and fine tuning	Part 7 Neural networks for classification of emotions	Part 8 Autoencoders	Part 9 Object detection with YOLO	Part 10 Recognition of gestures and actions
Part 11 Deep dream	Part 12 Style transfer	Part 13 GANs (Generative Adversarial Networks)	Part 14 Image segmentation	

COMPUTER VISION MASTERCLASS

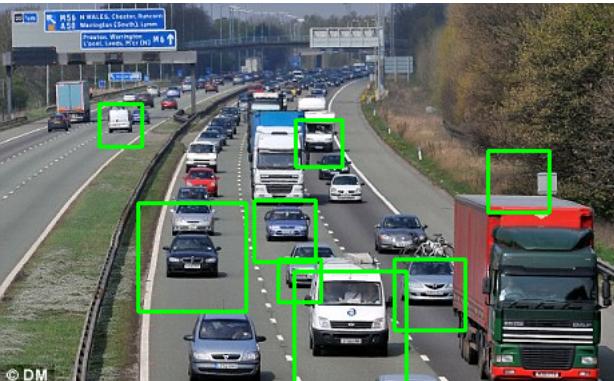


- Basic python programming
- Only the basic intuition (except for neural networks)
- Libraries
 - OpenCV
 - Dlib
 - TensorFlow
 - Darknet
 - Caffe framework
- Homeworks

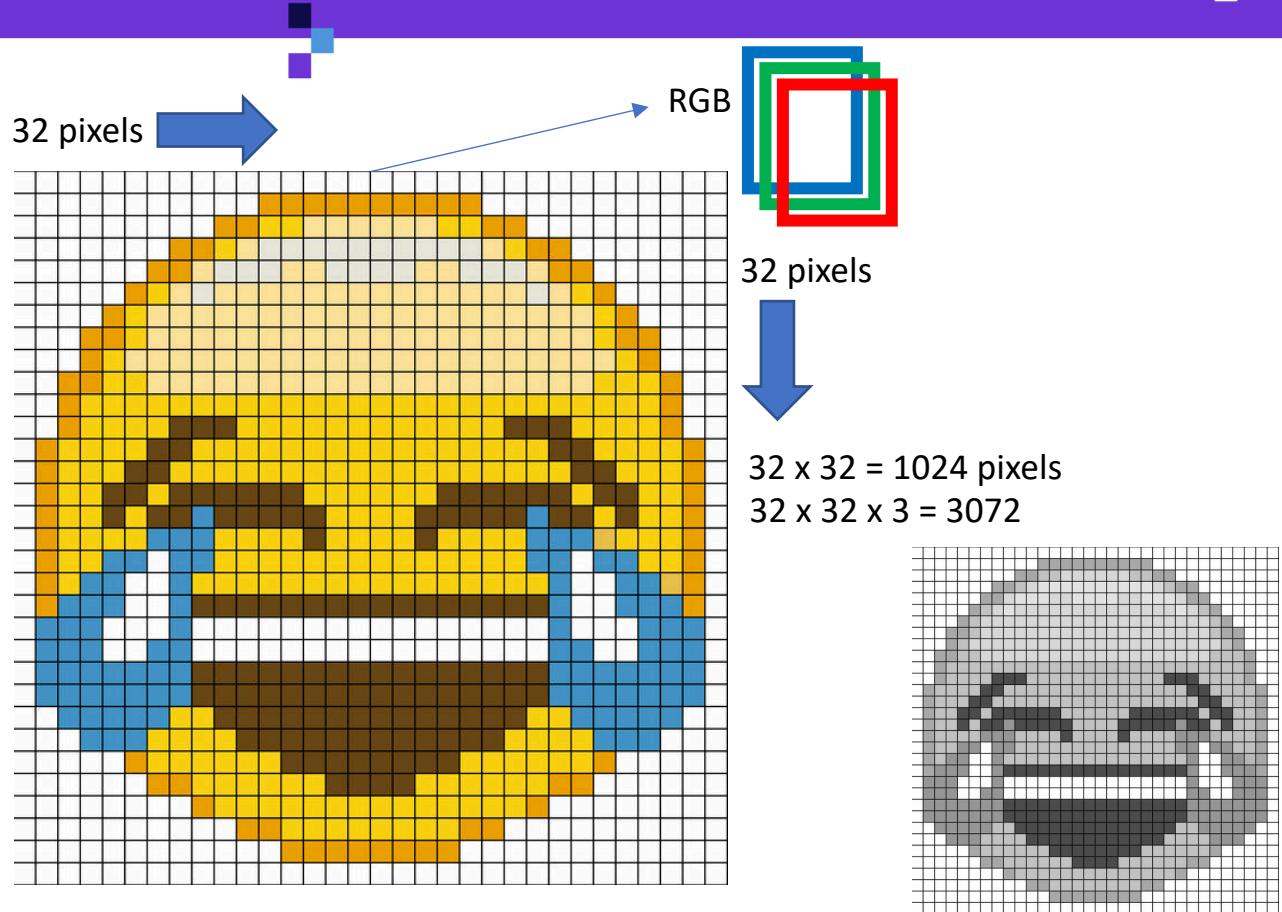


PLAN OF ATTACK – FACE DETECTION

1. Face detection with Haarcascade and OpenCV
2. Face detection with HOG and Dlib
3. Face detection with CNN and Dlib
4. Face detection using webcam



PIXELS

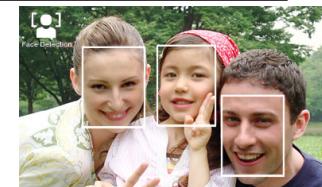
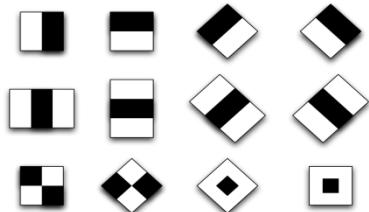


CASCADE CLASSIFIER



AdaBoost
Training

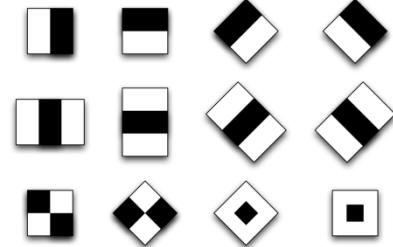
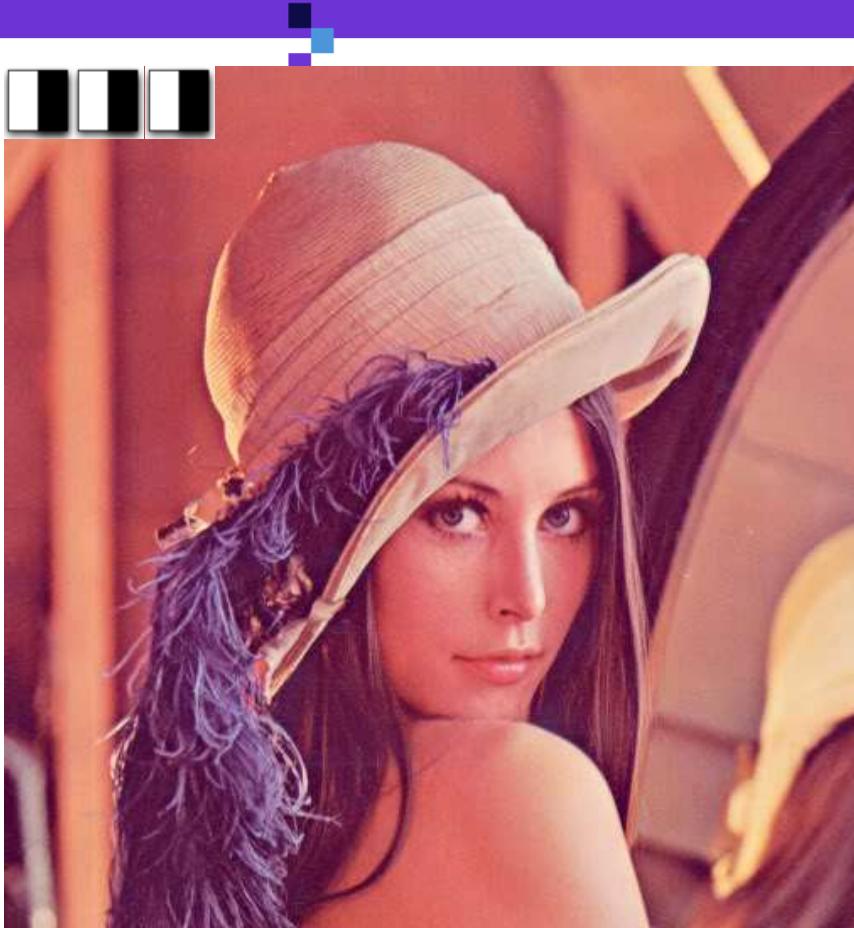
Feature
selection



Apply to each “sub-window”



CASCADE CLASSIFIER



2 3 5 6
8 9 2 1



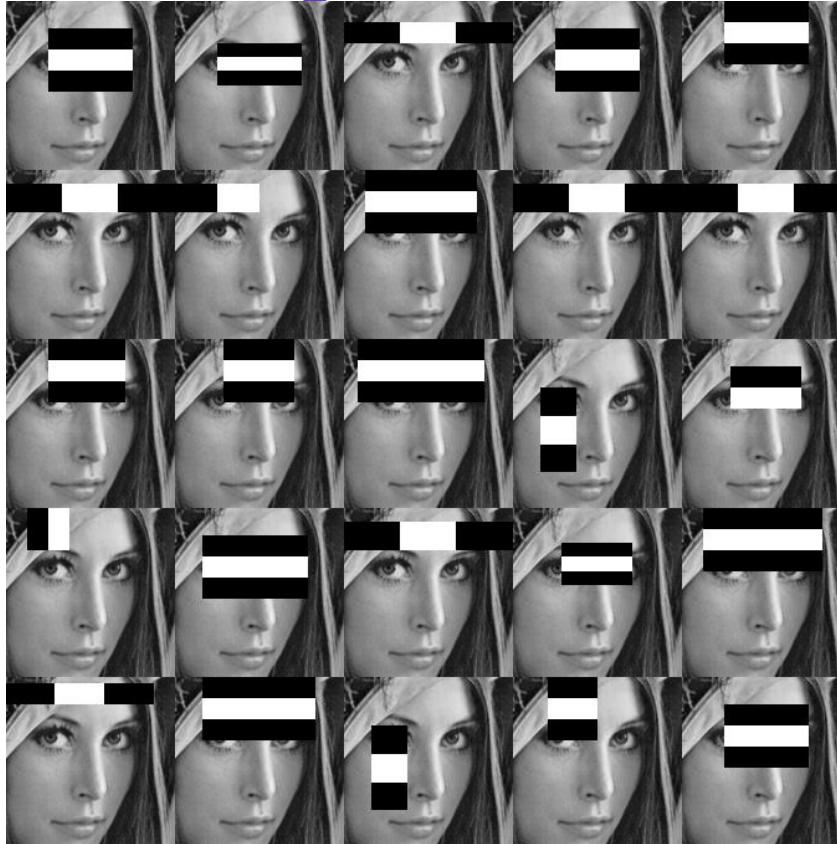
Sum of white pixels – sum
of black pixels



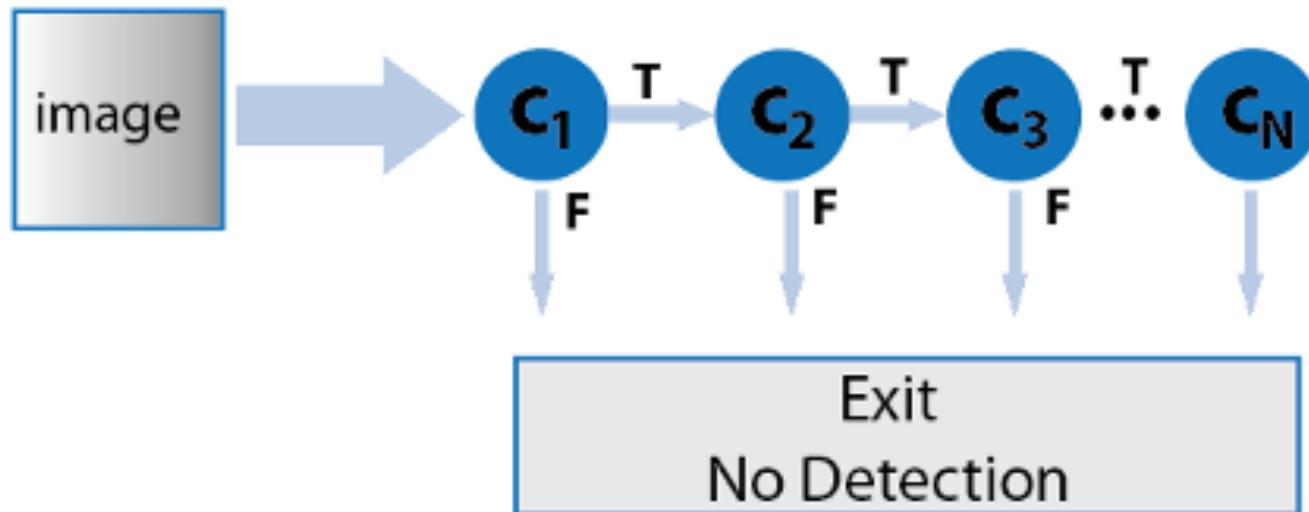
More than 160.000
combinations in a 24x24
image!



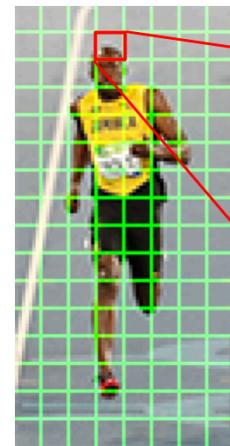
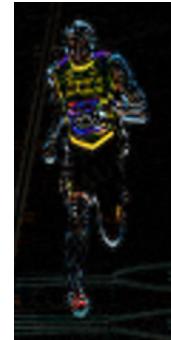
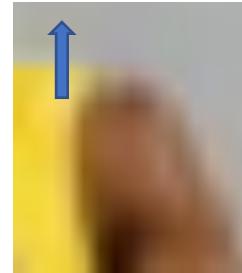
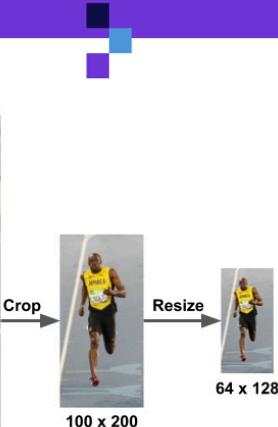
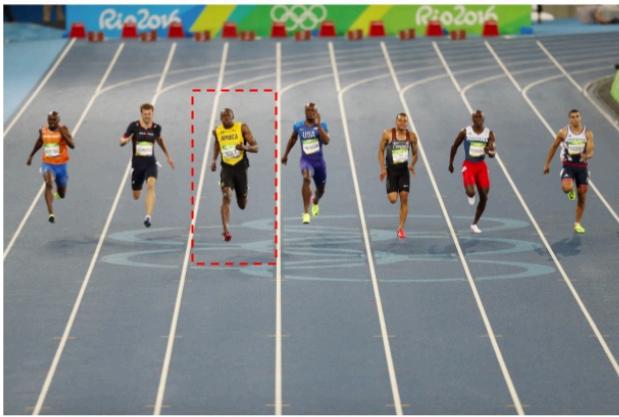
CASCADE CLASSIFIER



CASCADE CLASSIFIER



HOG – HISTOGRAMS OF ORIENTED GRADIENTS



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

Source of images: <https://www.learnopencv.com/histogram-of-oriented-gradients/>



HOG – HISTOGRAMS OF ORIENTED GRADIENTS

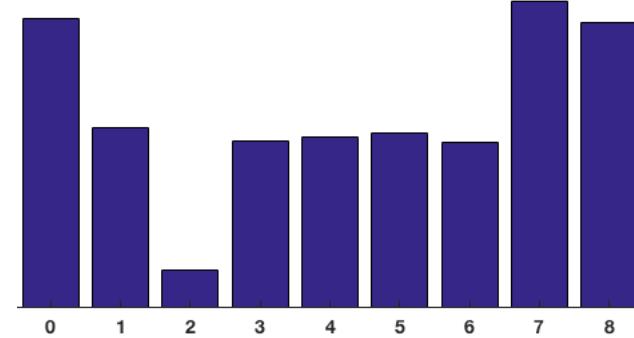
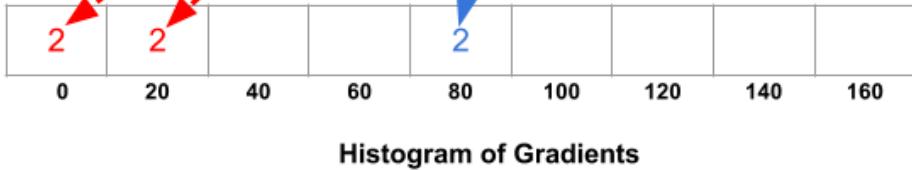


80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

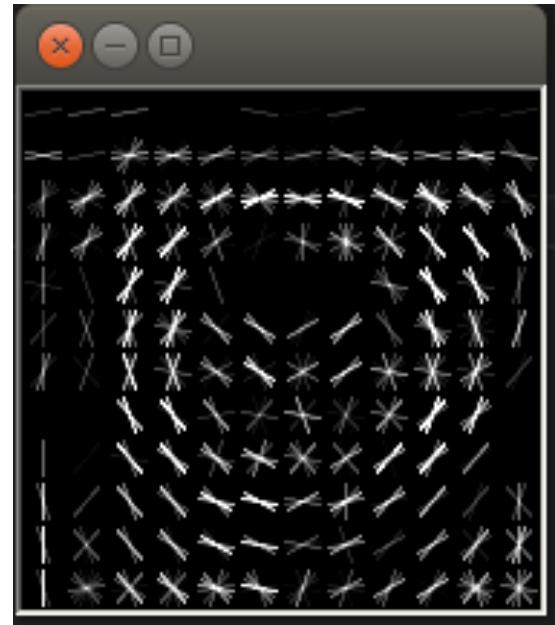
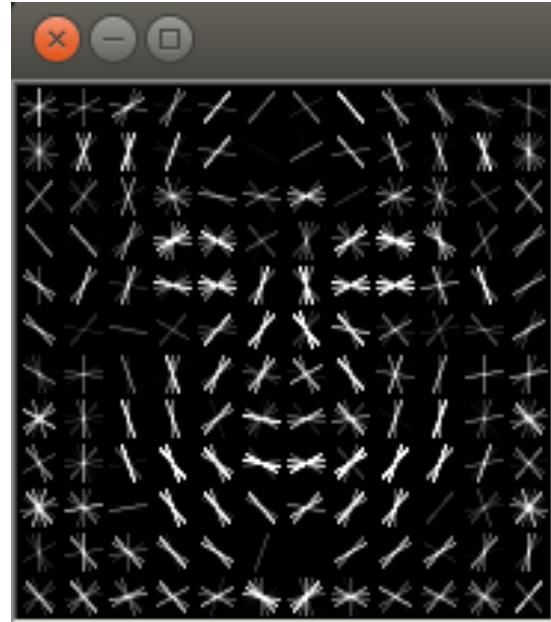
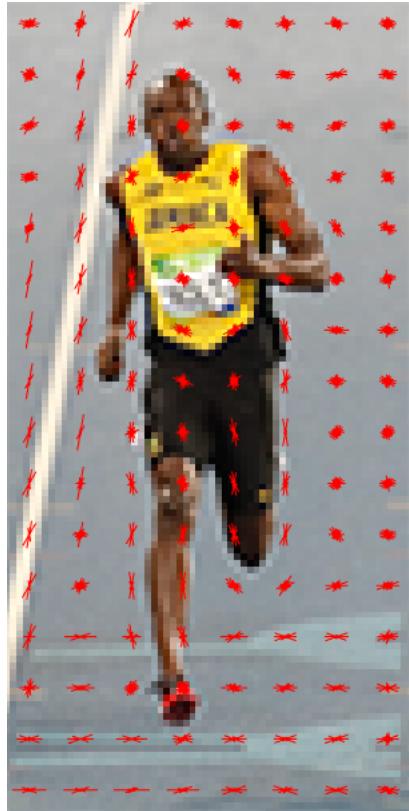
Gradient Direction

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
2	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude



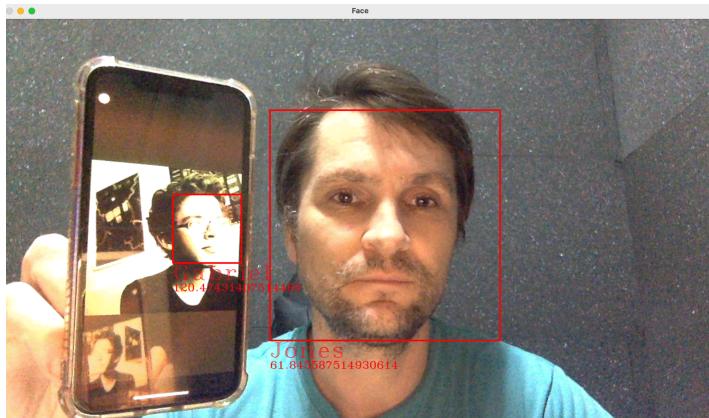
HOG – HISTOGRAMS OF ORIENTED GRADIENTS



PLAN OF ATTACK – FACE RECOGNITION



1. Face recognition with LBPH and OpenCV
2. Face recognition with Dlib, CNN and distance calculation
3. Face recognition using the webcam

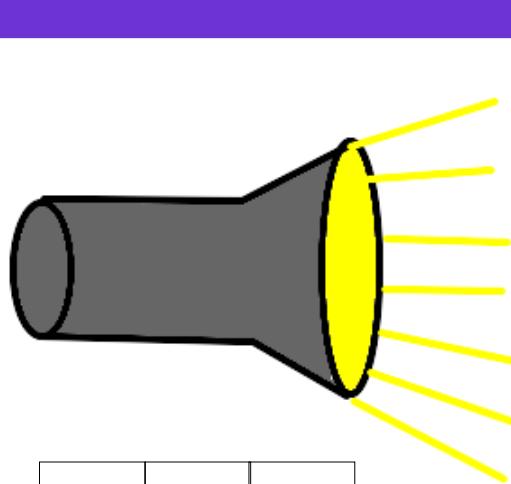


LBPH (LOCAL BINARY PATTERNS HISTOGRAMS)



12	15	18
5	8	3
8	1	2

If ≥ 8 : 1
If < 8 : 0



1	1	1
0	8	0
1	0	0

42	55	48
35	38	33
38	30	32

Binary = 11100010

Binary = 11100010

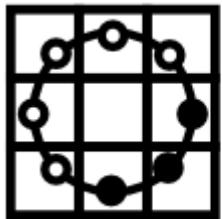
Decimal = 226



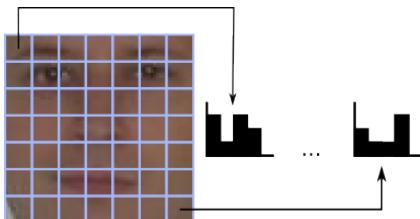
LBPH (LOCAL BINARY PATTERNS HISTOGRAMS)



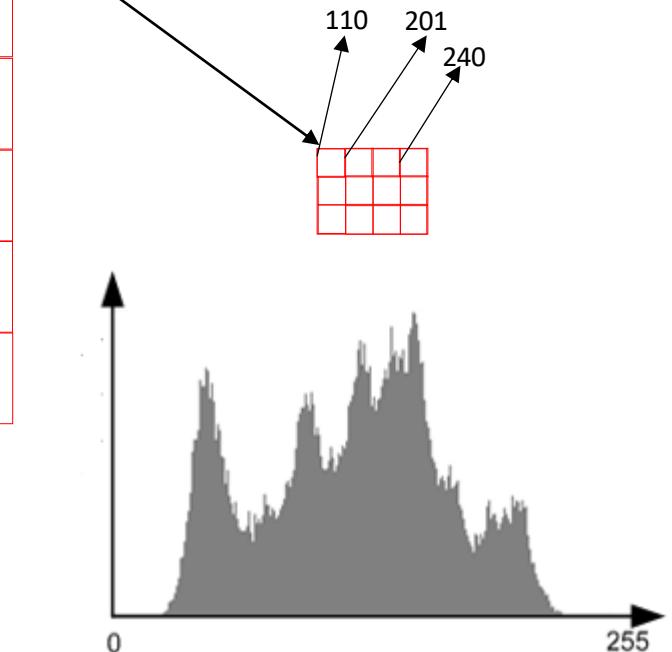
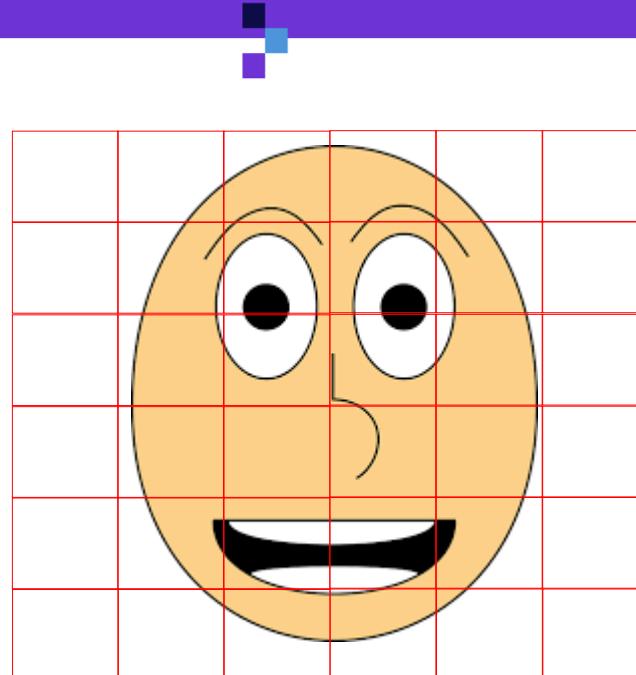
Edge



Corner

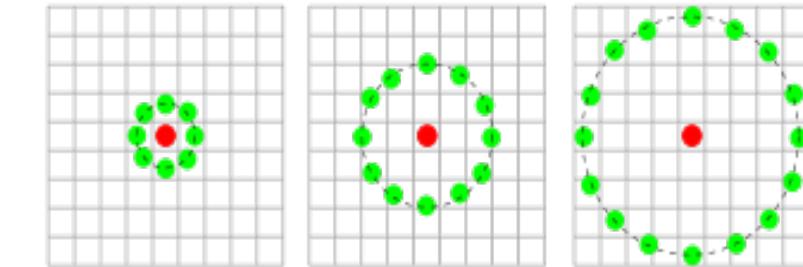
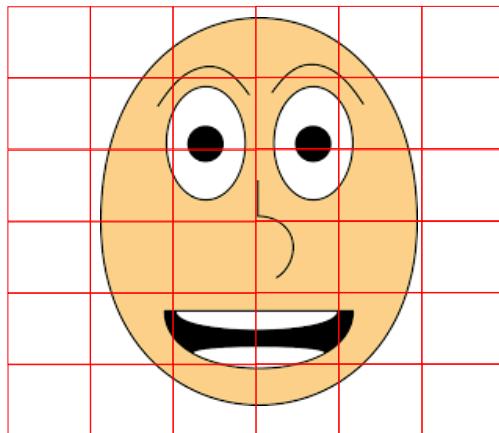


Source: https://bytefish.de/blog/local_binary_patterns/



LBPH (LOCAL BINARY PATTERNS HISTOGRAMS)

1. Radius
2. Neighbors
3. grid_x and grid_y
4. Threshold

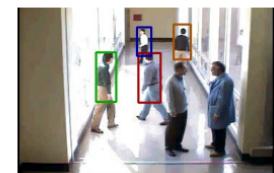
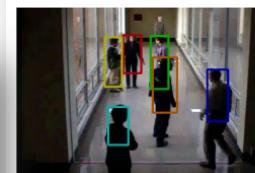
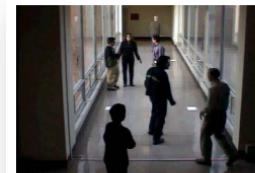
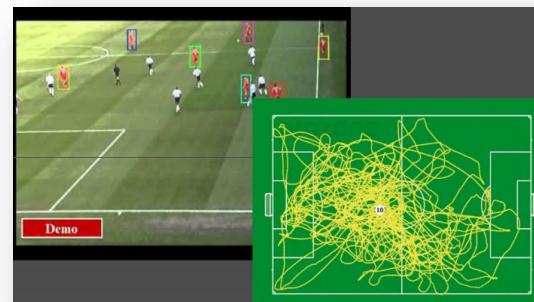
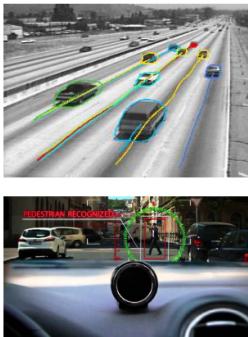


Source: https://en.wikipedia.org/wiki/Local_binary_patterns

PLAN OF ATTACK – OBJECT TRACKING



1. Object tracking vs. Object detection
2. KCF (Kernel Correlation Filters) algorithm
3. CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability) algorithm
4. KCF and CSRT implementation

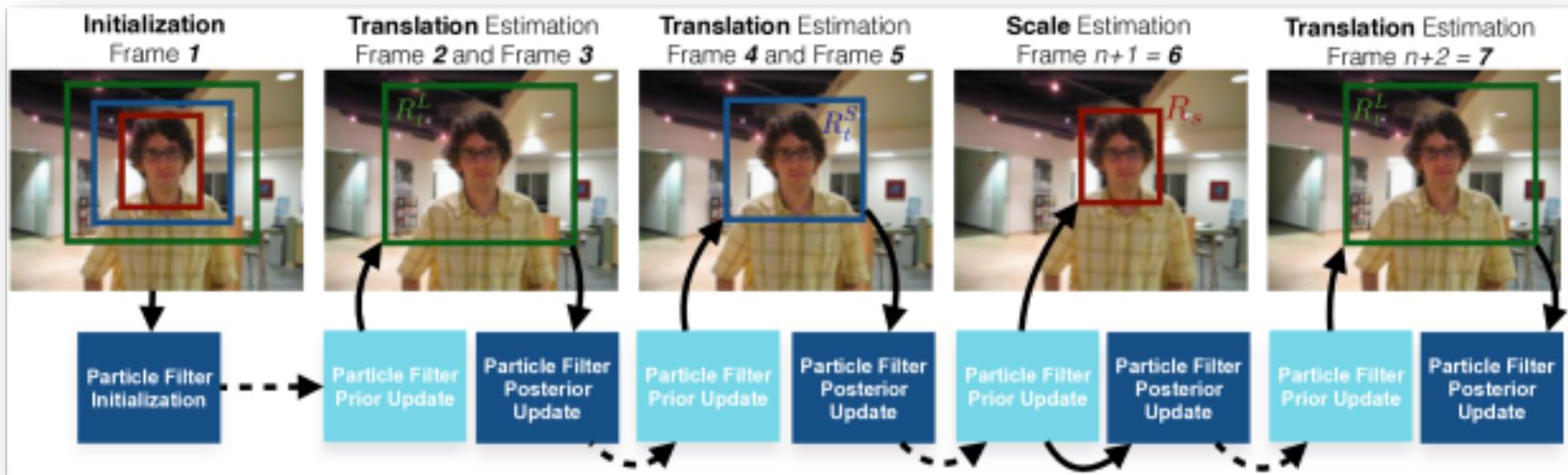


OBJECT TRACKING vs. OBJECT DETECTION

- Object tracking vs. Object detection



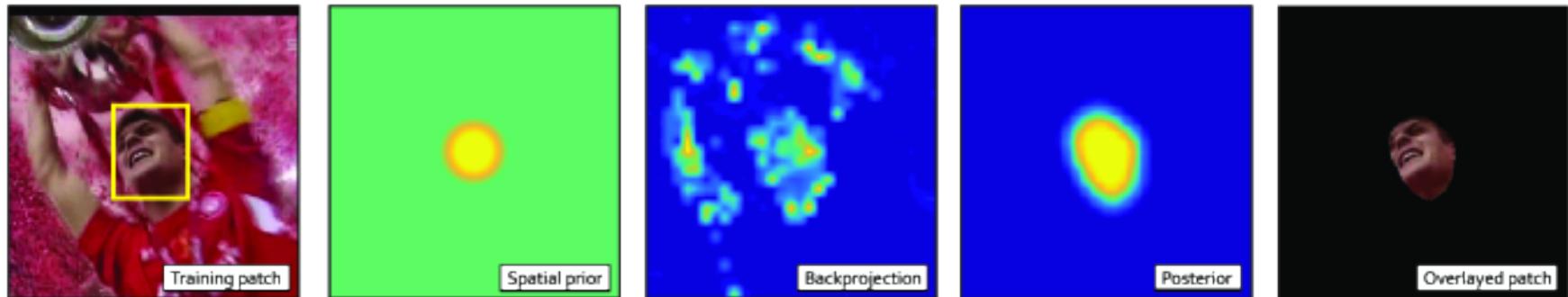
KCF (KERNEL CORRELATION FILTERS)



Source: [https://www.groundai.com/project/enkcf-ensemble-of-kernelized-correlation-filters-for-high-speed-object-tracking/1](https://www.groundai.com/project/enkcf-ensemble-of-kernelized-correlation-filters-for-high-speed-object-tracking/)



CSRT (DISCRIMINATIVE CORRELATION FILTER WITH CHANNEL AND SPATIAL RELIABILITY)



1. From left to right: training patch with the bounding box of the object
2. HOG to extract useful information of the image
3. Application of Random Markov Test to generate probabilities
4. Training patch masked using the confidence map

Source: <https://www.arxiv-vanity.com/papers/1611.08461/>

PLAN OF ATTACK



1. Part 1

Biological fundamentals
Single layer perceptron



2. Part 2

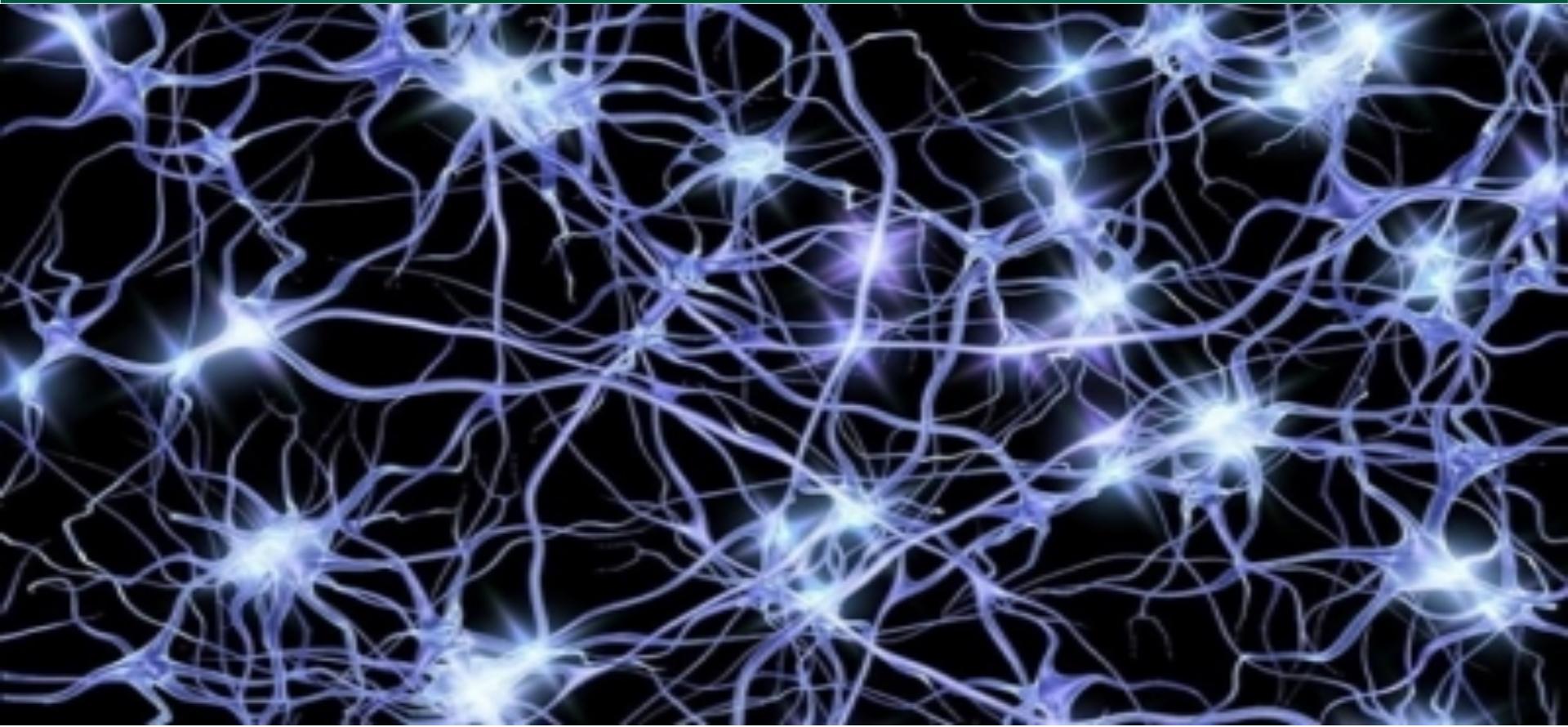
Multi-layer perceptron

3. Part 3

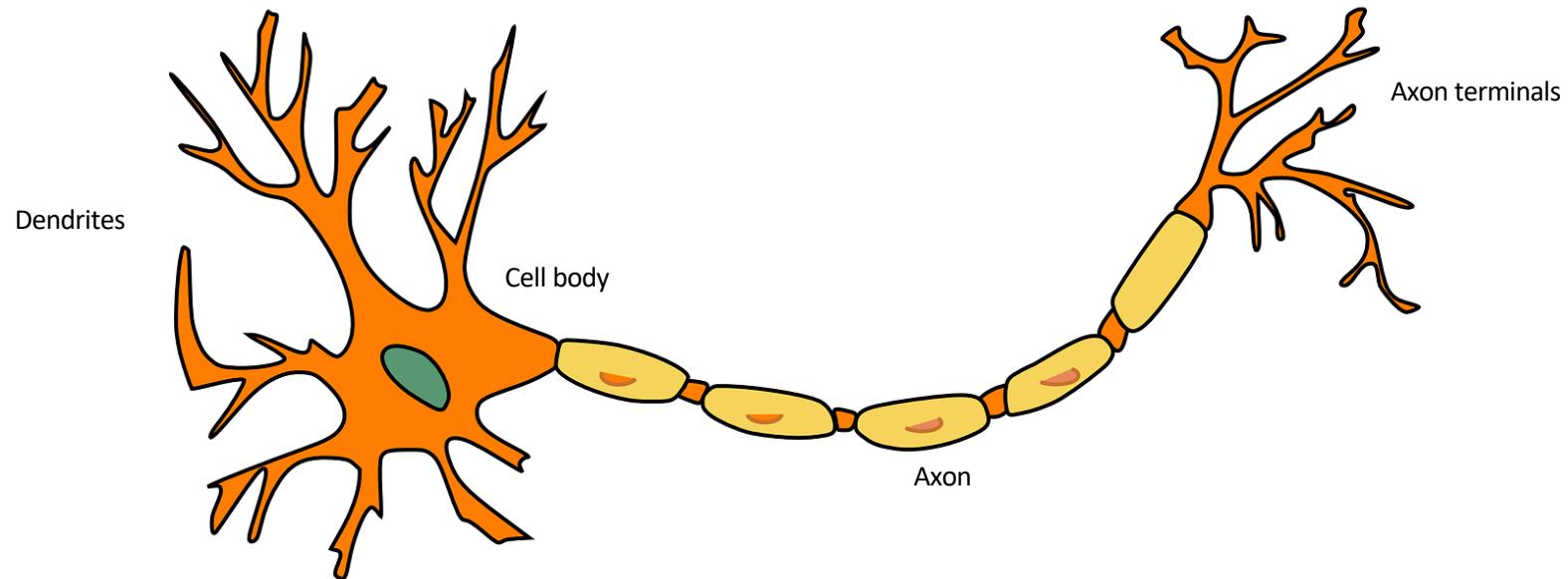
Pybrain
Sklearn
TensorFlow
PyTorch



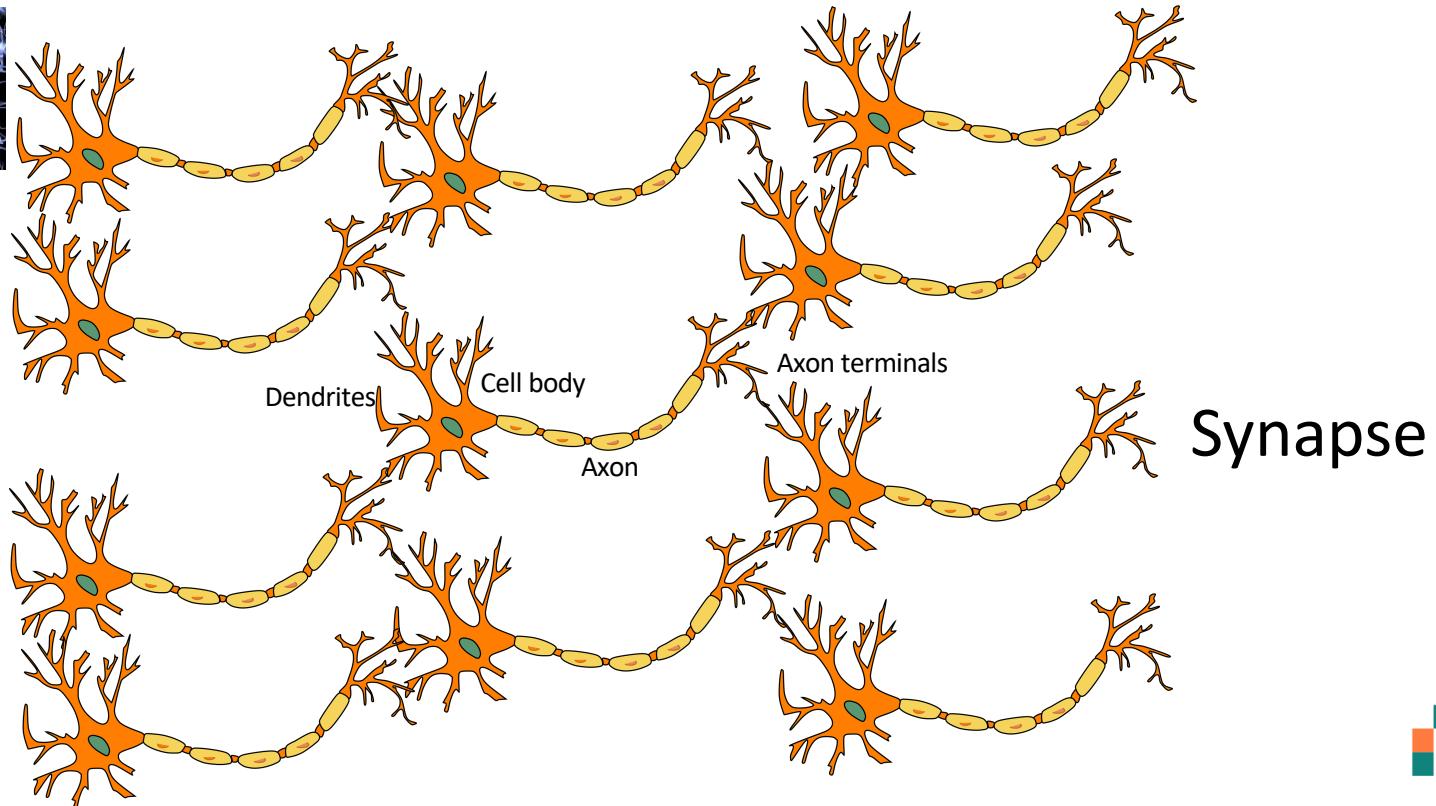
BIOLOGICAL FUNDAMENTALS



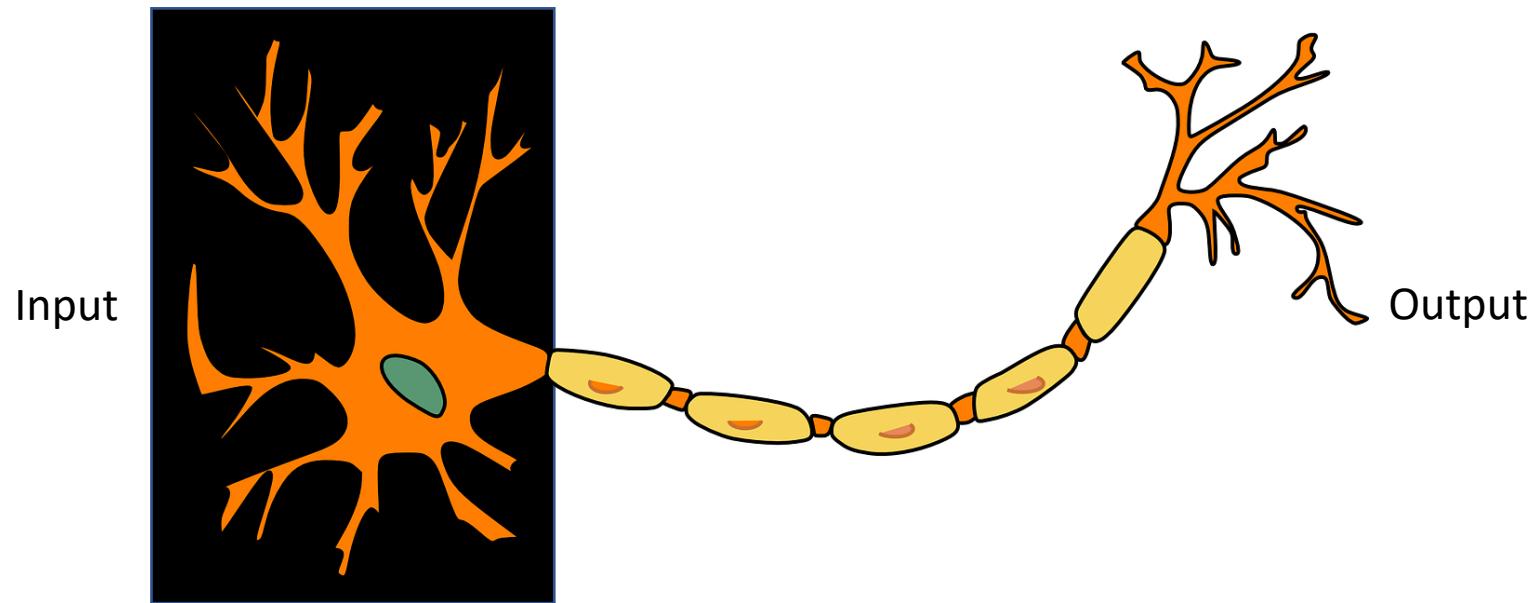
BIOLOGICAL FUNDAMENTALS



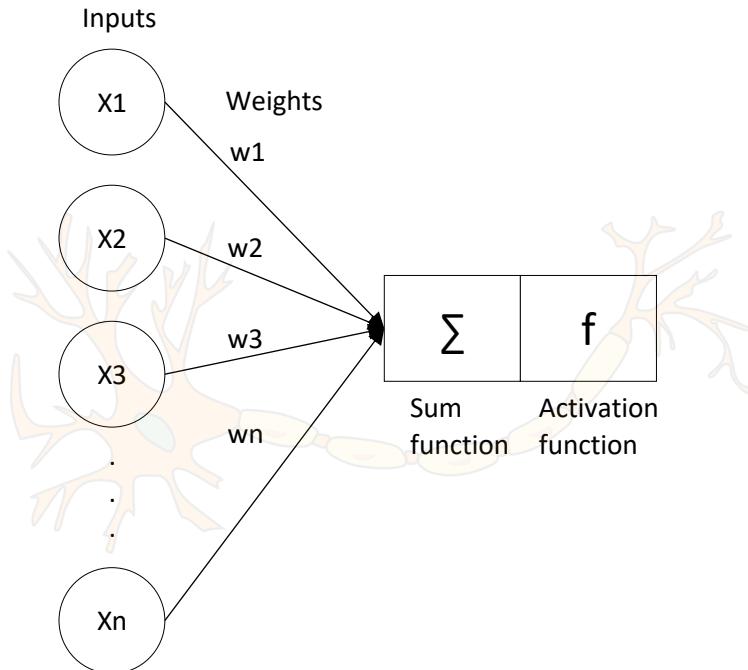
BIOLOGICAL FUNDAMENTALS



ARTIFICIAL NEURON



ARTIFICIAL NEURON

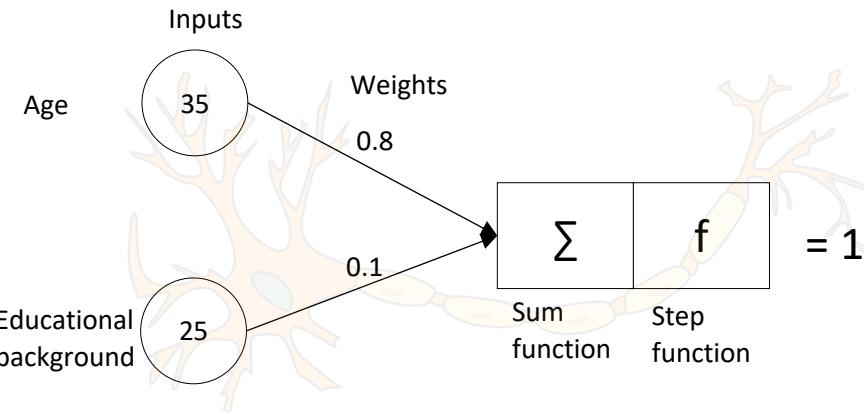


$$sum = \sum_{i=1}^n x_i * w_i$$

$$x_1 * w_1 + x_2 * w_2 + x_3 * w_3$$



PERCEPTRON



$$sum = \sum_{i=1}^n x_i * w_i$$

$$sum = (35 * 0.8) + (25 * 0.1)$$

$$sum = 28 + 2.5$$

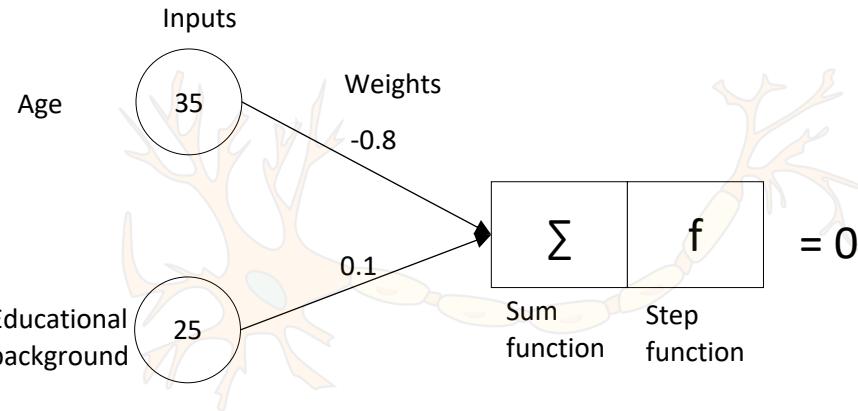
$$sum = 30.5$$

Greater or equal to 1 = 1

Otherwise = 0

"All or nothing" representation

PERCEPTRON



$$sum = \sum_{i=1}^n x_i * w_i$$

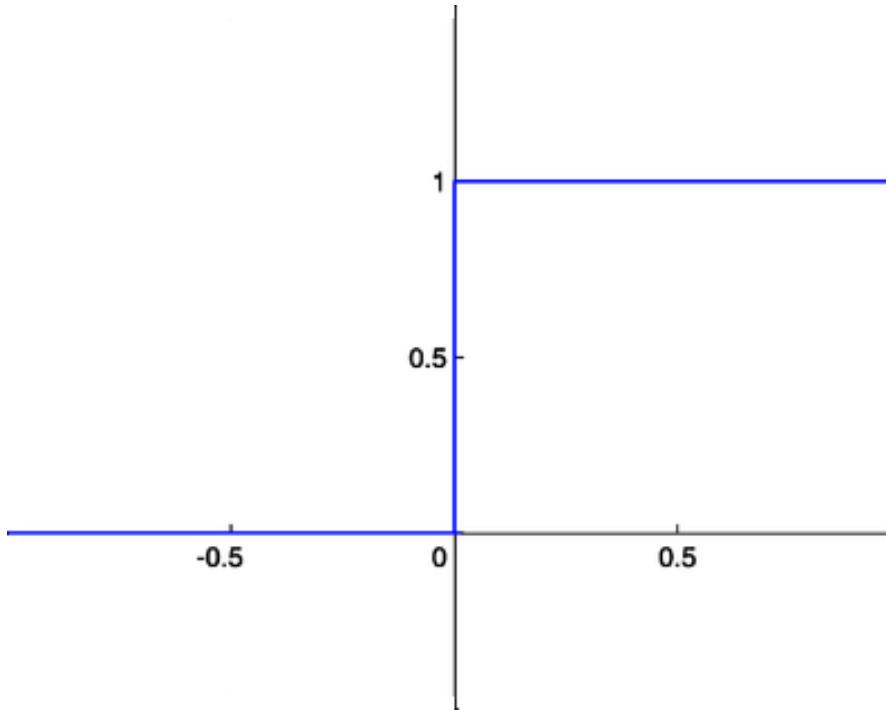
$$sum = (35 * -0.8) + (25 * 0.1)$$

$$sum = -28 + 2.5$$

$$sum = -25.5$$

Greater or equal to 1 = 1
Otherwise = 0

STEP FUNCTION



PERCEPTRON



- Positive weight – exciting synapse
- Negative weight – inhibitory synapse
- Weights are the synapses
- Weights amplify or reduce the input signal
- The knowledge of a neural network is the weights

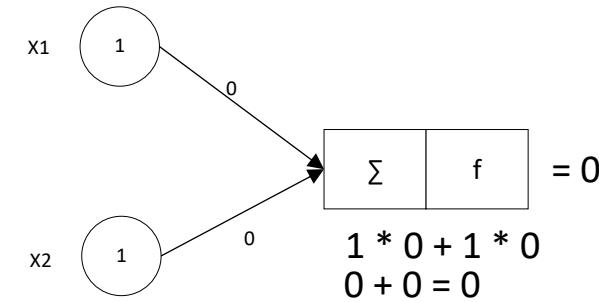
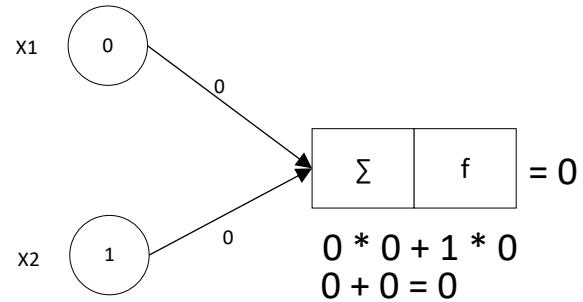
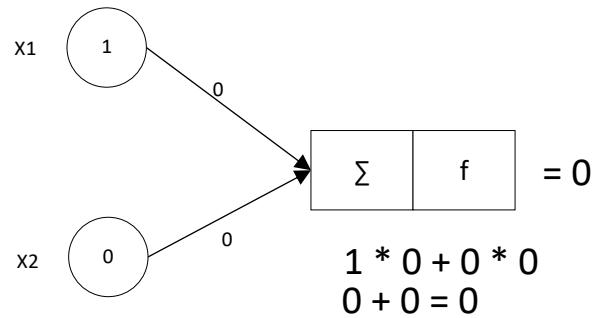
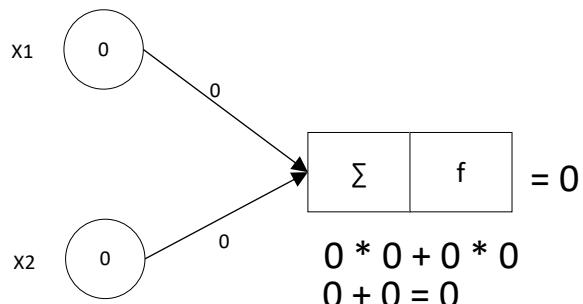


“AND” OPERATOR



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

“AND” OPERATOR



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

error = correct - prediction

Class	Prediction	Error
0	0	0
0	0	0
0	0	0
1	0	1

75%

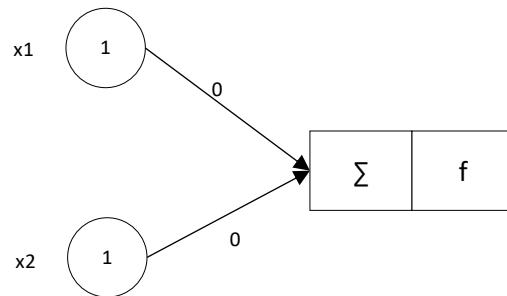
weight (n + 1) = weight(n) + (learning_rate * input * error)



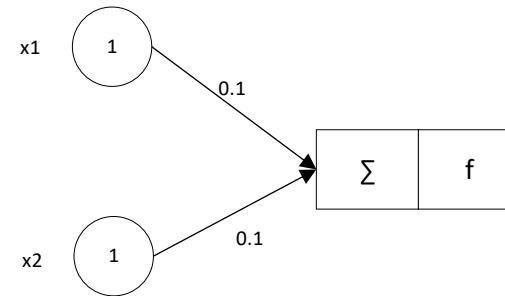
“AND” OPERATOR

error = correct - prediction

Class	Prediction	Error
0	0	0
0	0	0
0	0	0
1	0	1

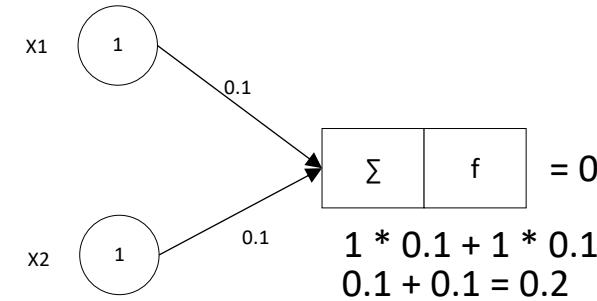
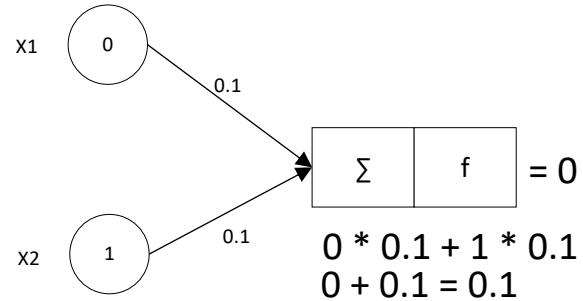
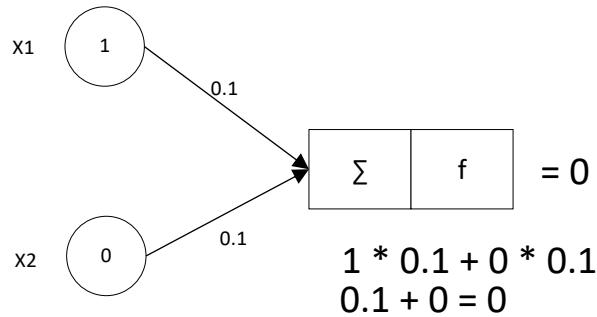
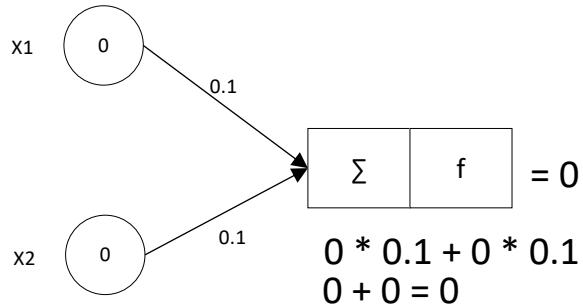


$$\begin{aligned} \text{weight (n + 1)} &= \text{weight(n)} + (\text{learning_rate} * \text{input} * \text{error}) \\ \text{weight (n + 1)} &= 0 + (0.1 * 1 * 1) \\ \text{weight (n + 1)} &= 0.1 \end{aligned}$$



$$\begin{aligned} \text{weight (n + 1)} &= \text{weight(n)} + (\text{learning_rate} * \text{input} * \text{error}) \\ \text{weight (n + 1)} &= 0 + (0.1 * 1 * 1) \\ \text{weight (n + 1)} &= 0.1 \end{aligned}$$

“AND” OPERATOR



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

error = correct - prediction

Class	Prediction	Error
0	0	0
0	0	0
0	0	0
1	0	1

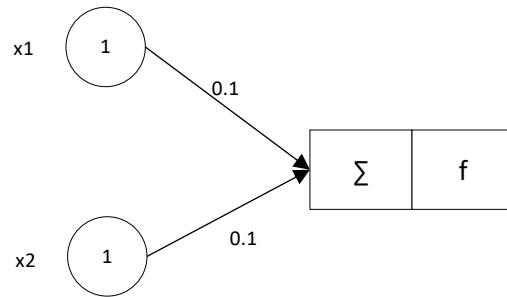
75%



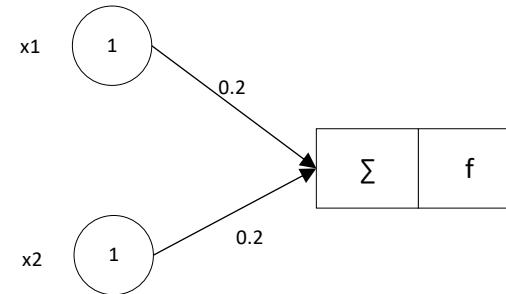
“AND” OPERATOR

error = correct - prediction

Class	Prediction	Error
0	0	0
0	0	0
0	0	0
1	0	1

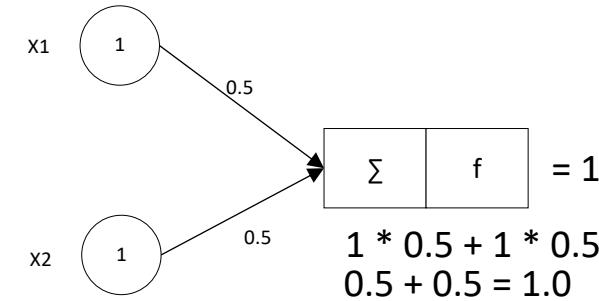
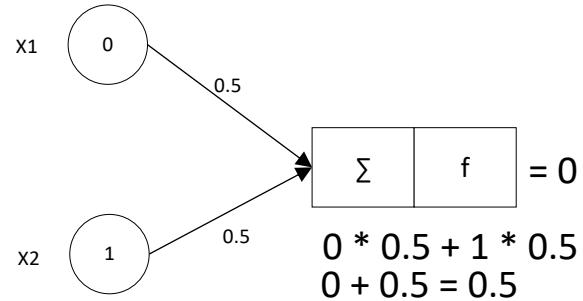
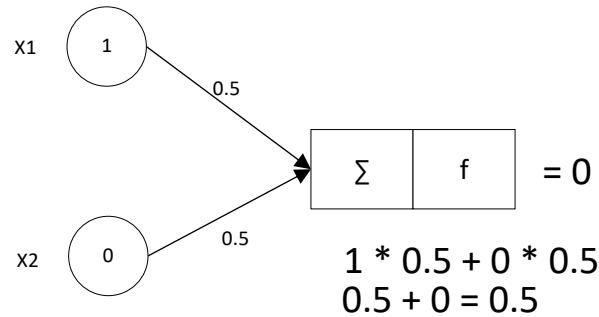
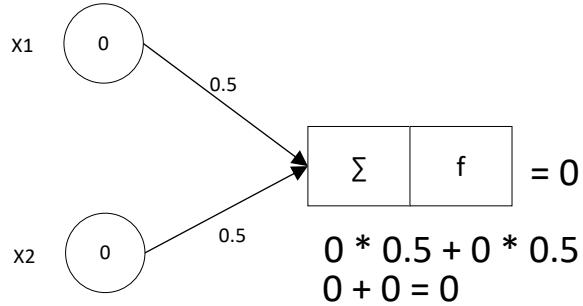


$$\begin{aligned} \text{weight (n + 1)} &= \text{weight(n)} + (\text{learning_rate} * \text{input} * \text{error}) \\ \text{weight (n + 1)} &= 0.1 + (0.1 * 1 * 1) \\ \text{weight (n + 1)} &= 0.2 \end{aligned}$$



$$\begin{aligned} \text{weight (n + 1)} &= \text{weight(n)} + (\text{learning_rate} * \text{input} * \text{error}) \\ \text{weight (n + 1)} &= 0.1 + (0.1 * 1 * 1) \\ \text{weight (n + 1)} &= 0.2 \end{aligned}$$

“AND” OPERATOR



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

error = correct - prediction

Class	Prediction	Error
0	0	0
0	0	0
0	0	0
1	1	0

100%





While error $\neq 0$

 For each row

 Calculate output

 Calculate error (correct - prediction)

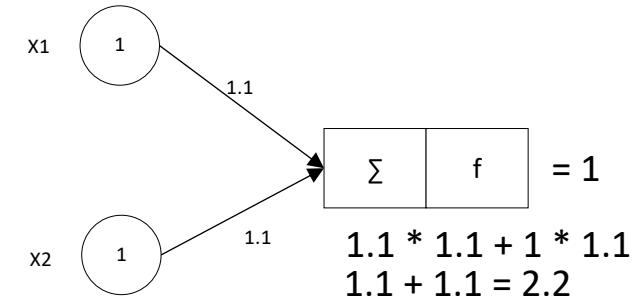
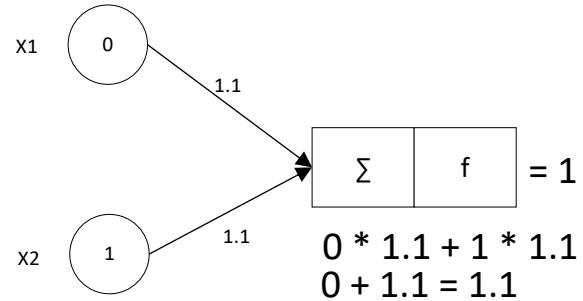
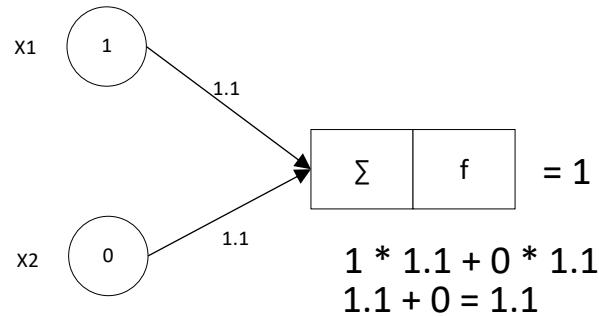
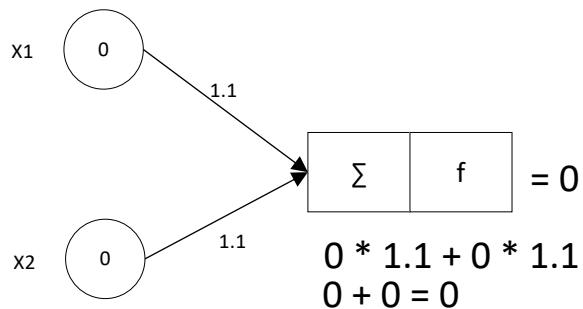
 If error > 0

 For each weight

 Update the weights



“OR” OPERATOR



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

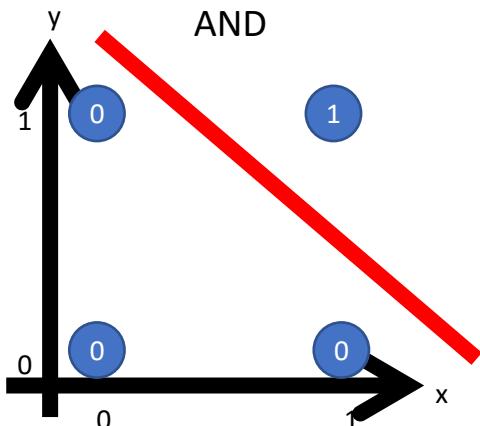
error = correct - prediction

Class	Prediction	Error
0	0	0
1	1	0
1	1	0
1	1	0

100%

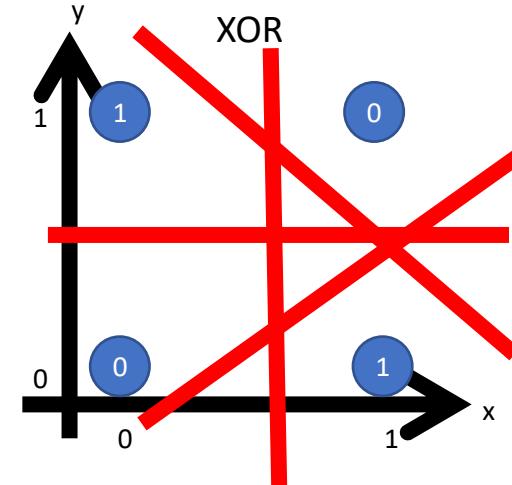
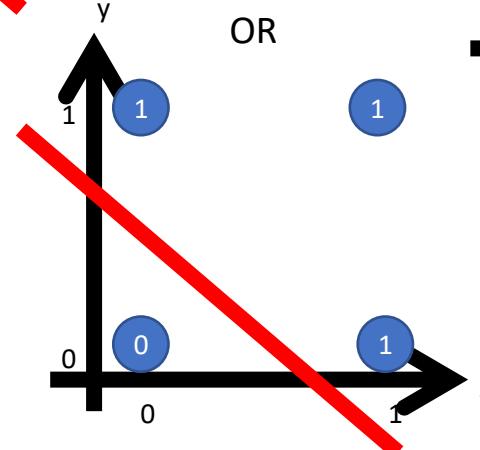


“XOR” OPERATOR



OR

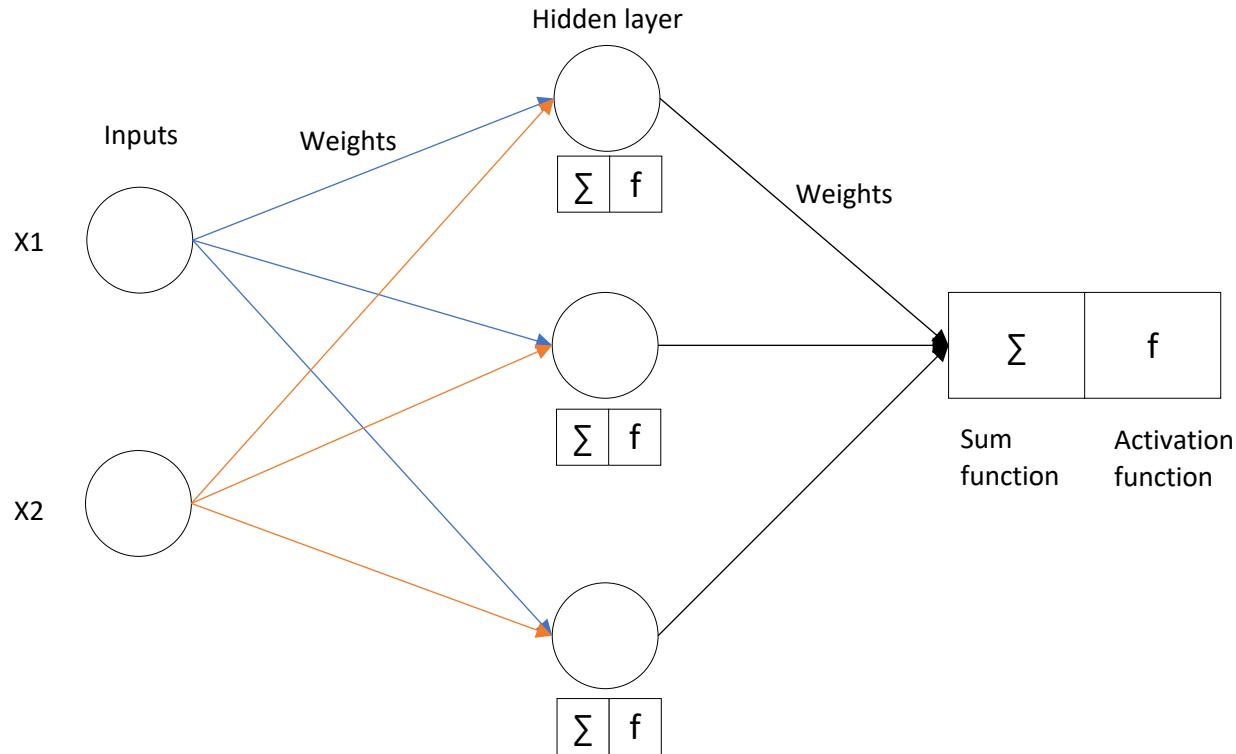
Linearly separable



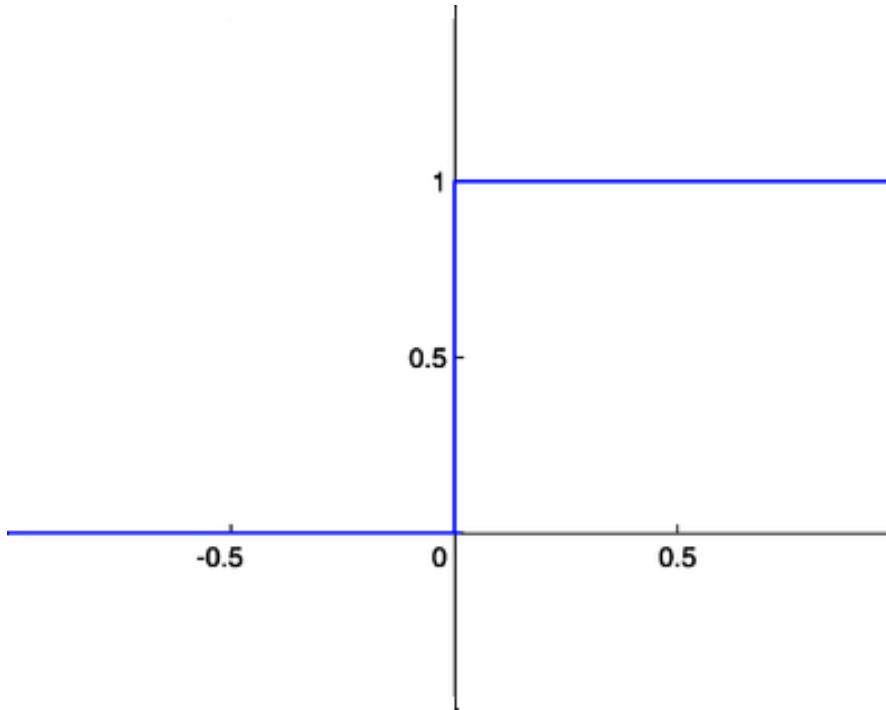
Non-linearly separable



MULTI-LAYER PERCEPTRON



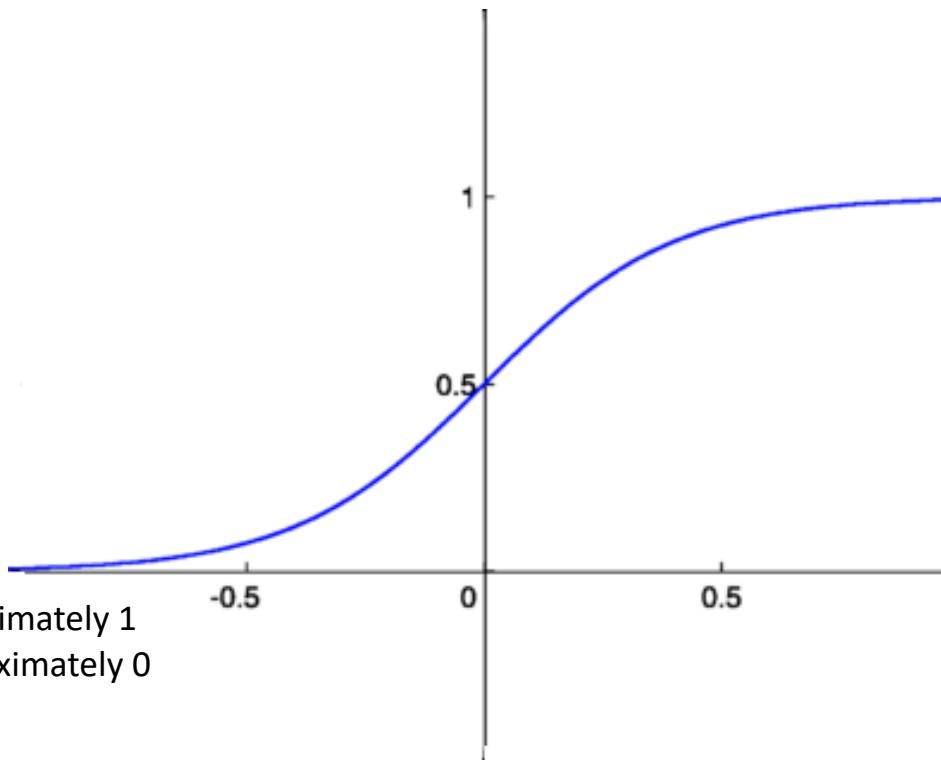
STEP FUNCTION



SIGMOID FUNCTION



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



If X is high, the value is approximately 1

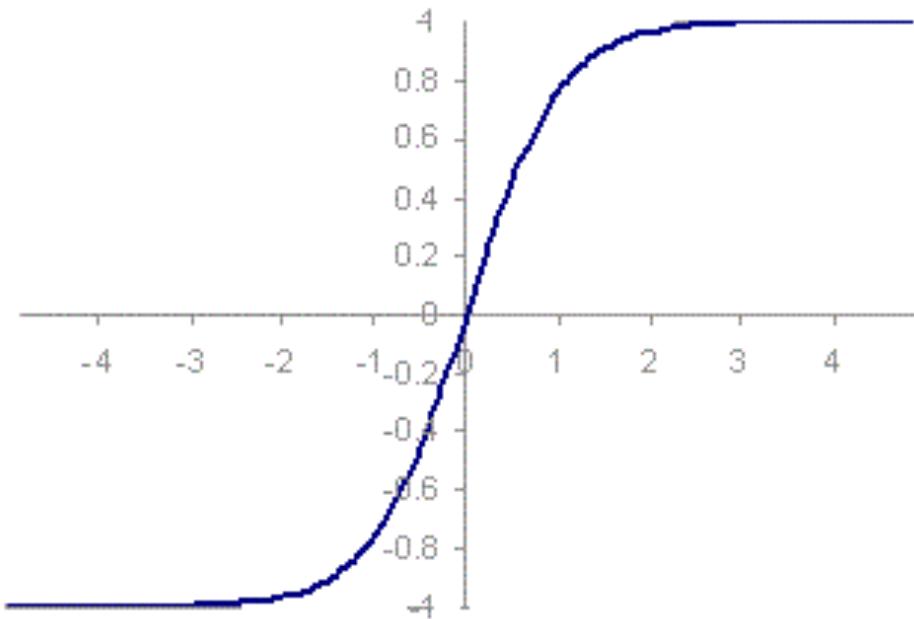
If X is small, the value is approximately 0



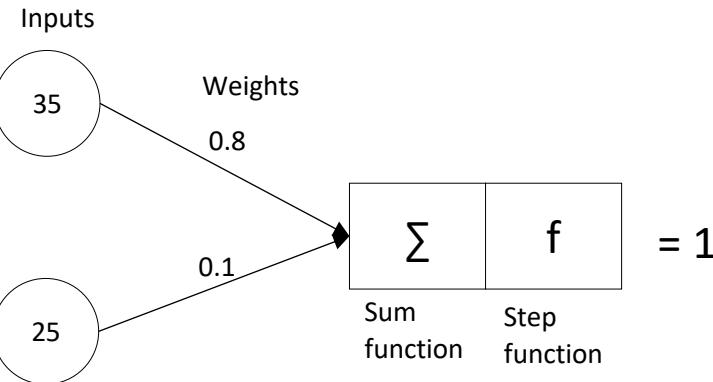
HYPERBOLIC TANGENT FUNCTION



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



STEP FUNCTION

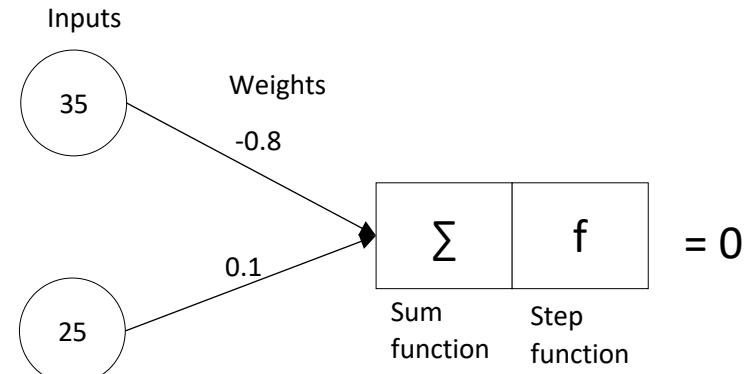


$$sum = \sum_{i=1}^n x_i * w_i$$

$$sum = (35 * 0.8) + (25 * 0.1)$$

$$sum = 28 + 2.5$$

$$sum = 30.5$$



$$sum = \sum_{i=1}^n x_i * w_i$$

$$sum = (35 * -0.8) + (25 * 0.1)$$

$$sum = -28 + 2.5$$

$$sum = -25.5$$



“XOR” OPERATOR

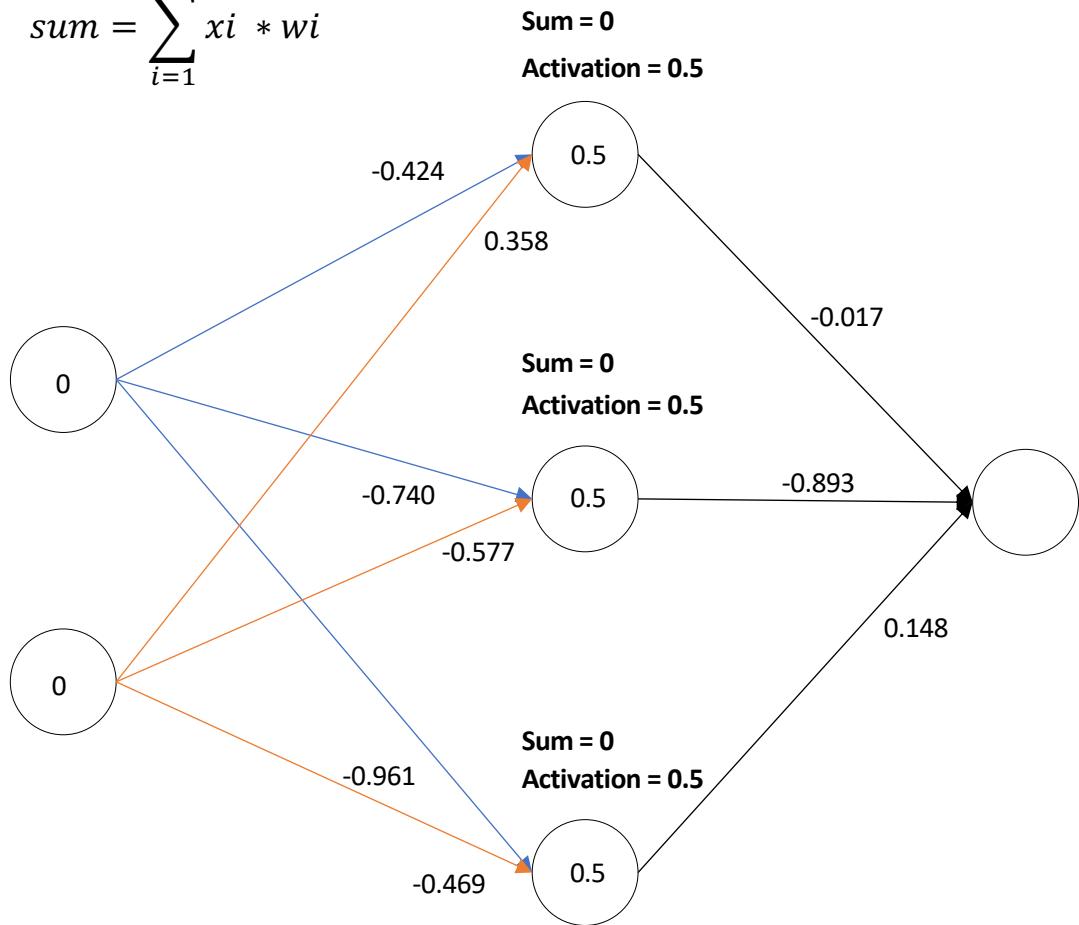


X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0

INPUT LAYER TO HIDDEN LAYER



$$sum = \sum_{i=1}^n x_i * w_i$$



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

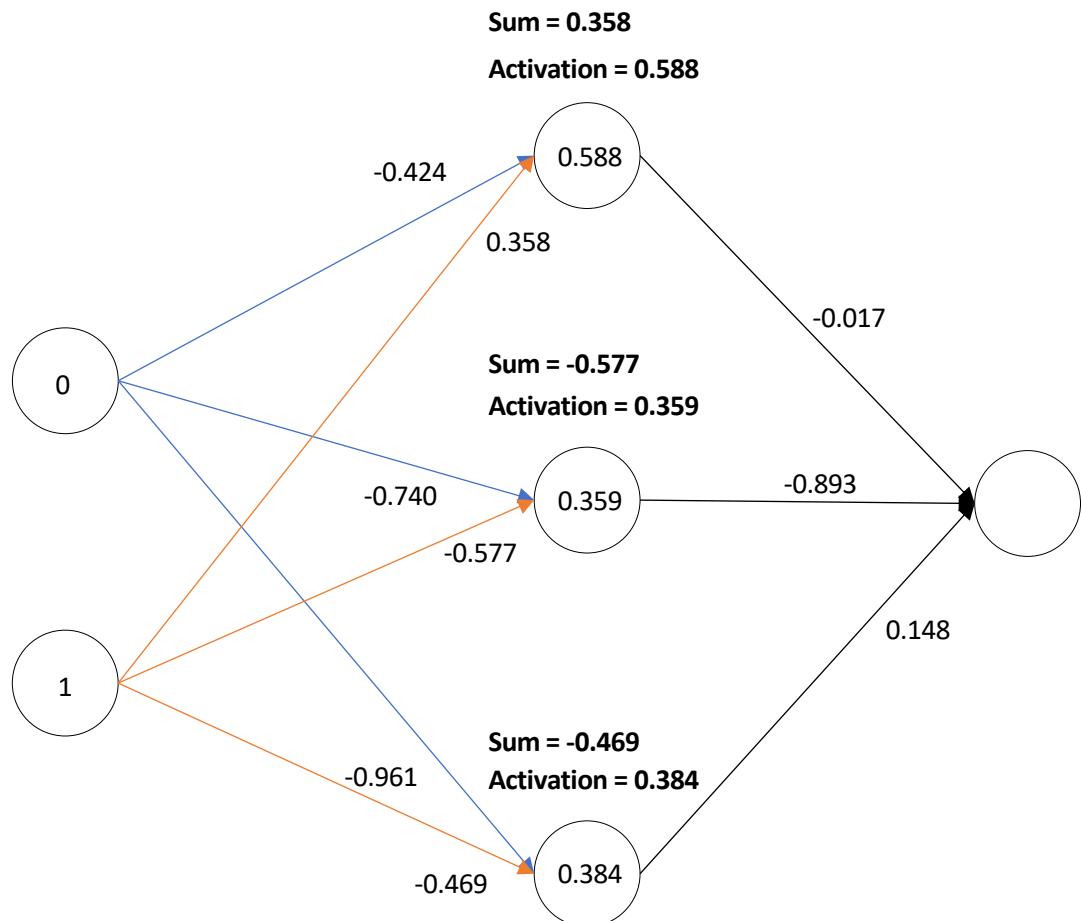
X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0

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

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

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

INPUT LAYER TO HIDDEN LAYER



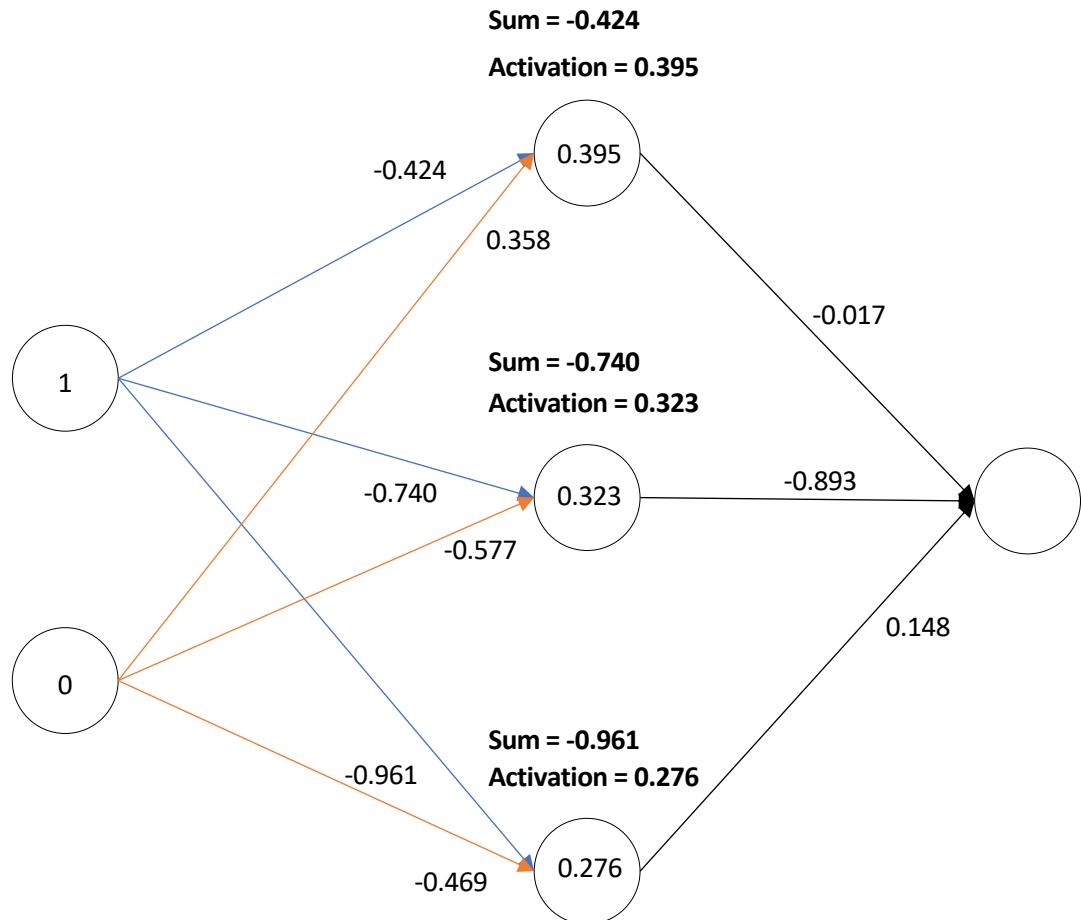
$$0 * (-0.424) + 1 * 0.358 = 0.358$$

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

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

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

INPUT LAYER TO HIDDEN LAYER



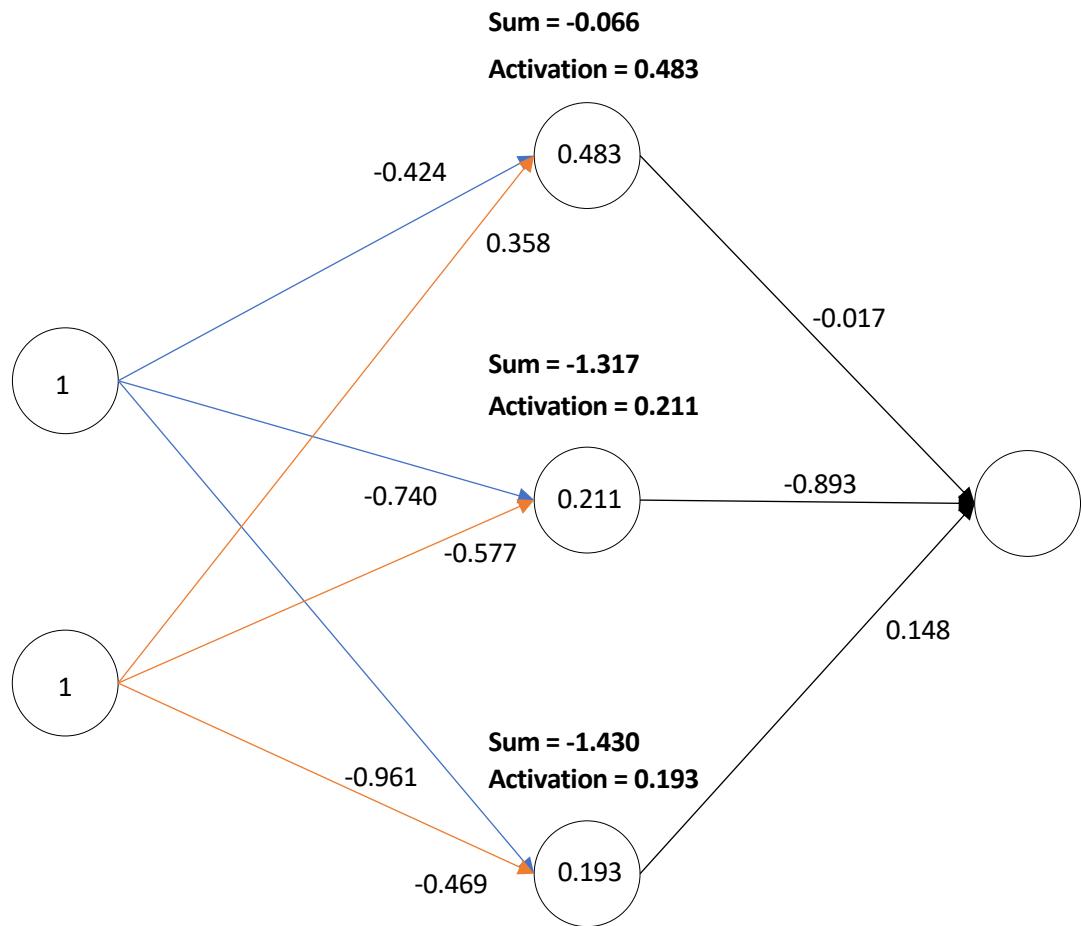
$$1 * (-0.424) + 0 * 0.358 = -0.424$$

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

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

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

INPUT LAYER TO HIDDEN LAYER



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

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

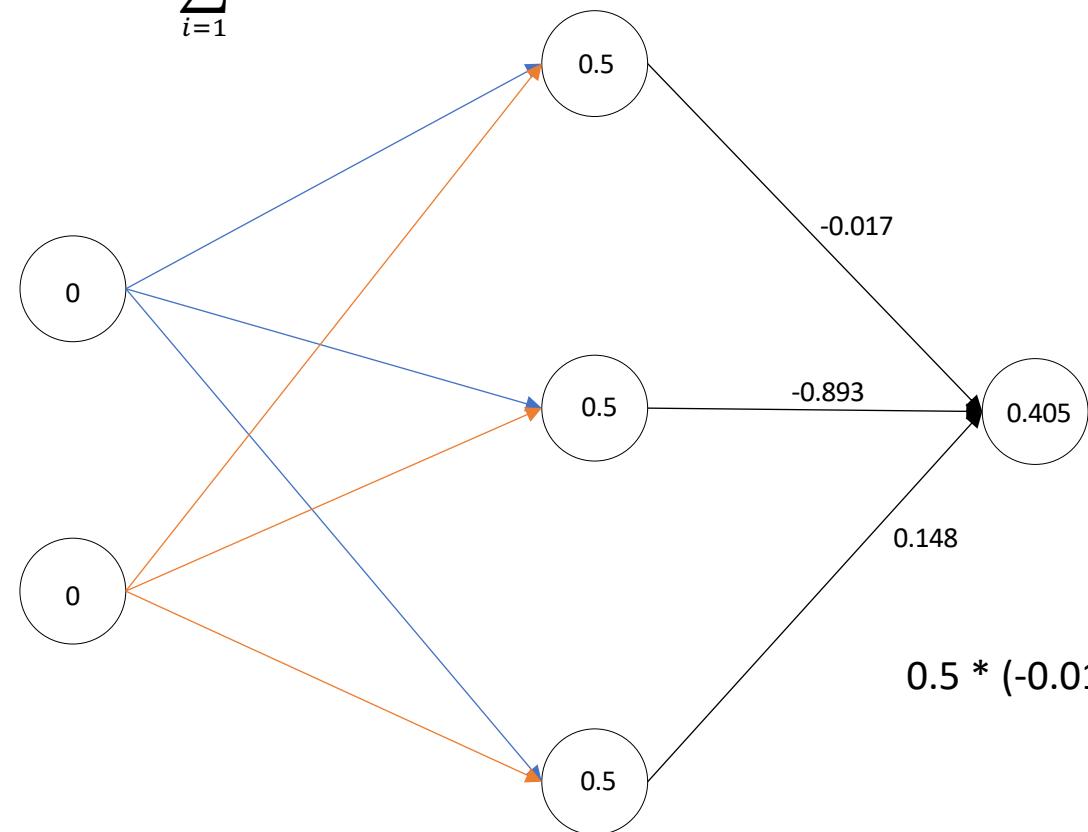
$$1 * (-0.740) + 1 * (-0.577) = -1.317$$

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

HIDDEN LAYER TO OUTPUT LAYER



$$sum = \sum_{i=1}^n x_i * w_i$$



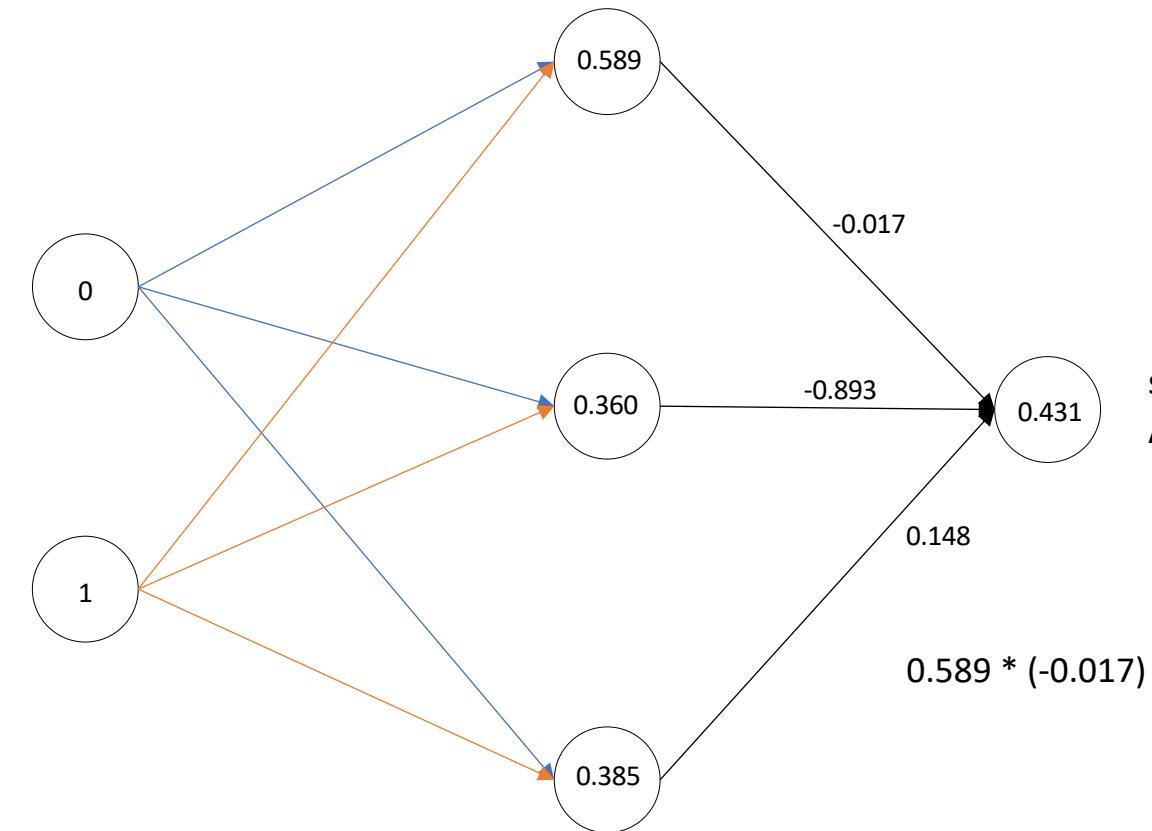
$$y = \frac{1}{1 + e^{-x}}$$

Sum = -0.381
Activation = 0.405

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

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

HIDDEN LAYER TO OUTPUT LAYER



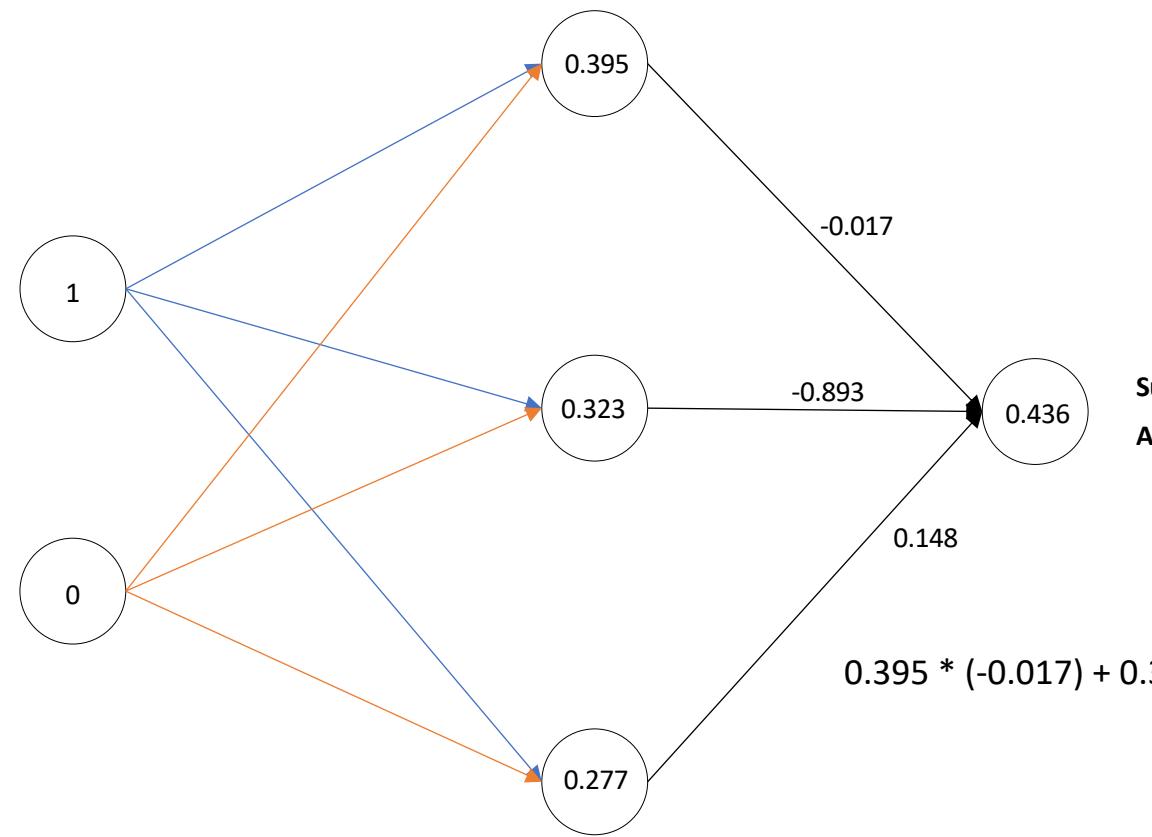
Sum = -0.274

Activation = 0.431

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

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

HIDDEN LAYER TO OUTPUT LAYER



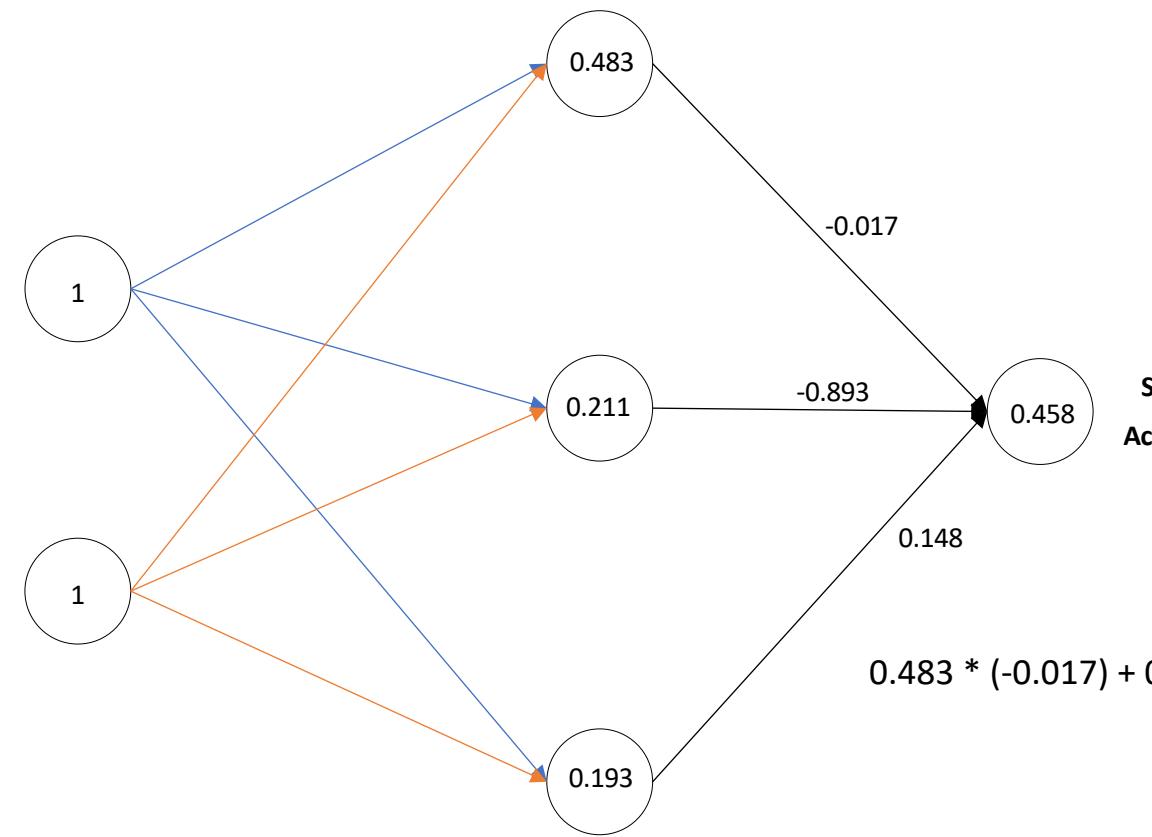
Sum = -0.254

Activation = 0.436

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

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

HIDDEN LAYER TO OUTPUT LAYER



Sum = -0.168
Activation = 0.458

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

X1	X2	Classe
0	0	0
0	1	1
1	0	1
1	1	0

“XOR” OPERATOR – ERROR (LOSS FUNCTION)



The simplest algorithm

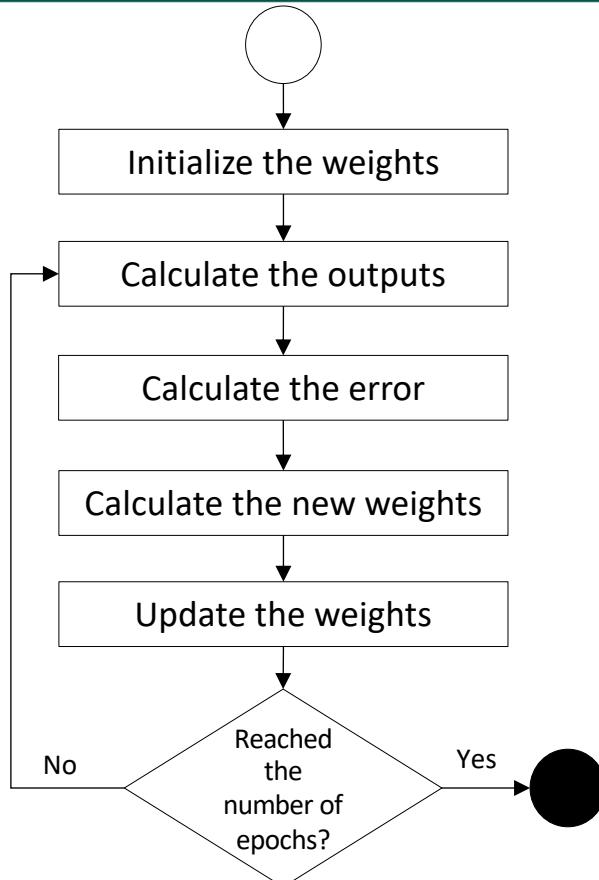
$$\text{error} = \text{correct} - \text{prediction}$$

X1	X2	Class	Prediction	Error
0	0	0	0.405	-0.405
0	1	1	0.431	0.569
1	0	1	0.436	0.564
1	1	0	0.458	-0.458

$$\text{Average} = 0.499$$



ALGORITHM



Cost function (loss function)
Gradient descent
Derivative
Delta
Backpropagation

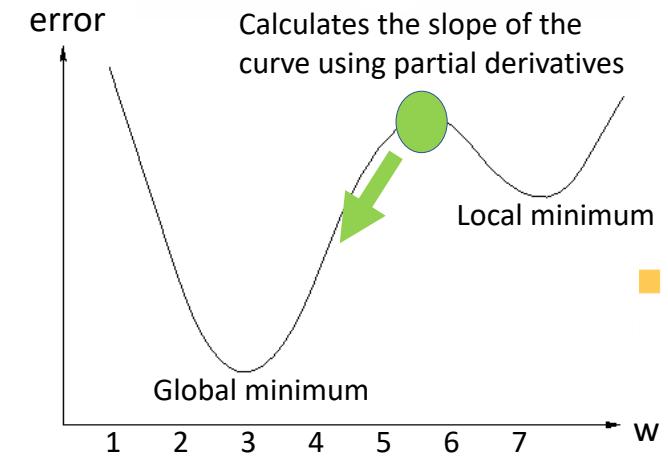
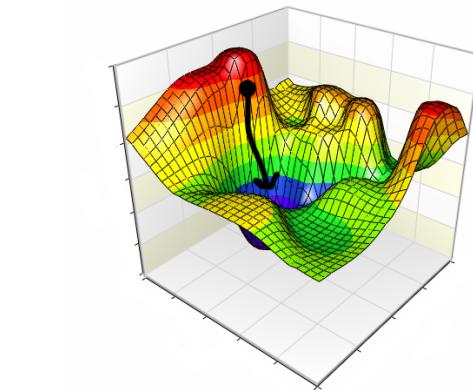
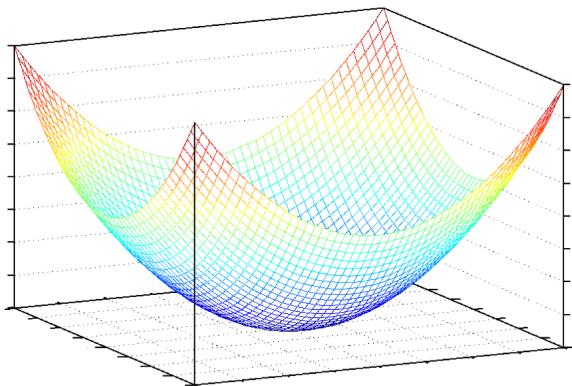


GRADIENT DESCENT



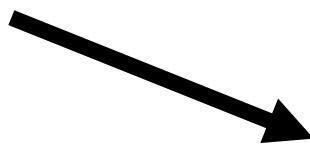
$$\min C(w_1, w_2 \dots w_n)$$

Calculate the partial derivative to move to the gradient direction



GRADIENT DESCENT (DERIVATIVE)

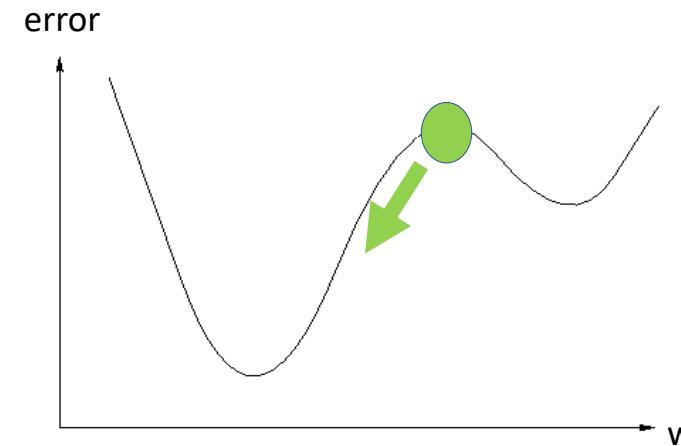
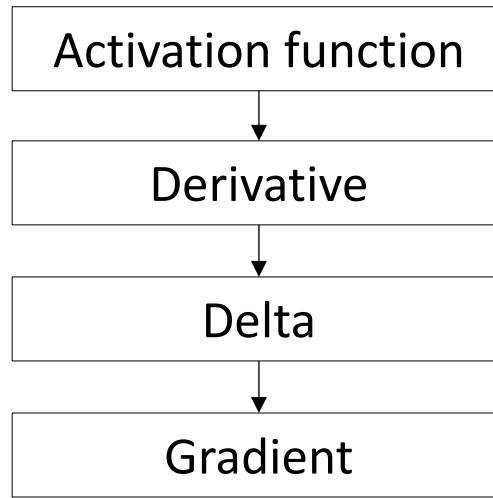
$$y = \frac{1}{1 + e^{-x}}$$



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

$$d = 0.1 * (1 - 0.1)$$

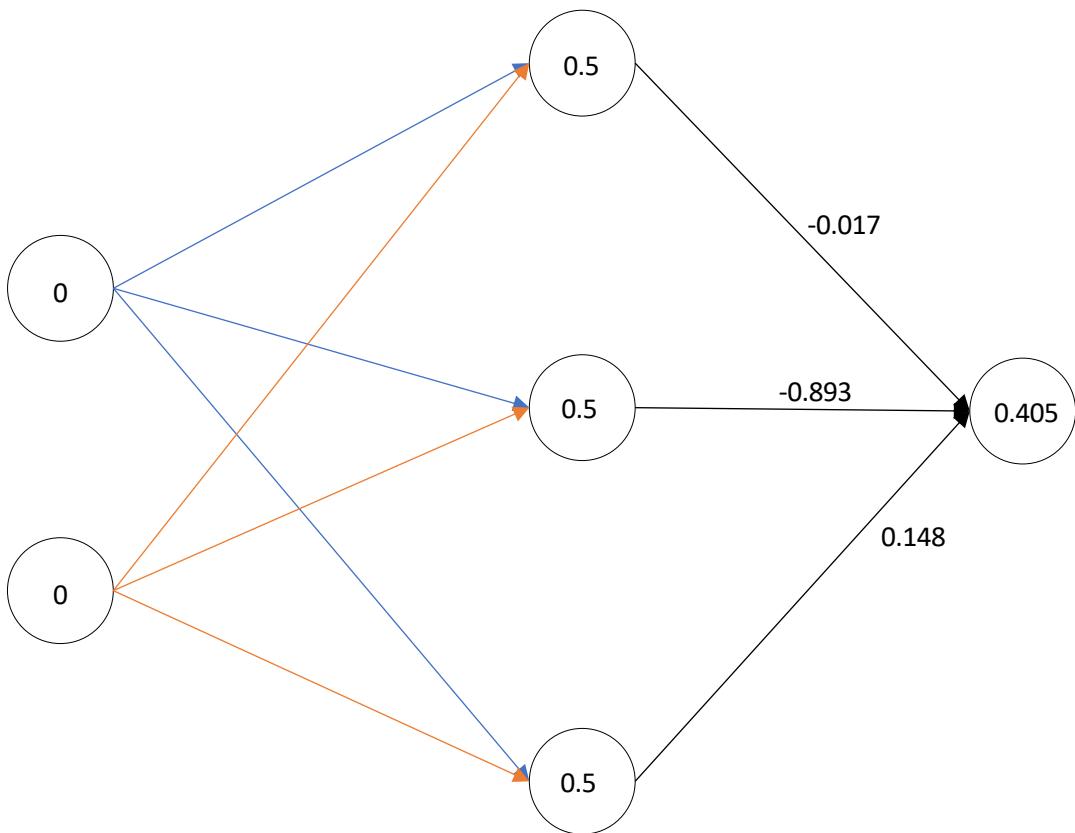
DELTA PARAMETER



OUTPUT LAYER – DELTA



$$\delta_{output} = error * sigmoid_{derivative}$$



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

Sum = -0.381

Activation = 0.405

Error = 0 – 0.405 = -0.405

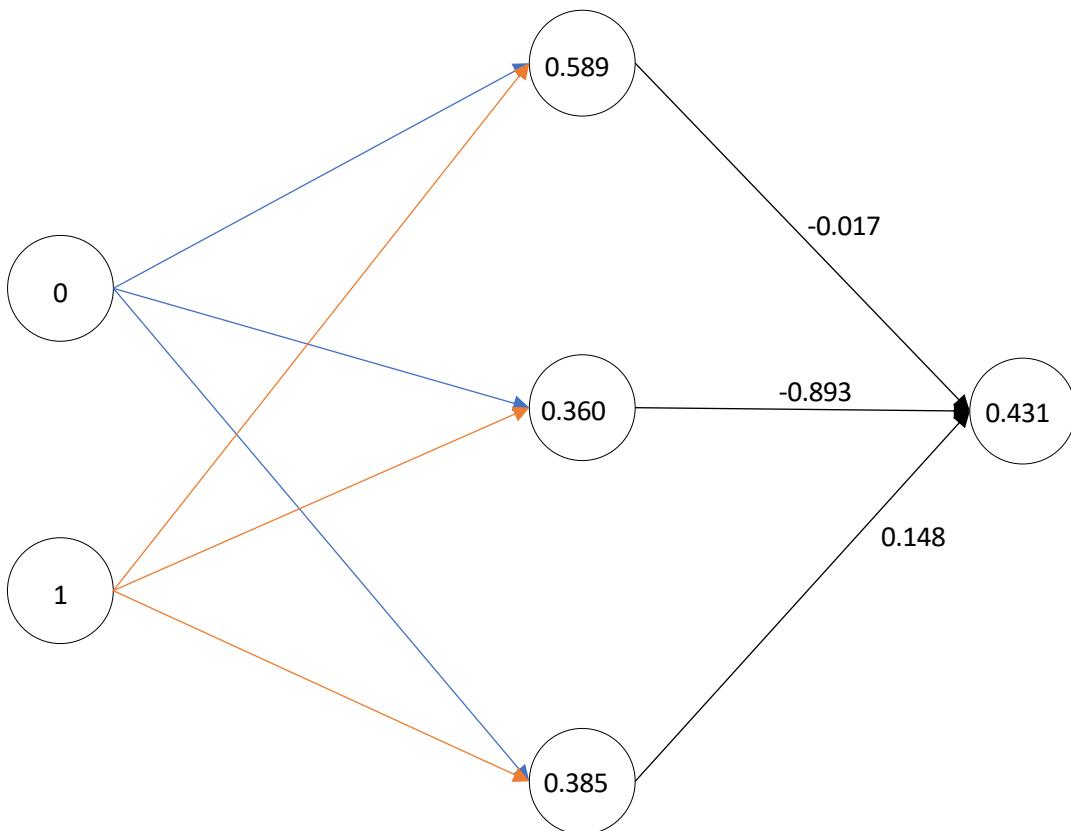
Derivative activation (sigmoid) = 0.241

Delta (output) = -0.405 * 0.241 = -0.097

OUTPUT LAYER – DELTA



$$\delta_{output} = error * sigmoid_{derivative}$$

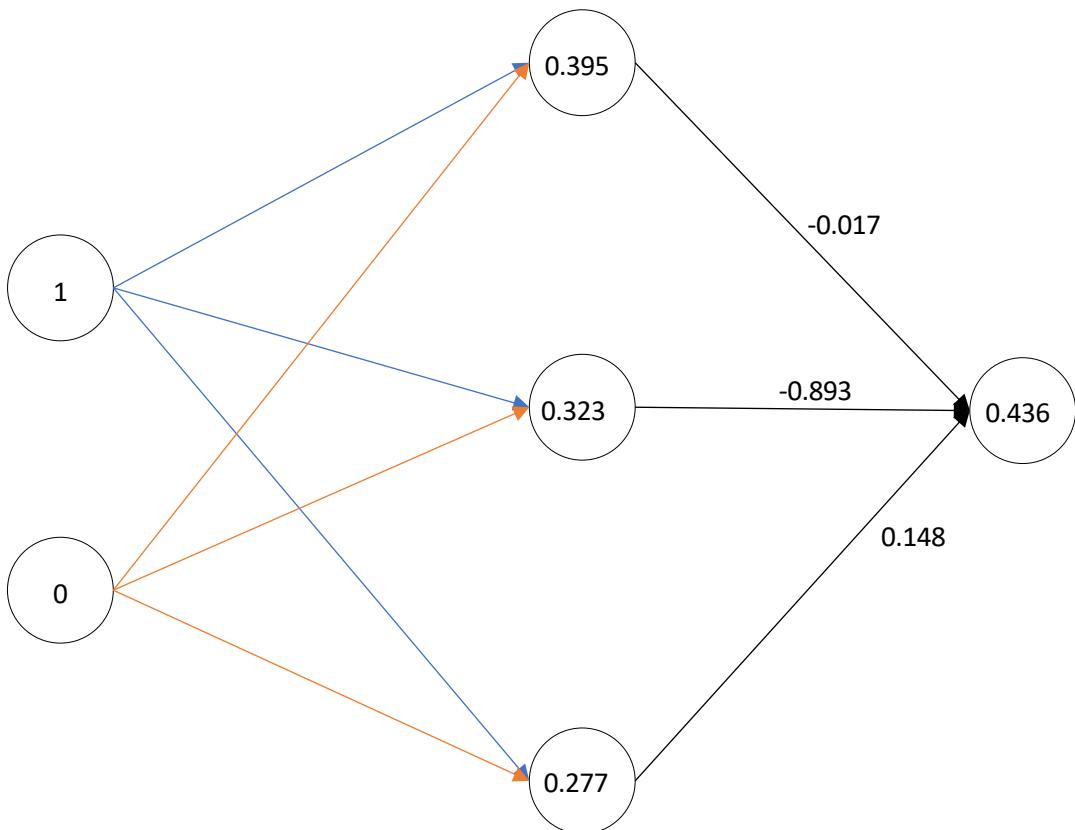


X1	X2	Class
0	0	0
0	1	1
1	0	1
1	1	0

OUTPUT LAYER – DELTA



$$\delta_{output} = error * sigmoid_{derivative}$$



Sum = -0.254

Activation = 0.436

Error = 1 - 0.436 = 0.564

Derivative activation (sigmoid) = 0.246

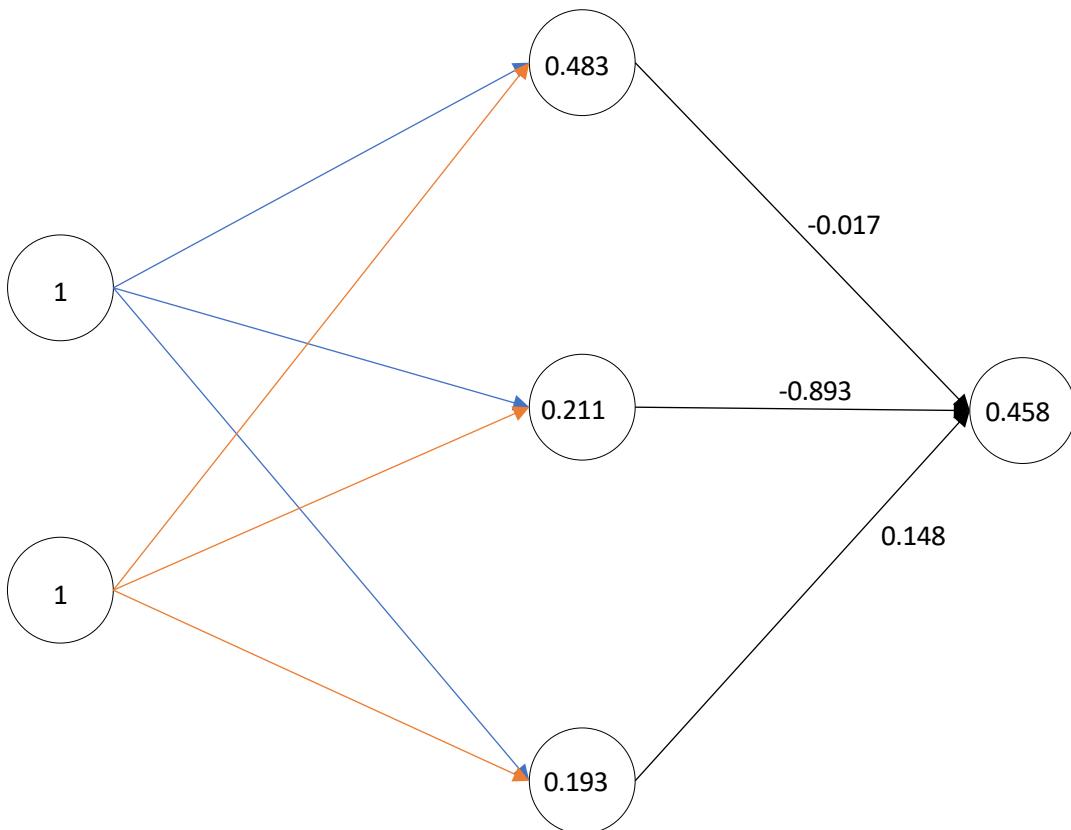
Delta (output) = 0.564 * 0.246 = 0.138

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

OUTPUT LAYER – DELTA



$$\delta_{output} = error * sigmoid_{derivative}$$



Sum = -0.168

Activation = 0.458

Error = 0 – 0.458 = -0.458

Derivative activation (sigmoid) = 0.248

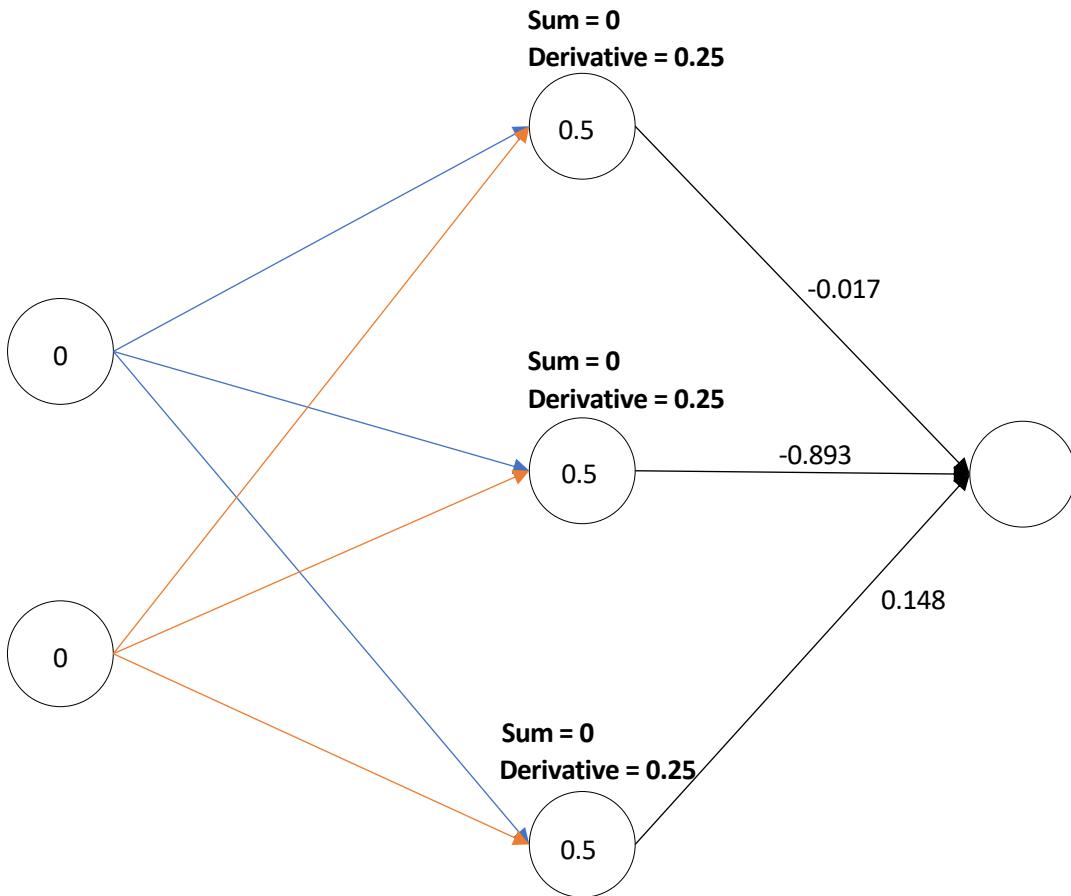
Delta (output) = -0.458 * 0.248 = -0.113

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

HIDDEN LAYER – DELTA



$$\delta_{hidden} = \text{sigmoid derivative} * weight * \delta_{output}$$



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

HIDDEN LAYER – DELTA



$$\delta_{hidden} = \text{sigmoid derivative} * weight * \delta_{output}$$

Sum = 0.358

Derivative = 0.242

0.589

Sum = -0.577

Derivative = 0.230

0.360

0.385

-0.017

-0.893

0.148

Delta (output) = 0.139

$$0.242 * (-0.017) * 0.139 = -0.000$$

$$0.230 * (-0.893) * 0.139 = -0.028$$

$$0.236 * 0.148 * 0.139 = 0.004$$

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

HIDDEN LAYER – DELTA



$$\delta_{hidden} = \text{sigmoid derivative} * weight * \delta_{output}$$

Sum = -0.424

Derivative = 0.239

0.396

Sum = -0.740

Derivative = 0.218

0.323

Sum = -0.961

Derivative = 0.200

0.277

-0.017

-0.893

0.148

Delta (output) = 0.138

$$0.239 * (-0.017) * 0.138 = -0.000$$

$$0.218 * (-0.893) * 0.138 = -0.026$$

$$0.200 * 0.148 * 0.138 = 0.004$$

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

HIDDEN LAYER – DELTA



$$\delta_{hidden} = \text{sigmoid derivative} * weight * \delta_{output}$$

Sum = -0.066
Derivative = 0.249

0.484

Sum = -1.317
Derivative = 0.166

0.211

Sum = -1.430
Derivative = 0.155

0.193

-0.017

-0.893

0.148

Delta (output) = -0.113

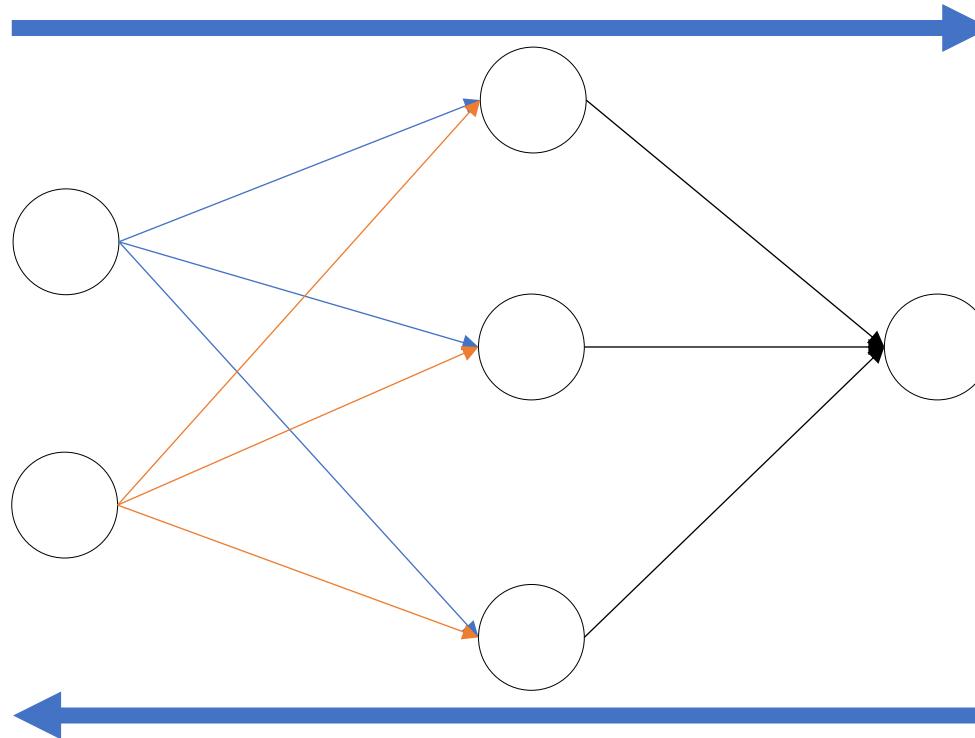
$$0.249 * (-0.017) * (-0.113) = 0.000$$

$$0.166 * (-0.893) * (-0.113) = 0.016$$

$$0.155 * 0.148 * (-0.113) = -0.002$$

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

WEIGHT UPDATE

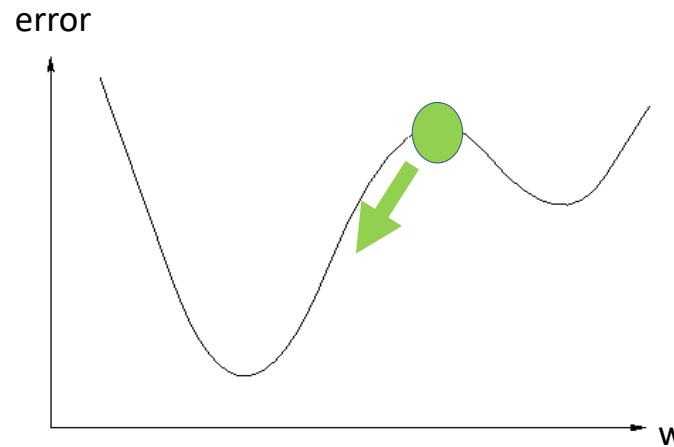


$$weight_{n+1} = weight_n + (input * delta * learning_rate)$$

LEARNING RATE



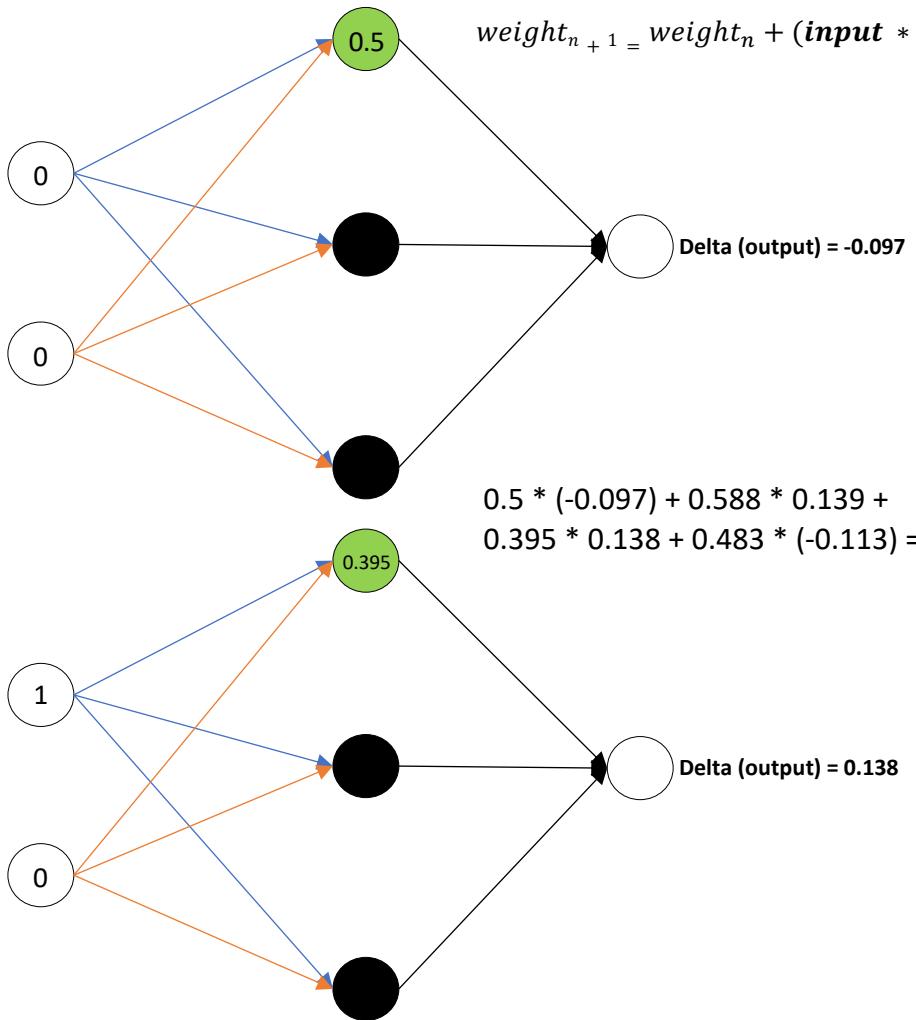
- Defines how fast the algorithm will learn
- High: the convergence is fast but may lose the global minimum
- Low: the convergence will be slower but more likely to reach the global minimum



WEIGHT UPDATE – OUTPUT LAYER TO HIDDEN LAYER



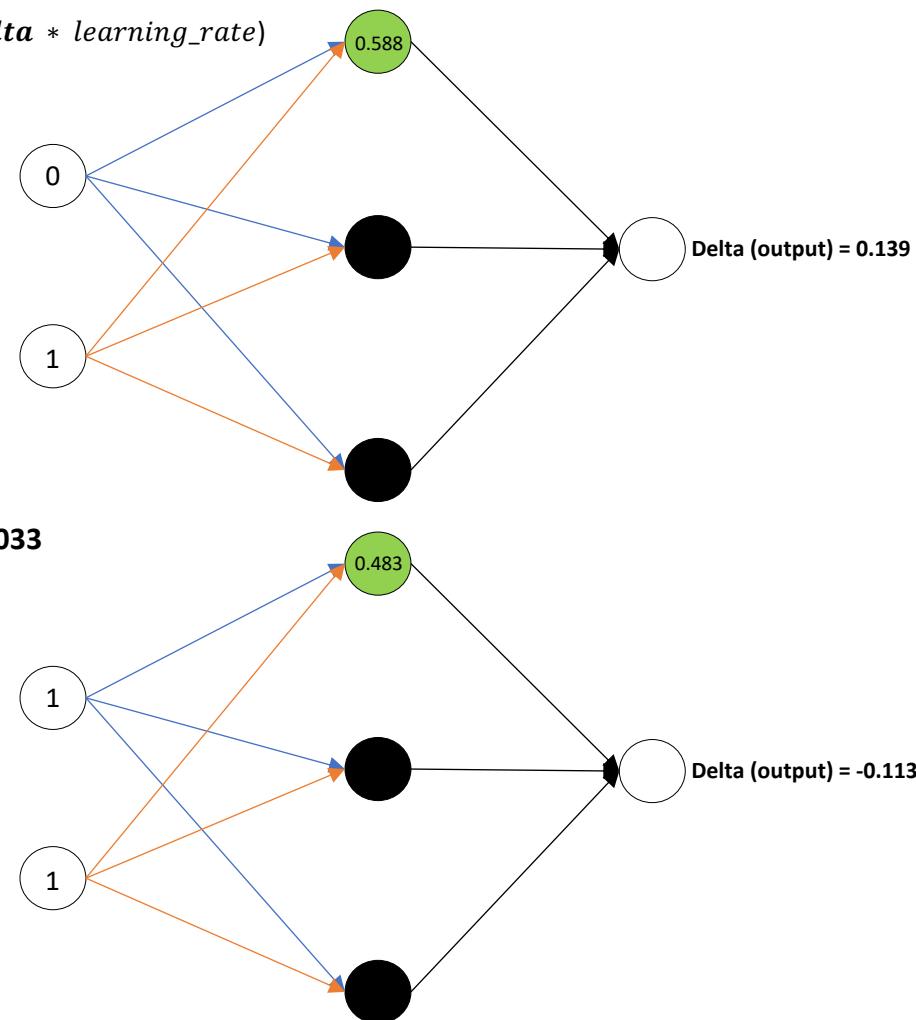
$$weight_{n+1} = weight_n + (input * delta * learning_rate)$$



Delta (output) = -0.097

$$0.5 * (-0.097) + 0.588 * 0.139 + 0.395 * 0.138 + 0.483 * (-0.113) = \mathbf{0.033}$$

Delta (output) = 0.138



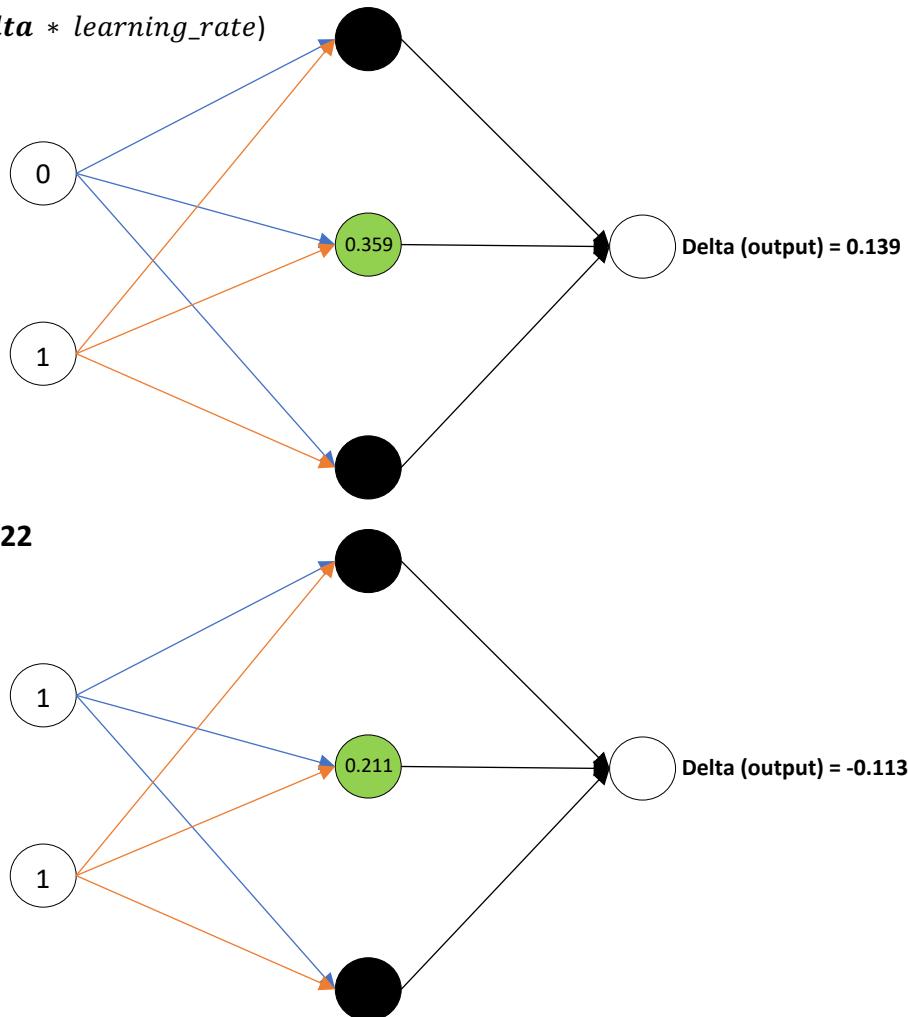
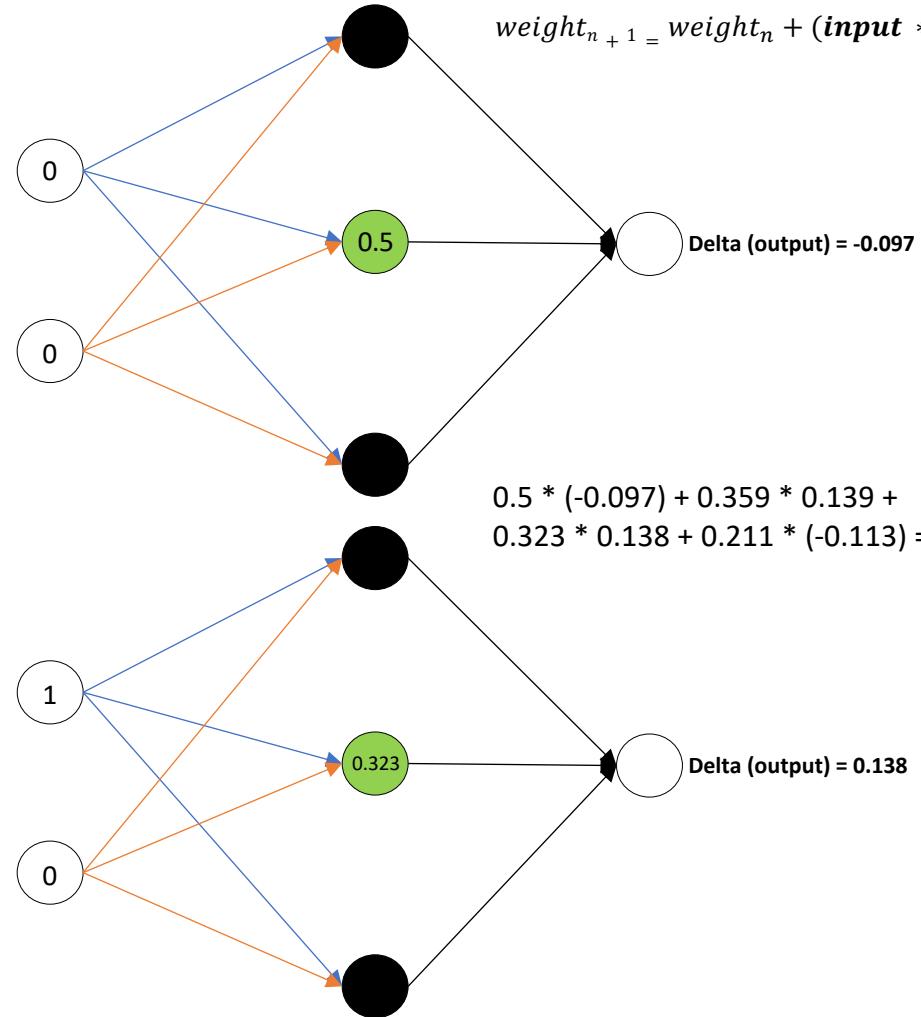
Delta (output) = 0.139

Delta (output) = -0.113

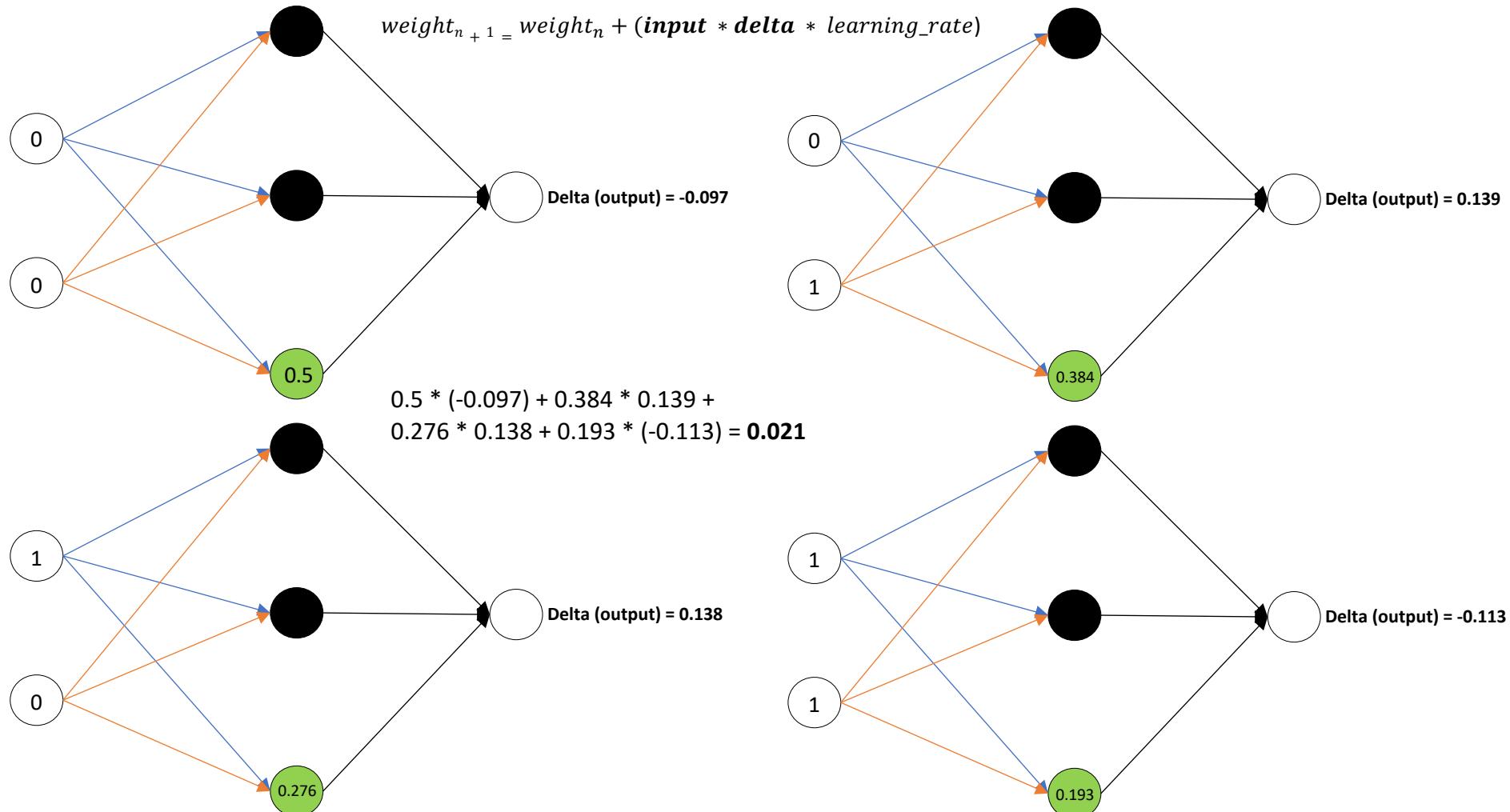
WEIGHT UPDATE – OUTPUT LAYER TO HIDDEN LAYER



$$weight_{n+1} = weight_n + (input * delta * learning_rate)$$



WEIGHT UPDATE – OUTPUT LAYER TO HIDDEN LAYER



WEIGHT UPDATE – OUTPUT LAYER TO HIDDEN LAYER



Learning rate = 0.3

Input x delta

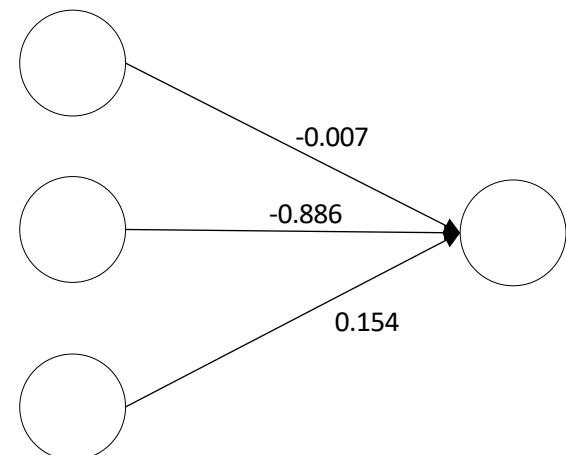
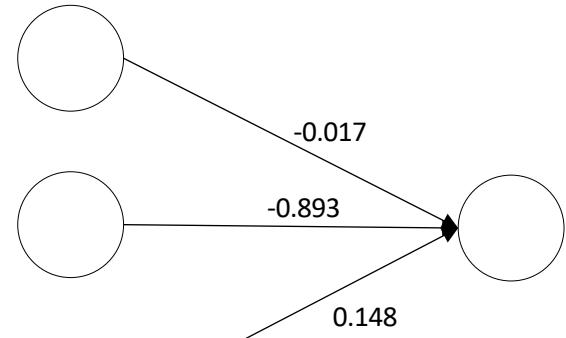
0.033
0.022
0.021

$$weight_{n+1} = weight_n + (input * delta * learning_rate)$$

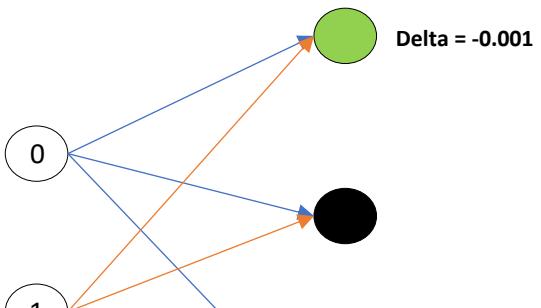
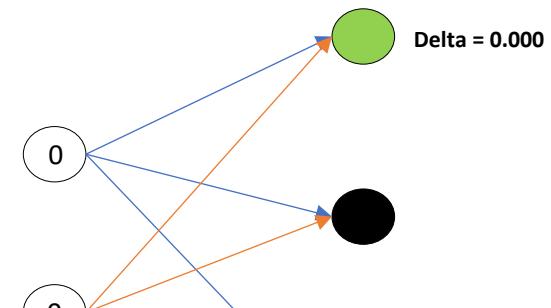
$$-0.017 + 0.033 * 0.3 = \mathbf{-0.007}$$

$$-0.893 + 0.022 * 0.3 = \mathbf{-0.886}$$

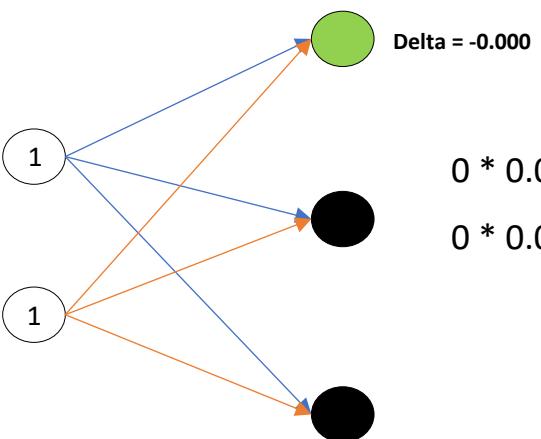
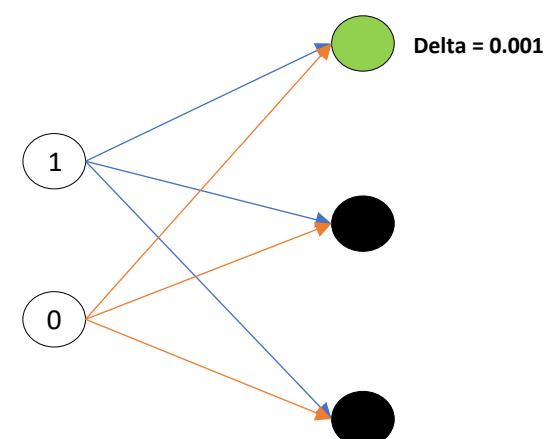
$$0.148 + 0.021 * 0.3 = \mathbf{0.154}$$



WEIGHT UPDATE – HIDDEN LAYER TO INPUT LAYER



$$weight_{n+1} = weight_n + (input * delta * learning_rate)$$



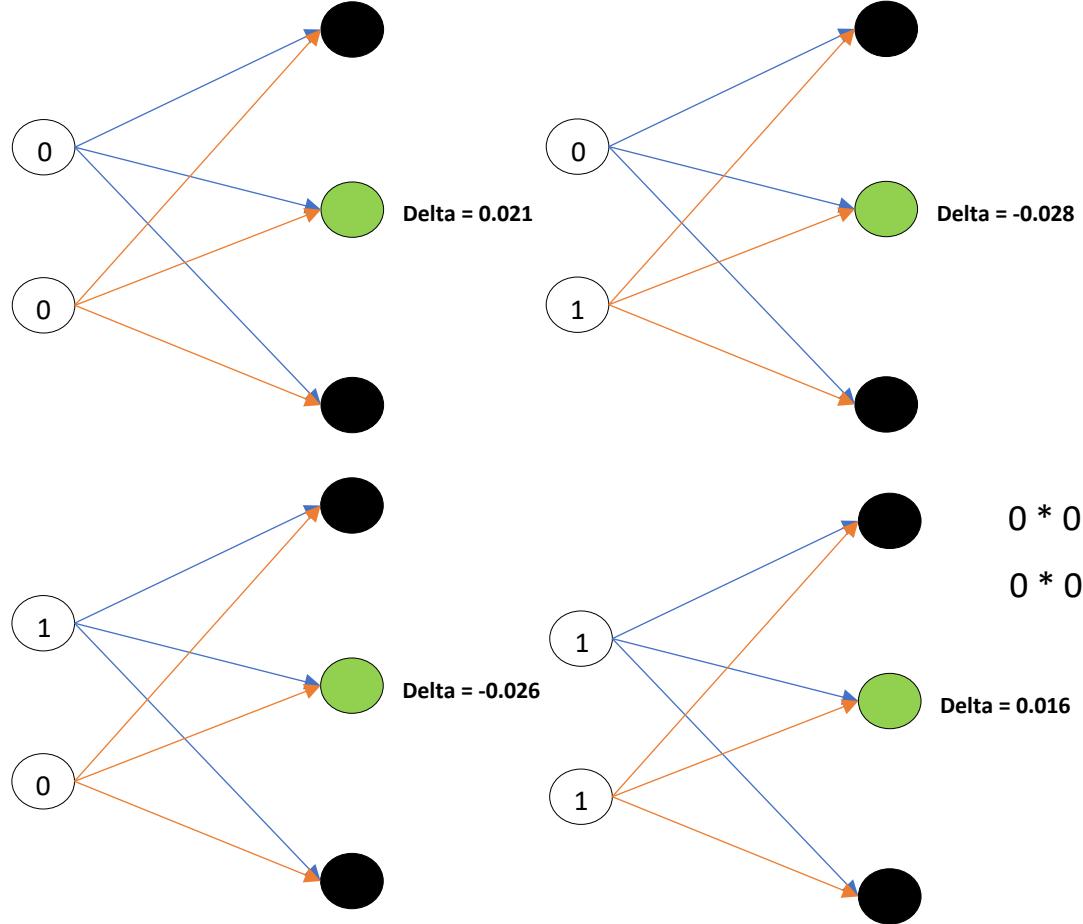
$$0 * 0.000 + 0 * (-0.001) + 1 * (0.001) + 1 * -0.000 = -0.000$$

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

WEIGHT UPDATE – HIDDEN LAYER TO INPUT LAYER



$$weight_{n+1} = weight_n + (input * delta * learning_rate)$$



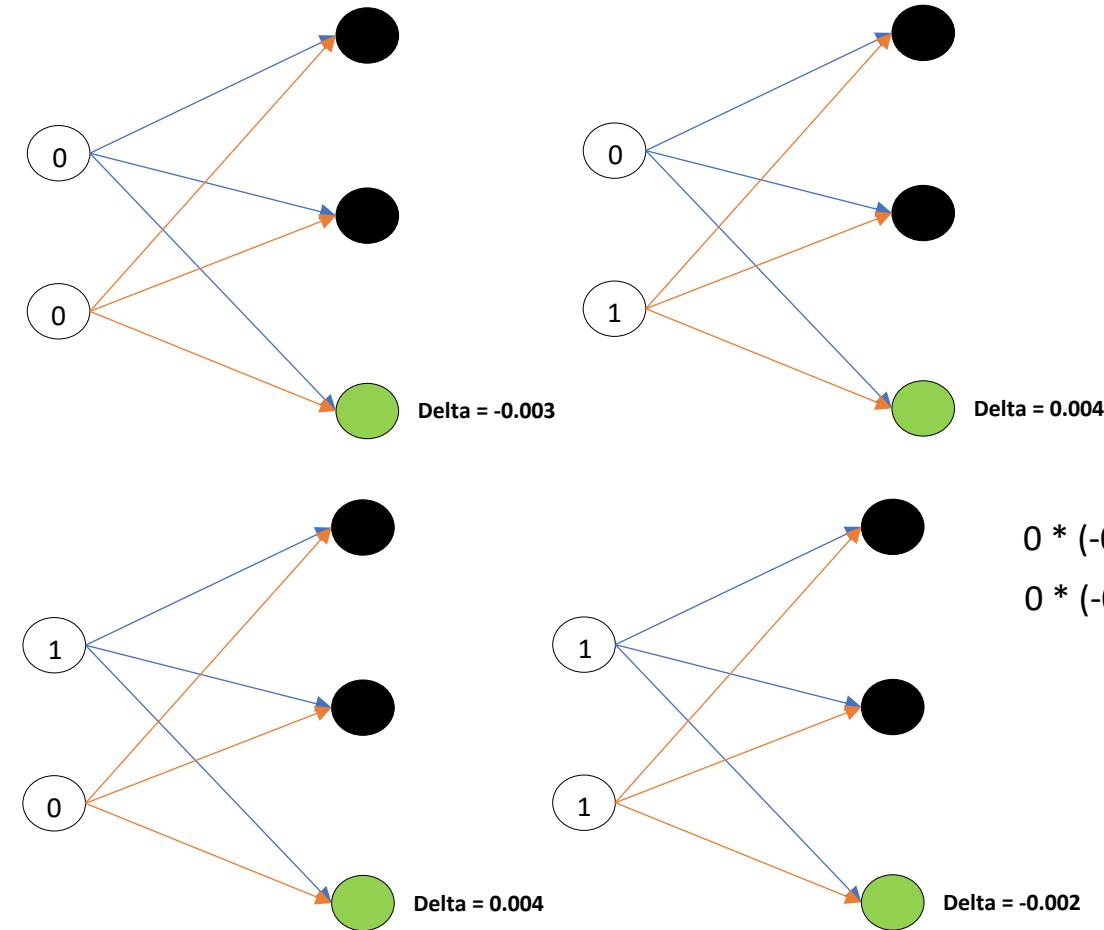
$$0 * 0.021 + 0 * (-0.028) + 1 * (-0.026) + 1 * 0.016 = -0.009$$

$$0 * 0.021 + 1 * (-0.028) + 0 * (-0.026) + 1 * 0.016 = -0.012$$

WEIGHT UPDATE – HIDDEN LAYER TO INPUT LAYER



$$weight_{n+1} = weight_n + (input * delta * learning_rate)$$



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

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

WEIGHT UPDATE – HIDDEN LAYER TO INPUT LAYER



Learning rate = 0.3

Input x delta

-0.000 -0.009 0.002
-0.000 -0.012 0.002

$$weight_{n+1} = weight_n + (input * delta * learning_rate)$$

$$-0.424 + (-0.000) * 0.3 = \mathbf{-0.424}$$

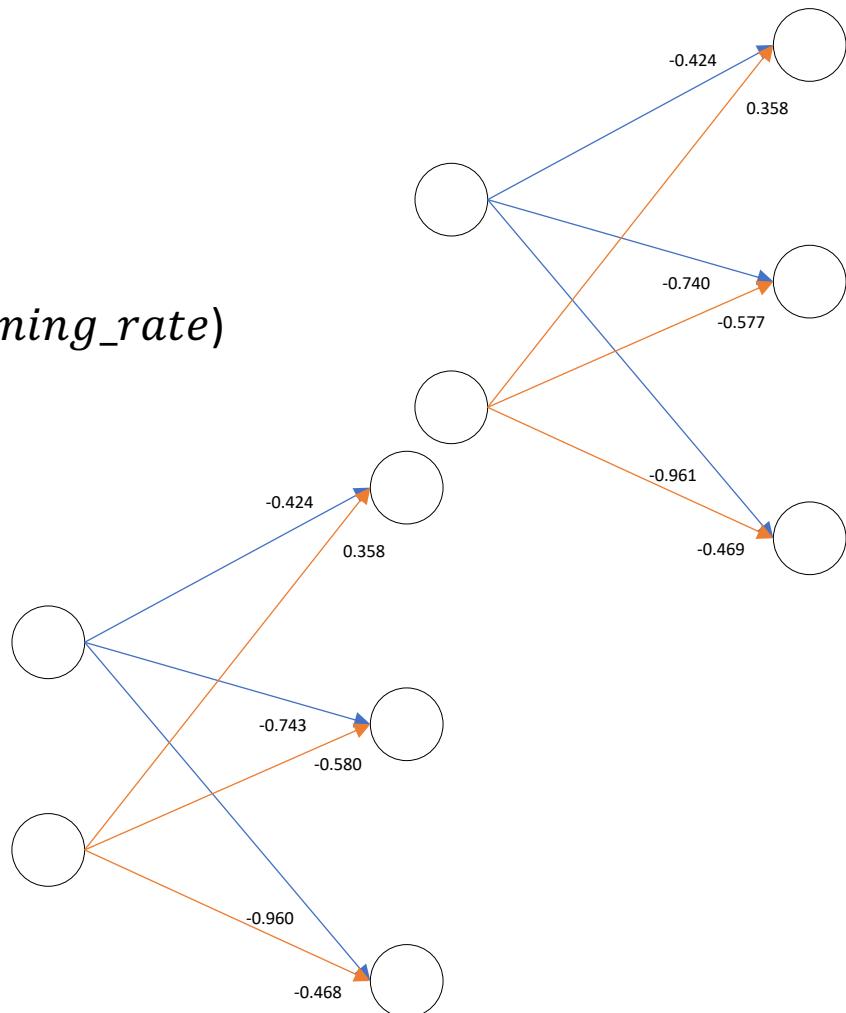
$$0.358 + (-0.000) * 0.3 = \mathbf{0.358}$$

$$-0.740 + (-0.009) * 0.3 = \mathbf{-0.743}$$

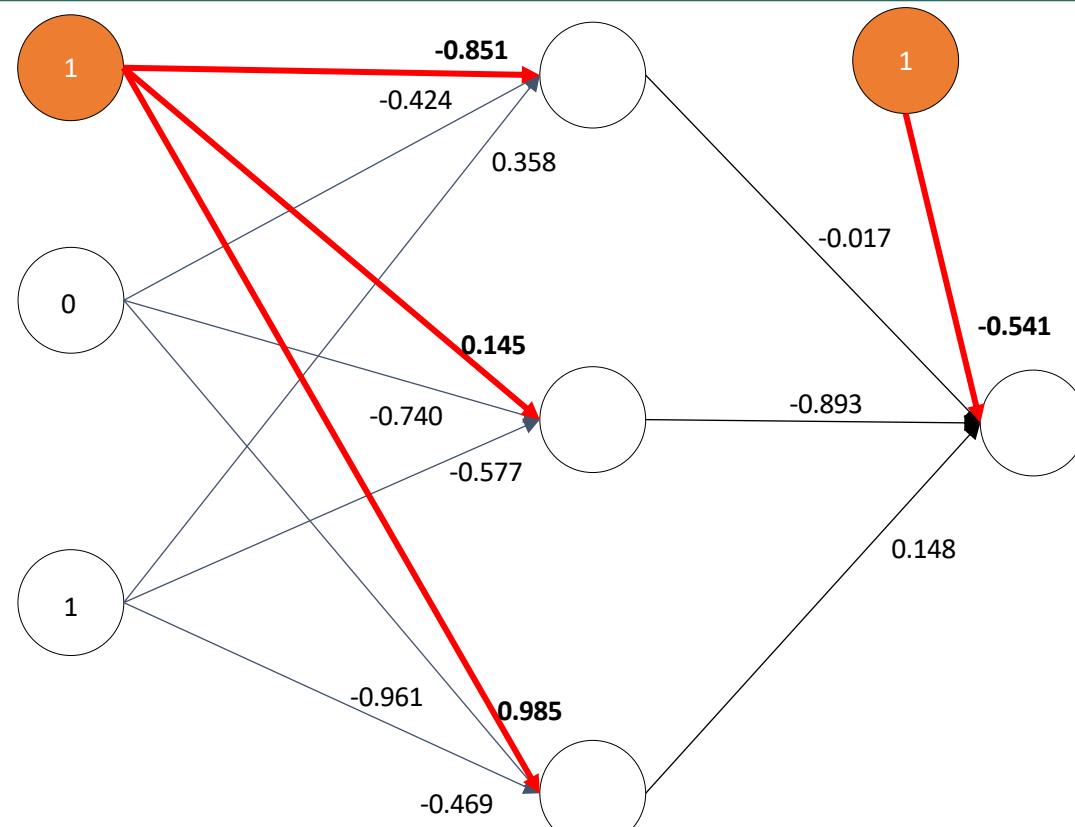
$$-0.577 + (-0.012) * 0.3 = \mathbf{-0.580}$$

$$-0.961 + 0.002 * 0.3 = \mathbf{-0.960}$$

$$-0.469 + 0.002 * 0.3 = \mathbf{-0.468}$$



BIAS



$$output = \sum(inputs * weights) + bias$$

ERROR (LOSS FUNCTION)



The simplest algorithm

$$\text{error} = \text{correct} - \text{prediction}$$

X1	X2	Class	Prediction	Error
0	0	0	0.405	-0.405
0	1	1	0.431	0.569
1	0	1	0.436	0.564
1	1	0	0.458	-0.458

$$\text{Average} = 0.499$$



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

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

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

X1	X2	Class	Prediction	Error
0	0	0	0.405	$(0 - 0.405)^2 = 0.164$
0	1	1	0.431	$(1 - 0.431)^2 = 0.322$
1	0	1	0.436	$(1 - 0.436)^2 = 0.316$
1	1	0	0.458	$(0 - 0.458)^2 = 0.209$

10 (expected) – 5 (prediction) = 5 ($5^2 = 25$)

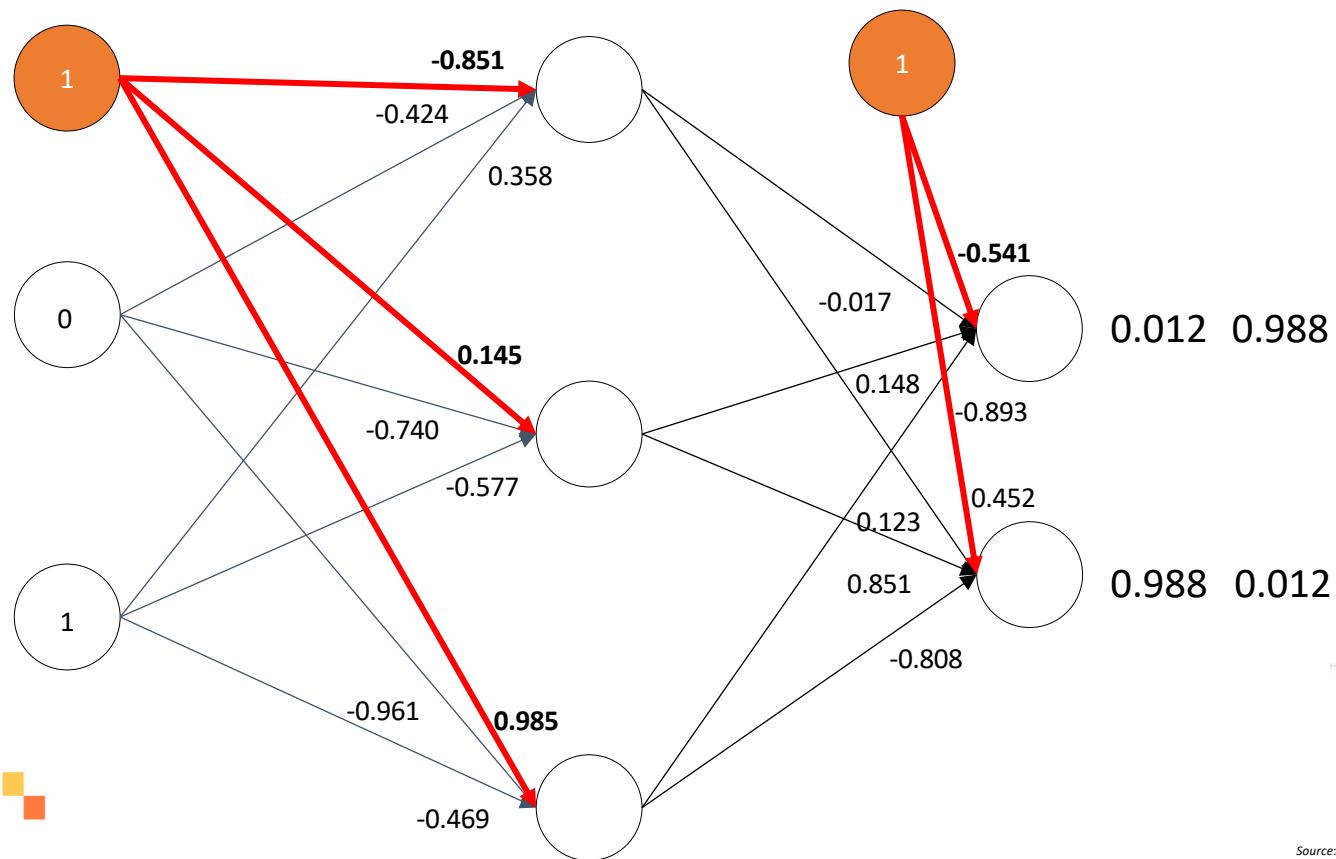
10 (expected) – 8 (prediction) = 2 ($2^2 = 4$)

Sum = 1.011

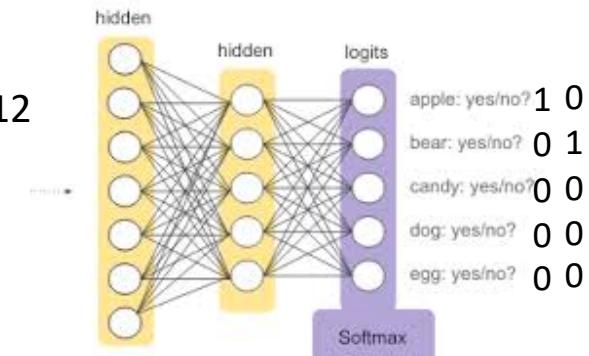
MSE = $1.011 / 4 = 0.252$

RMSE = 0.501

MULTIPLE OUTPUTS



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



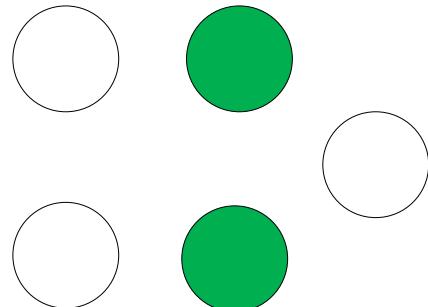
Source: <https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax>

HIDDEN LAYERS



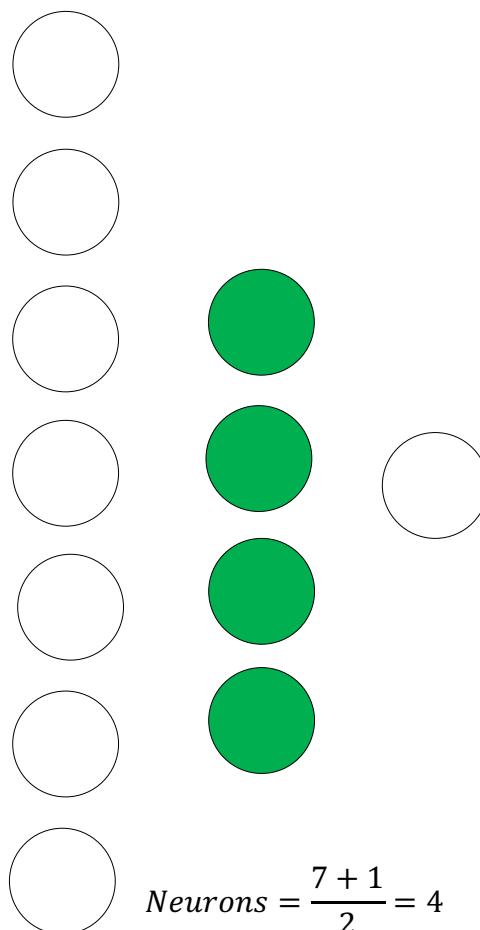
$$\text{Neurons} = \frac{\text{Inputs} + \text{Outputs}}{2}$$

Inputs

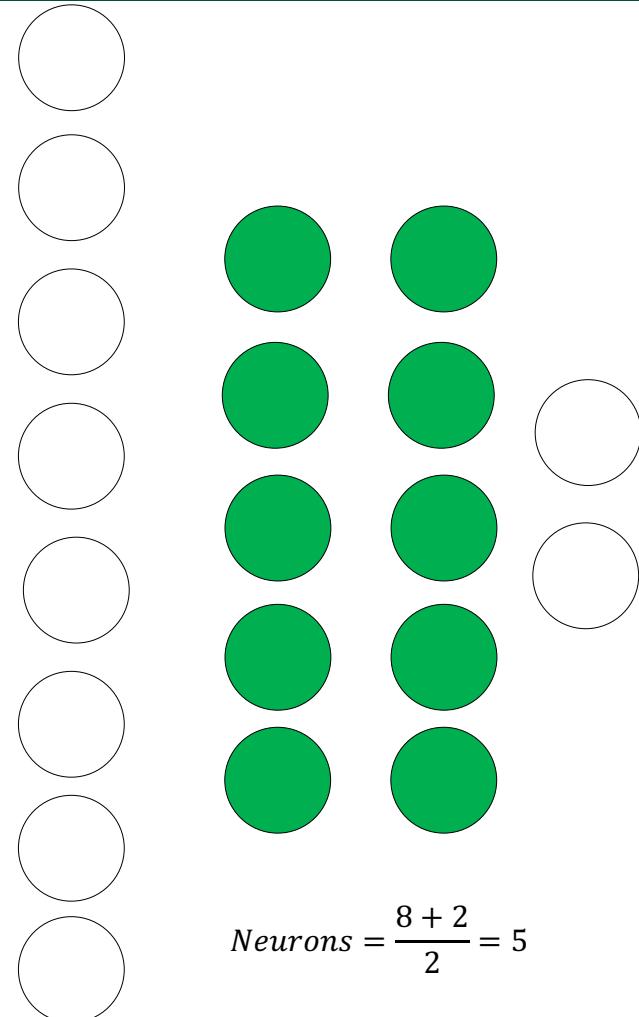


$$\text{Neurons} = \frac{2 + 1}{2} = 1.5$$

Output



$$\text{Neurons} = \frac{7 + 1}{2} = 4$$



$$\text{Neurons} = \frac{8 + 2}{2} = 5$$

HIDDEN LAYERS



- Linearly separable problems do not require hidden layers
- In general, two layers work well
- Deep learning research shows that more layers are essential for more complex problems



HIDDEN LAYERS

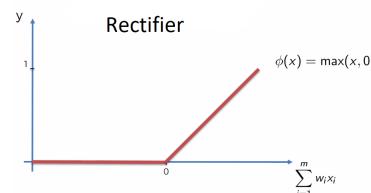
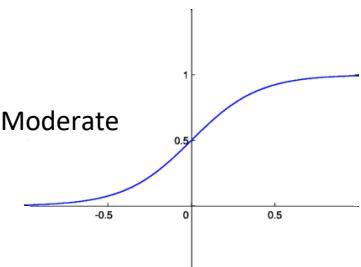
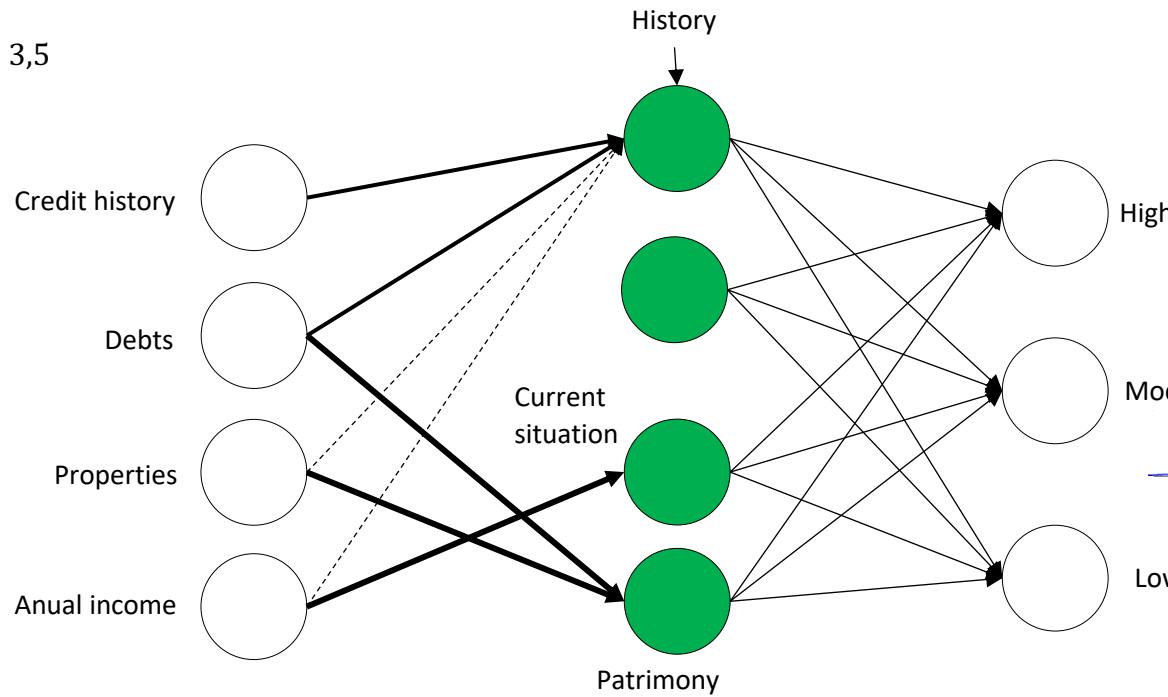


Credit history	Debts	Properties	Anual income	Risk
Bad	High	No	< 15.000	High
Unknown	High	No	>= 15.000 a <= 35.000	High
Unknown	Low	No	>= 15.000 a <= 35.000	Moderate
Unknown	Low	No	> 35.000	High
Unknown	Low	No	> 35.000	Low
Unknown	Low	Yes	> 35.000	Low
Bad	Low	No	< 15.000	High
Bad	Low	Yes	> 35.000	Moderate
Good	Low	No	> 35.000	Low
Good	High	Yes	> 35.000	Low
Good	High	No	< 15.000	High
Good	High	No	>= 15.000 a <= 35.000	Moderate
Good	High	No	> 35.0000	Low
Bad	High	No	>= 15.000 a <= 35.000	High

HIDDEN LAYERS

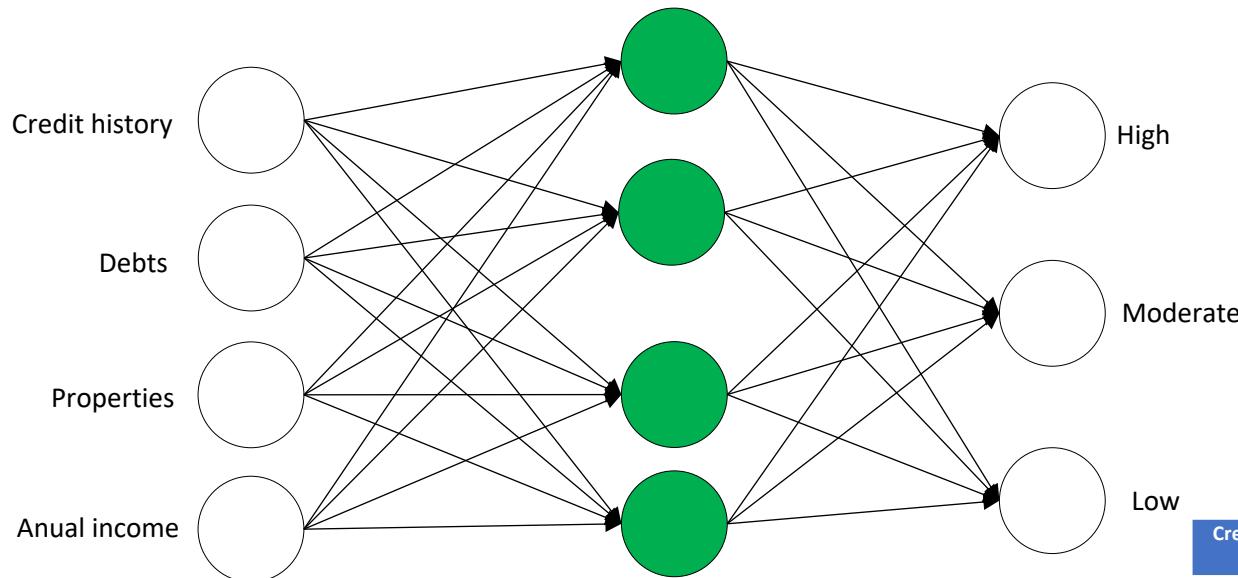


$$Neurons = \frac{4 + 3}{2} = 3,5$$



The higher the activation value,
the more impact the neuron has

OUTPUT LAYER WITH CATEGORICAL DATA



error = correct – prediction

expected output = 1 0 0

prediction = 0.95 0.02 0.03

$$\text{error} = (1 - 0.95) + (0 - 0.02) + (0 - 0.03)$$

$$\text{error} = 0.05 + 0.02 + 0.03 = 0.08$$

Credit history	Debts	Properties	Annual income	Risk
3	1	1	1	100
2	1	1	2	100
2	2	1	2	010
2	2	1	3	100
2	2	1	3	001
2	2	2	3	001
3	2	1	1	100
3	2	2	3	010
1	2	1	3	001
1	1	2	3	001
1	1	1	1	100
1	1	1	2	010
1	1	1	3	001
3	1	1	2	100

STOCHASTIC GRADIENT DESCENT



Credit history	Debts	Properties	Anual income	Risk
3	1	1	1	100
2	1	1	2	100
2	2	1	2	010
2	2	1	3	100
2	2	1	3	001
2	2	2	3	001
3	2	1	1	100
3	2	2	3	010
1	2	1	3	001
1	1	2	3	001
1	1	1	1	100
1	1	1	2	010
1	1	1	3	001
3	1	1	2	100

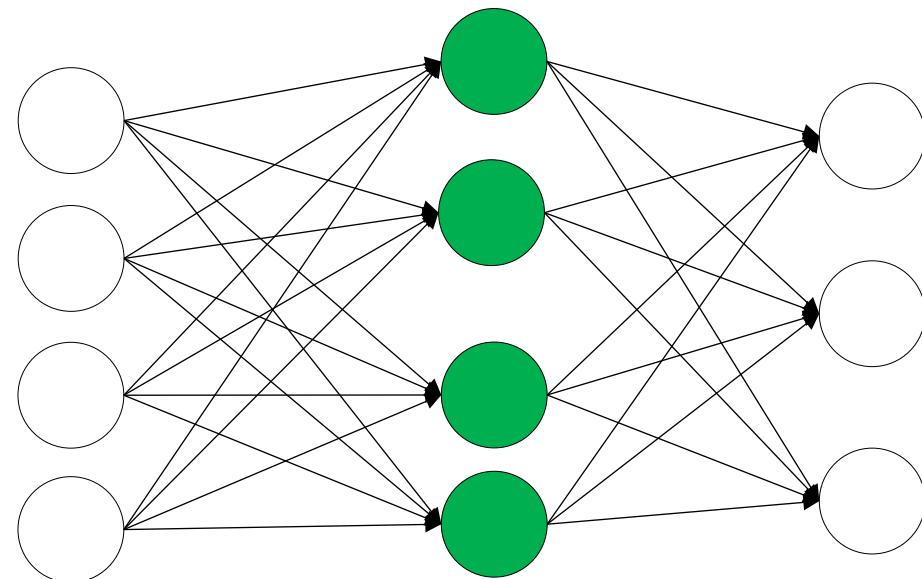
Credit history	Debts	Properties	Anual income	Risk
3	1	1	1	100
2	1	1	2	100
2	2	1	2	010
2	2	1	3	100
2	2	2	3	001
3	2	1	1	100
3	2	2	3	010
1	2	1	3	001
1	1	2	3	001
1	1	1	1	100
1	1	1	2	010
1	1	1	3	001
3	1	1	2	100

Batch gradient descent

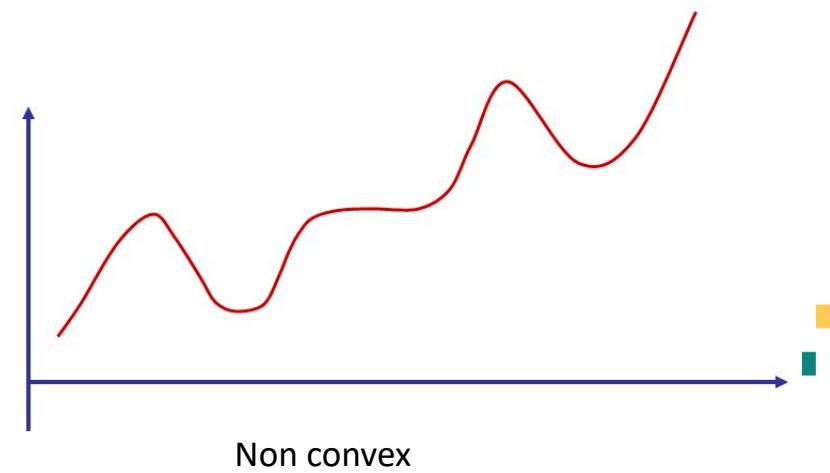
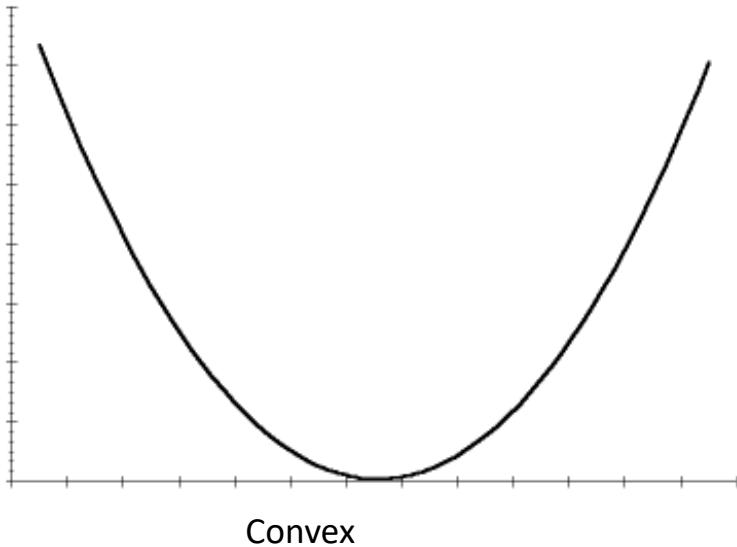
Calculate the error for all instances and then update the weights

Stochastic gradient descent

Calculate the error for each instance and then update the weights



CONVEX AND NON CONVEX



GRADIENT DESCENT

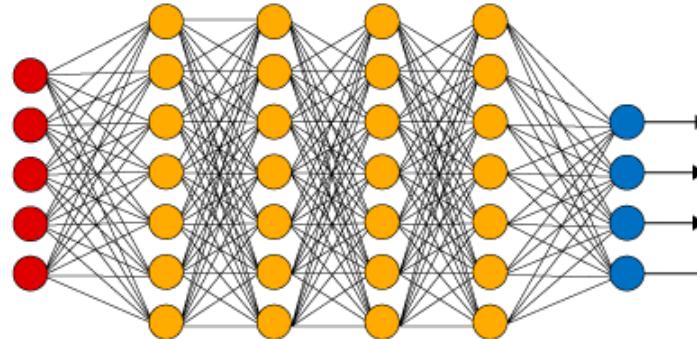


- Stochastic gradient descent
 - Prevent local minimums (non convex)
 - Faster
- Mini batch gradient descent
 - Select a pre-defined number of instances in order to calculate the error and update the weights





- 90's: SVM (Support Vector Machines)
- From 2006, several algorithms were created for training neural networks
- Two or more hidden layers



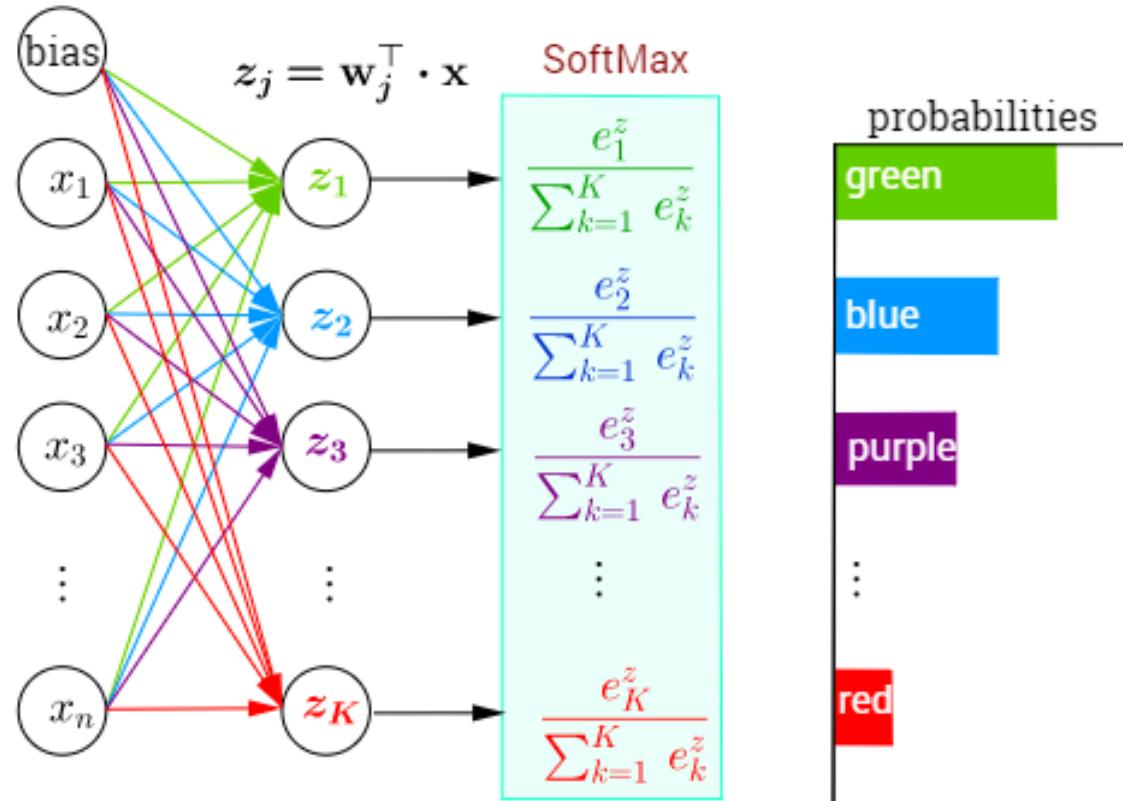


- Convolutional neural networks
- Recurrent neural networks
- Autoencoders
- GANs (Generative adversarial networks)



SOFTMAX

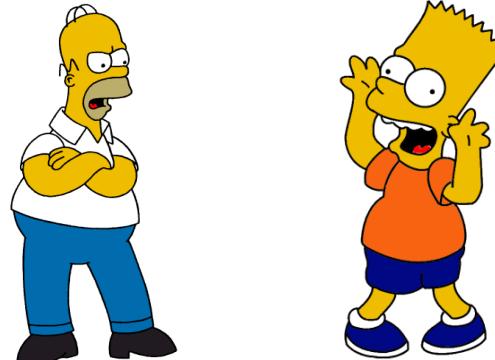
$$Y = \frac{e(x)}{\sum e(x)}$$



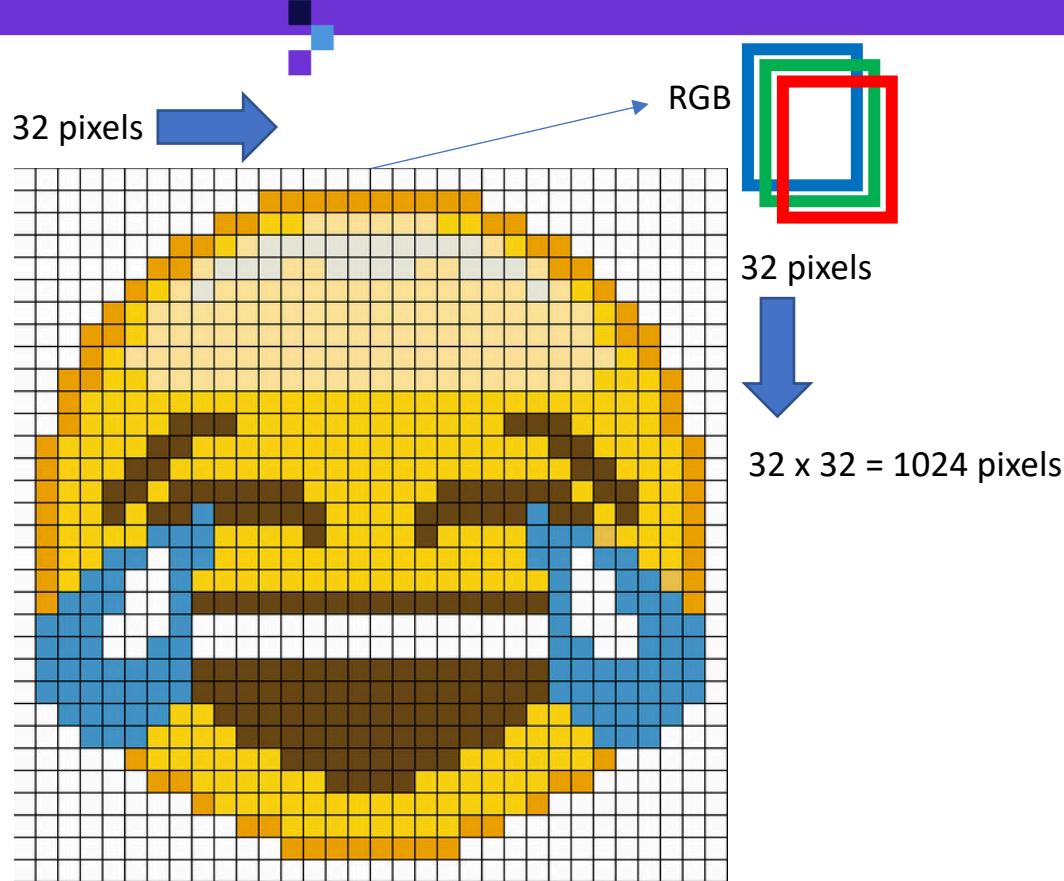
PLAN OF ATTACK – NEURAL NETWORKS FOR IMAGE CLASSIFICATION



1. Intuition about neural networks
2. Neural network using the pixels of the images
3. Feature extractor with OpenCV
4. Neural network using feature extraction

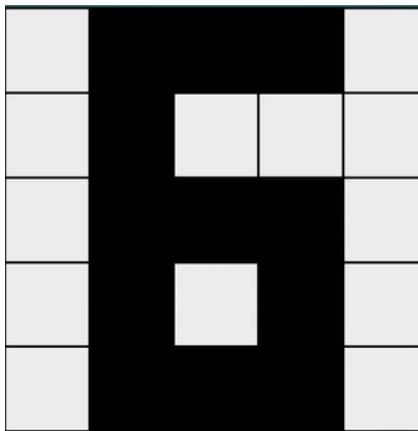


PIXELS

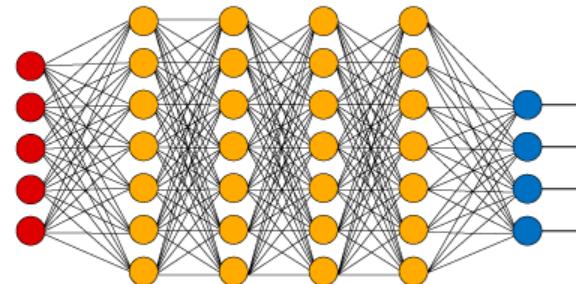


PIXELS

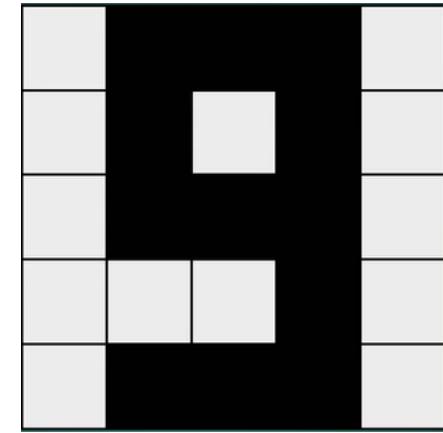
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0



0,1,1,1,0,0,1,0,0,0,1,1,1,0,0,0,1,0,1,0,0,1,1,1,0,6



0,1,1,1,0,0,1,0,1,0,0,1,1,1,0,0,0,0,1,0,0,1,1,1,0,9



FEATURE EXTRACTION



Brown (mouth)



Blue (pants)



Gray (shoes)



Orange (T-Shirt)

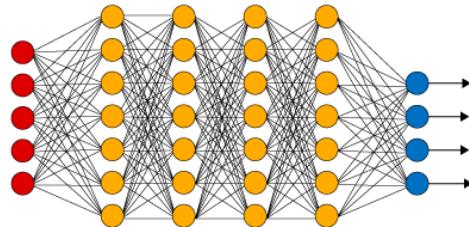


Blue (shorts)

Blue (sneakers)

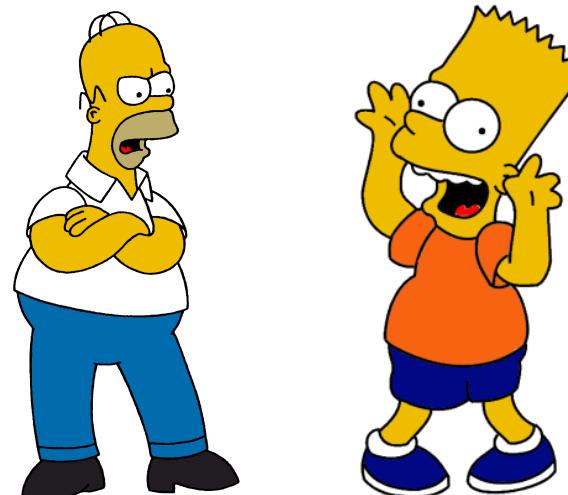


```
0.0,0.0,0.087449393,9.861538462,8.660728745,2.257489879,0  
0.0,0.0,0.012995452,4.980506823,0.0,0.0,0  
0.0,0.0,0.014219662,9.668452534,1.034365711,1.034365711,0  
0.0,0.0,0.013168984,10.734478157,5.981352718,2.126352016,0  
0.979401078,16.958948664,2.349976559,2.48769339,0.0,0.0,0,1  
0.058665773,15.566543748,0.306738183,0.0,0.0,0.0,1  
1.829063147,14.074792961,2.135093168,0.0,0.0,0.0,1  
2.804717308,0.152271939,0.026014568,0.0,0.0,0.0,1  
2.510348847,0.0,0.037677839,0.0,0.0,0.0,1
```



PLAN OF ATTACK – CONVOLUTIONAL NEURAL NETWORKS FOR IMAGE CLASSIFICATION

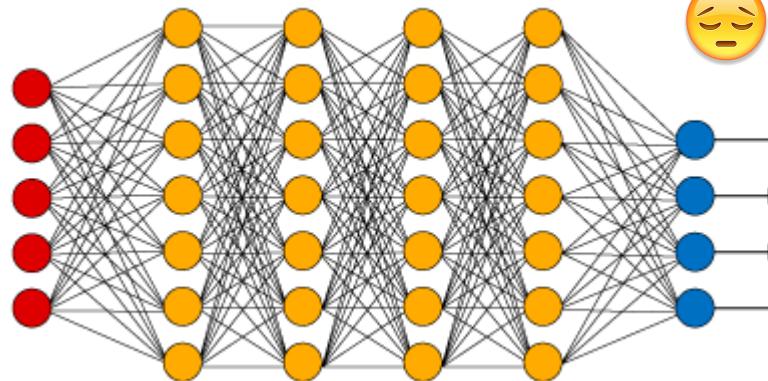
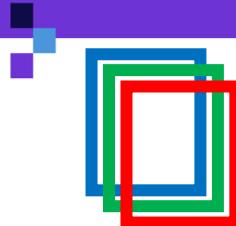
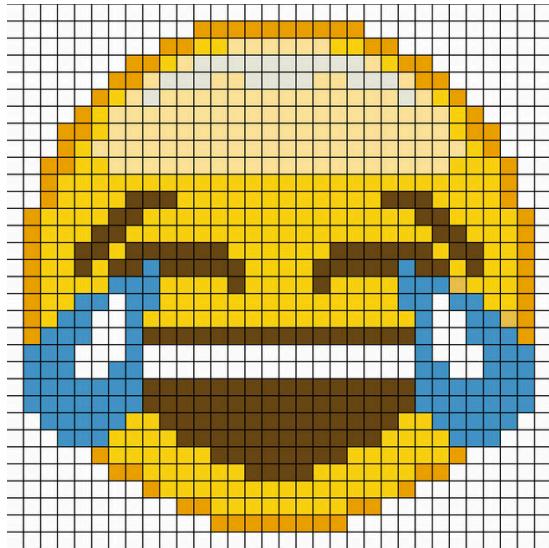
1. Intuition about convolutional neural networks
2. Types of neural networks for image classification
3. Convolutional neural network for image classification



PIXELS



$$32 \times 32 = 1.024 \times 3 = \mathbf{3.072 \text{ inputs}}$$



- 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



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

Image

x



1	0	0
1	0	1
0	1	1

Feature detector

=

0				

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

\times

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



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

Image



x

1	0	0
1	0	1
0	1	1

Feature detector

=

0	1	0	1	

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

Image



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

Feature maps

The network will decide which feature detector to use

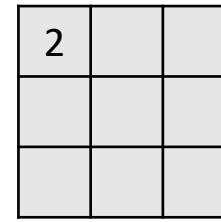
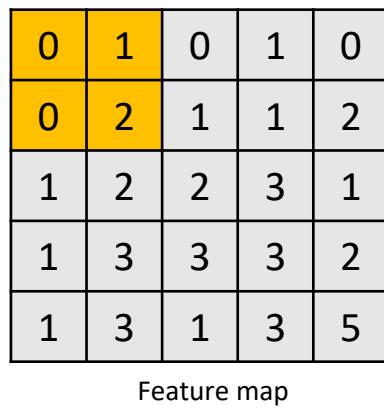
The convolutional layer is the set of feature maps



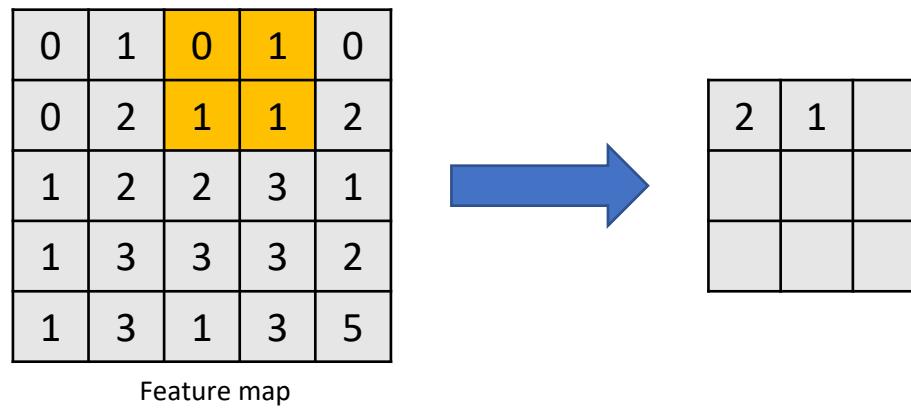
STEP 2: POOLING



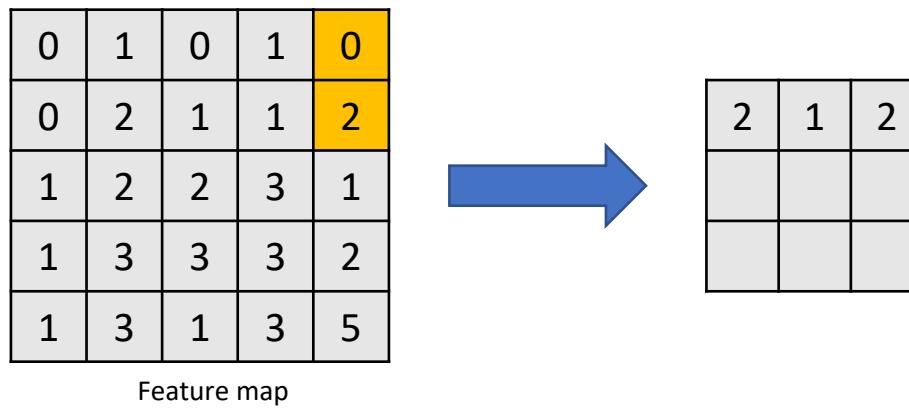
STEP 2: POOLING



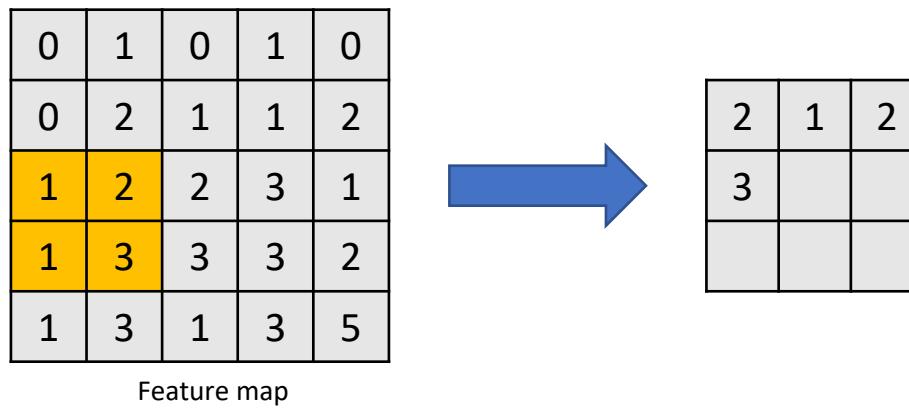
STEP 2: POOLING



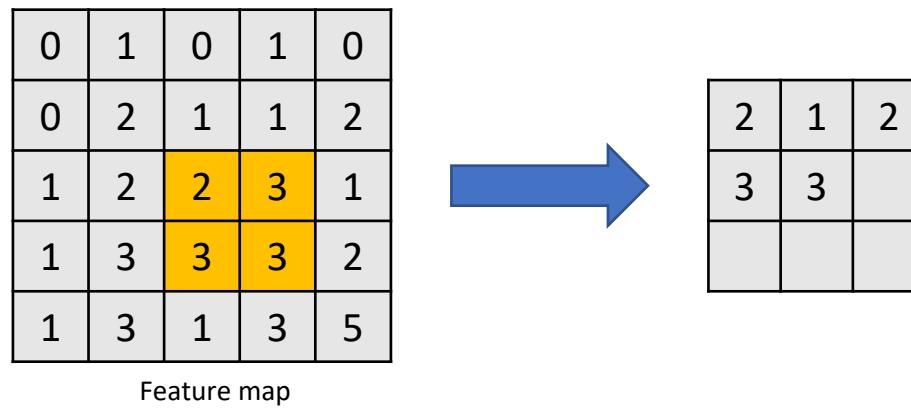
STEP 2: POOLING



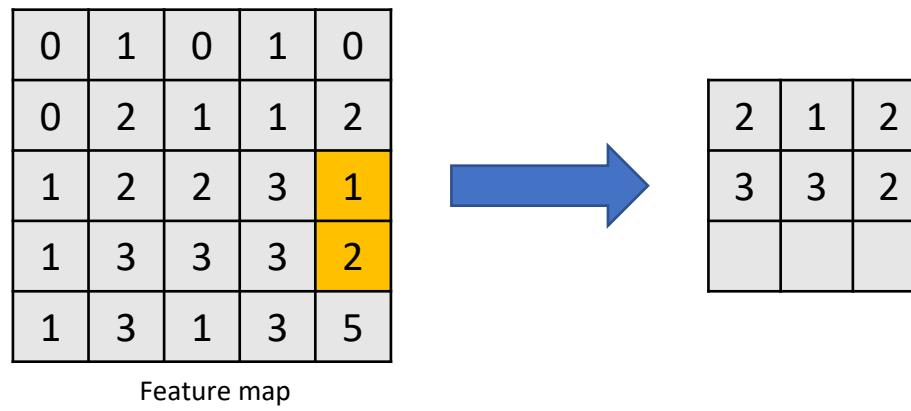
STEP 2: POOLING



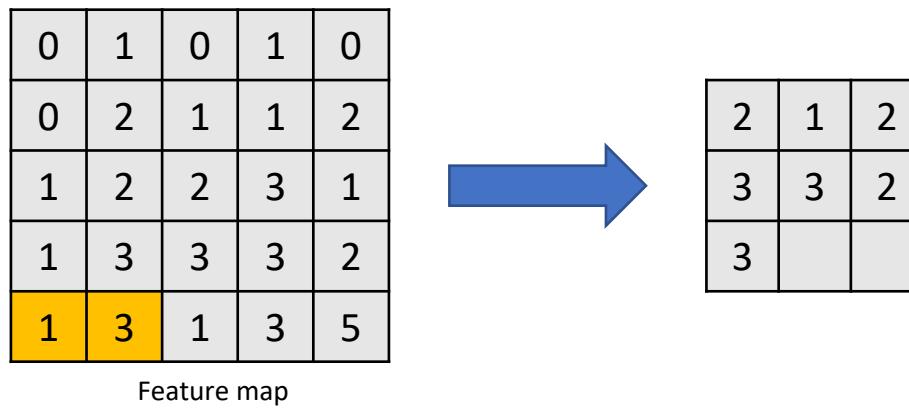
STEP 2: POOLING



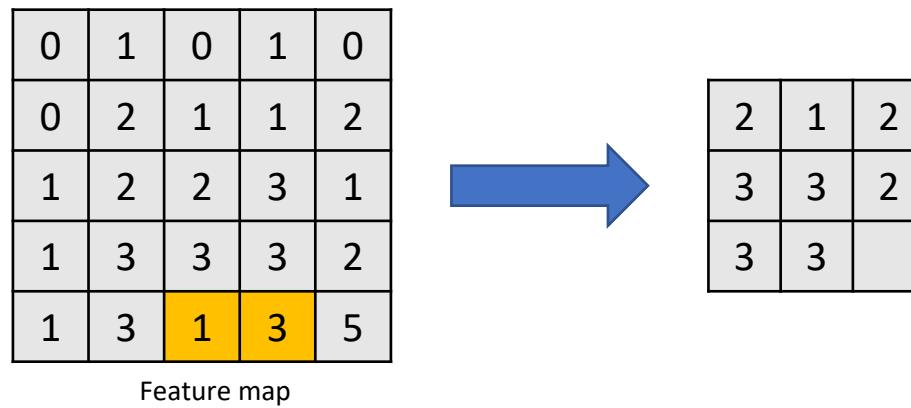
STEP 2: POOLING



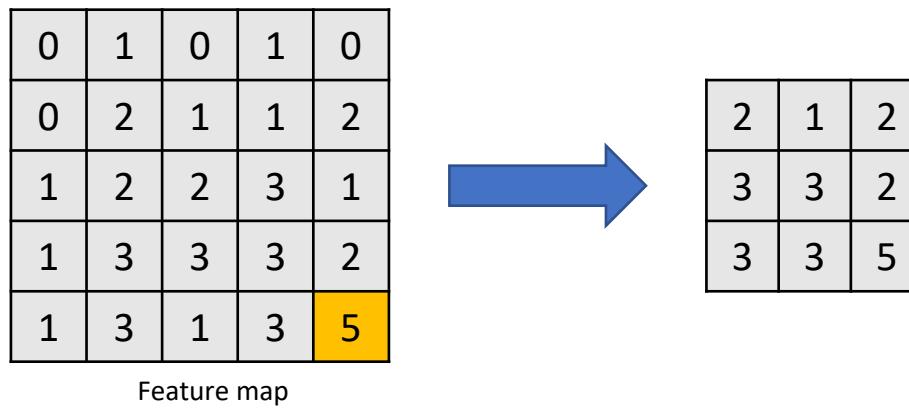
STEP 2: POOLING



STEP 2: POOLING



STEP 2: POOLING

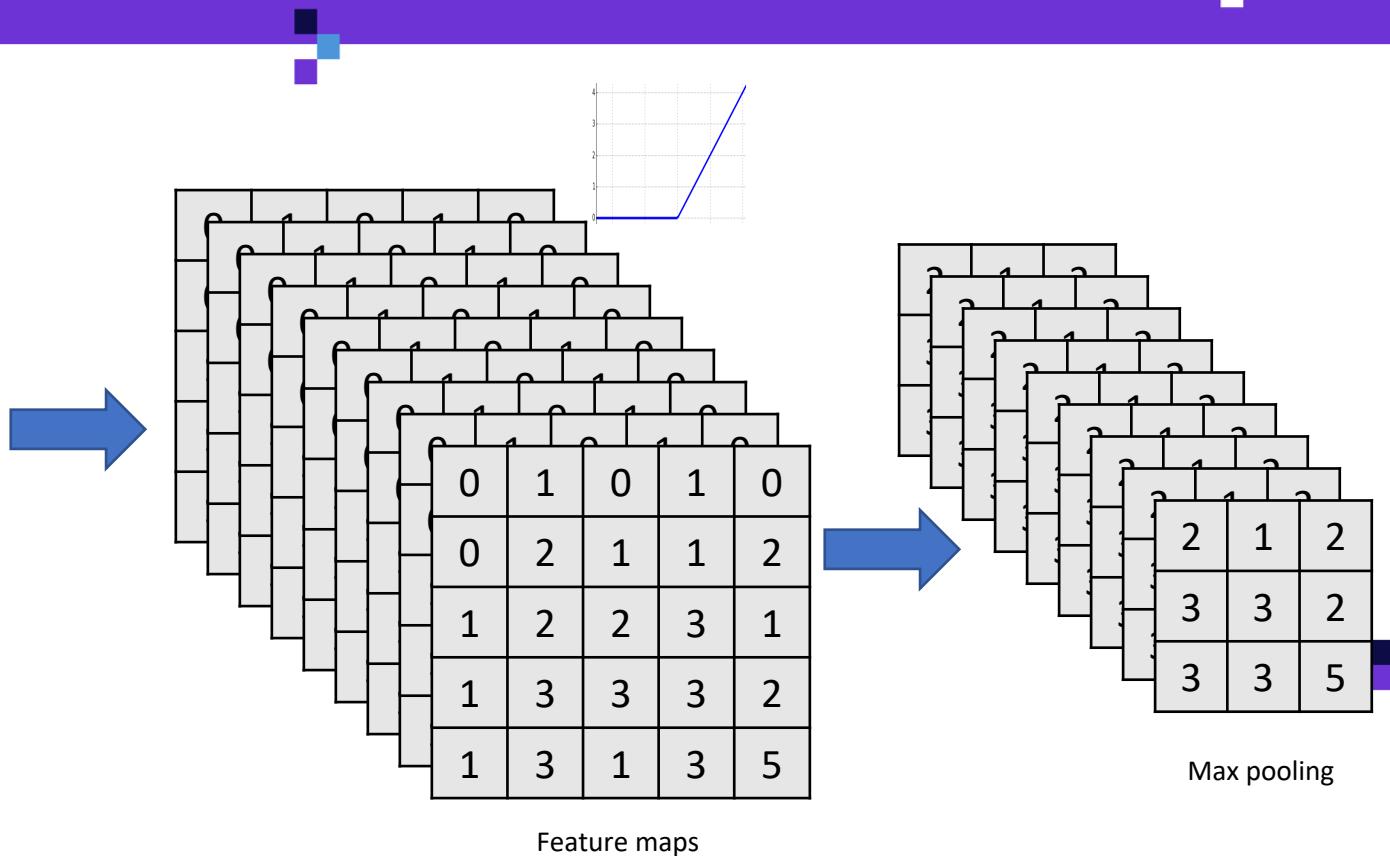


CONVOLUTIONAL NEURAL NETWORK – POOLING

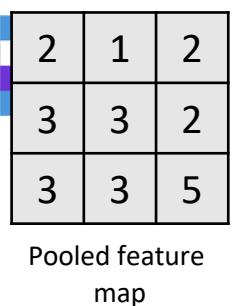


0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	1	0	1	1	1
0	1	0	1	1	0	0	0
0	1	0	1	1	0	1	1
0	1	0	0	0	1	1	1

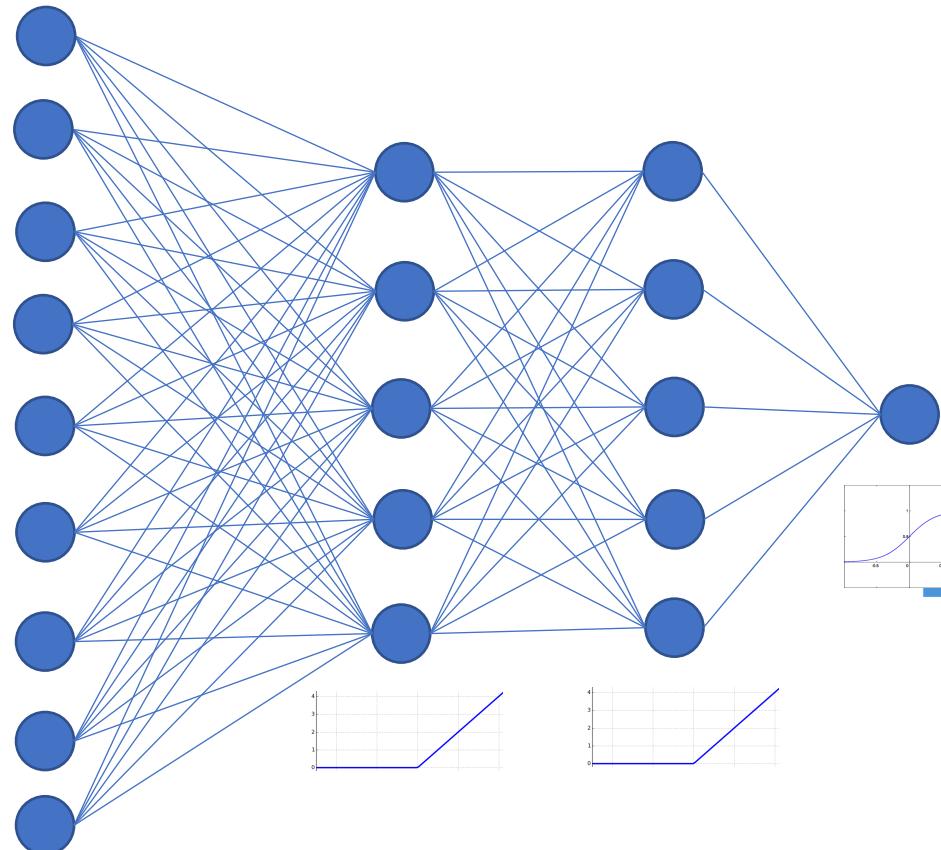
Image



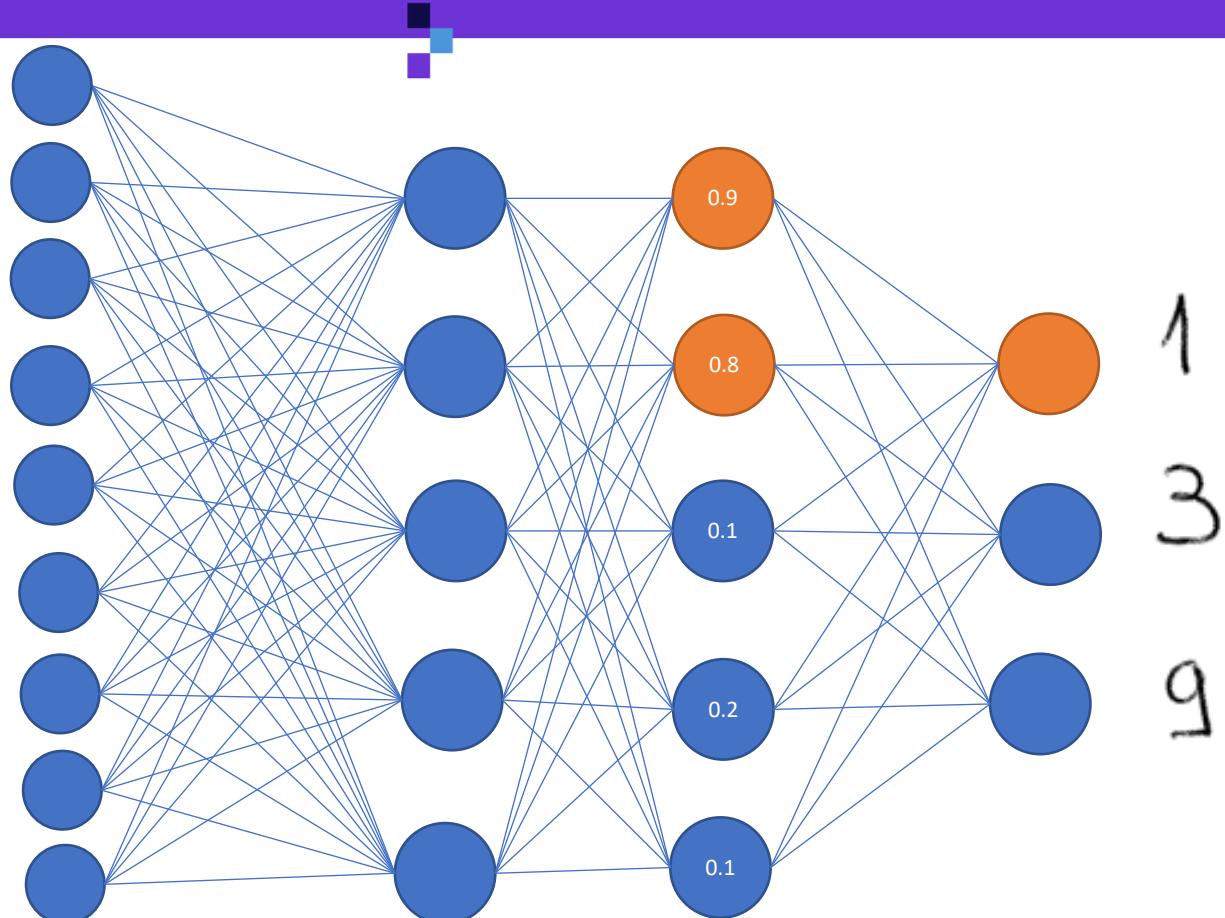
STEP 3: FLATTENING



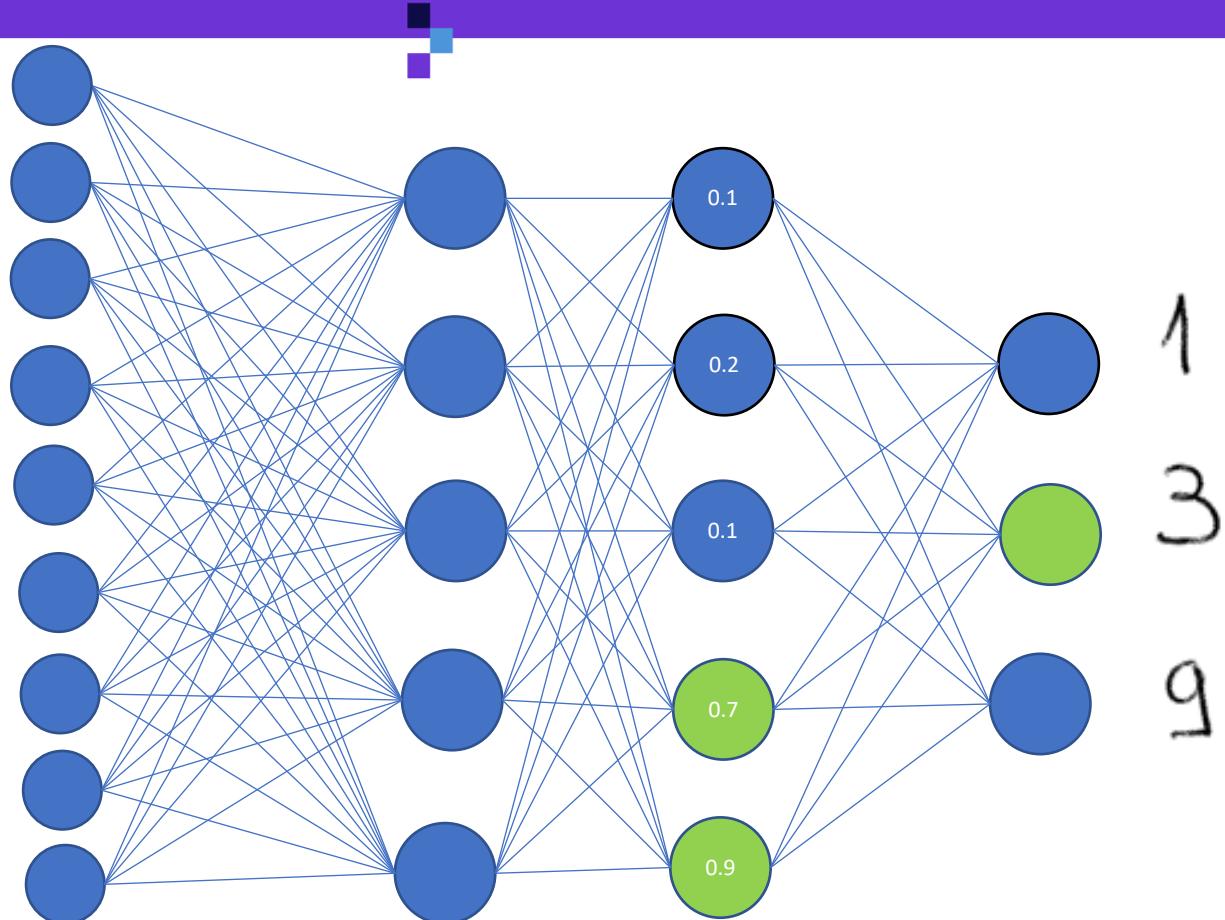
2
1
2
3
3
2
3
3
5



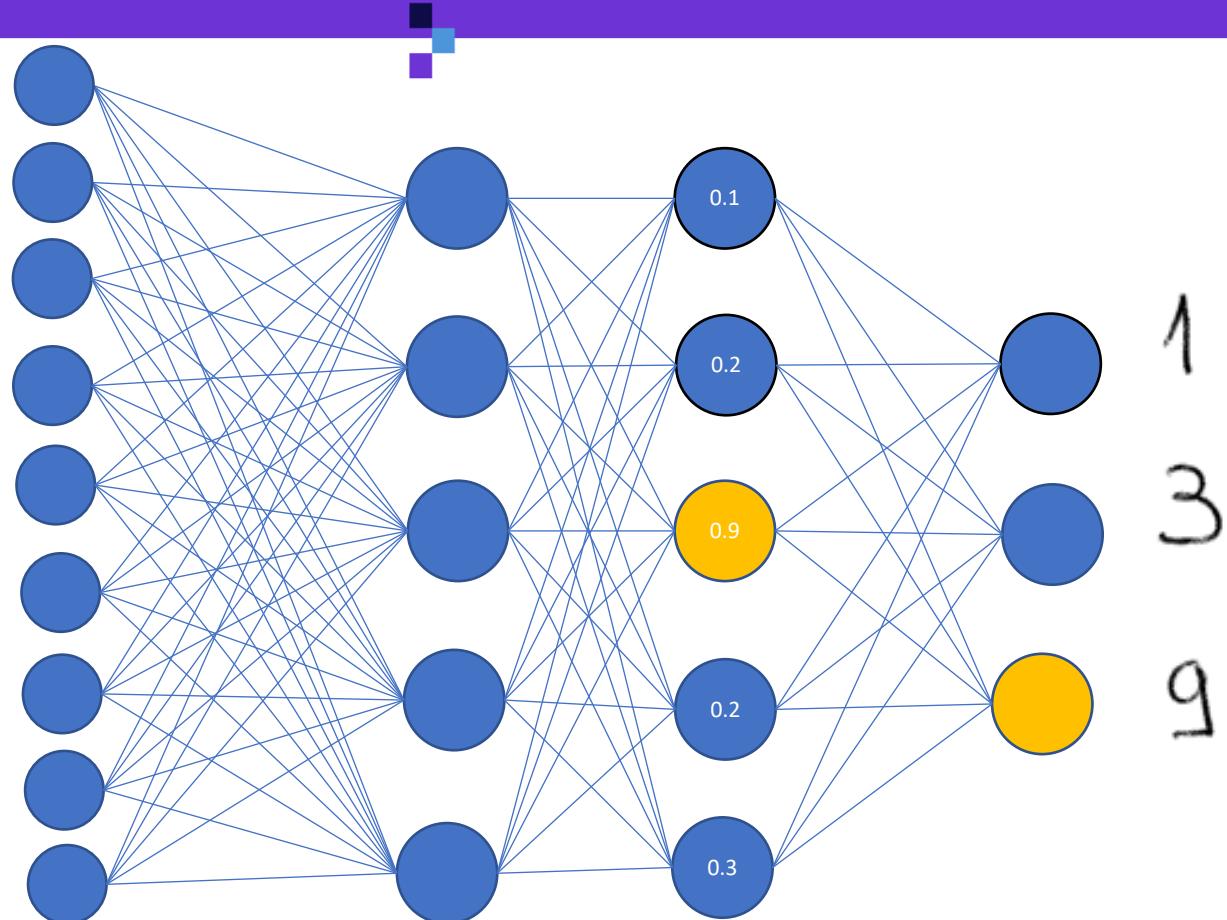
STEP 3: DENSE NEURAL NETWORK



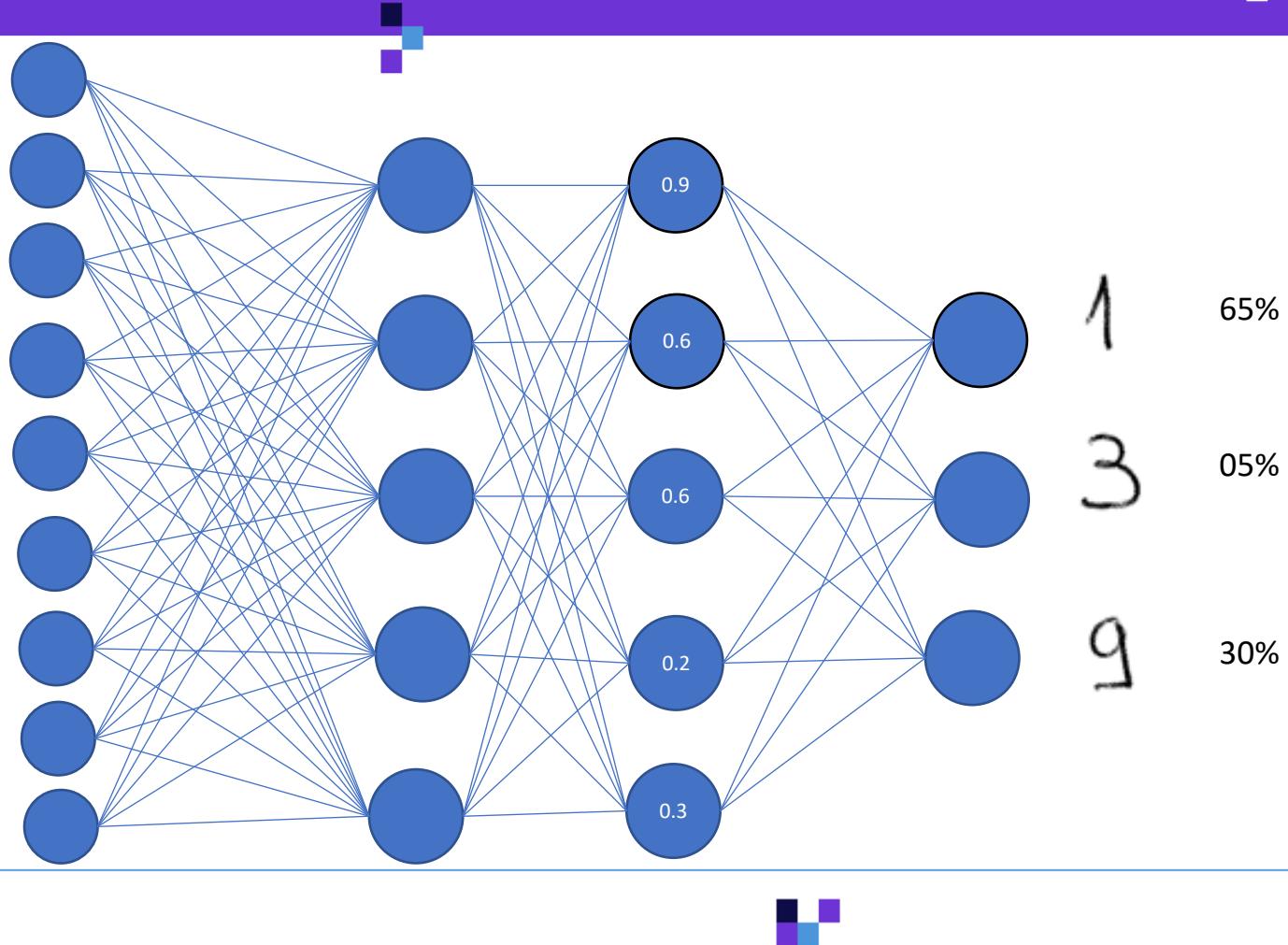
STEP 3: DENSE NEURAL NETWORK



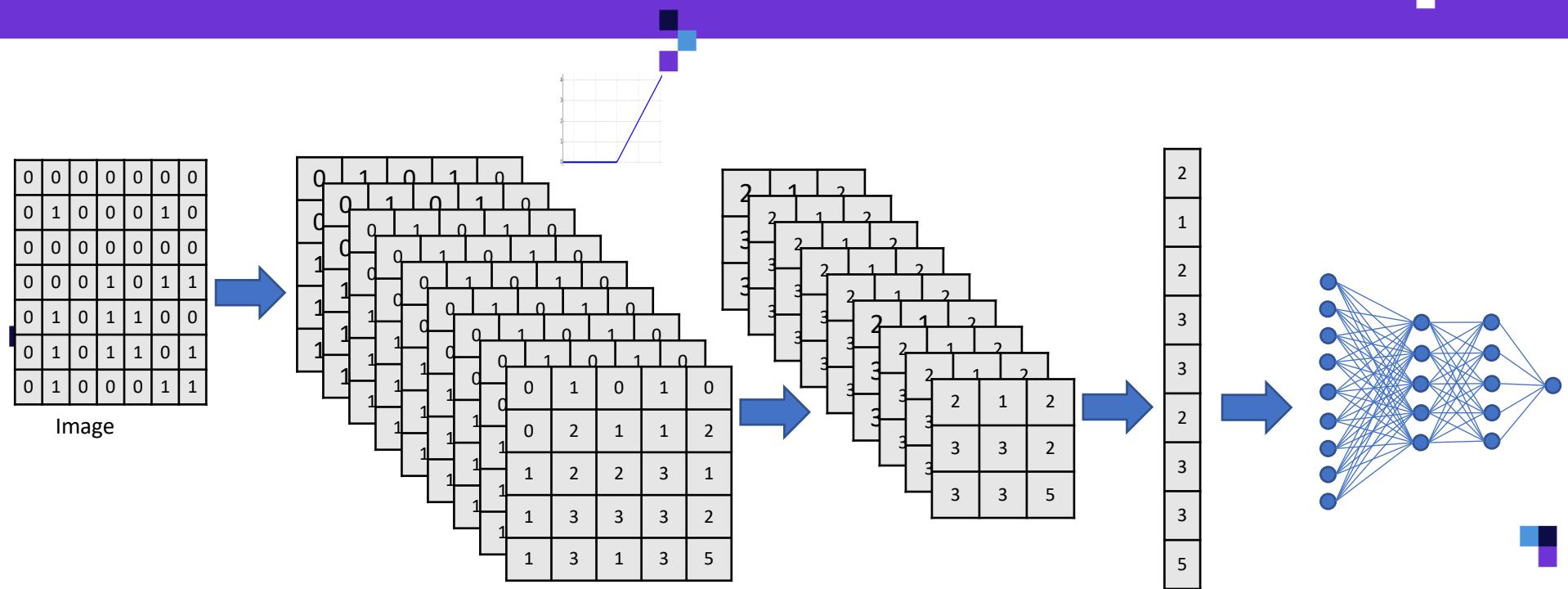
STEP 3: DENSE NEURAL NETWORK



STEP 3: DENSE NEURAL NETWORK



CONVOLUTIONAL NEURAL NETWORK



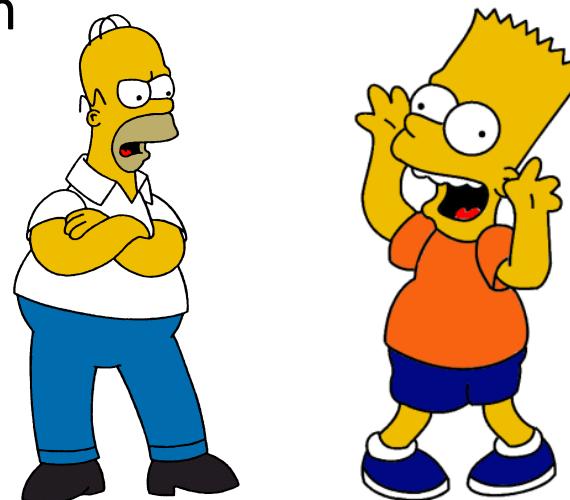
Training using gradient descent

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



PLAN OF ATTACK – TRANSFER LEARNING AND FINE TUNING FOR IMAGE CLASSIFICATION

1. Intuition about transfer learning
2. Implementation
3. Intuition about fine tuning
4. Implementation



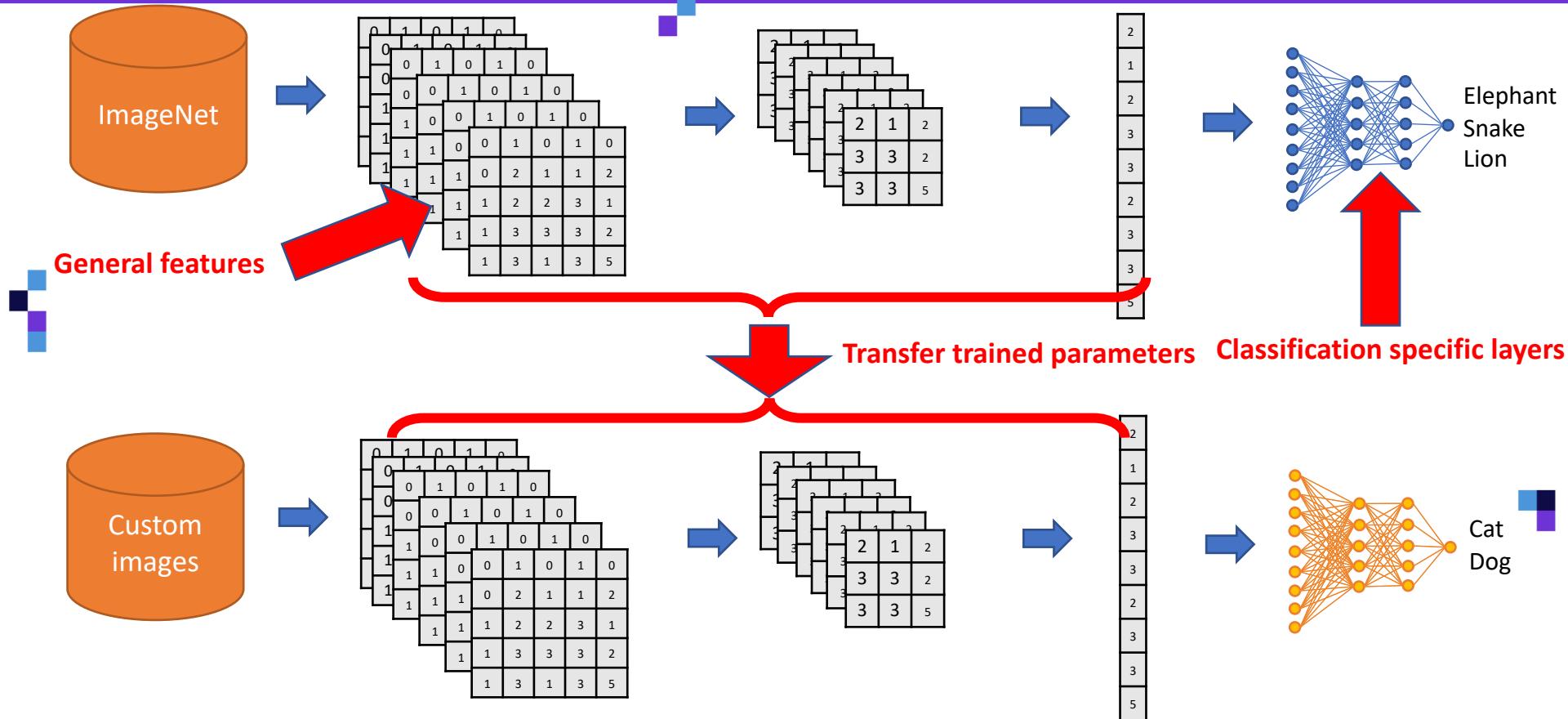
TRANSFER LEARNING



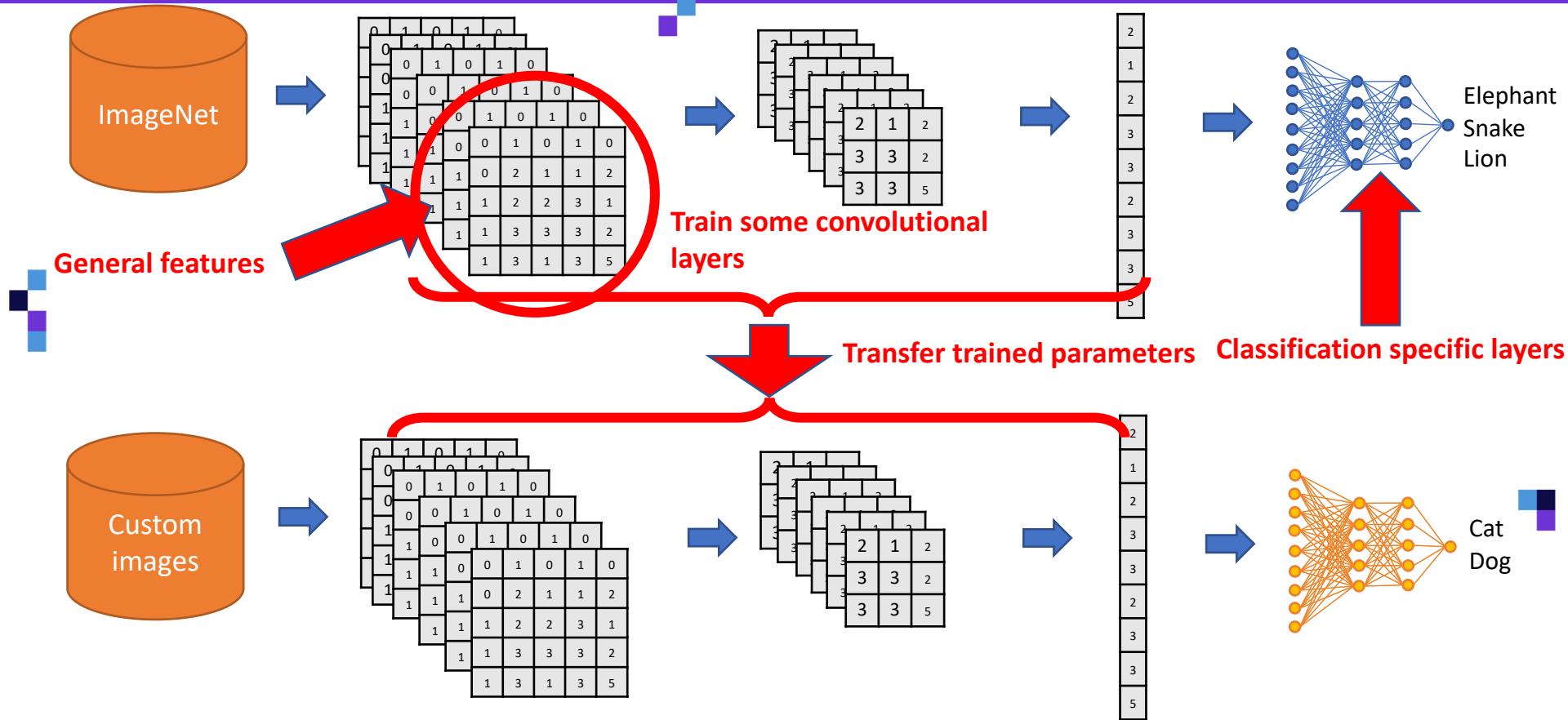
KNOWLEDGE TRANSFER



TRANSFER LEARNING



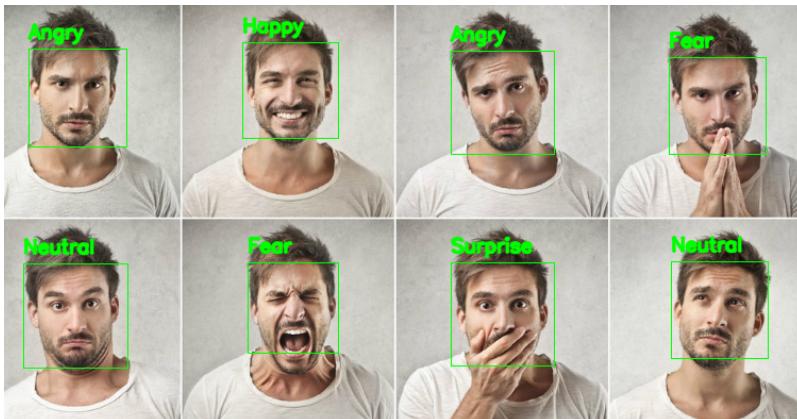
FINE TUNING



PLAN OF ATTACK – NEURAL NETWORKS FOR CLASSIFICATION OF EMOTIONS



1. Implementation of convolutional neural networks
2. Detecting emotions in images
3. Detecting emotions in videos



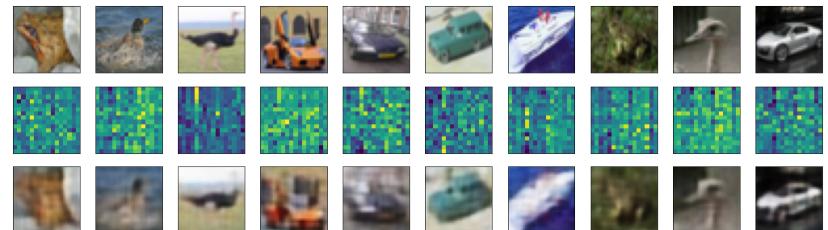
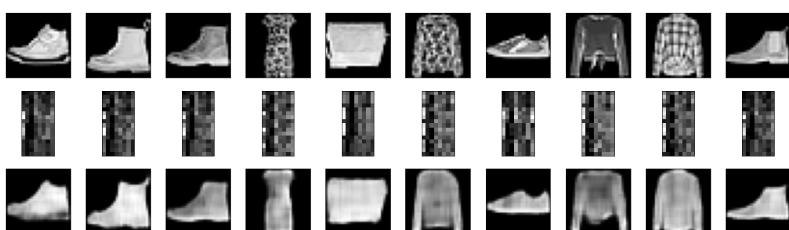
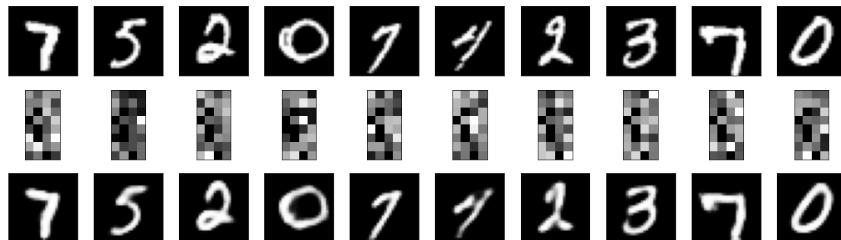
Source: EPFL



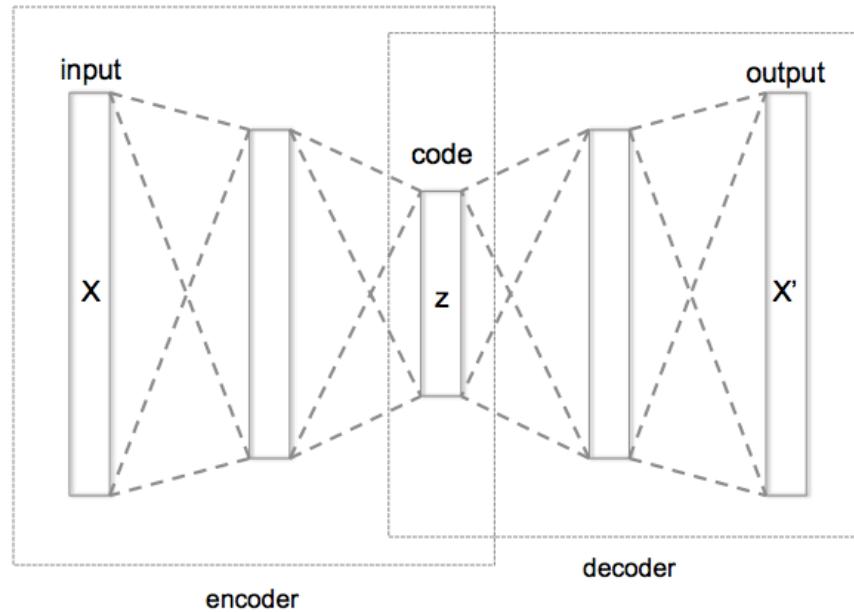
PLAN OF ATTACK – AUTOENCODERS



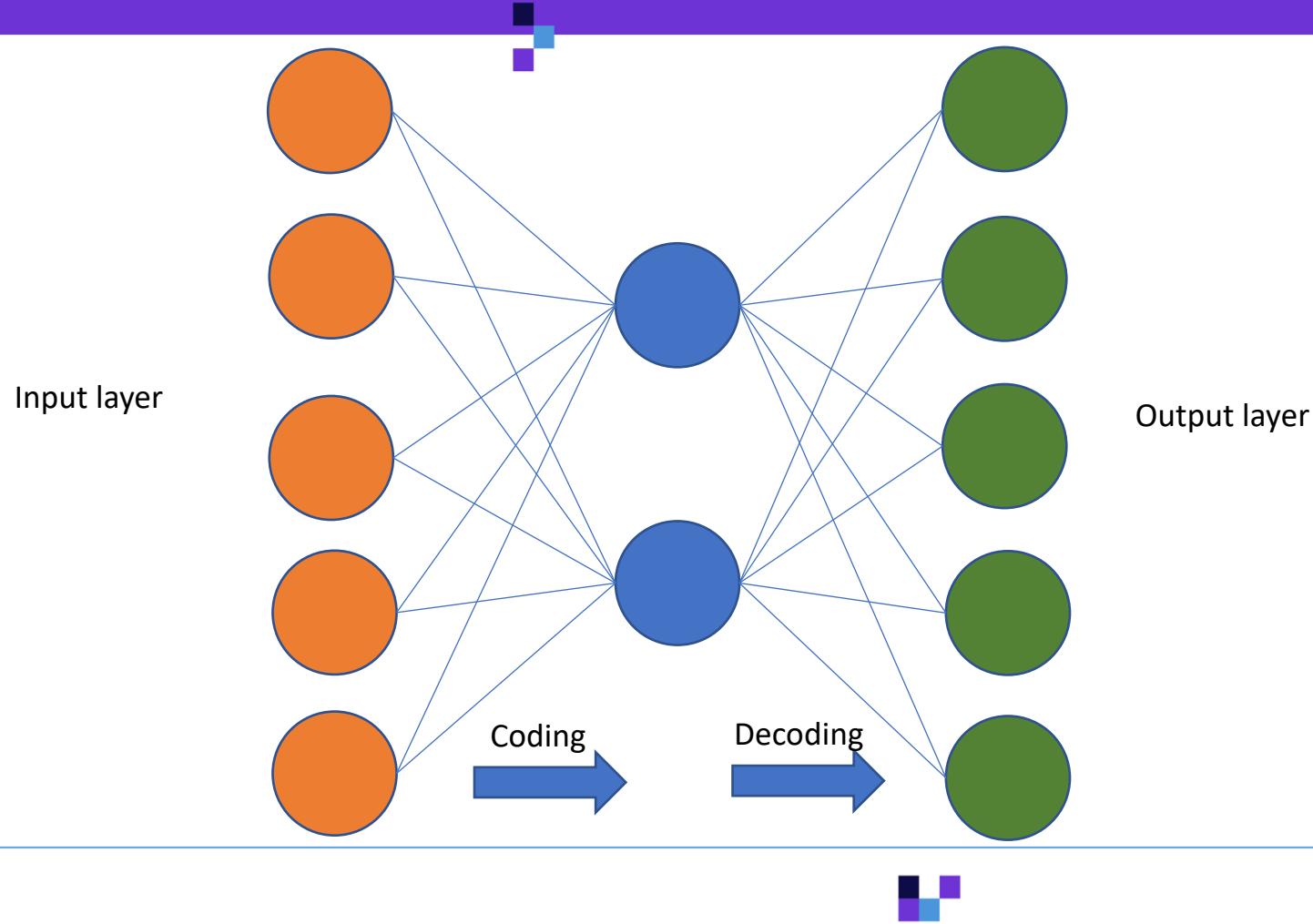
1. Intuition about autoencoders
2. Implementation of linear autoencoders
3. Implementation of convolutional autoencoders



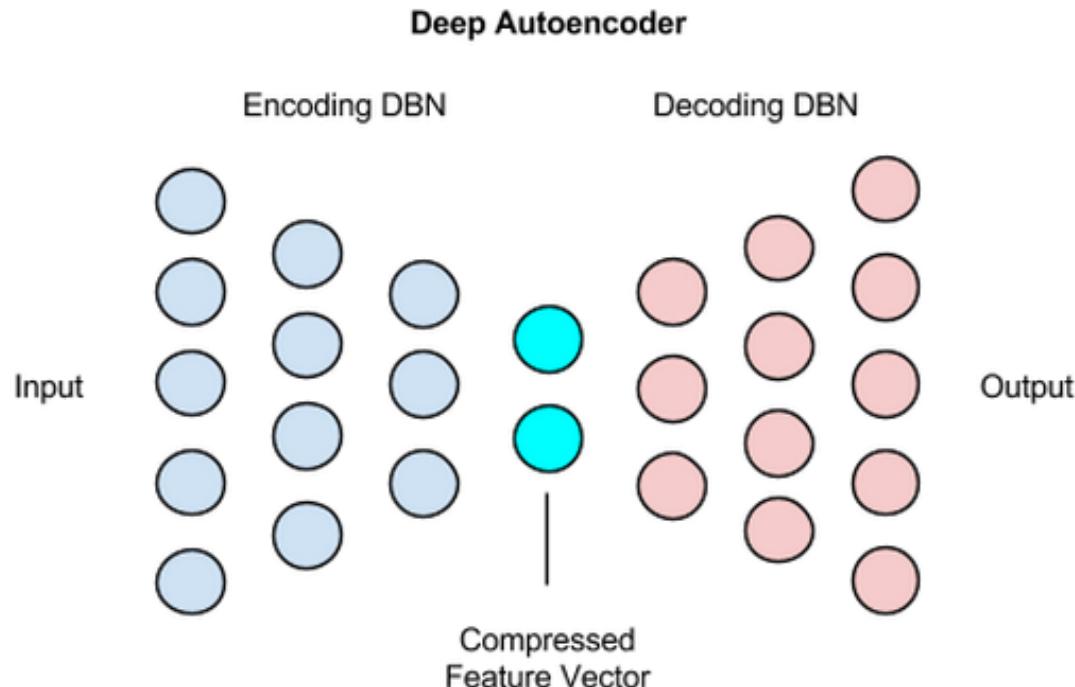
AUTOENCODERS



LINEAR AUTOENCODERS



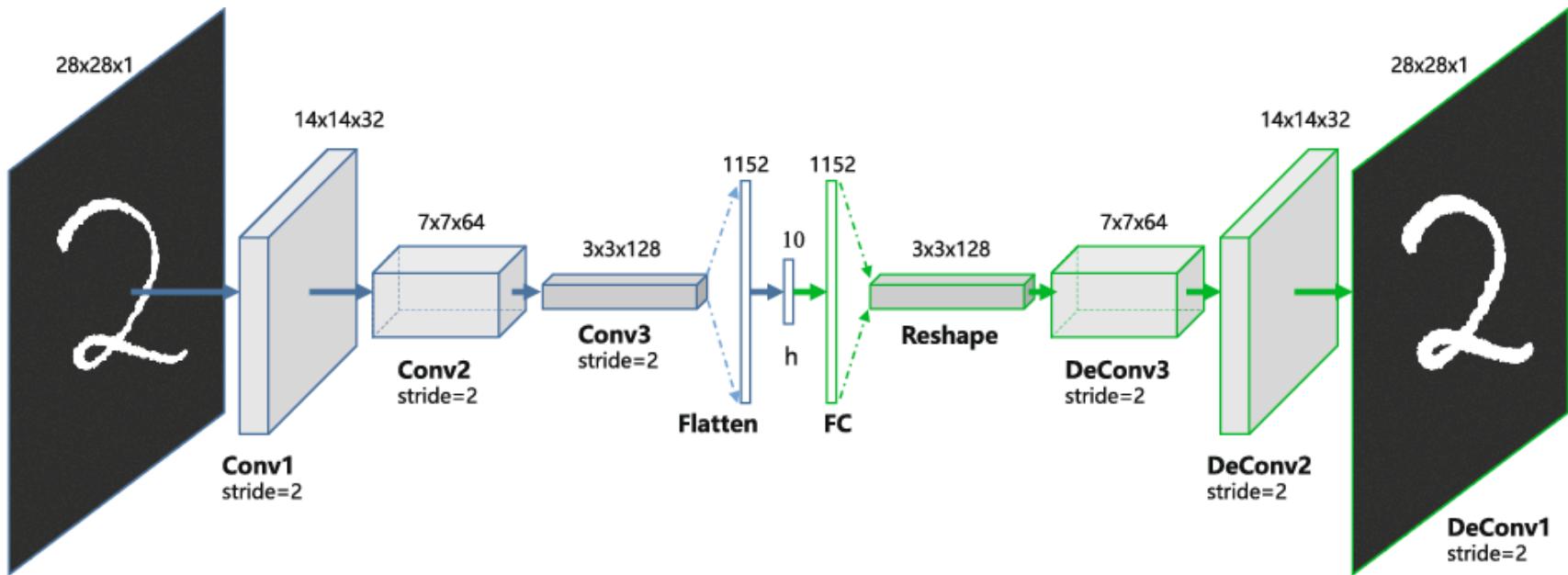
LINEAR AUTOENCODERS



Source: <https://skymind.ai/wiki/deep-autoencoder>



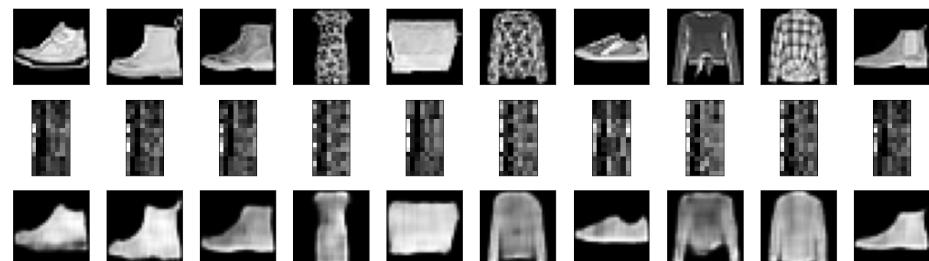
CONVOLUTIONAL AUTOENCODERS



Source: <https://www.semanticscholar.org/paper/Deep-Clustering-with-Convolutional-Autoencoders-Guo-Liu/b4c8c77fe8ac7aa07a9e41827955d38a24f6137f>

AUTOENCODERS – APPLICATIONS

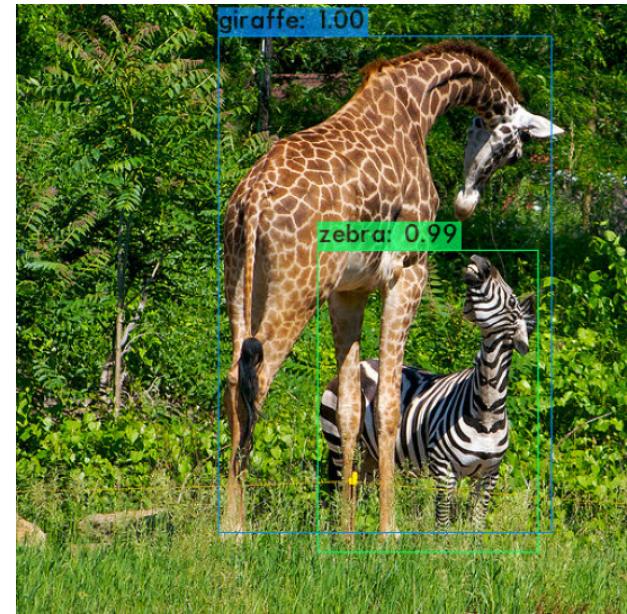
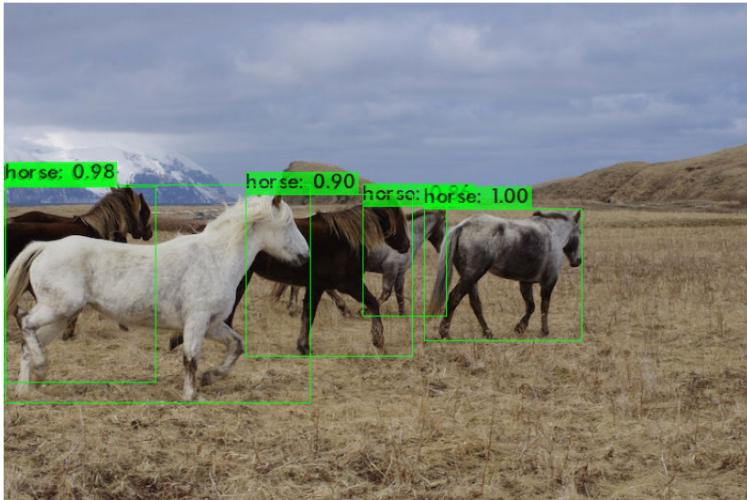
1. Noise removal
2. Image compression
3. Fraud detection
4. Dimensionality reduction (similar to PCA)



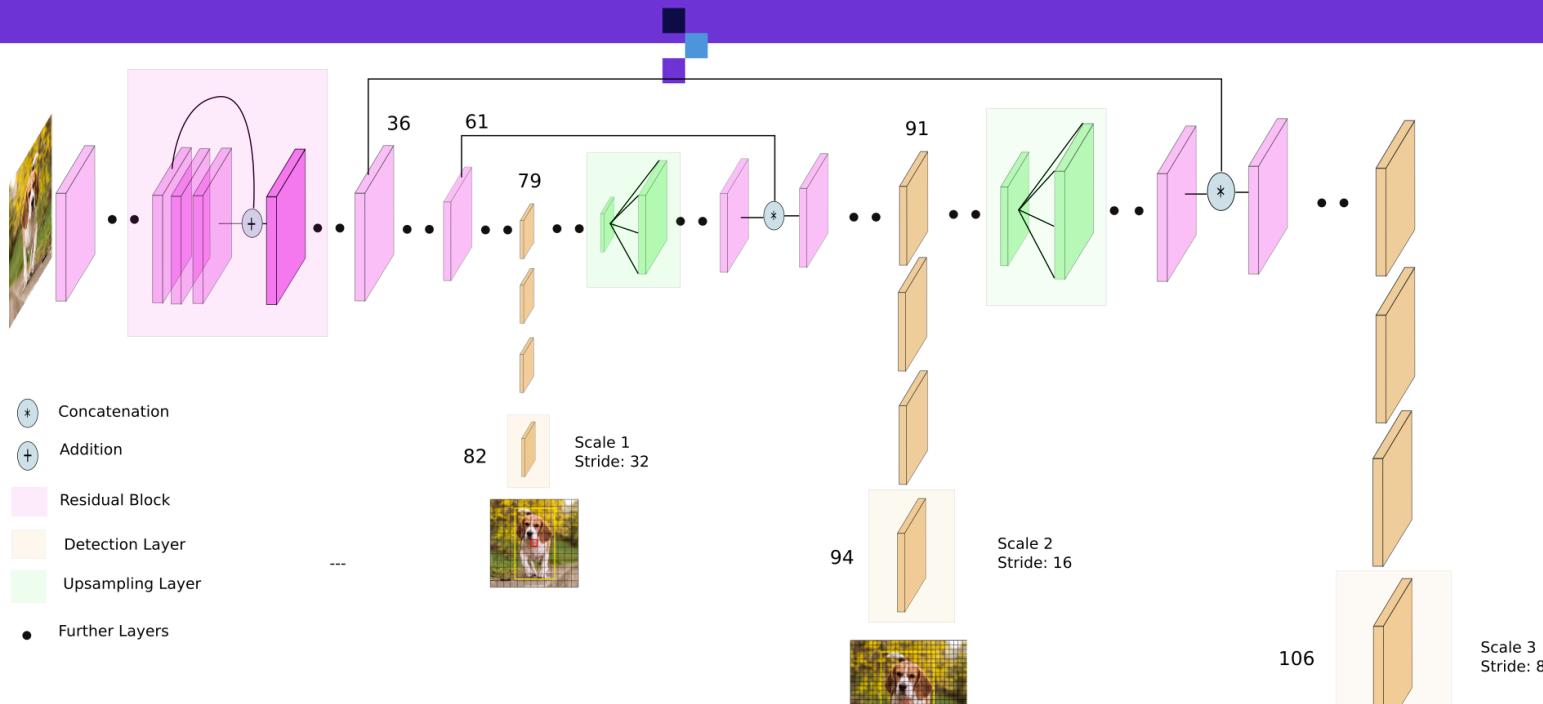
PLAN OF ATTACK – YOLO (YOU ONLY LOOK ONCE)



- 1. Intuition about YOLO
- 2. Object detection in images (Darknet)
- 3. Object detection in videos



YOLO (YOU ONLY LOOK ONCE)

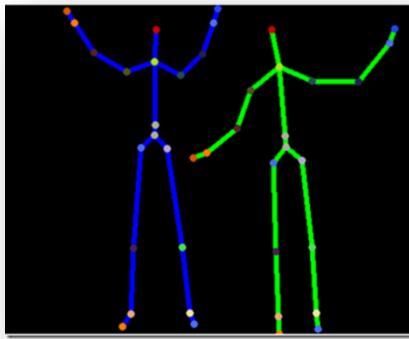


YOLO v3 network Architecture

PLAN OF ATTACK – RECOGNITION OF GESTURES AND ACTIONS



1. Intuition about recognition of gestures and actions
2. Implementation using images and videos



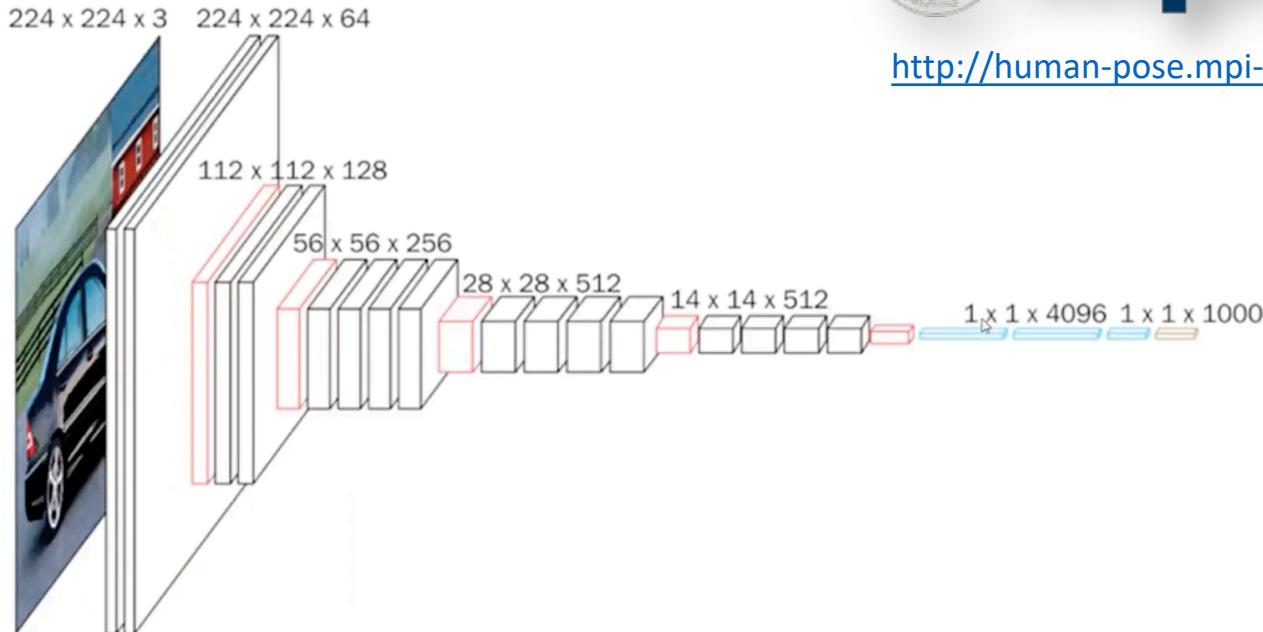
MPII MODEL



mpi

max planck institut
informatik

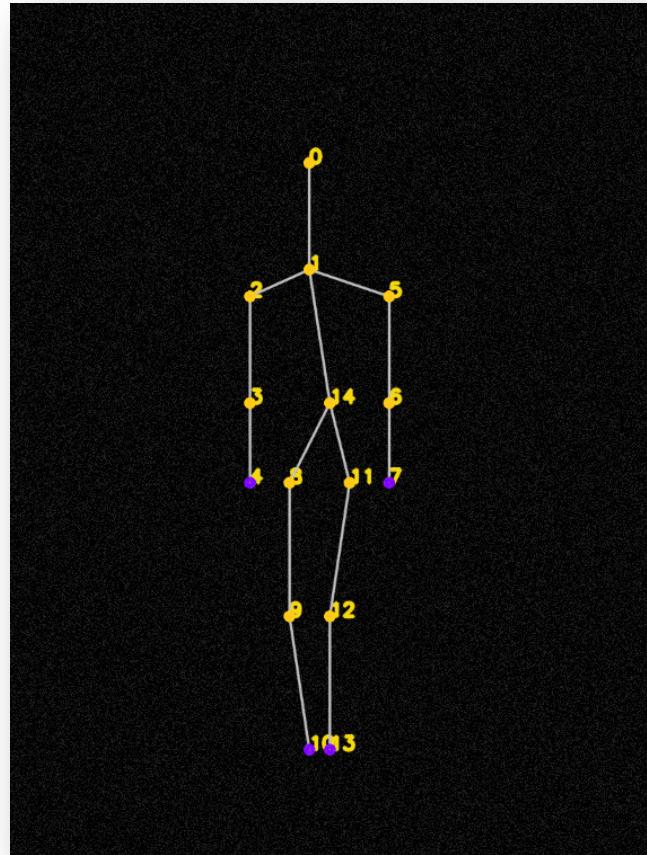
<http://human-pose.mpi-inf.mpg.de/>



MPII – BODY POINTS

0	Head
1	Neck
2	Righth shoulder
3	Righth elbow
4	Righth wrist
5	Left shoulder
6	Left elbow
7	Left wrist
8	Righth hip

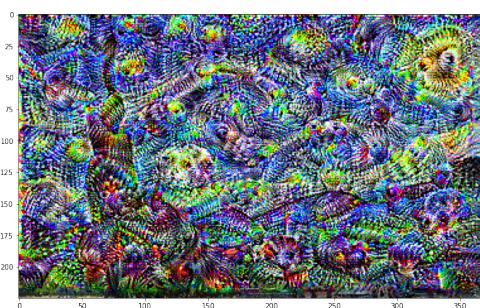
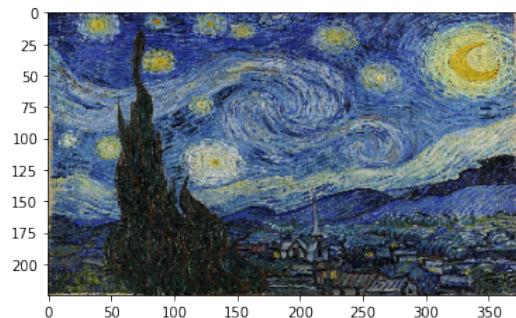
9	Righth knee
10	Righth ankle
11	Left hip
12	Left knee
13	Left ankle
14	Chest
15	Background



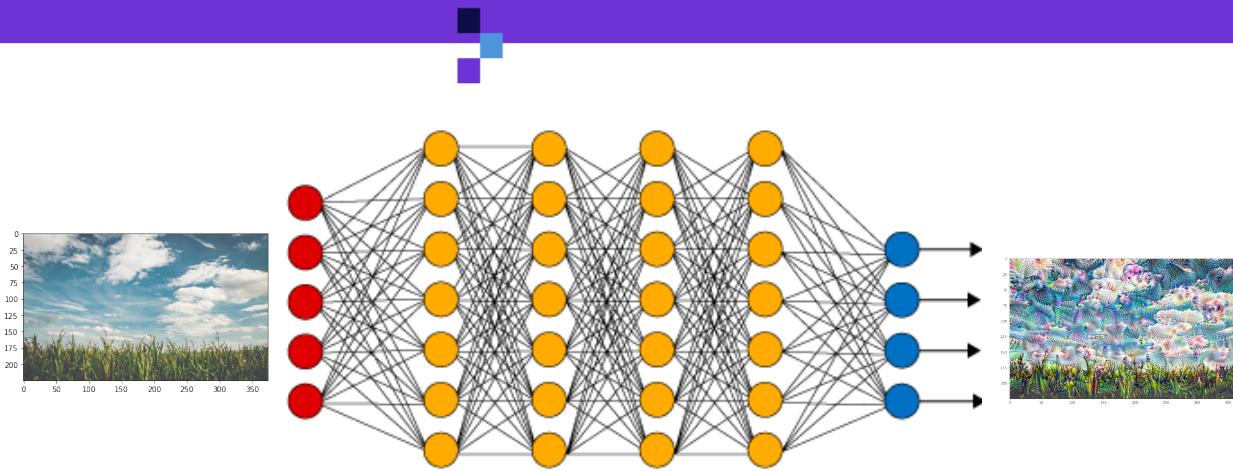
PLAN OF ATTACK – DEEP DREAM



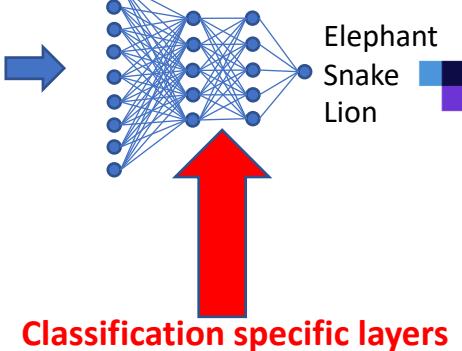
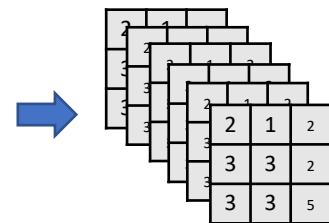
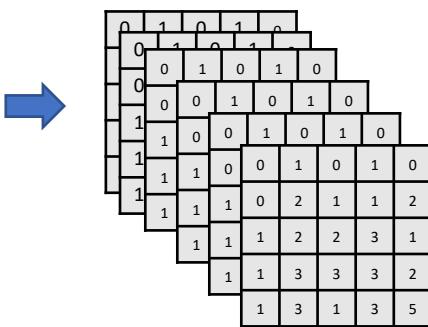
1. Intuition about Deep Dream
2. Implementation



DEEP DREAM



General features



PLAN OF ATTACK – STYLE TRANSFER



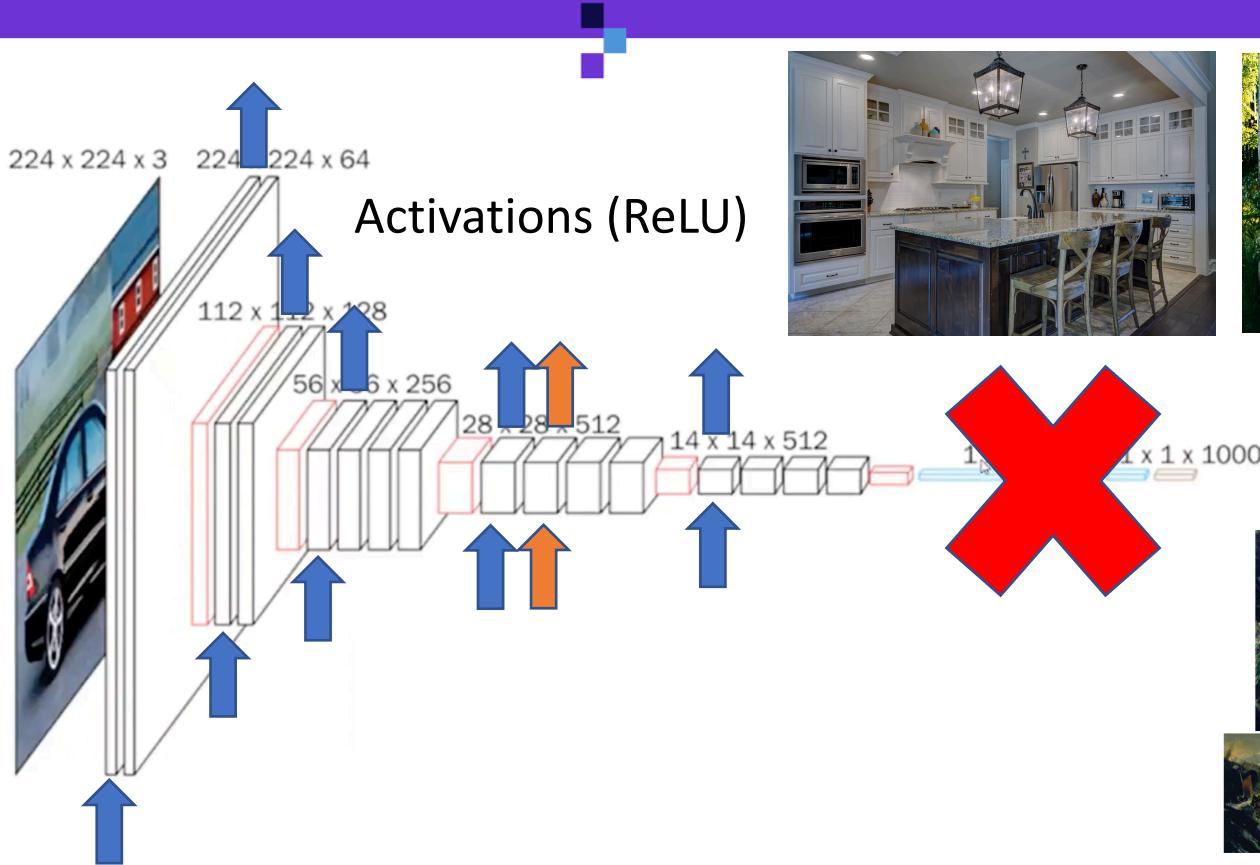
1. Intuition about Style Transfer
2. Implementation



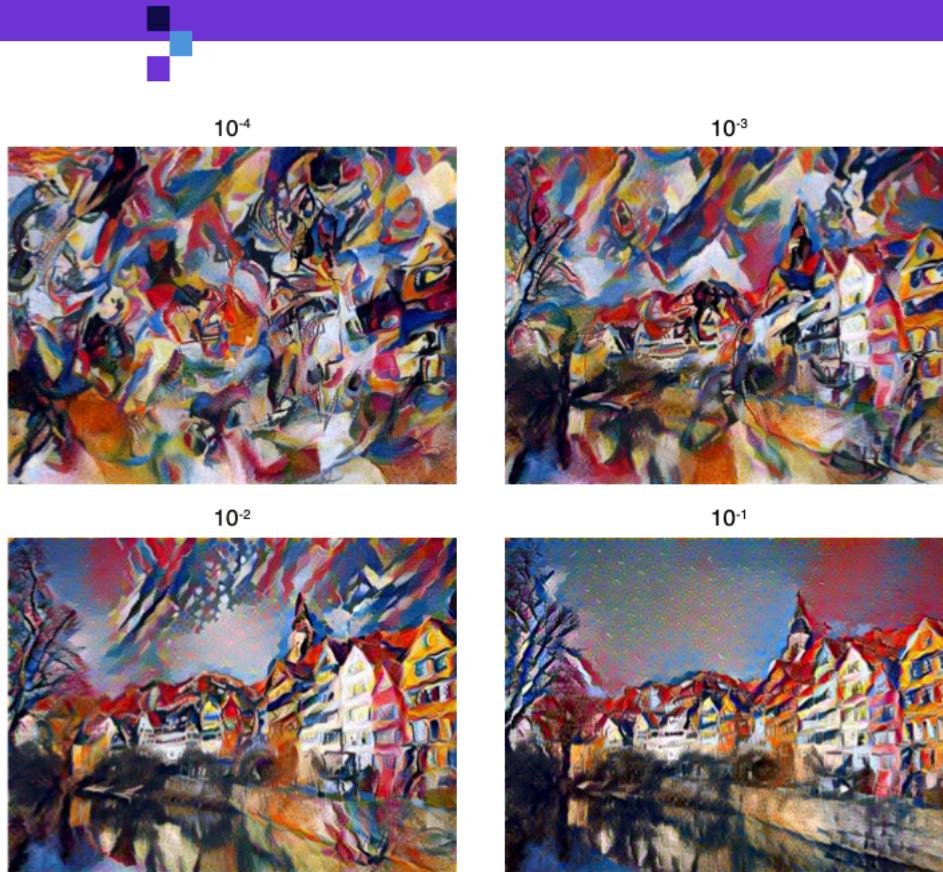
Source of images: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf



STYLE TRANSFER



STYLE TRANSFER



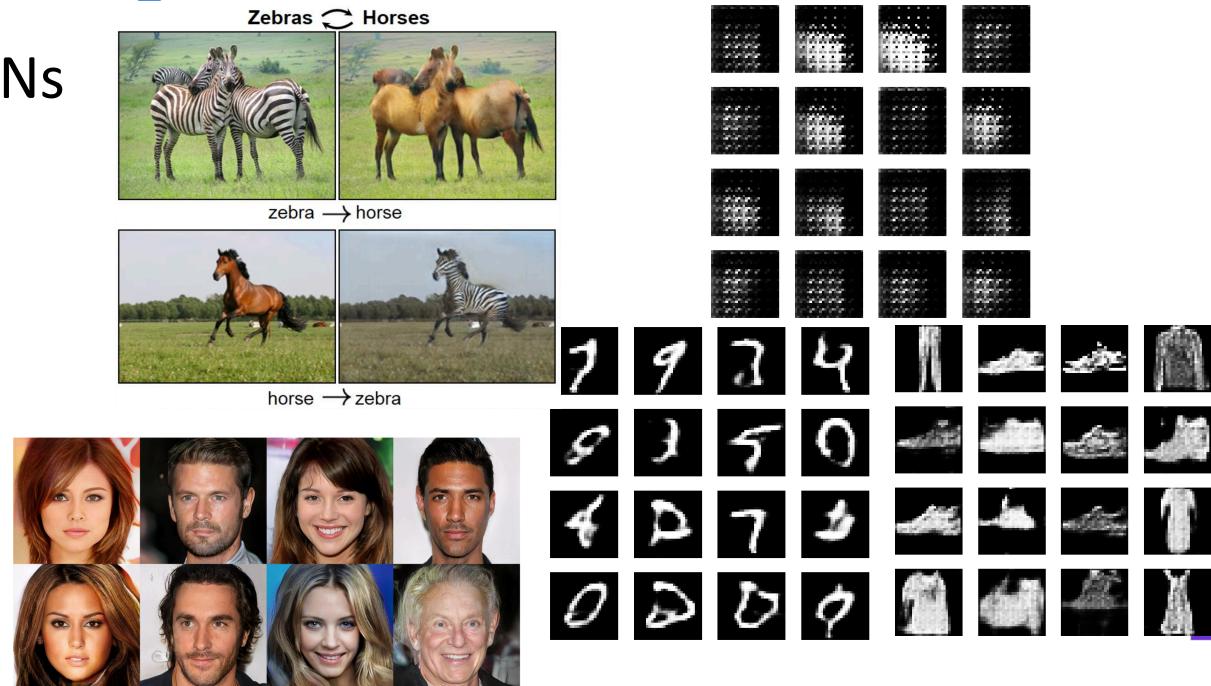
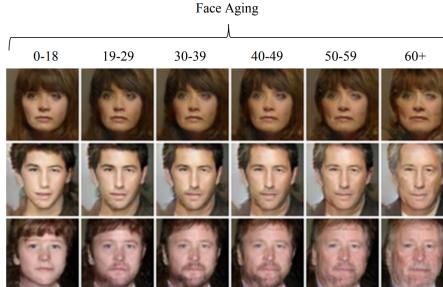
PLAN OF ATTACK – GANs (GENERATIVE ADVERSARIAL NETWORKS)



1. Intuition about GANs
2. Implementation



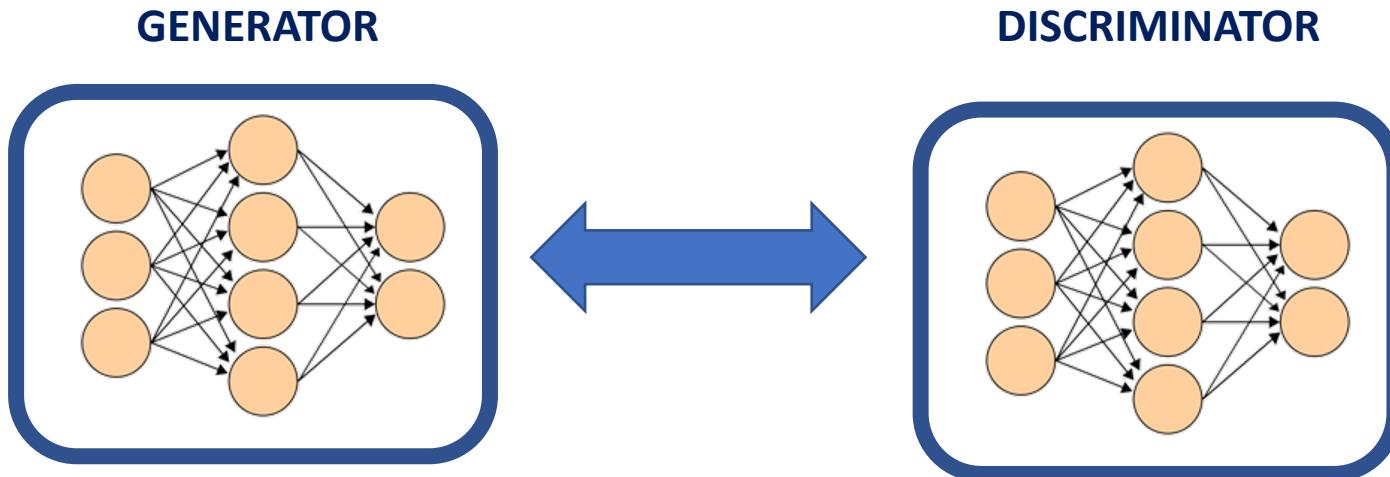
Figure 3: Generated samples



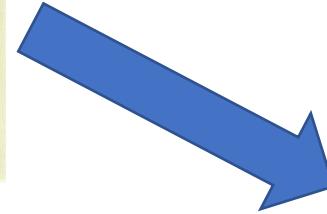
Source of images: <https://jonathan-hui.medium.com/gan-some-cool-applications-of-gans-4c9ecca35900>



GANs (GENERATIVE ADVERSARIAL NETWORKS)



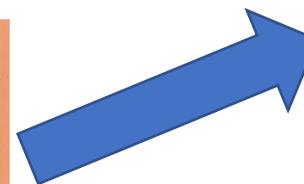
GANs (GENERATIVE ADVERSARIAL NETWORKS)



GENERATOR



FAKE MONEY



DISCRIMINATOR

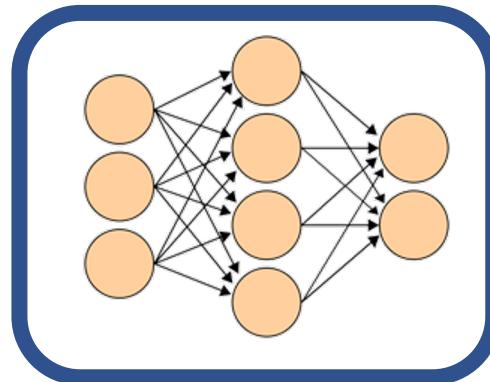
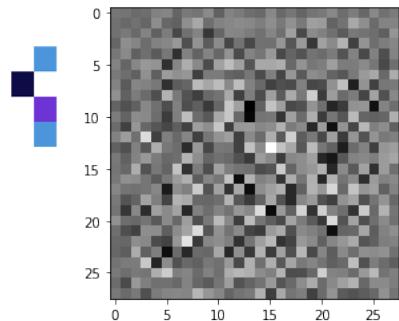


GANs – GENERATOR



1 – Real images
0 – Fake images

GENERATOR



GANs - DISCRIMINATOR

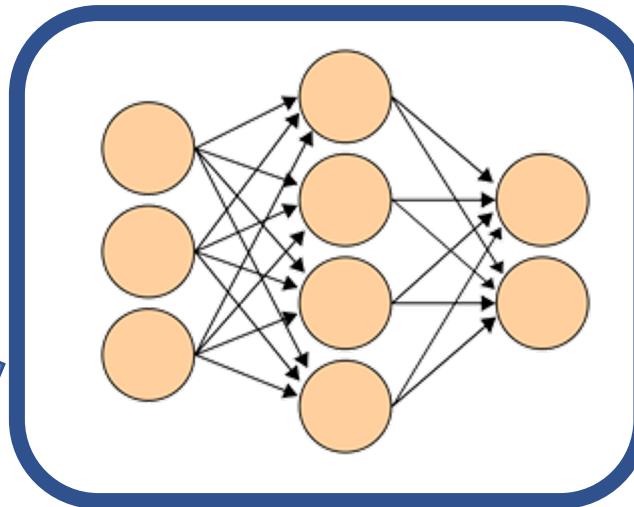


FAKE MONEY



REAL MONEY

DISCRIMINATOR



0



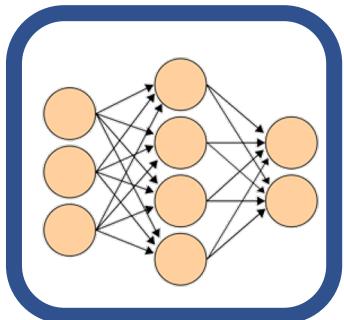
1



GANs – DISCRIMINATOR TRAINING



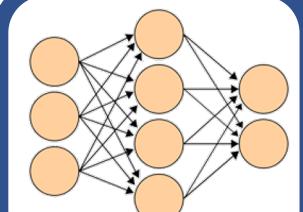
GENERATOR



FAKE MONEY



DISCRIMINATOR



0

1

REAL MONEY



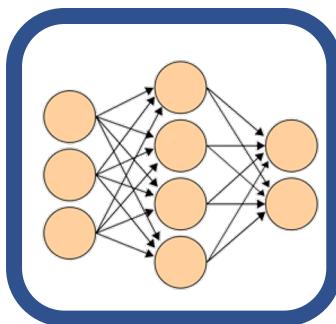
UPDATE WEIGHTS



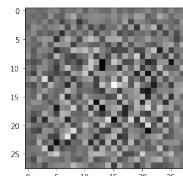
GANs – GENERATOR TRAINING



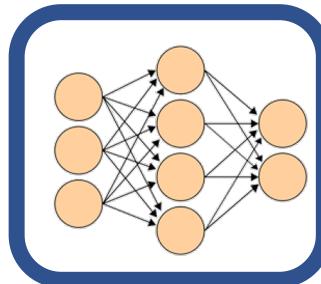
GENERATOR



FAKE MONEY



DISCRIMINATOR



1

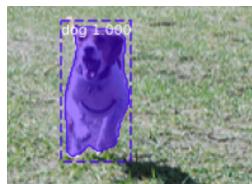
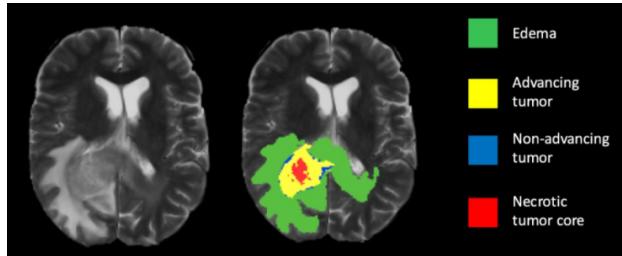
UPDATE WEIGHTS



PLAN OF ATTACK – IMAGE SEGMENTATION



1. Intuition about segmentation
2. Implementation



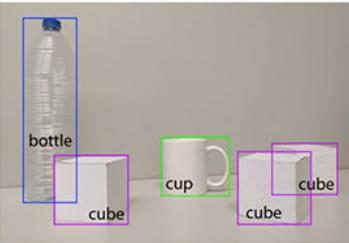
Source of images: [data-flair.training](#) & [deepsense.ai](#)



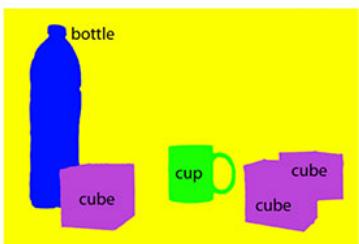
INSTANCE VS. SEMANTIC SEGMENTATION



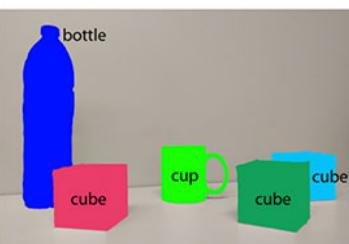
(a) Image classification



(b) Object localization



(c) Semantic segmentation



(d) Instance segmentation

Source: [Garcia et al](#)



Source: [NVIDIA Developer Blog](#)

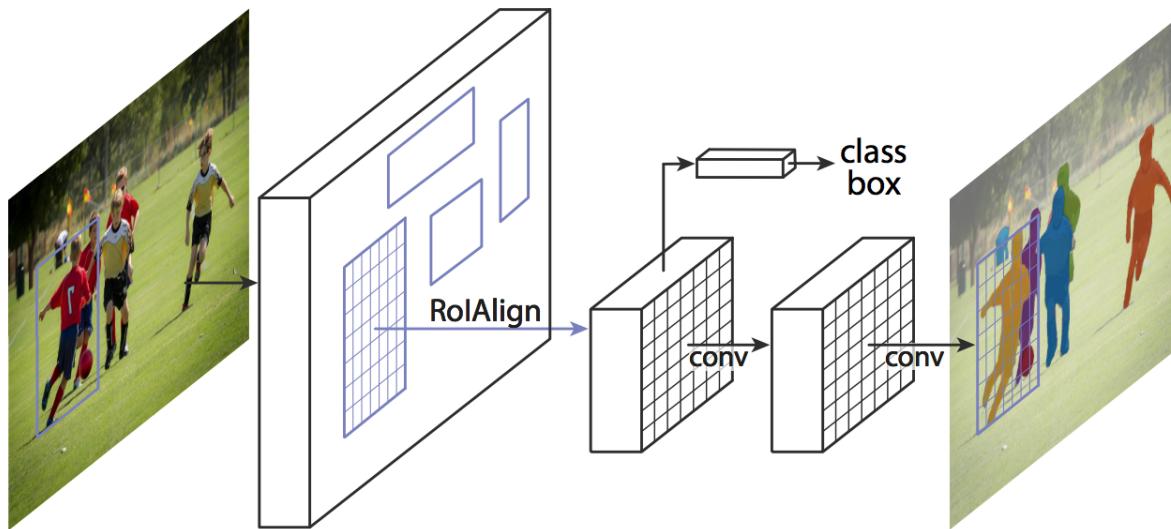


Source: [Pexels](#)

MASK R-CNN



- It was built based on the previous object detection works: R-CNN (2013), Fast R-CNN (2015), and Faster R-CNN (2015), all by Girshick et al.



Source: <https://arxiv.org/abs/1703.06870>



R-CNN, FAST R-CNNs AND FASTER R-CNN



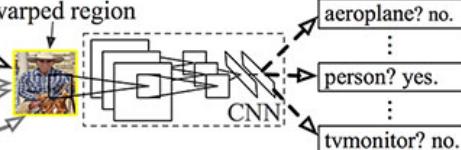
Original R-CNN Architecture



1. Input image



2. Extract region proposals (~2k)

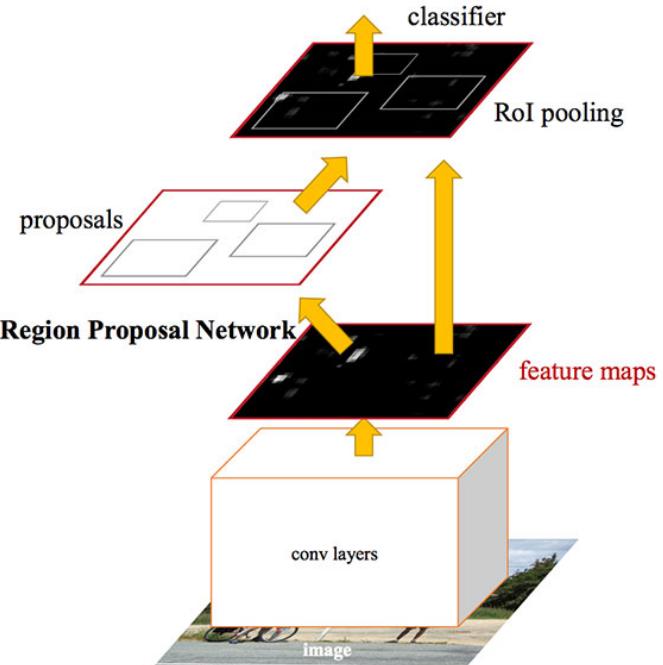


3. Compute CNN features

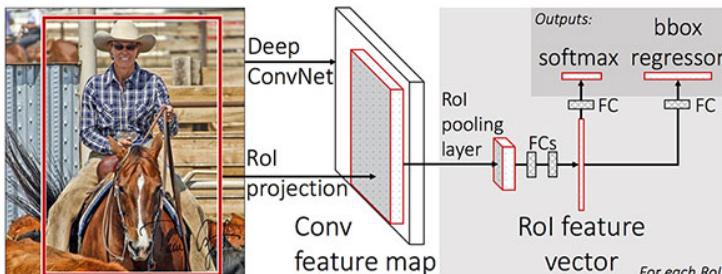
4. Classify regions

Source: [Girshick et al., 2013](#)

Faster R-CNN Architecture



Fast R-CNN Architecture



Source: [Girshick et al., 2015](#)

Source: [Girshick et al., 2015](#)