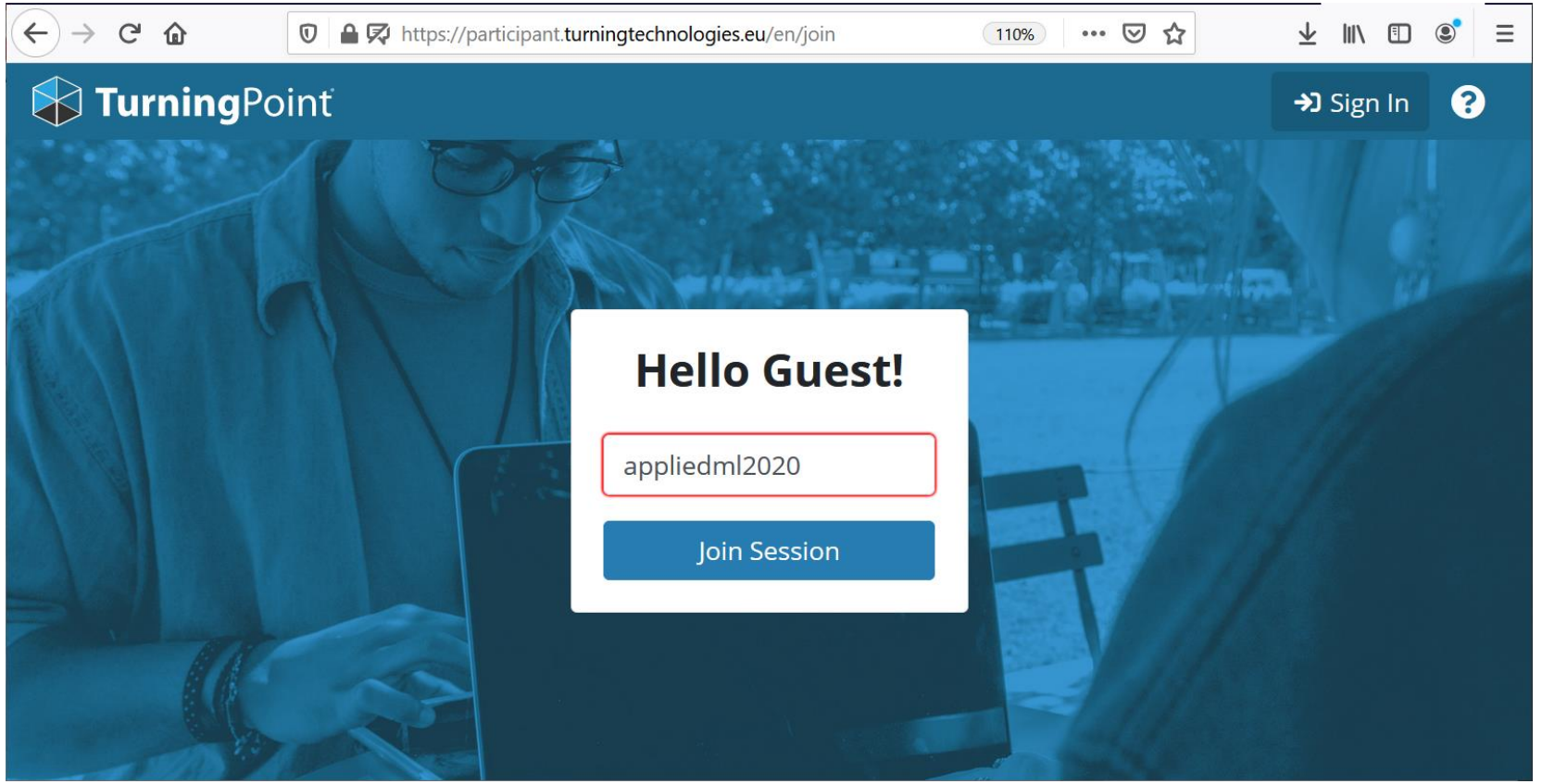
APPLIED MACHINE LEARNING

Introduction to Neural Networks
Interactive session

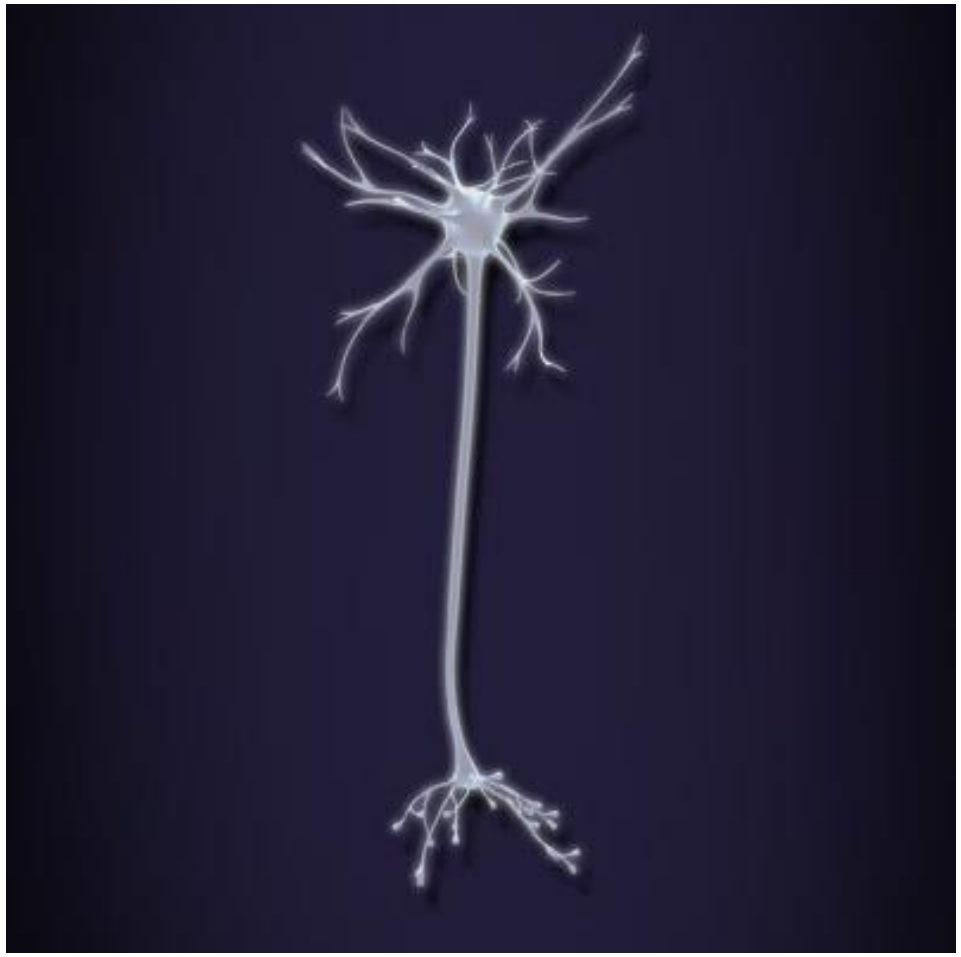
Launch polling system

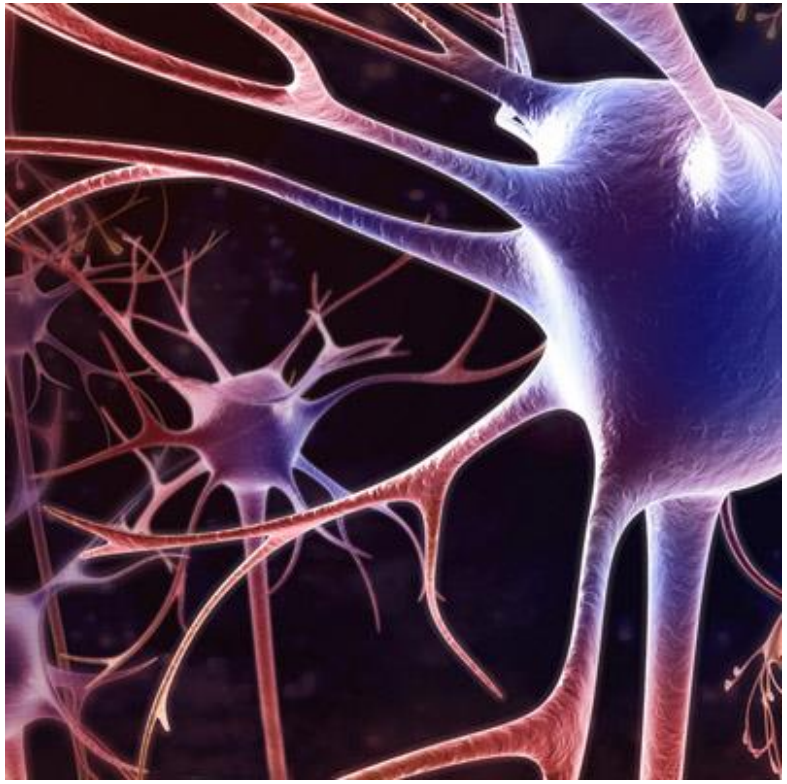
https://participant.turningtechnologies.eu/en/join

Acces as GUEST and enter the session id: *appliedml2020*



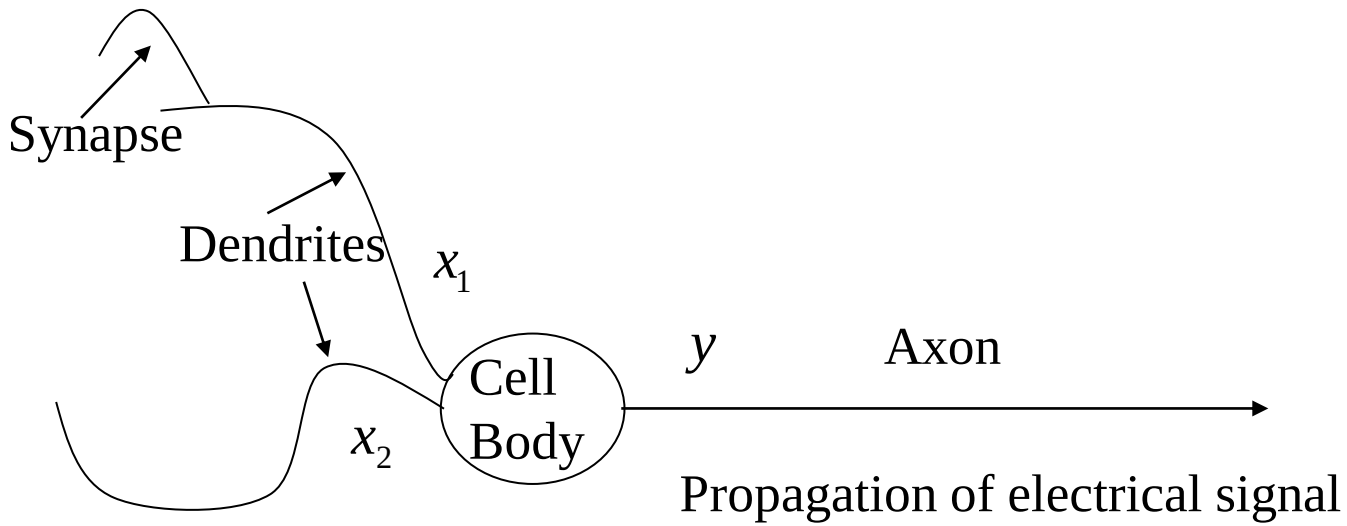
The Neuron



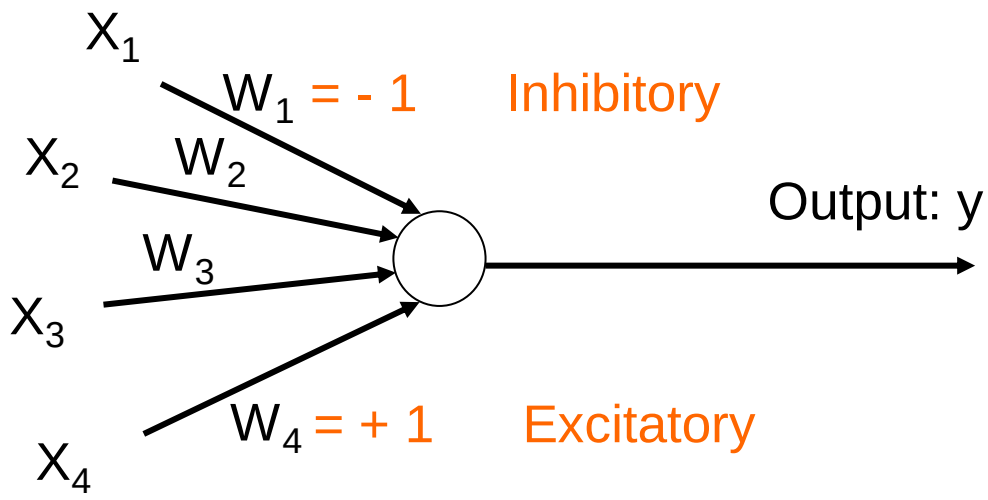


Courtesy of 3DScience.com

Simple Model of Biological Neurons' Signal Processing: The Leaky – Integrator Neuron



Simple Model – The Perceptron

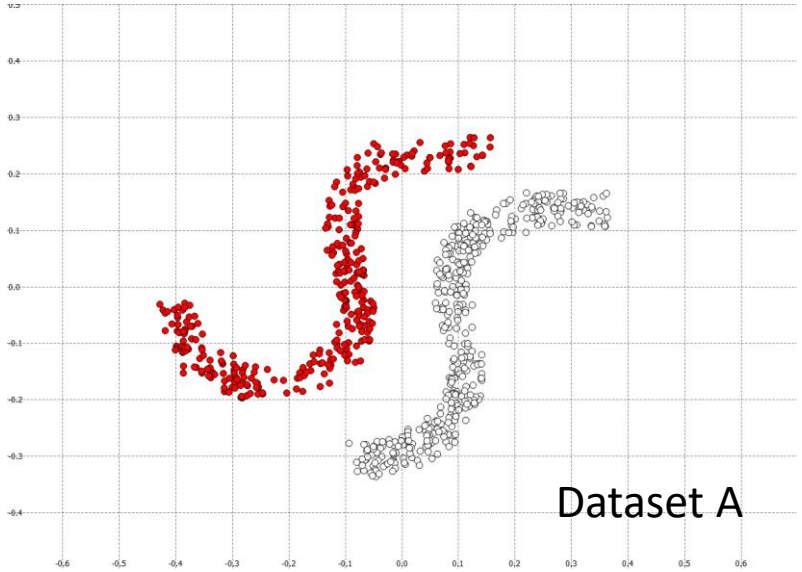


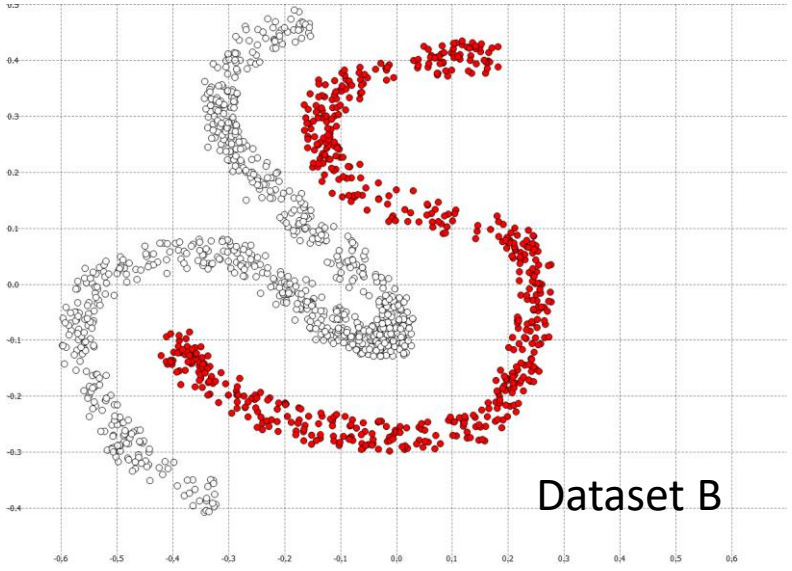
X: Input vector from all other neurons
w: the strength of each synapse
y: neuron output
f: transfer function

$$y = \theta\left(\sum_{i=1}^{4} w_i \cdot x_i + w_0\right)$$

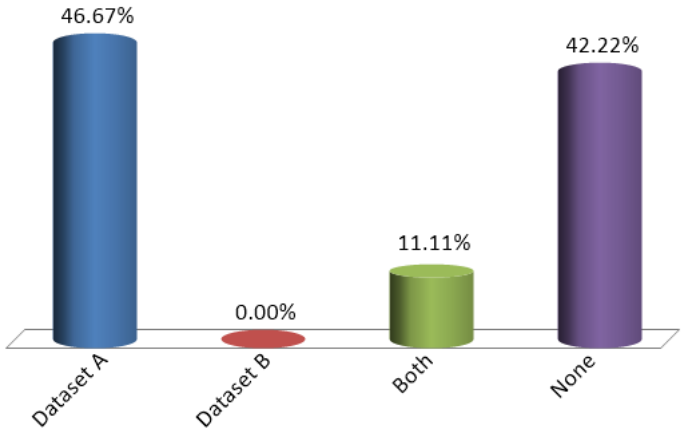
$$\theta(..) = \begin{cases} 1 \text{ if } \mathbf{w}^T\mathbf{x} + w_0 \geq 0 \\ 0 \text{ if } \mathbf{w}^T\mathbf{x} + w_0 < 0 \end{cases}$$

For which dataset the perceptron model can achieve perfect classification (100% accuracy)

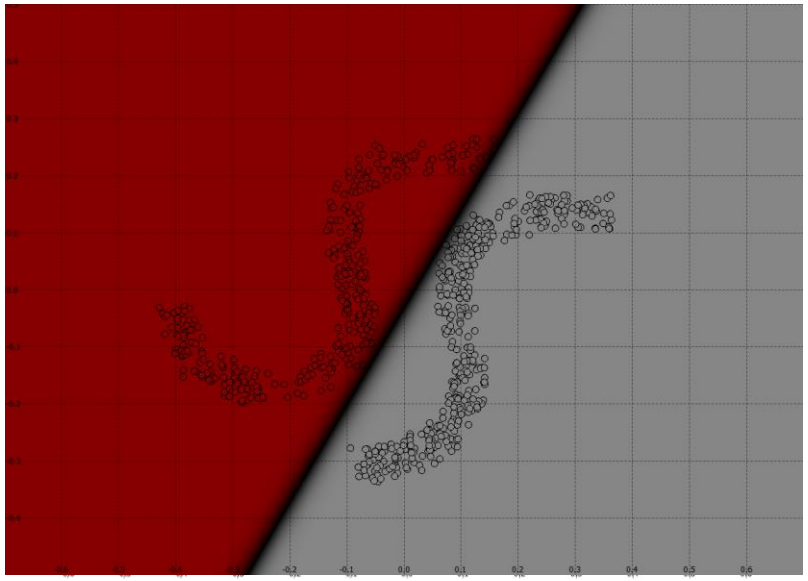


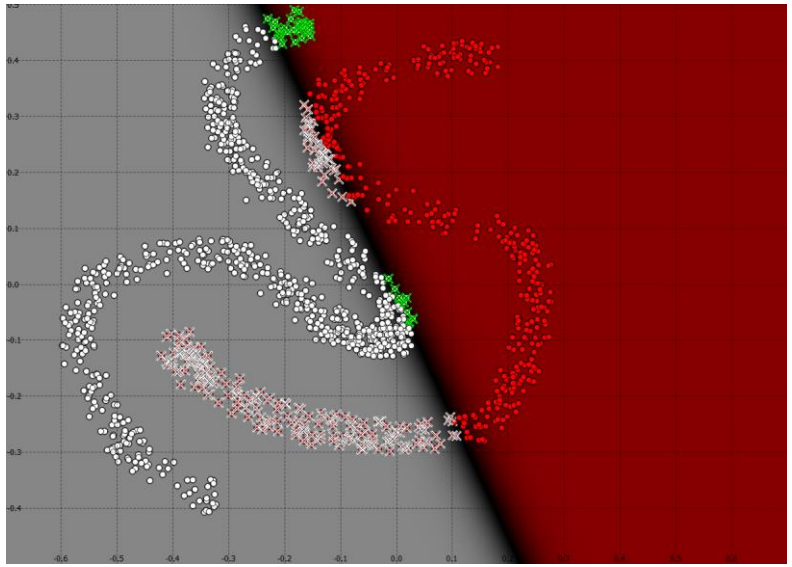


A. Dataset A
B. Dataset B
C. Both
D. None



For which dataset the perceptron model can achieve perfect classification (100% accuracy)

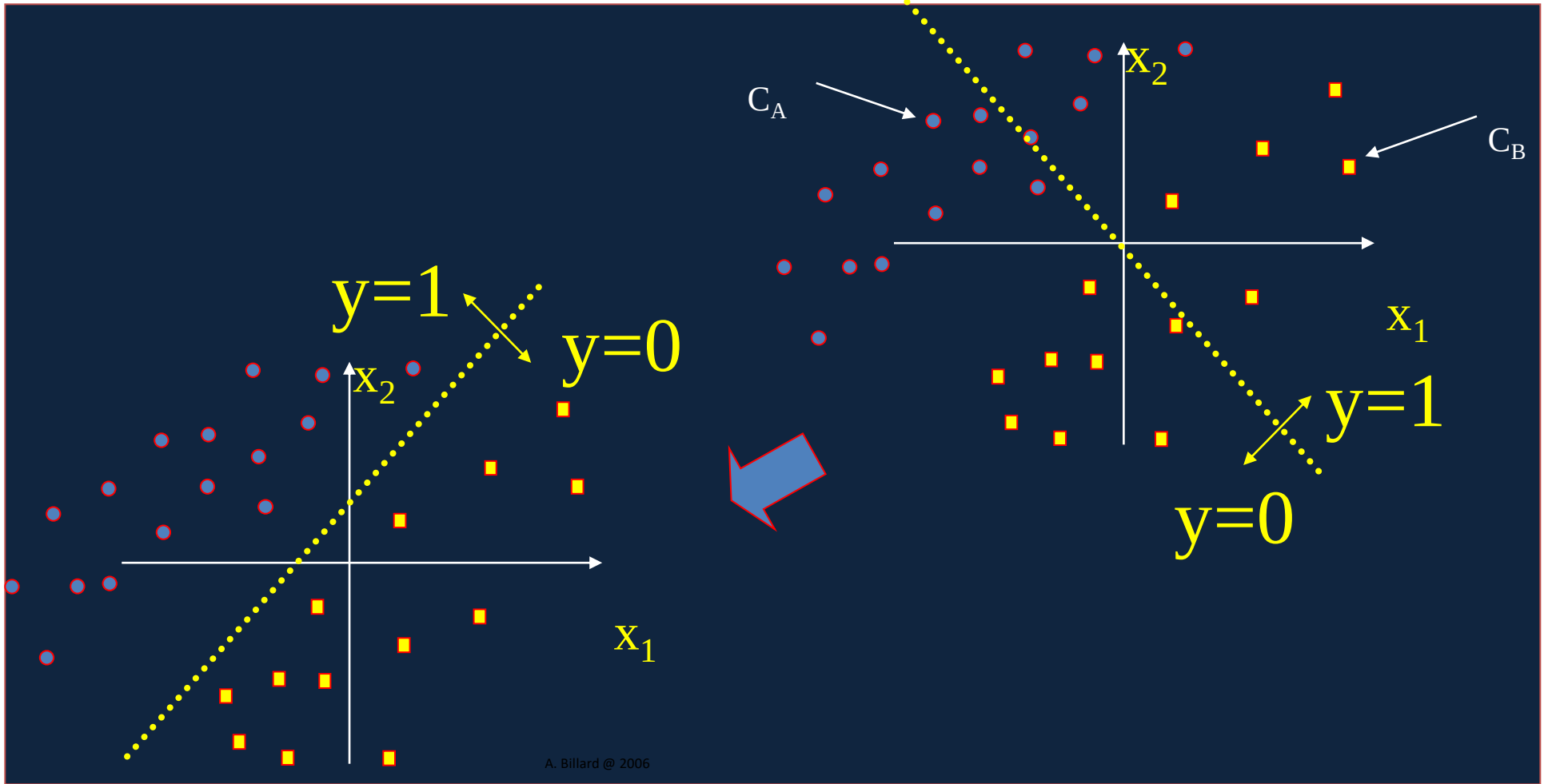




The perceptron is a simple linear classifier – identical to linear SVM

Learning in one Neuron

Goal: we would like the neuron:
to output 1 when (x_1,x_2) belongs to C_A, and
to output 0 when (x_1,x_2) belongs to C_B



Learning in one Neuron

The perceptron iteratively updates the weight vector w until the classification is correct

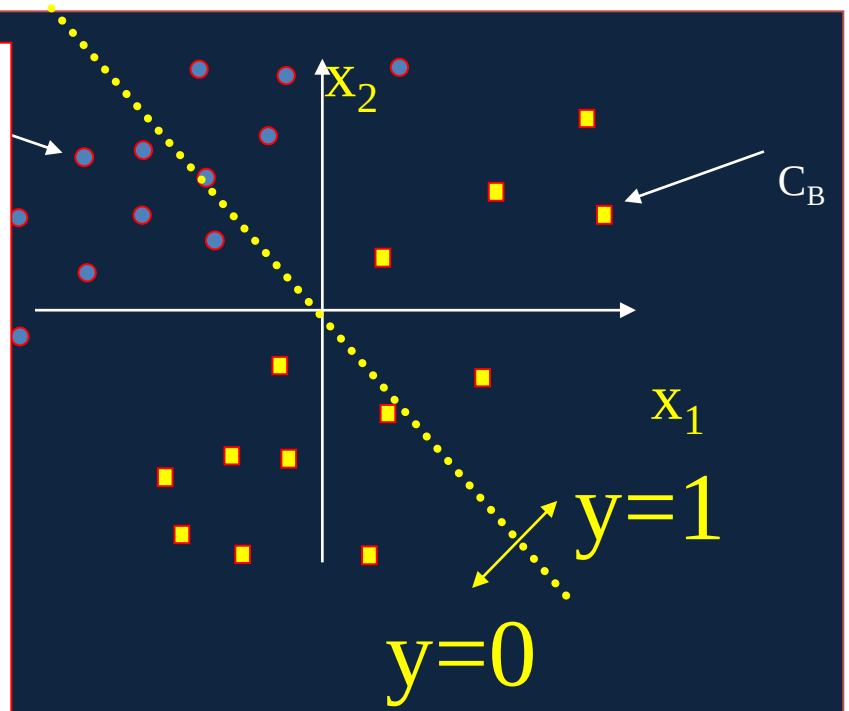
For each training pair (x_1,x_2):
If correctly classified:
do nothing
If wrongly classified:

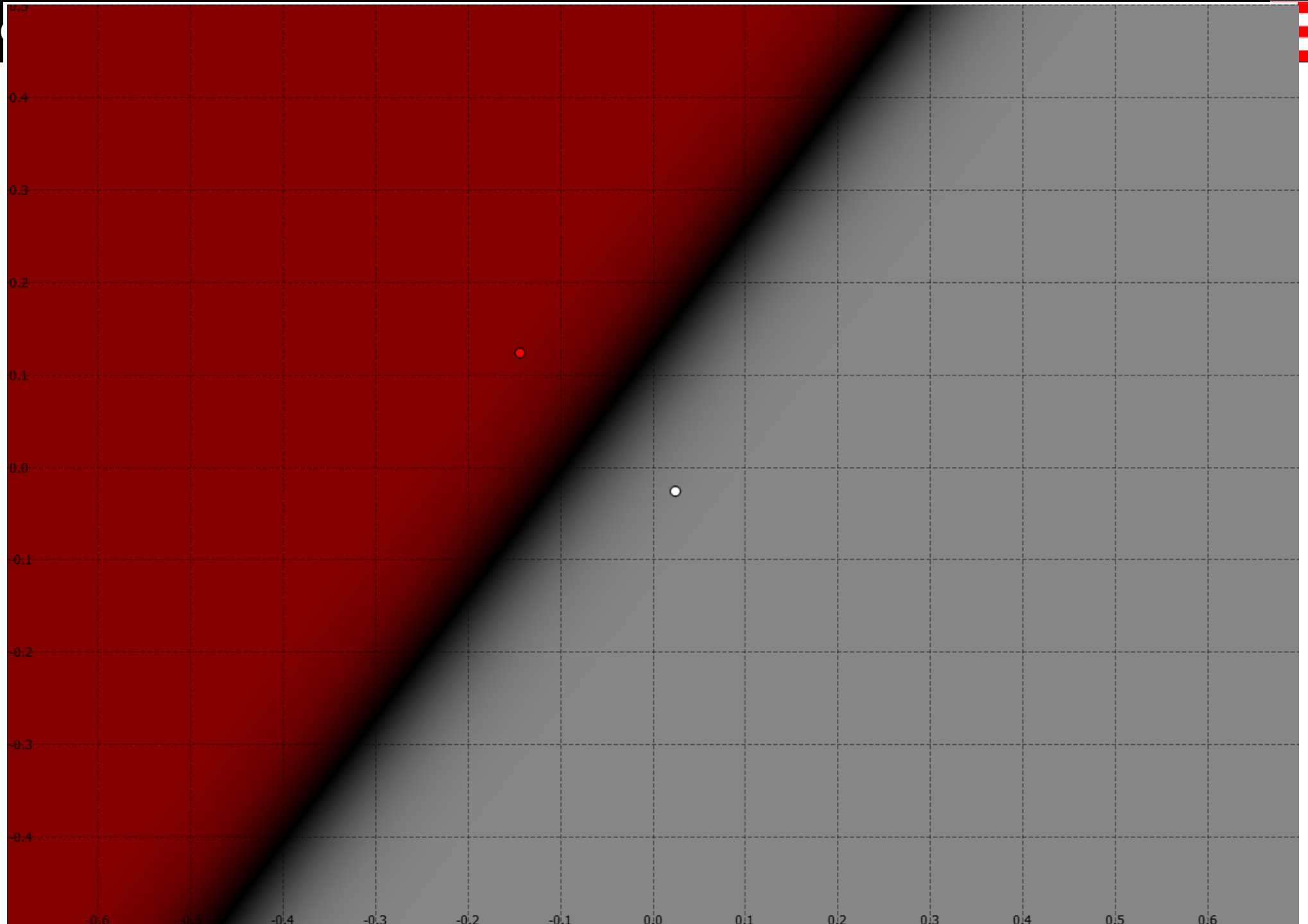
$$\vec{w}(t+1) = \vec{w}(t) - \eta \cdot \vec{x}$$

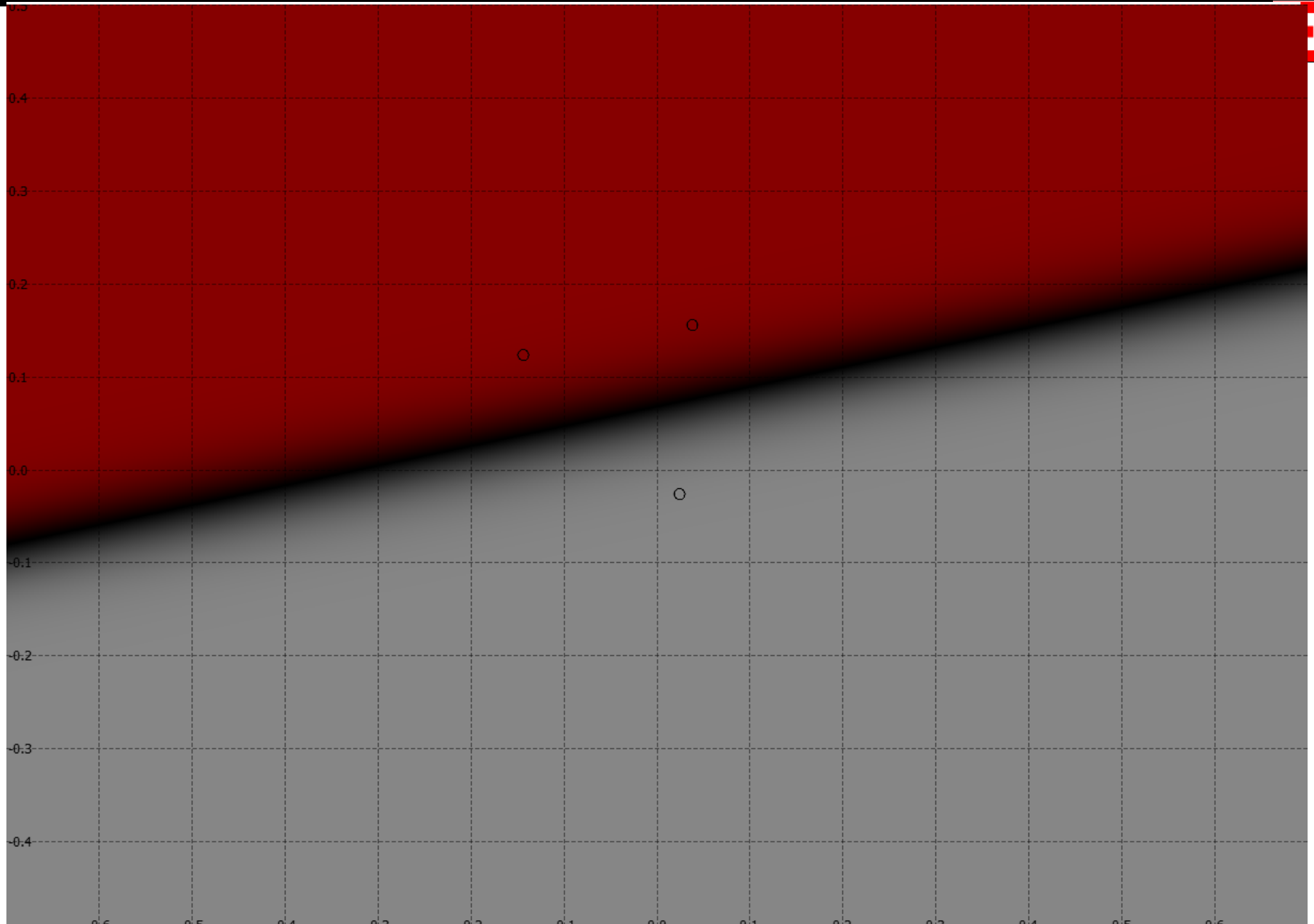
if $\vec{w}^T \cdot \vec{x} \geq 0$ and $\vec{x}$ belongs to C_B

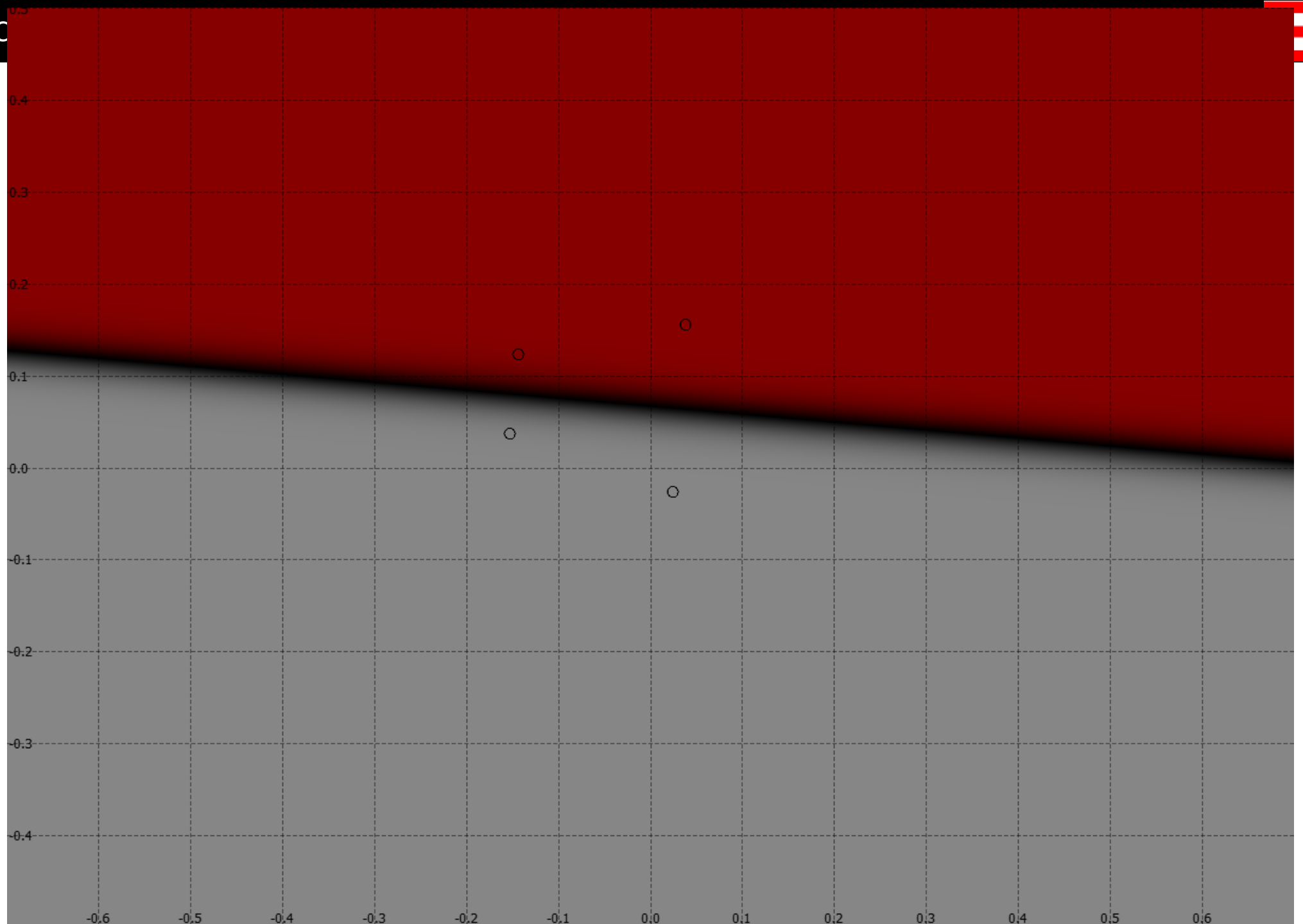
$$\vec{w}(t+1) = \vec{w}(t) + \eta \cdot \vec{x}$$

if $\vec{w}^T \cdot \vec{x} < 0$ and $\vec{x}$ belongs to C_A

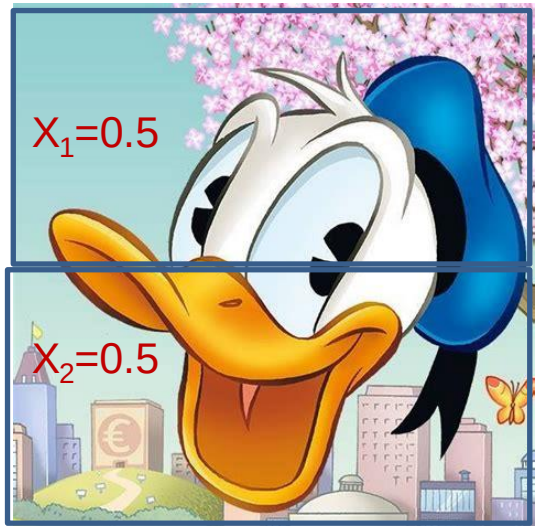


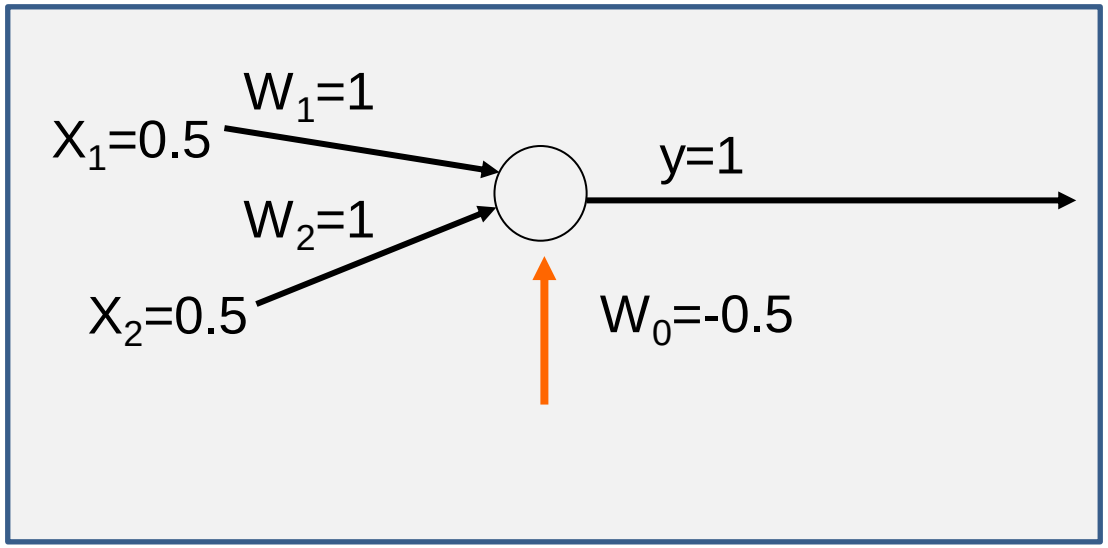

0.4
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
-0.4
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6


0.4
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
-0.4


0.4
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
-0.4
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6

The perceptron



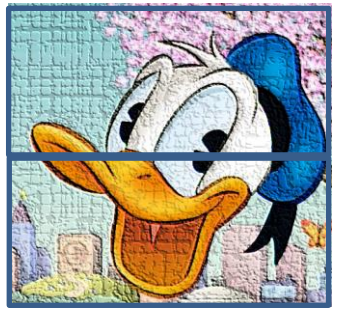


$$y = \theta\left(\sum_{i=1}^{4} w_i \cdot x_i + w_0\right)$$

Assume X_i codes average grey scale intensity in part i of the image.

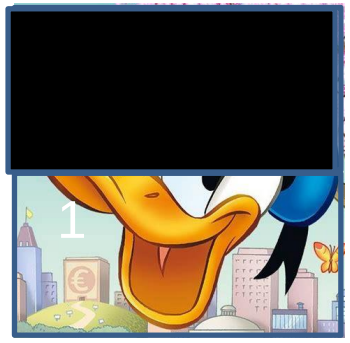
Is it Donald?

$$\theta(..) = \begin{cases} 1 \text{ Yes} & \text{if } \mathbf{w}^T\mathbf{x} + w_0 \geq 0 \\ 0 \text{ No} & \text{if } \mathbf{w}^T\mathbf{x} + w_0 < 0 \end{cases}$$



X_1=0.4
X_2=0.4

1

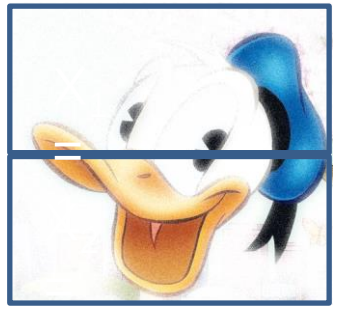


X_1=0
X_2=1

2

X_1=0.1
X_2=0.1

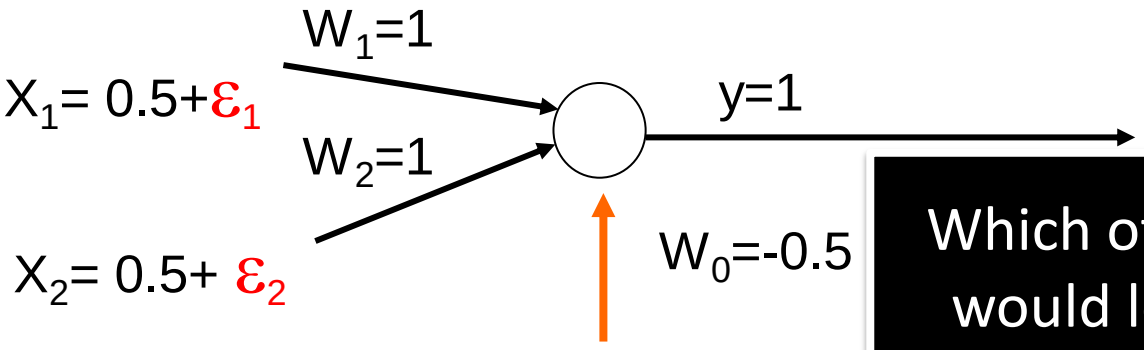
3

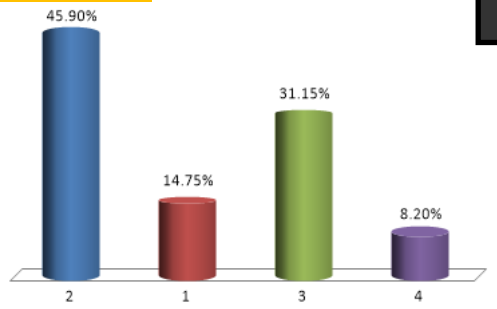


X_1=0.9
X_2=0.9

4

Consider you apply a disturbance on the inputs





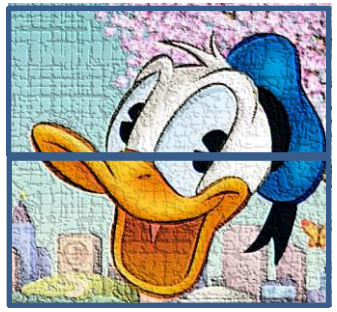
Which of the four types of disturbances would lead to incorrect classification?

Is it Donald?

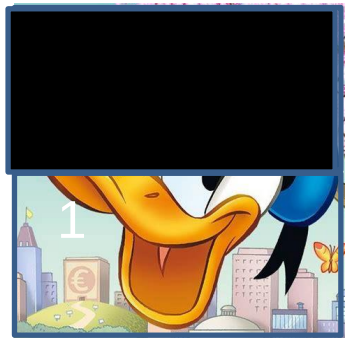
$$\theta(..) = \begin{cases} 1 \text{ Yes} & \text{if } \mathbf{w}^T\mathbf{x} + w_0 \geq 0 \\ 0 \text{ No} & \text{if } \mathbf{w}^T\mathbf{x} + w_0 < 0 \end{cases}$$

A. 2
B. 1
C. 3
D. 4

The classifier is robust to large but local disturbances (case 2), but not to diffuse disturbances (case 3). How can we enlarge robustness?



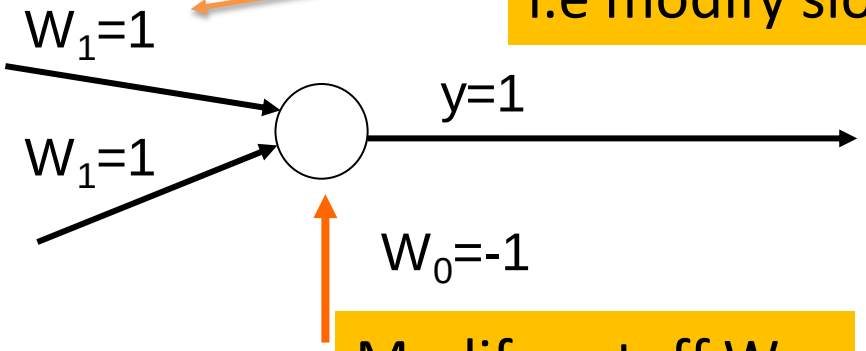
X_1=0.4
X_2=0.4
1



X_1=0
X_2=1
2

X_1= 1+ε_1

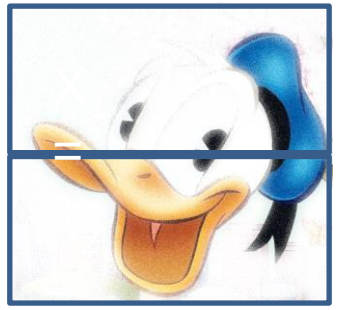
X_2= 1+ ε_2



Modify the weights,
i.e modify slope of the separating line

Modify cutoff W_0

X_1=0.1
X_2=0.1
3



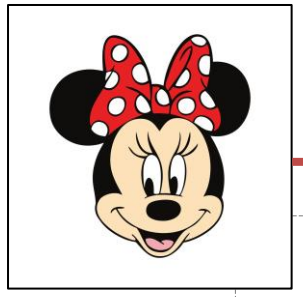
X_1=0.9
X_2=0.9
4

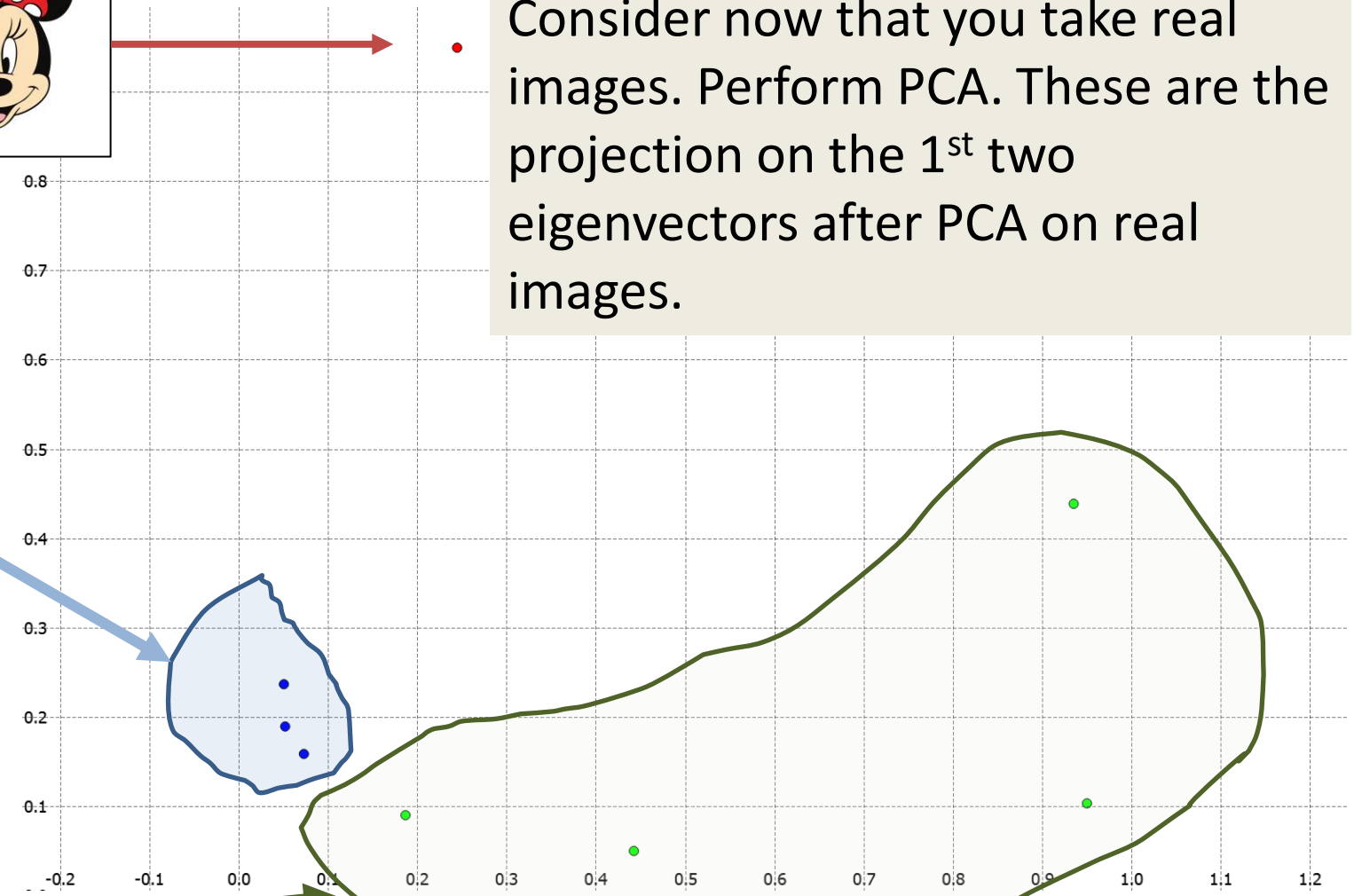
Is it Donald?

$$\theta(..) = \begin{cases} 1 \text{ Yes} & \text{if } \mathbf{w}^T\mathbf{x} + w_0 \geq 0 \\ 0 \text{ No} & \text{if } \mathbf{w}^T\mathbf{x} + w_0 < 0 \end{cases}$$

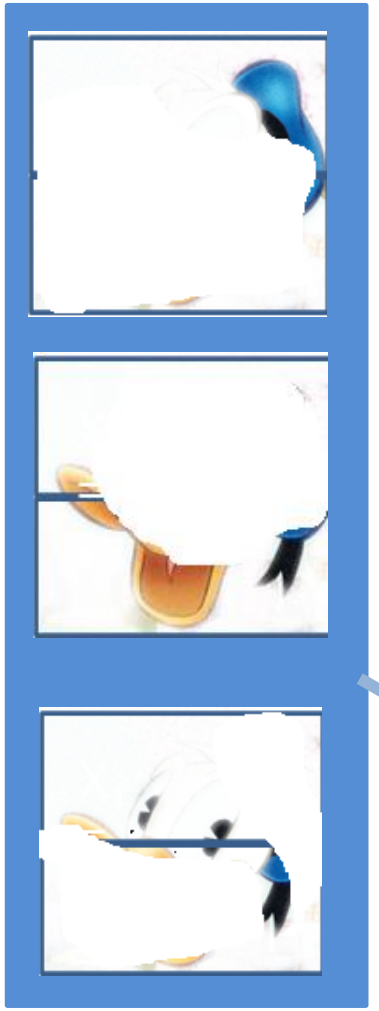
Modify the encoding

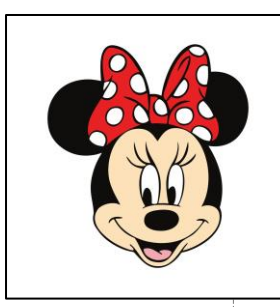
The classifier is robust to large but local disturbances (case 2), but not to diffuse disturbances (case 3). How can we enlarge robustness?





Consider now that you take real images. Perform PCA. These are the projection on the 1st two eigenvectors after PCA on real images.





W_0=+0.8

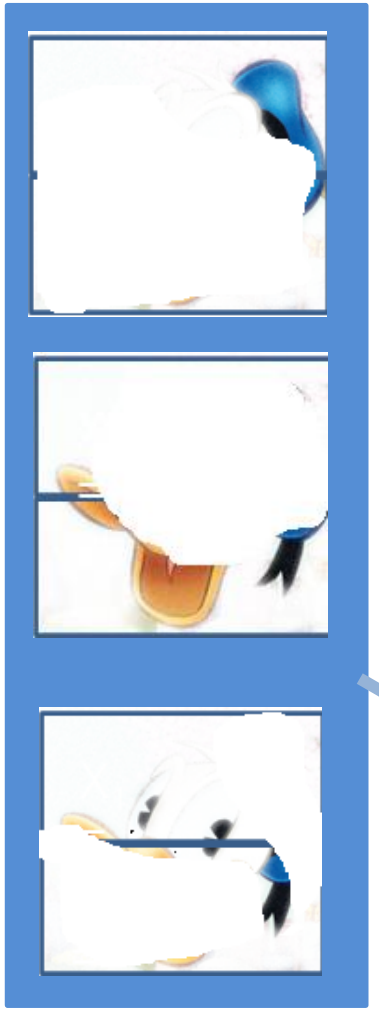
W_0=+0.3

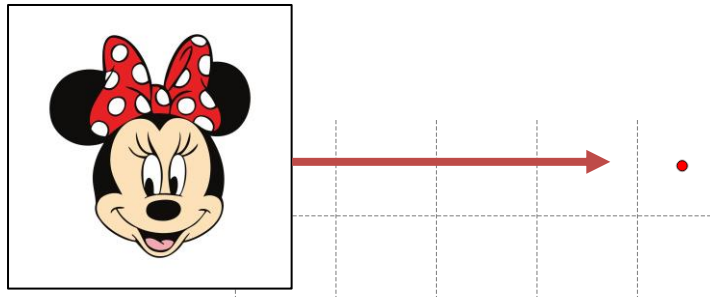
weights=$[W_1\ W_2]^T$ =$[1\ \ -1]^T$

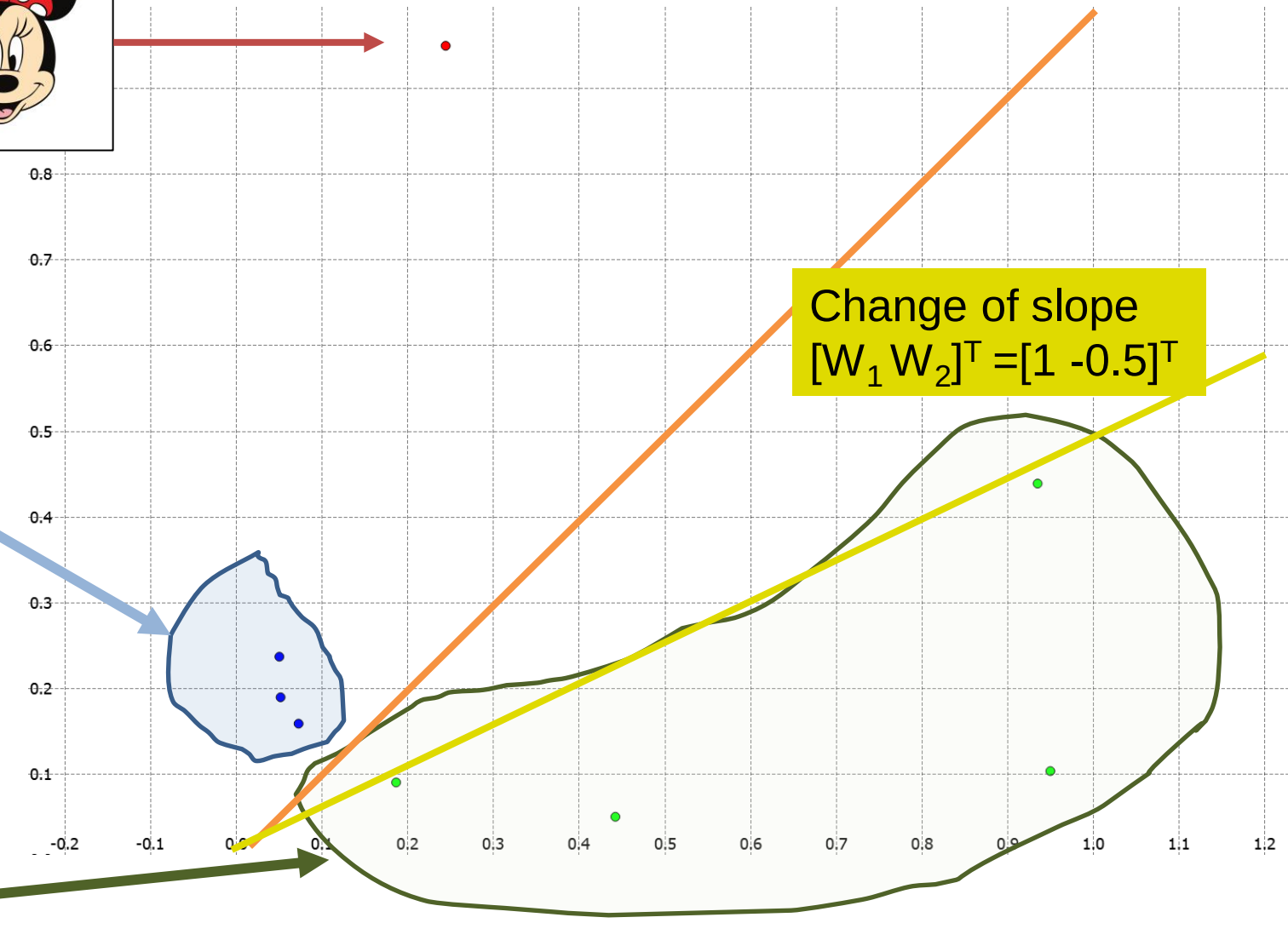
Bias W_0=0

W_0=-0.5

Modify cutoff W_0→ Shift to // plane

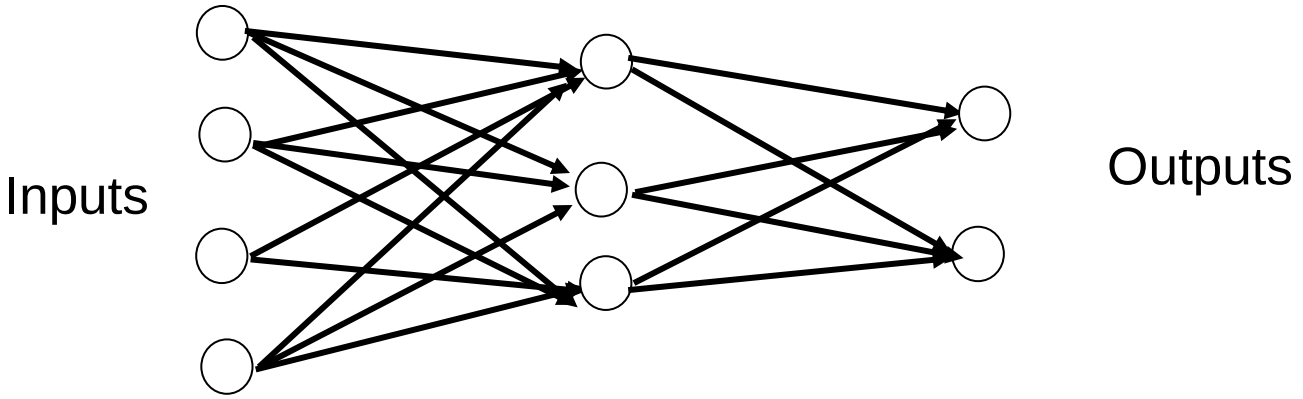





0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-0.2
-0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
1.0
1.1
1.2
Change of slope
$[W_1\ W_2]^T = [1\ -0.5]^T$

Multi-Layer

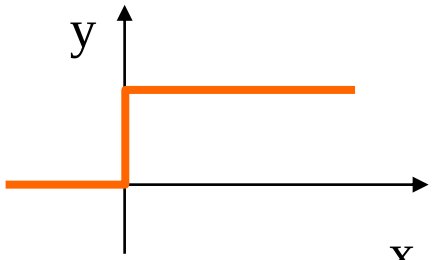
A two-layer Feed-Forward Neural Network



PERCEPTRON (1940, McCulloch & Pitts)

Threshold Function:

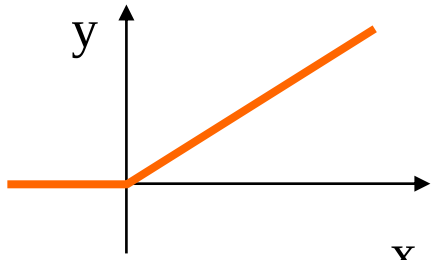
$$f(z)=\theta(z)=\begin{cases}1 & \text{if } z\geq 0\\ 0 & \text{if } z<0\end{cases} \quad \Rightarrow \quad y=\theta\left(\sum_i w_i\cdot x_i\right)$$



RELU (Rectified Linear Unit) (Hahnloser & Seung 2011)

Threshold Function:

$$f(z)=\theta(z)=\max(0,z)=\begin{cases}z & \text{if } z\geq 0\\ 0 & \text{if } z<0\end{cases}$$



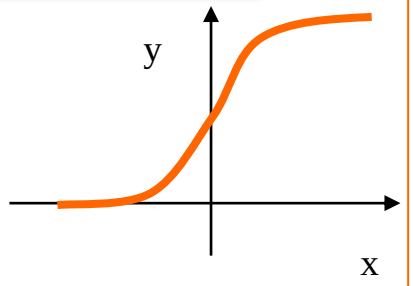
Sigmoid

The steepest the sigmoid slope, the closer it is to perceptron

The flatter the sigmoid slope, the closer it is to RELU

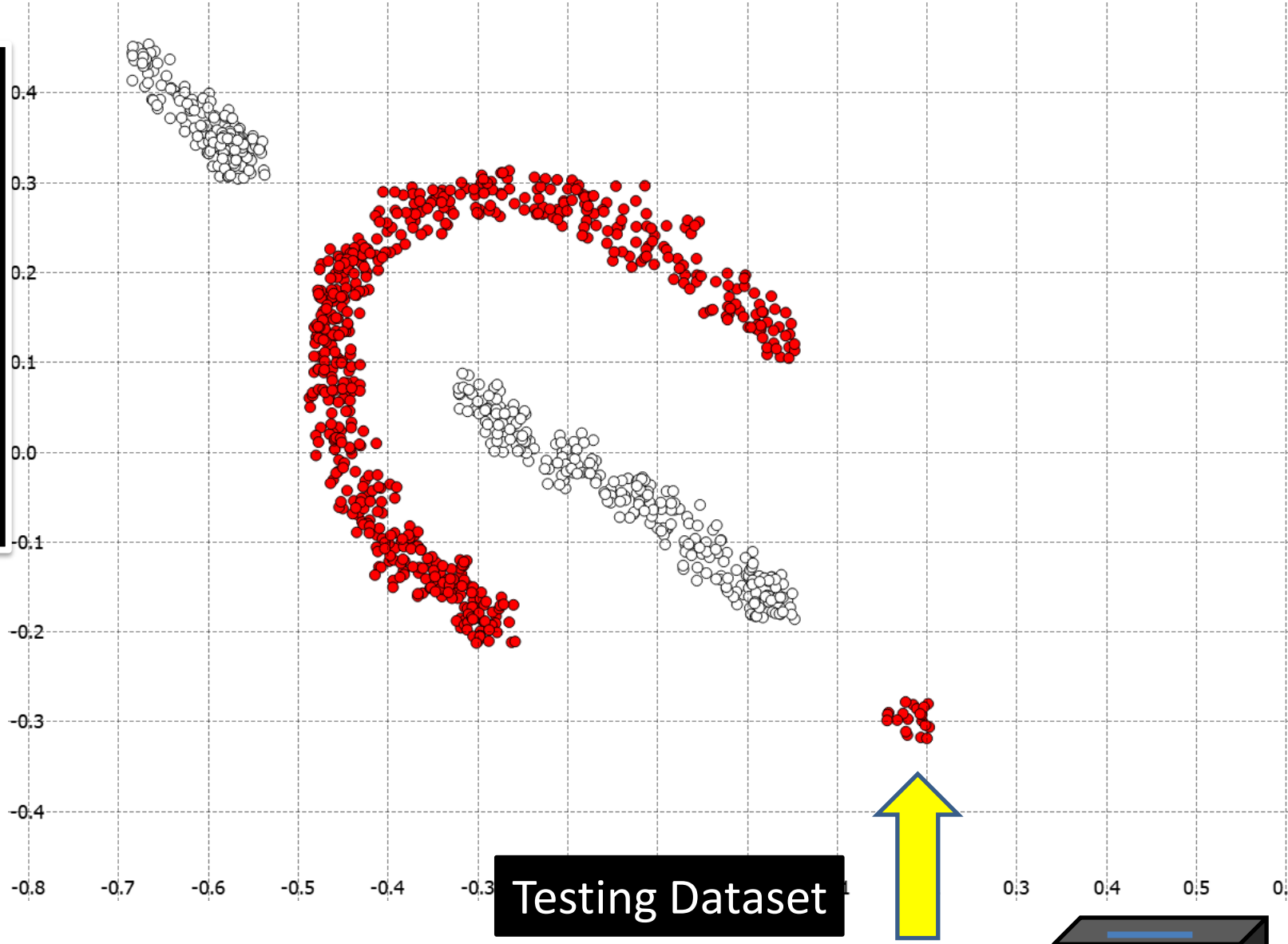
Positive form:

$$f(z)=g(z)=\frac{1}{1+e^{-D\cdot z}} \quad \Rightarrow \quad y=\frac{1}{1+e^{-D\cdot\sum_i w_i\cdot x_i}}$$



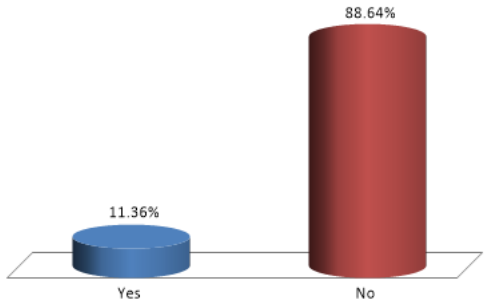
D is the slope of the function

Would a multi-layer feed-forward neural network achieve correct classification (100% accuracy) when tested on the B set and trained on the rest?

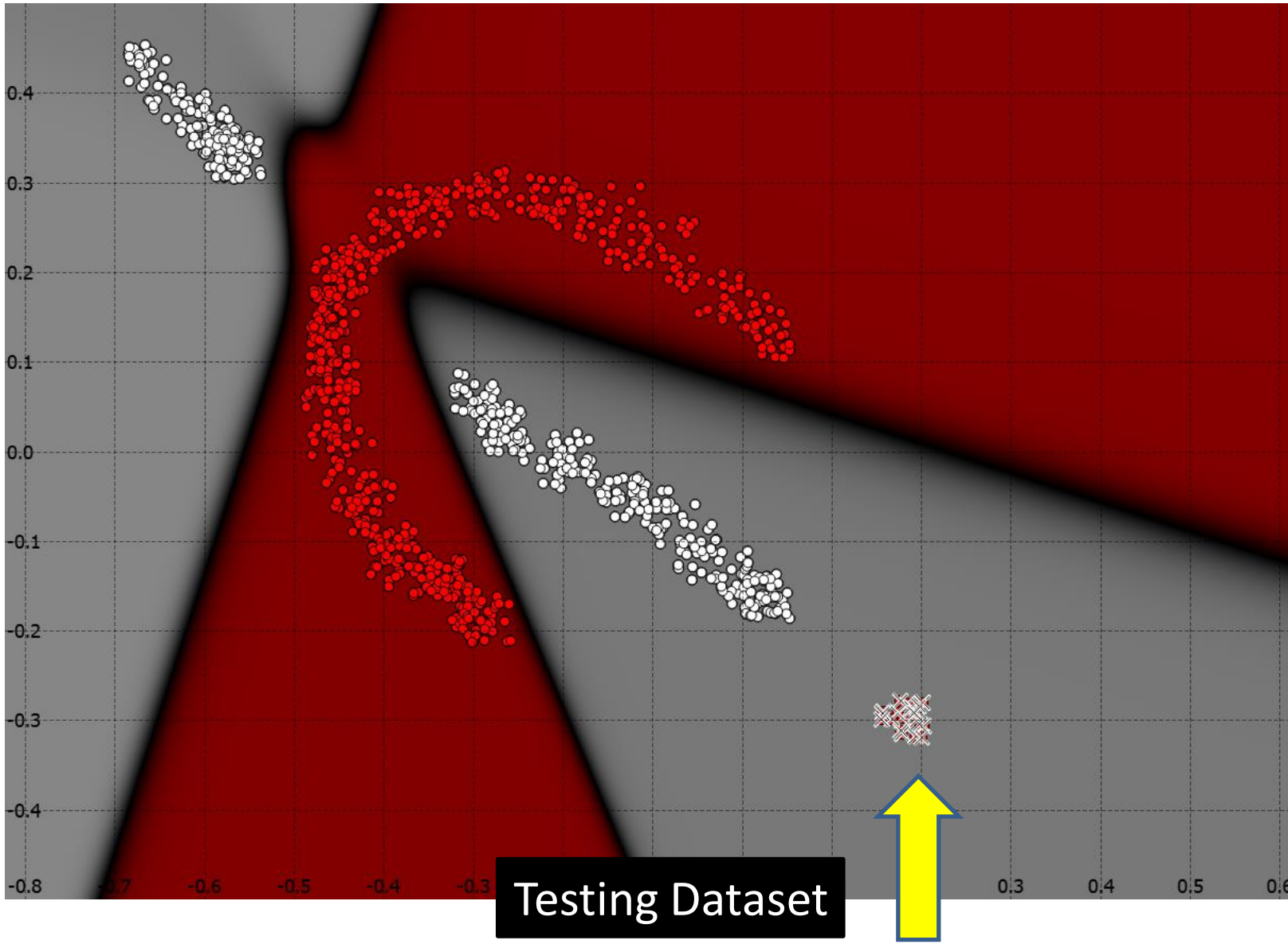


A. Yes

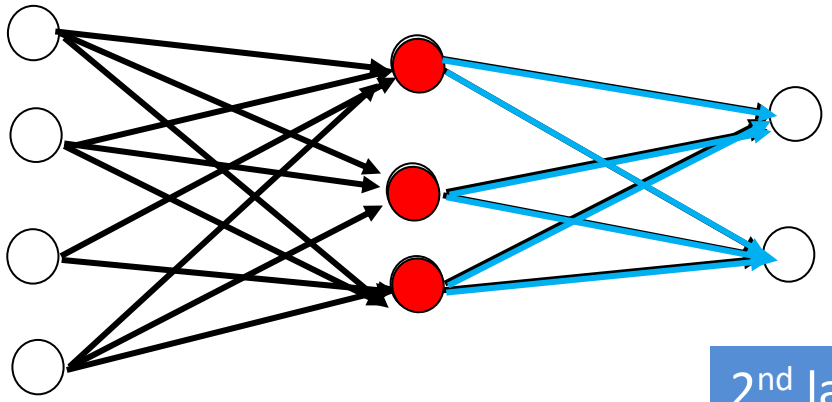
B. No



No, as the NN will learn piece-wise decomposition of the space.
Set of linear functions with ReLu or quasi linear with sigmoid activation fct.

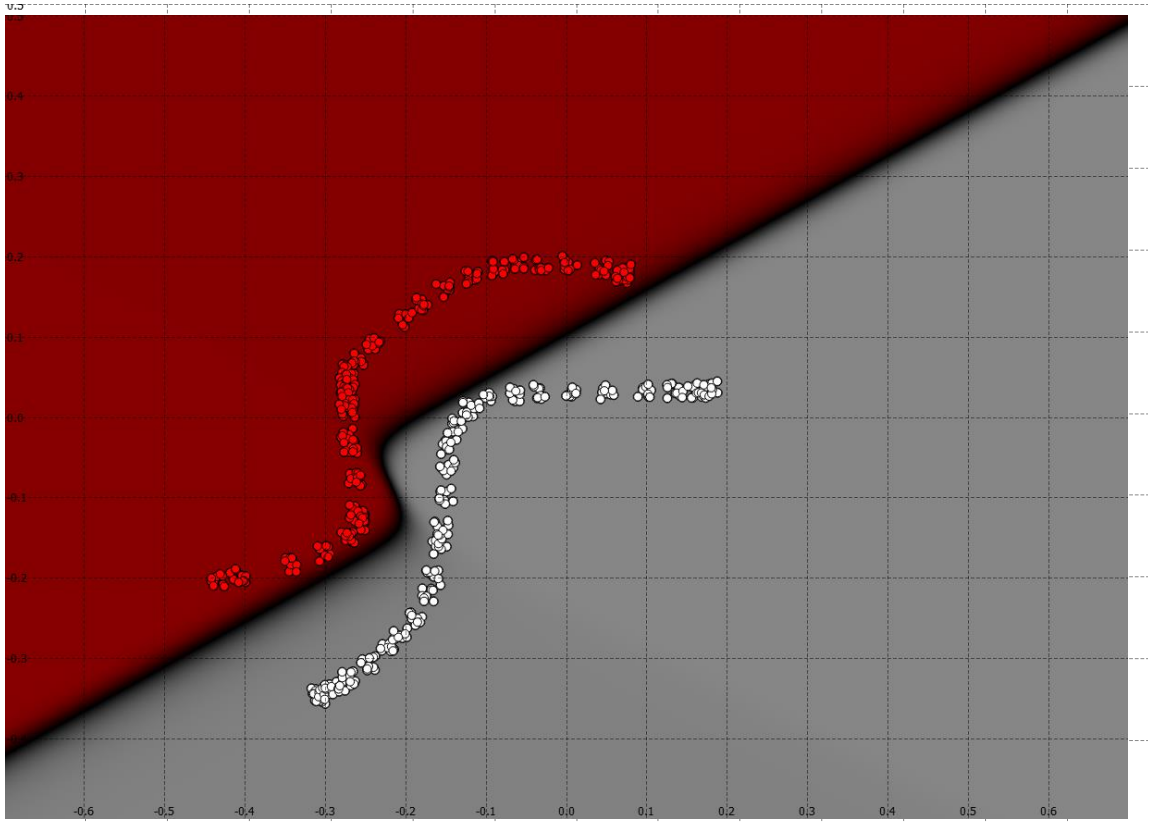


Each hidden neuron creates a dividing plane.

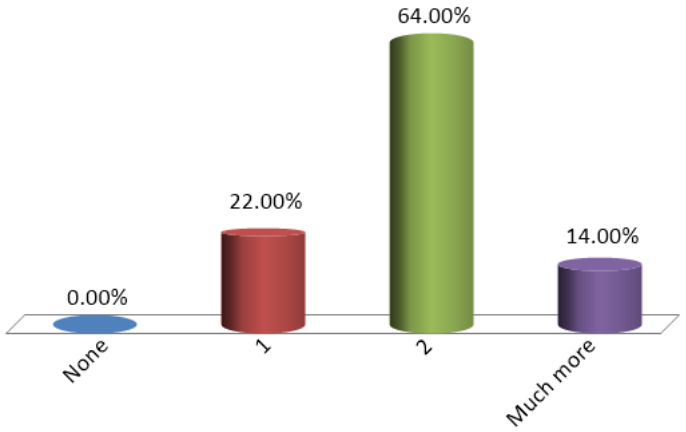


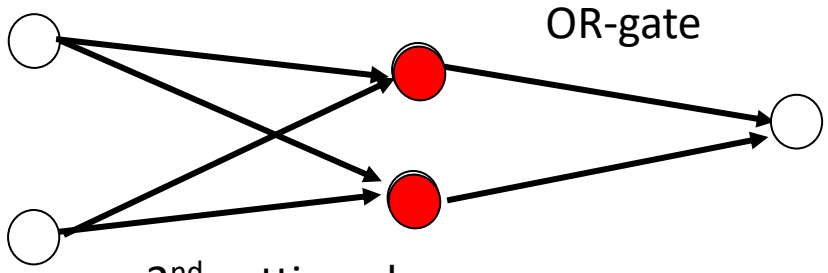
2nd layer combines planes through AND / OR gates

How many neurons do you need at minimum in the hidden layer to separate correctly these two classes?



A. None
B. 1
C. 2
D. 3

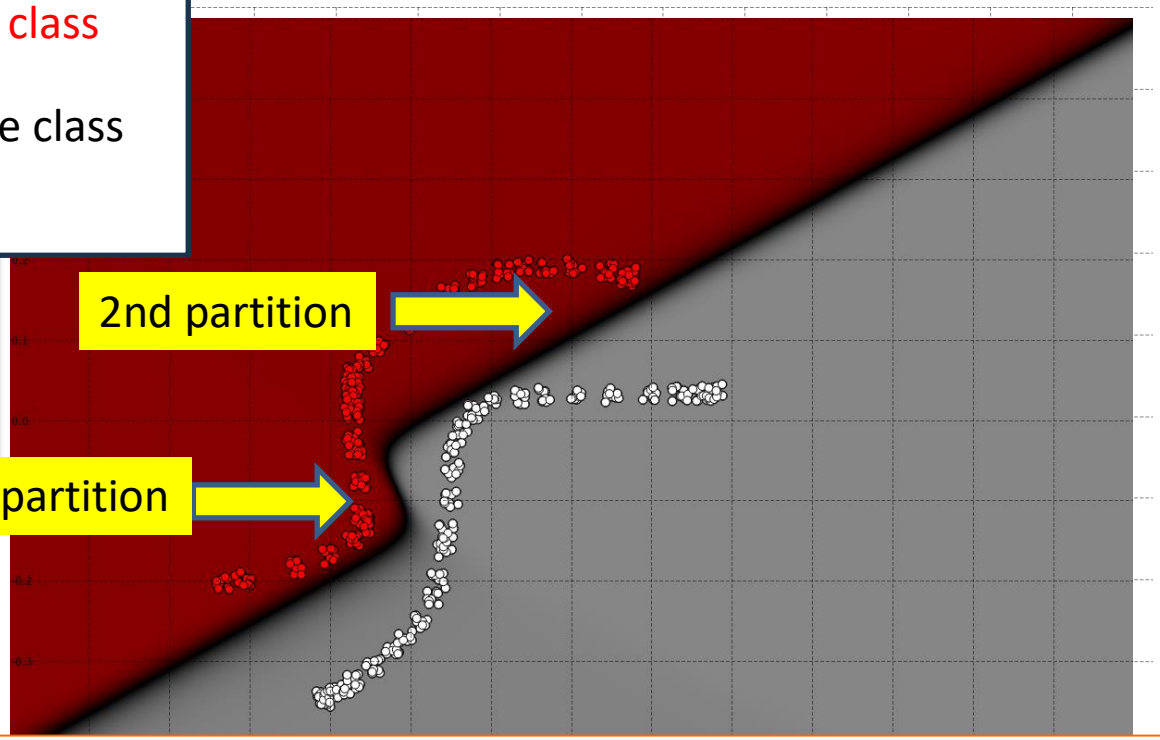


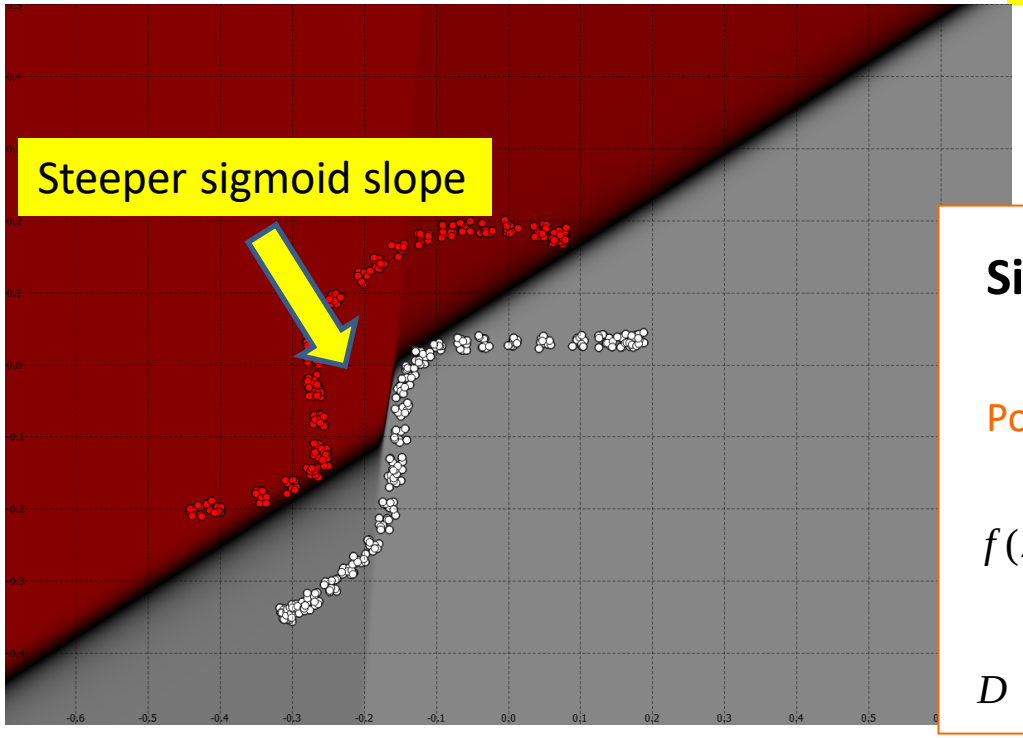


y>=0 → red class

y<0 → white class

2 neurons suffice as the boundary is composed of 2 partitions, but it depends on the slope of the activation function.

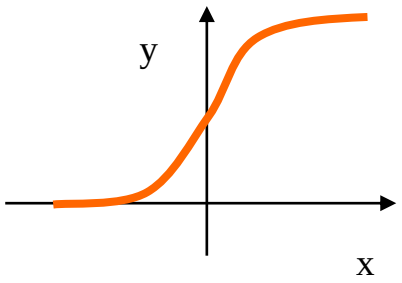




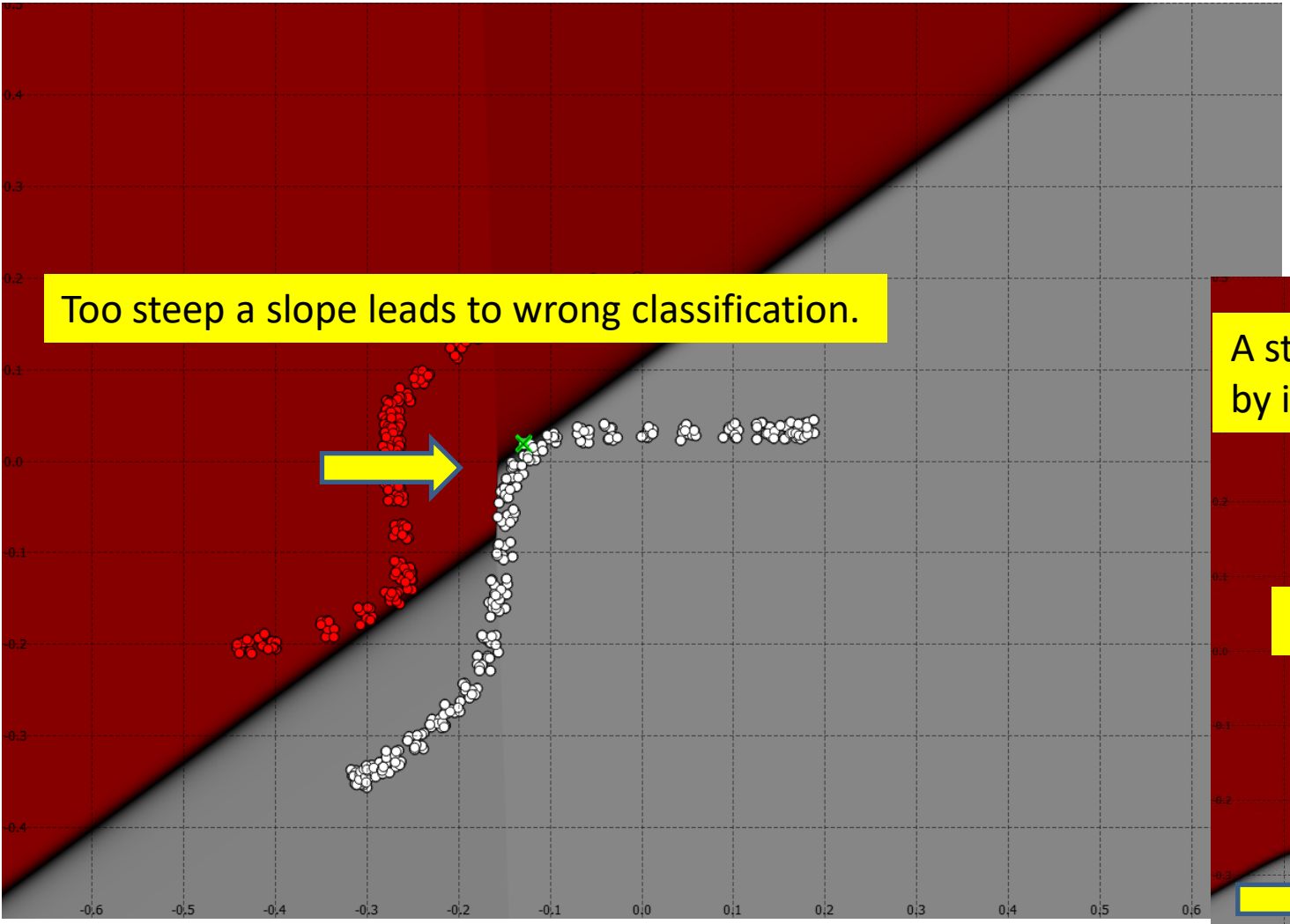
Sigmoid

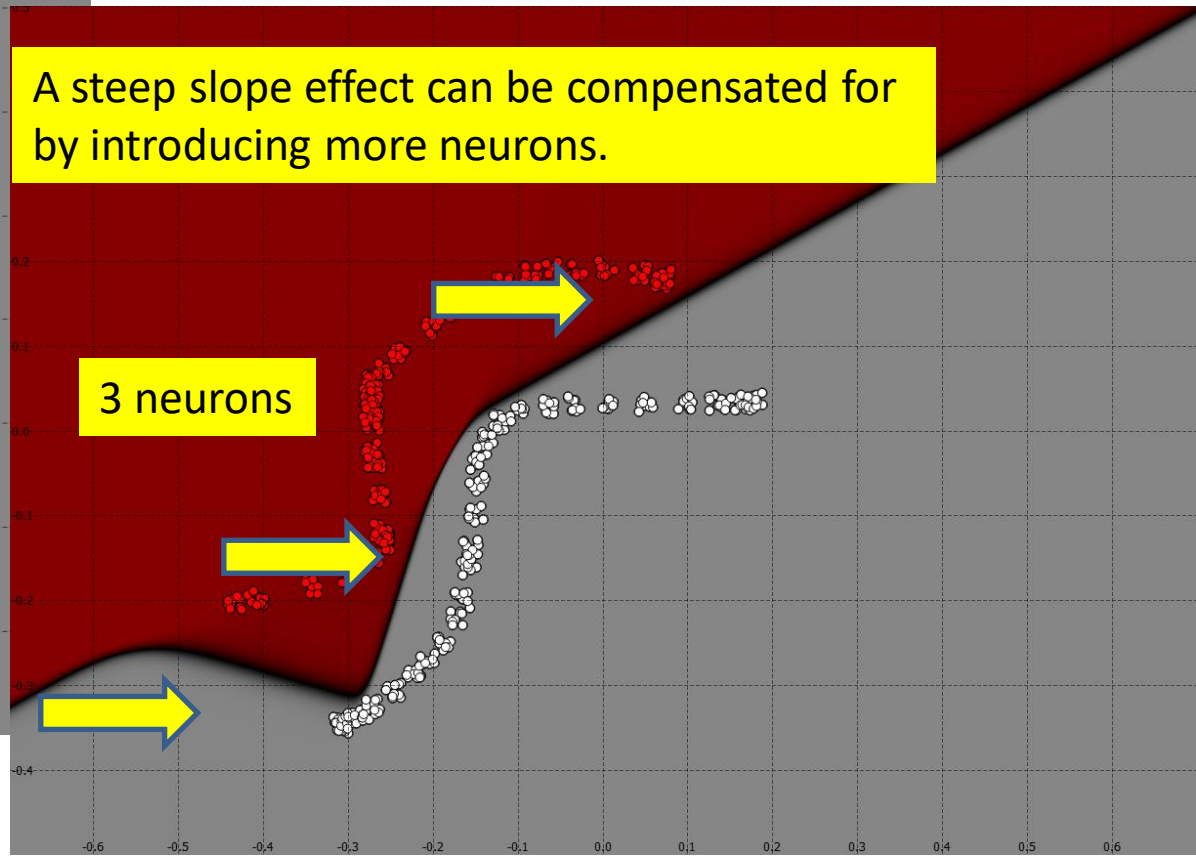
Positive form:

$$f(z) = g(z) = \frac{1}{1+e^{-D\cdot z}} \quad \Rightarrow \quad y = \frac{1}{1+e^{-D\cdot\sum_i w_i\cdot x_i}}$$

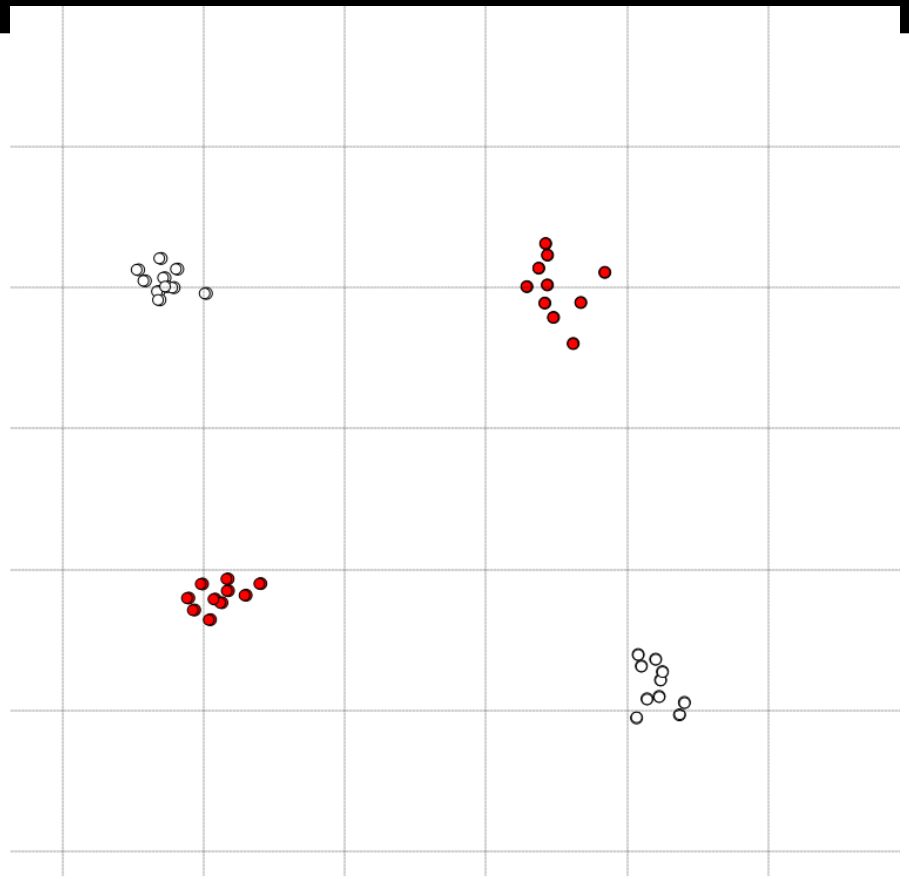


D is the slope of the function

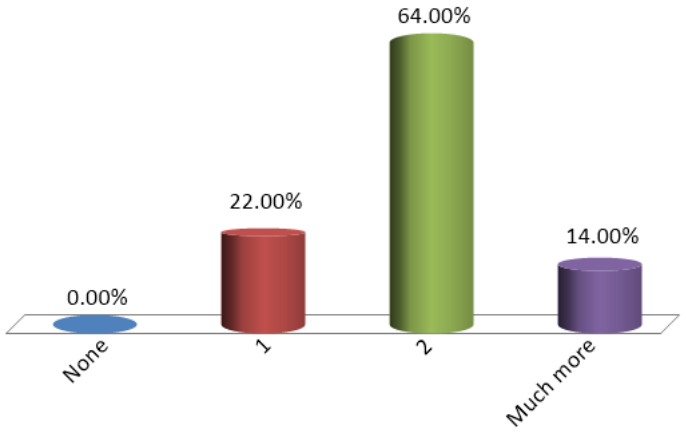

Too steep a slope leads to wrong classification.

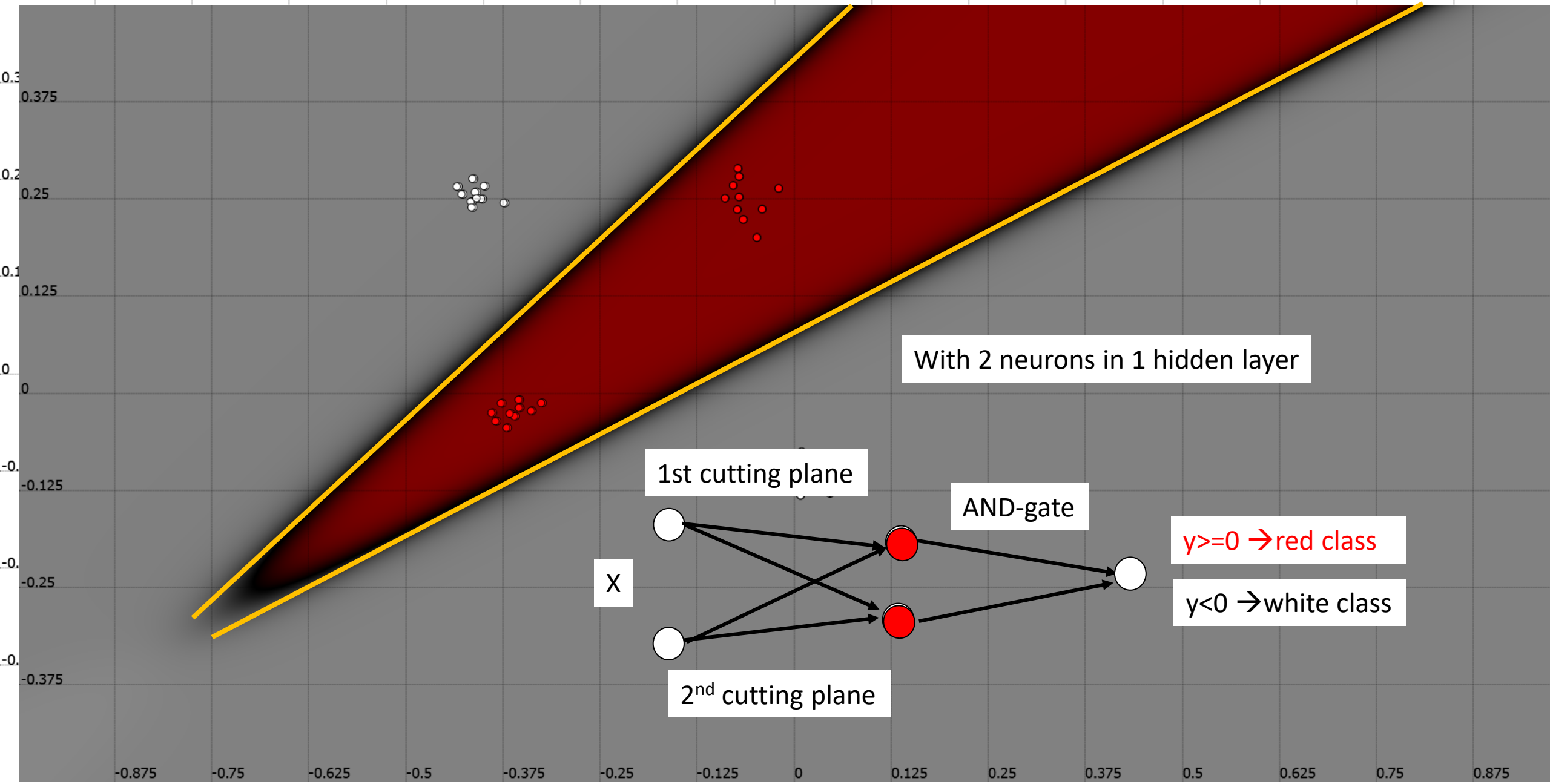

A steep slope effect can be compensated for by introducing more neurons.
3 neurons

How many neurons do you need at minimum in the hidden layer to separate correctly these two classes?



A. None
B. 2
C. 4
D. 6

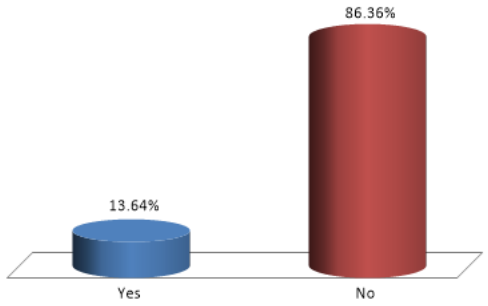



0.375
0.25
0.125
0
-0.125
-0.25
-0.375
-0.875
-0.75
-0.625
-0.5
-0.375
-0.25
-0.125
0
0.125
0.25
0.375
0.5
0.625
0.75
0.875
With 2 neurons in 1 hidden layer
1st cutting plane
AND-gate
X
y>=0 →red class
y<0 →white class
2nd cutting plane

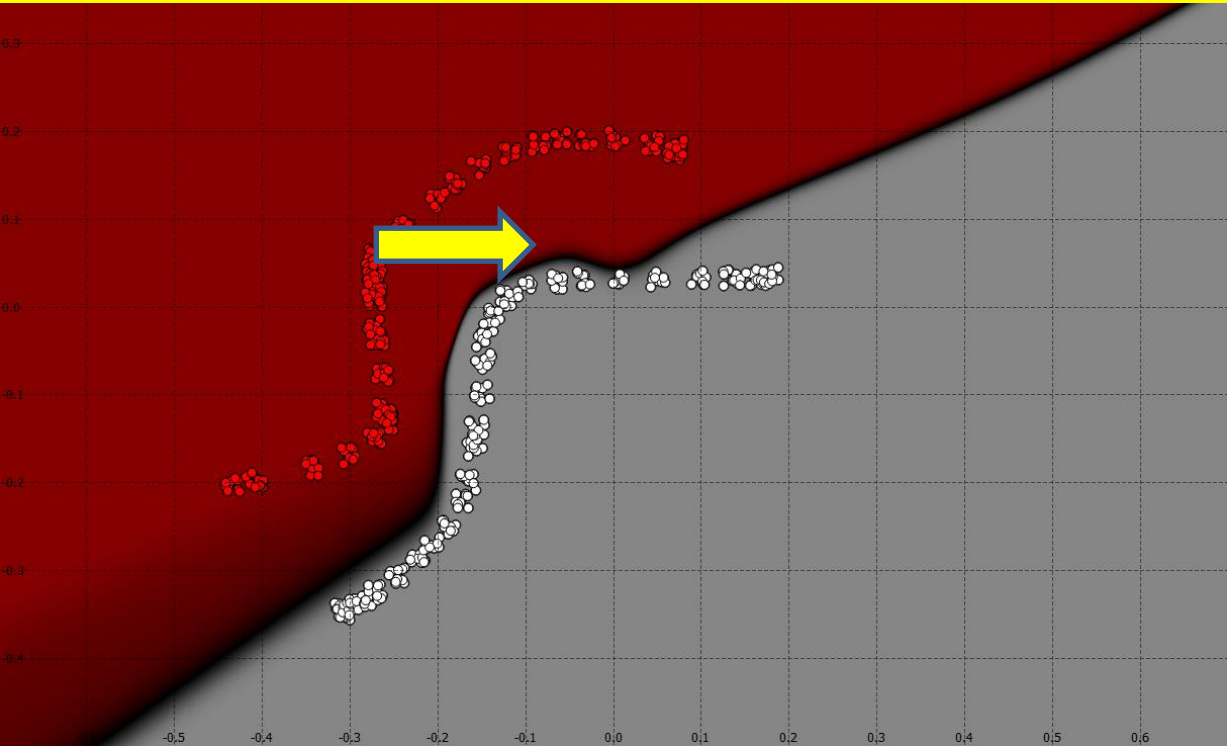
The more neurons, the more layers, the merrier?

A. Yes

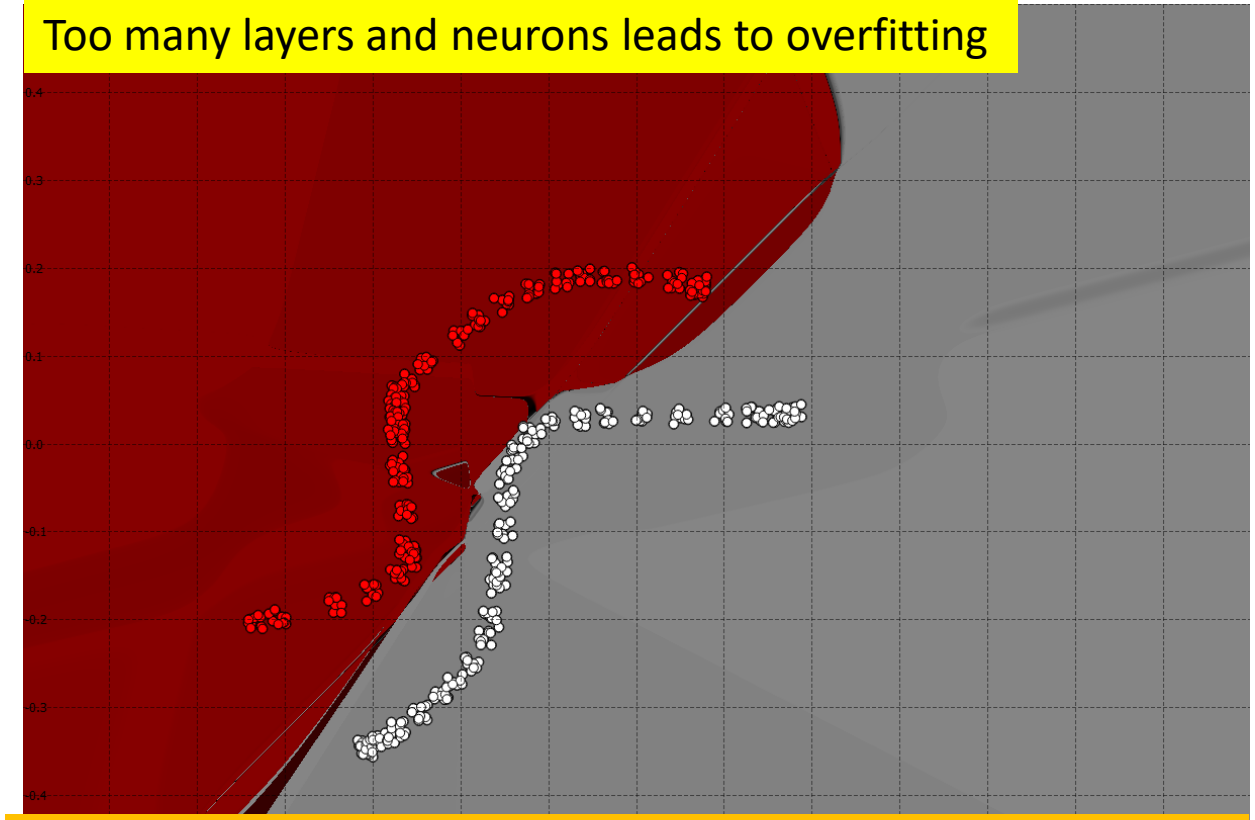
B. No



More neurons on 1-hidden layer leads to smoother and tighter curve

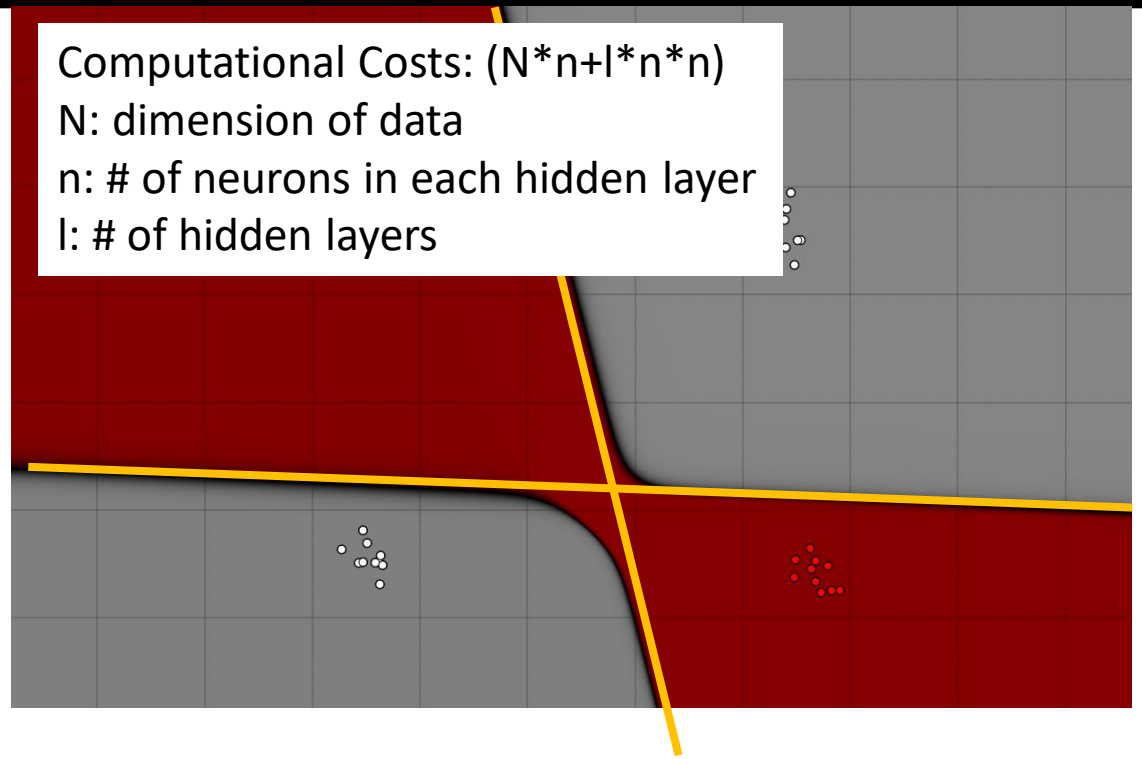


Too many layers and neurons leads to overfitting

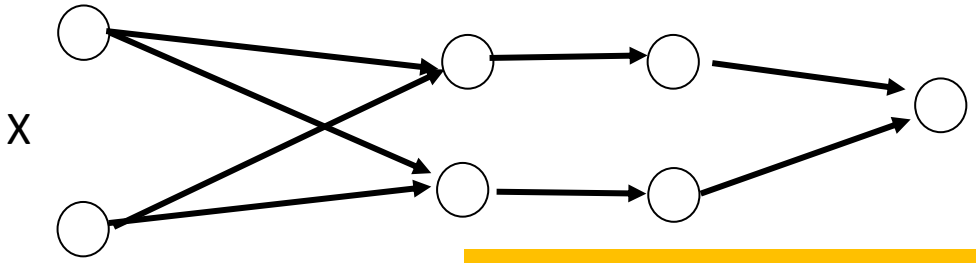


Here 5 layers, 10 neurons per layer → overfitting
With 500 datapoints total / too many parameters to estimate

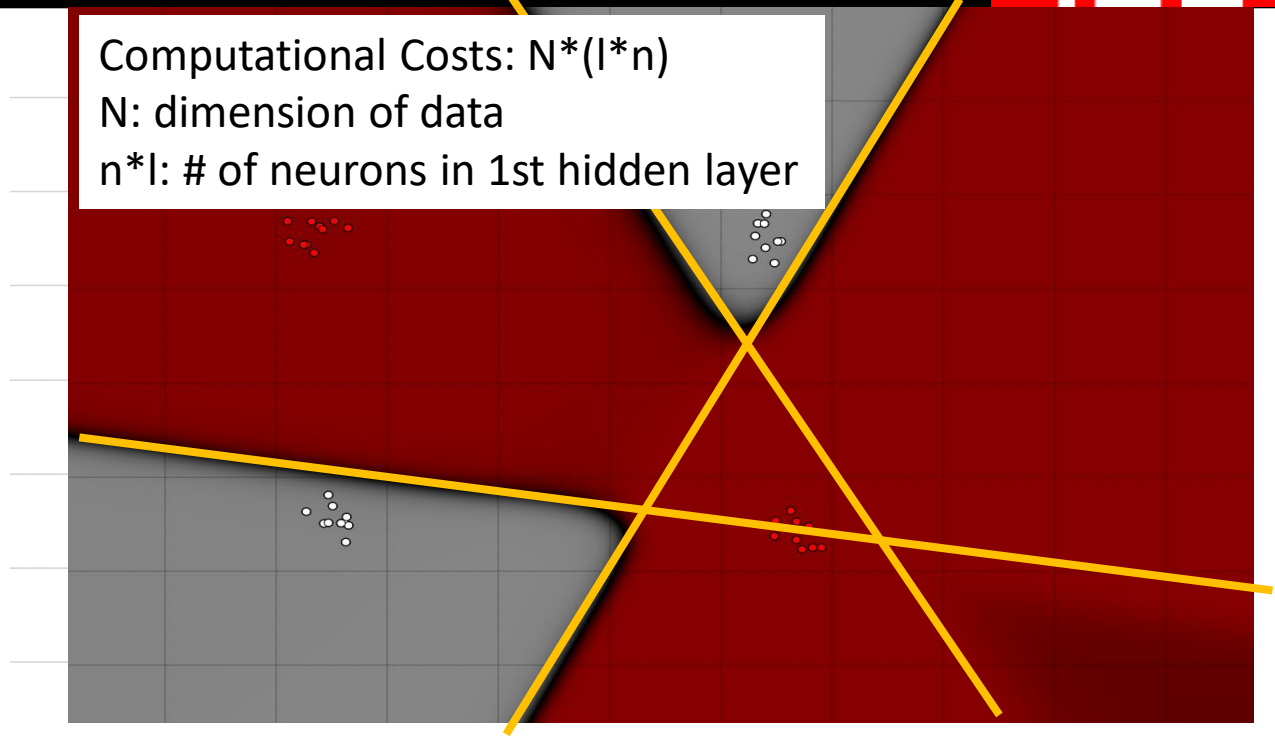
Early stopping with dropout and weight constraint can mitigate this effect.



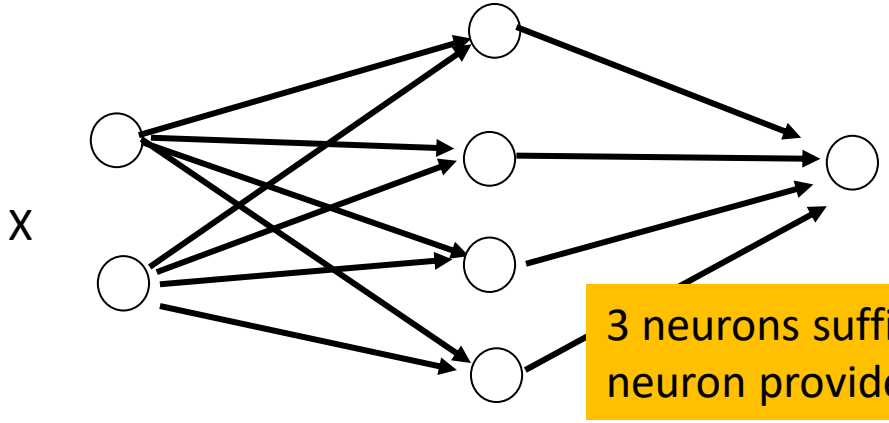
With 2 neurons in each of 2 hidden layers



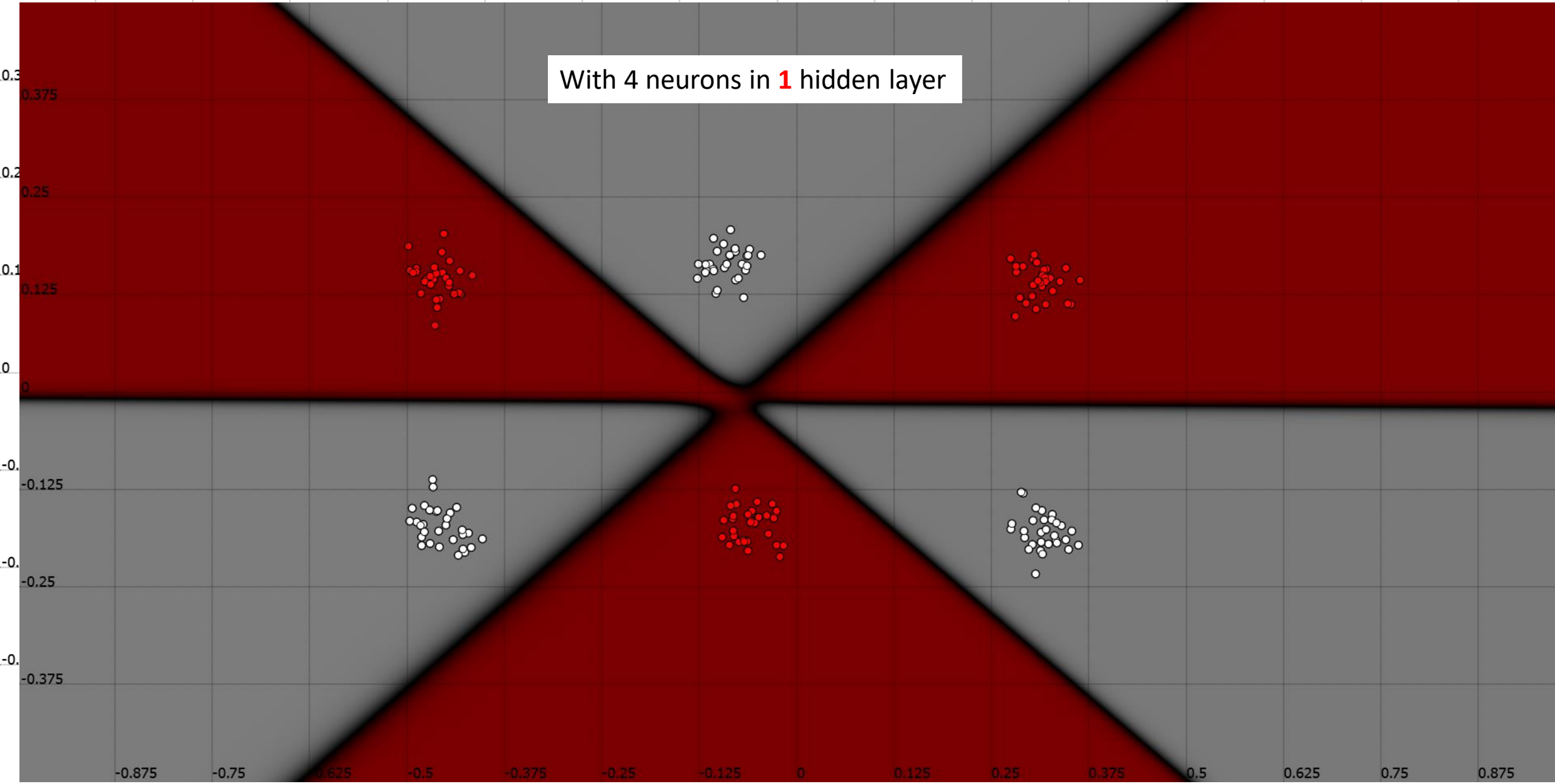
Layers apply logic functions
If, then rule for decision

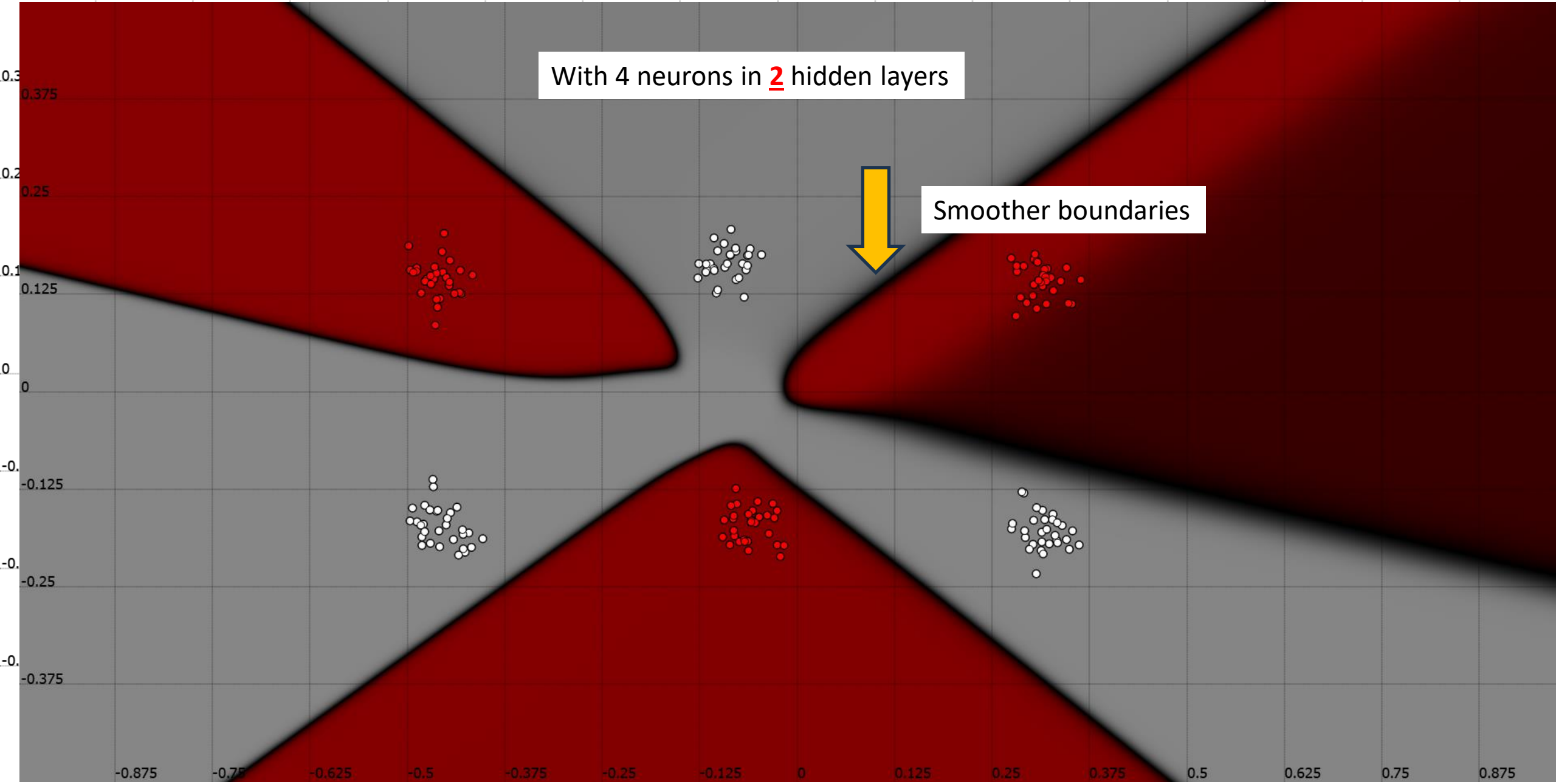


With 4 neurons in hidden layer



3 neurons suffice, 4th neuron provides redundancy


With 4 neurons in 1 hidden layer
0.375
0.25
0.125
0
-0.125
-0.25
-0.375
-0.875
-0.75
-0.625
-0.5
-0.375
-0.25
-0.125
0
0.125
0.25
0.375
0.5
0.625
0.75
0.875


With 4 neurons in 2 hidden layers
Smoother boundaries
0.375
0.25
0.125
0
-0.125
-0.25
-0.375
-0.875
-0.625
-0.5
-0.375
-0.25
-0.125
0
0.125
0.25
0.375
0.5
0.625
0.75
0.875

THE BACKPROPAGATION ALGORITHM

For each input-output pair, do:

1) Calculate the output y_j of each output unit j of the net

2) Calculate $\frac{\partial E}{\partial w_{ij}}$ for all weights on arcs to output nodes

3) Calculate $\frac{\partial E}{\partial s_j}$ for each of these nodes

4) Calculate $\frac{\partial E}{\partial s_j}$ for the last hidden layer

5) ... and so on, propagating $\frac{\partial E}{\partial s_j}$ backward towards the first layer

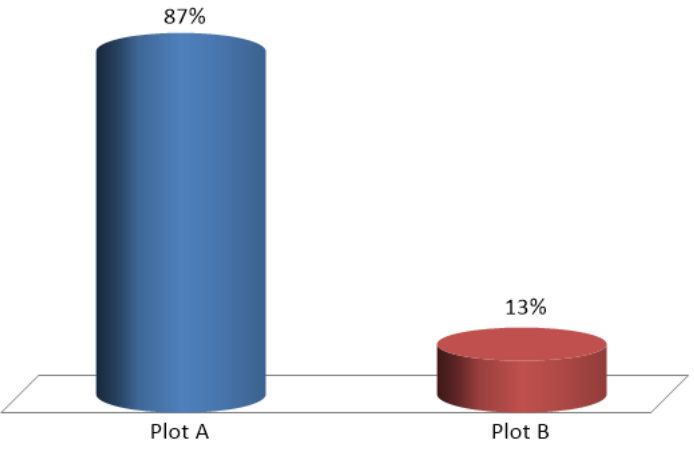
6) finally, compute all the weight changes:

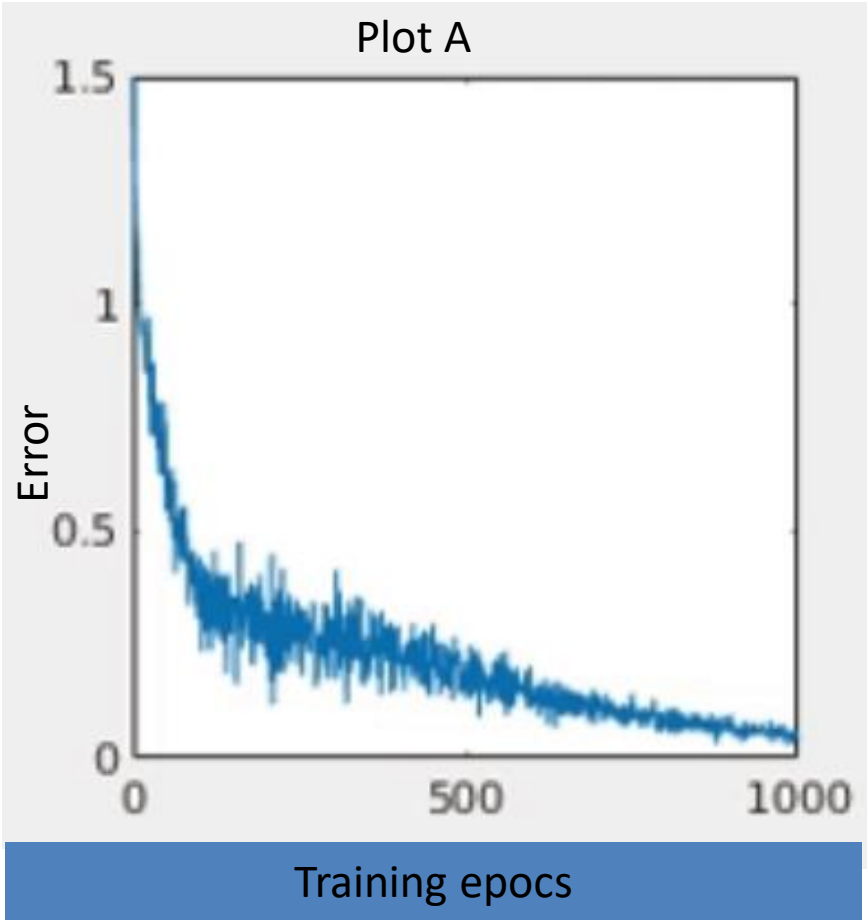
$$\Delta w_{ij} = -\eta \cdot \frac{\partial E}{\partial w_{ij}}$$

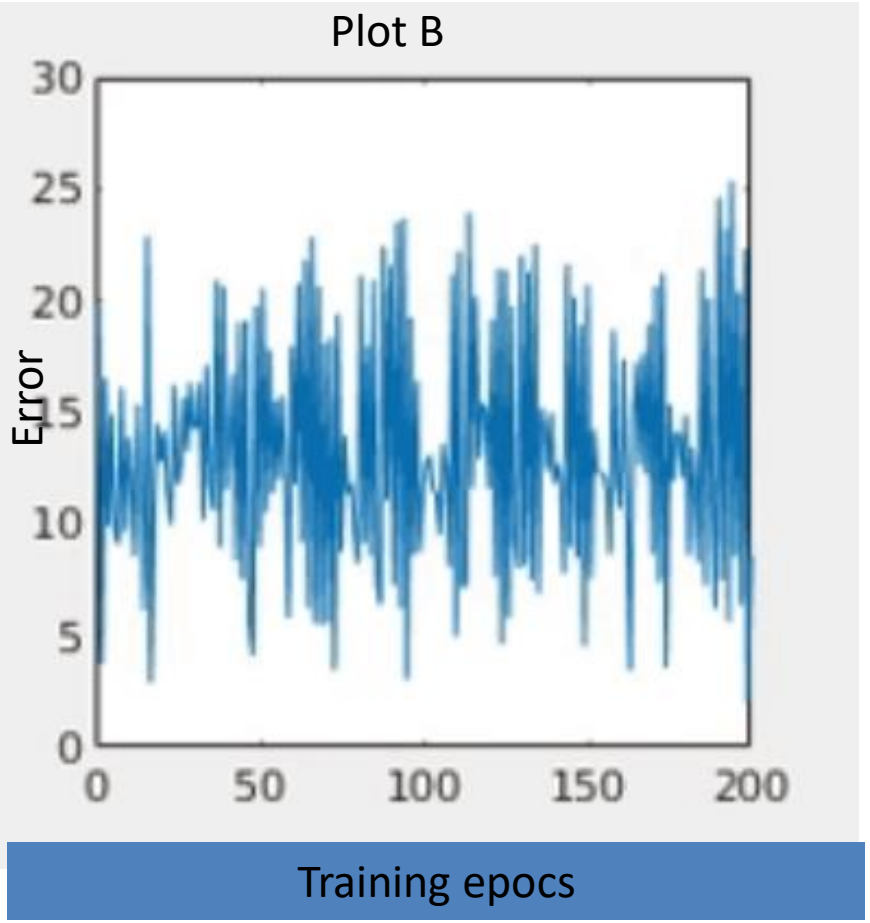
Which of the following error-plot corresponds to a NN trained with a lower learning rate?

A. Plot A

B. Plot B

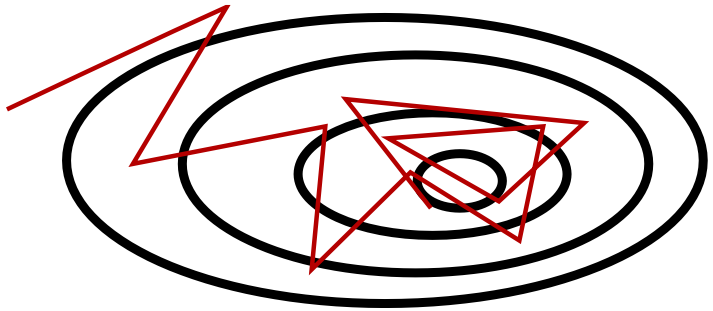






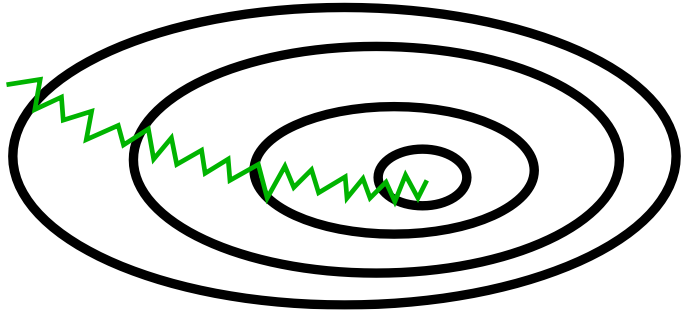
Effects of the learning rate

large learning rate



Might not converge

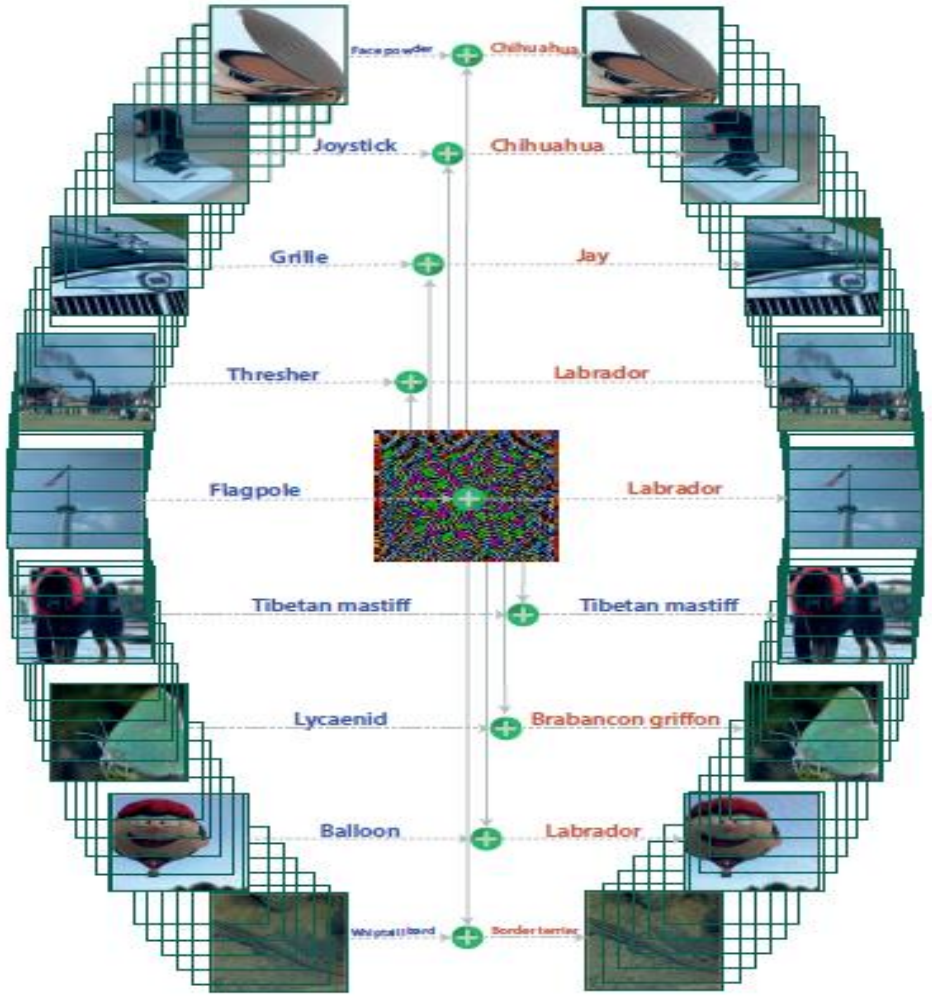
small learning rate

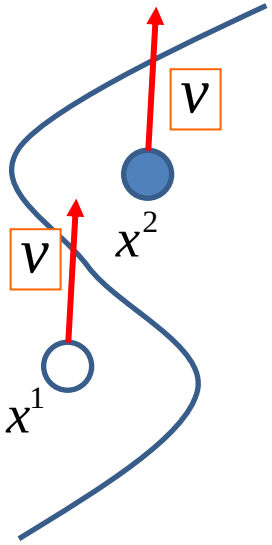


Slower learning

Adaptive learning rate (learning rate decay)

How to Fool Neural Networks





- Class 1
- Class 0

Add a small "image-agnostic" perturbation v

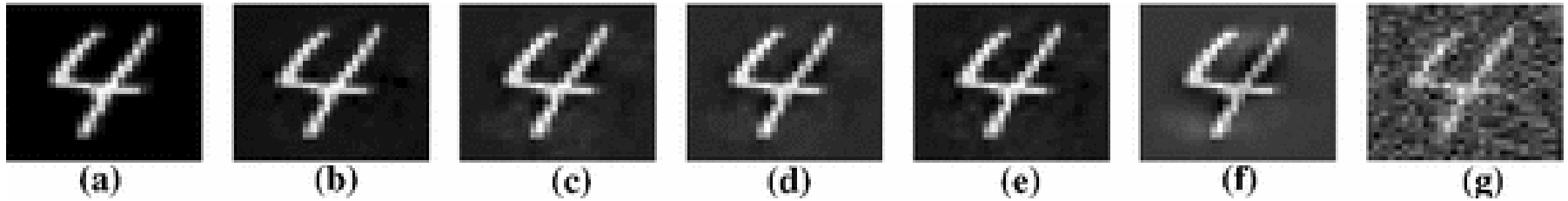
$x' = x + v$

that leads to missclassification.

Algorithm that can learn from few training datapoints which "minimal" perturbation ($\min \|v\|$) that leads to missclassification with some probability p.

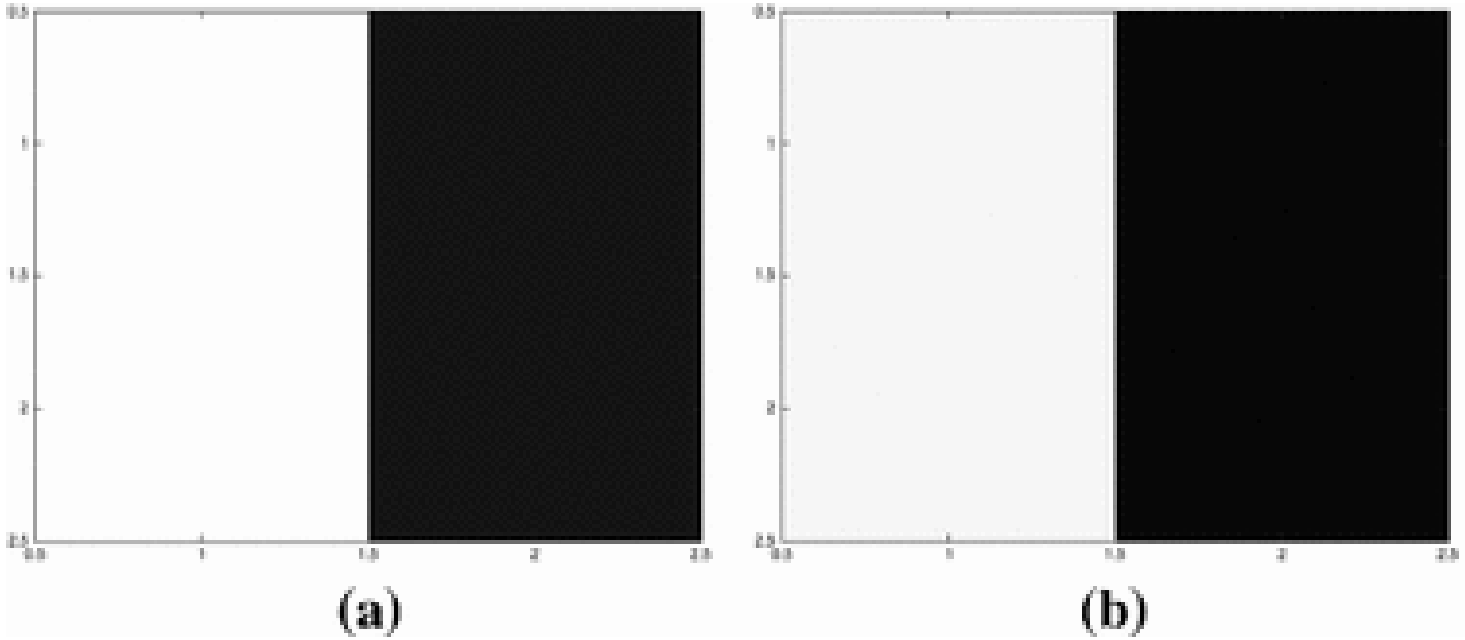
Moosavi-Dezfooli, S. M., Fawzi, A., Fawzi, O., & Frossard, P. Universal adversarial perturbations. CVPR (2017).

How to Fool Neural Networks



Original image **a** and minimally perturbed images (**b**)–(**f**) that switch the estimated label of linear (**b**), quadratic (**c**), cubic (**d**), RBF(1) (**e**), RBF(0.1) (**f**) classifiers.

Moosavi-Dezfooli, S. M., Fawzi, A., Fawzi, O., & Frossard, P. Universal adversarial perturbations. CVPR (2017).

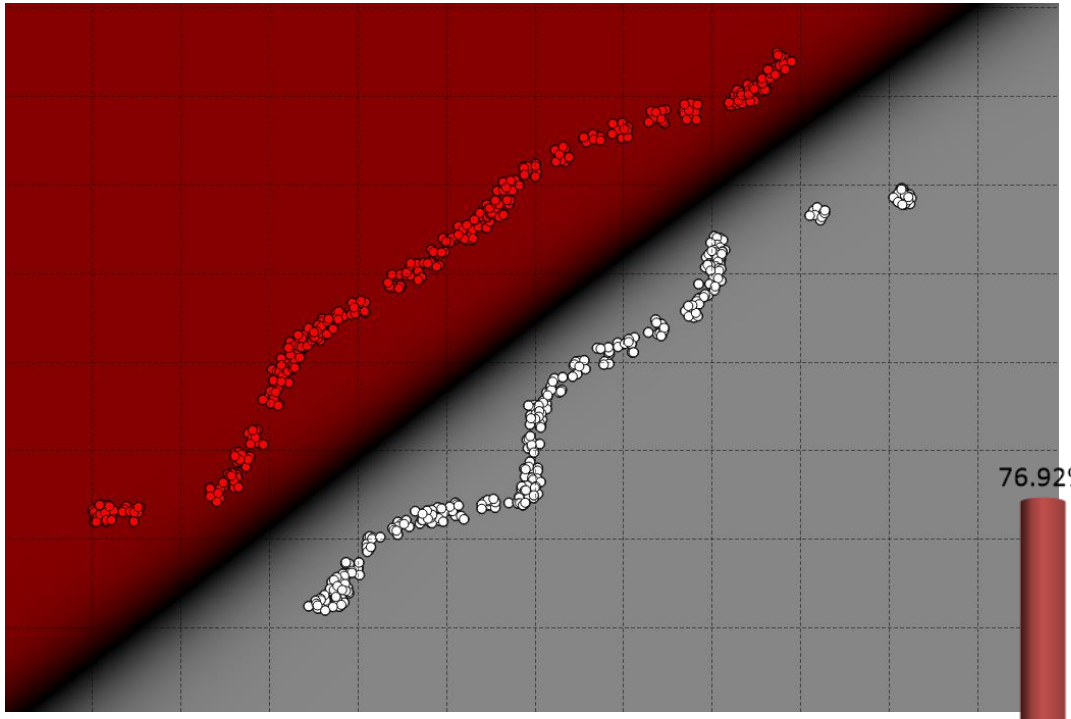


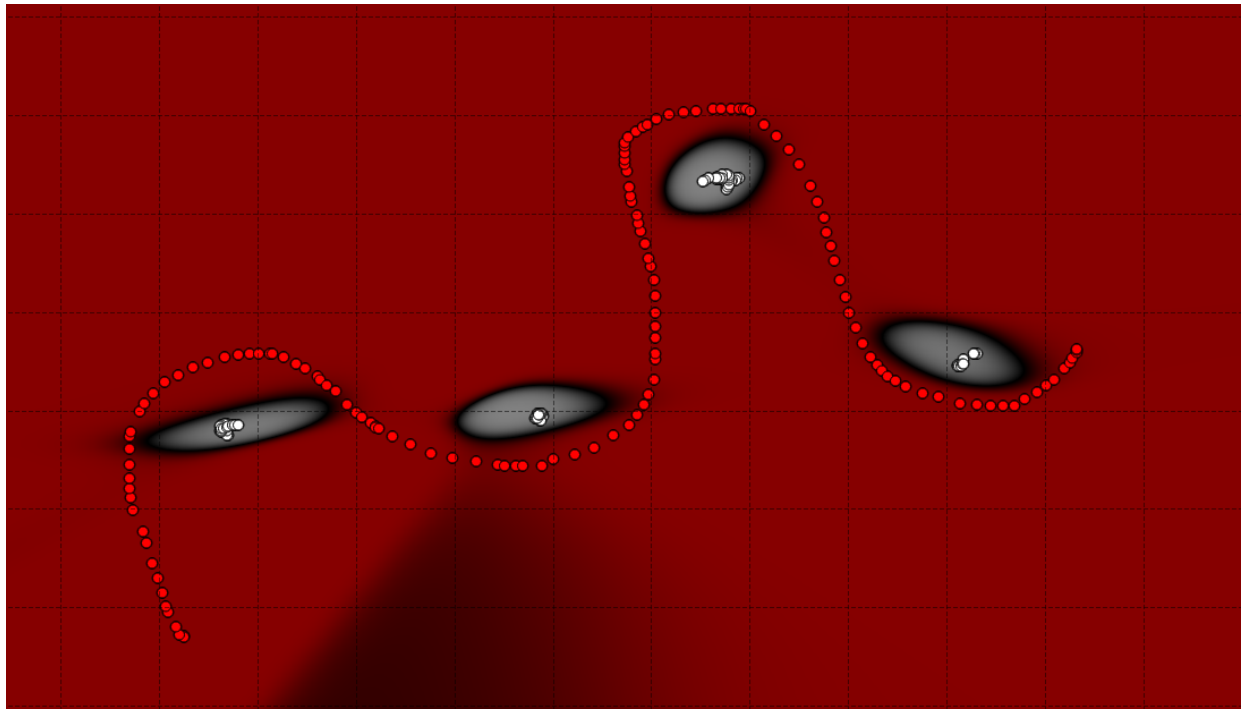
A small perturbation, quasi invisible, is applied to pattern (a) and makes it look like pattern (b). Pattern (b) changes label with a linear classifier.

Can you explain why?

The linear classifier has learned an AND-gate for all white (positive pixels).
A small perturbation that reduces the grey scale value of the white pixels leads to incorrect prediction.

Which of the two datasets would be easiest to fool?



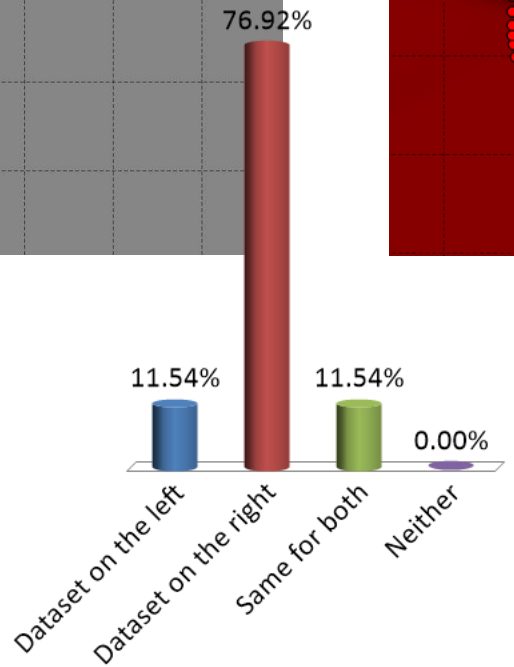


A. Dataset on the left

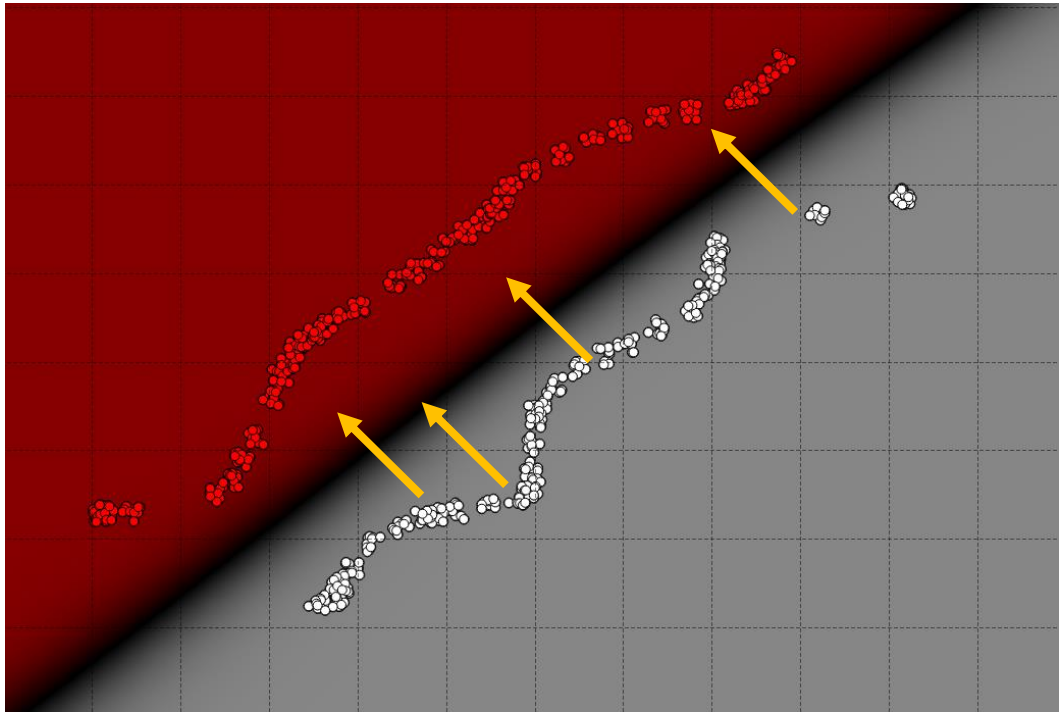
B. Dataset on the right

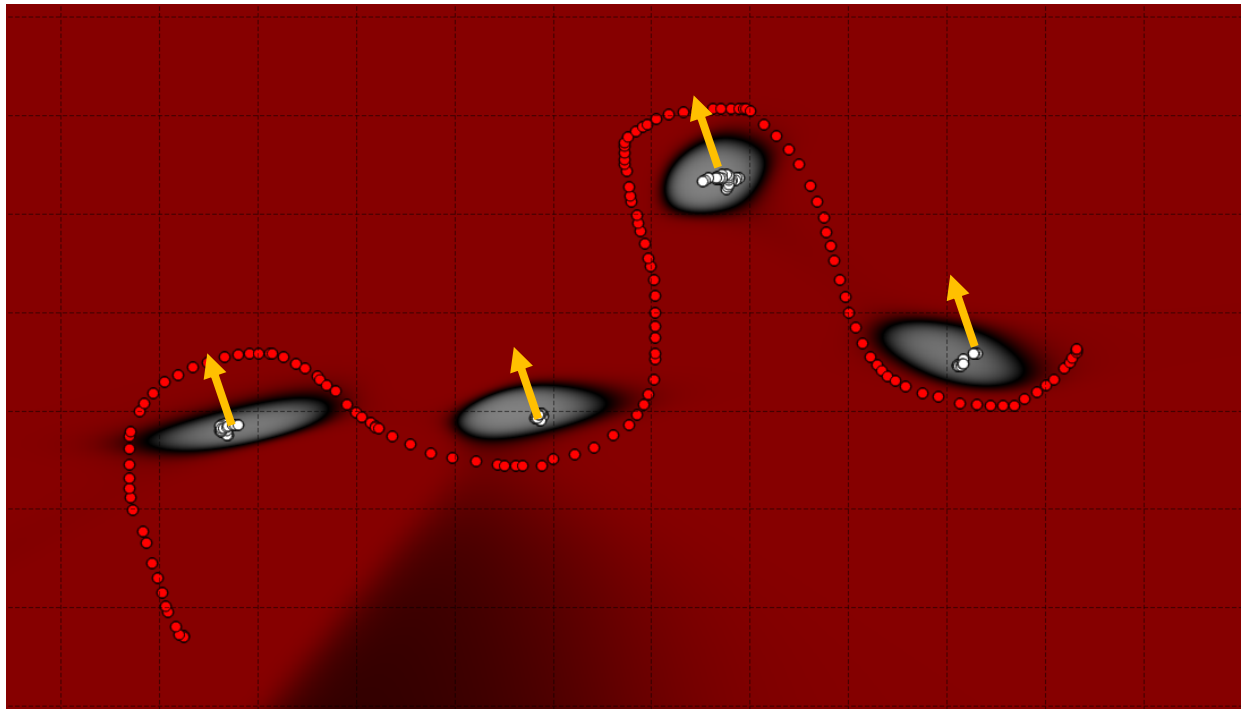
C. Same for both

D. Neither



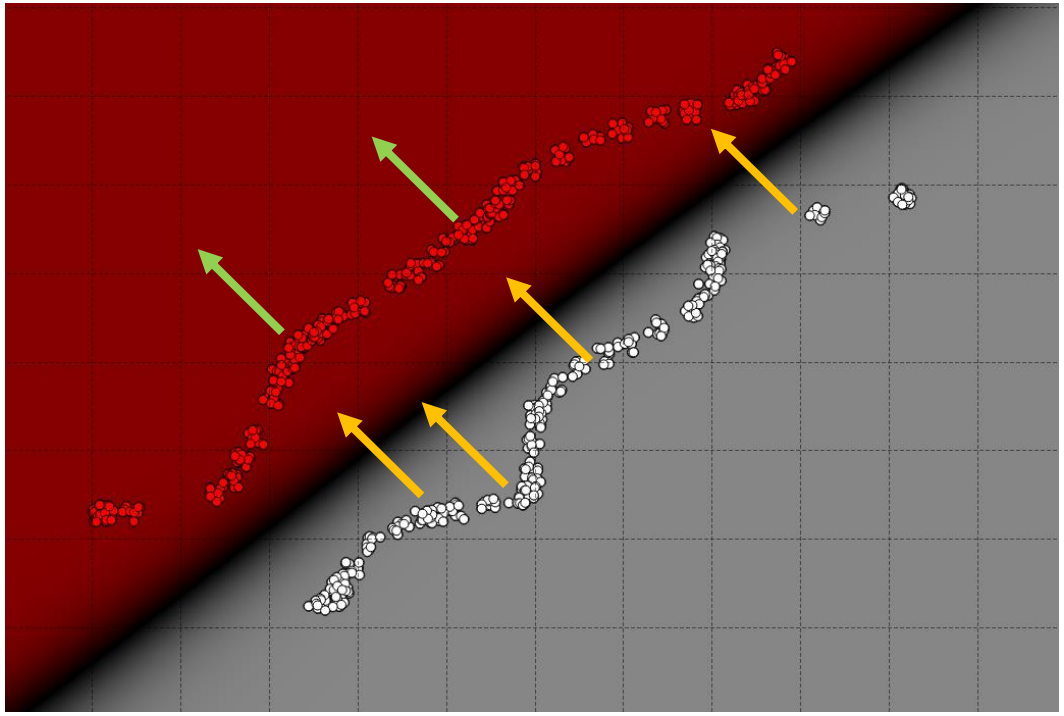
Which of the two datasets would be easiest to fool?

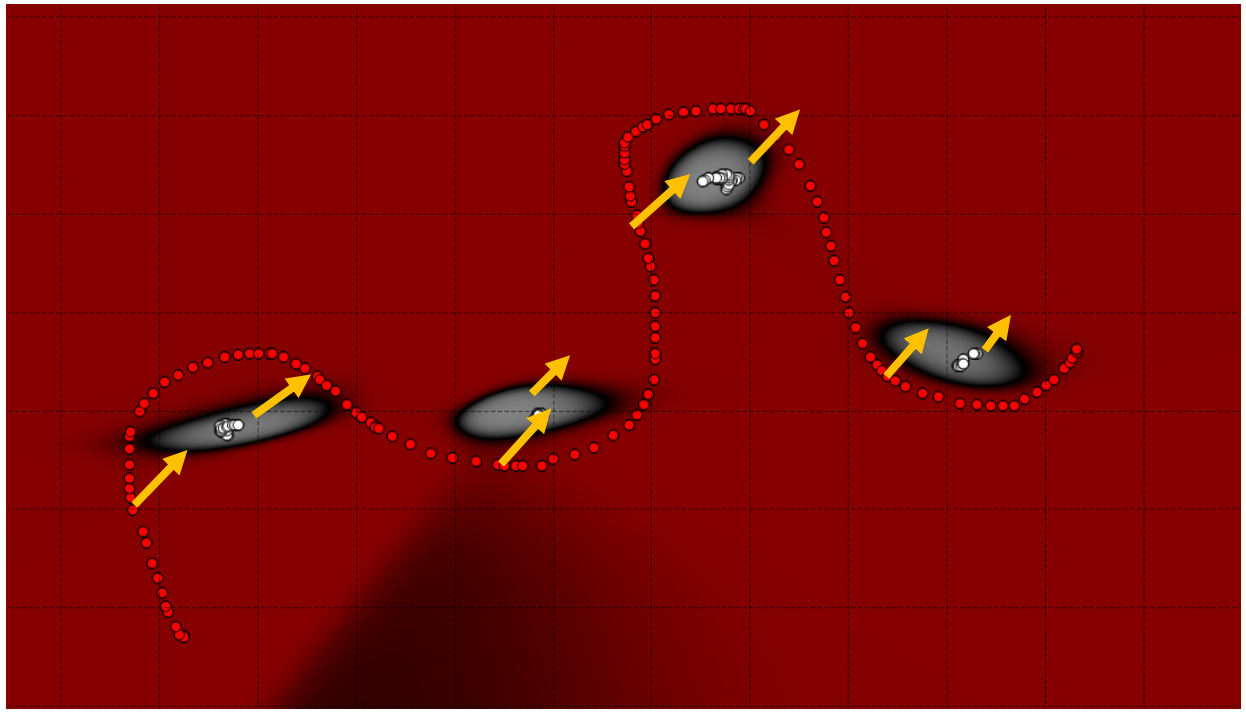




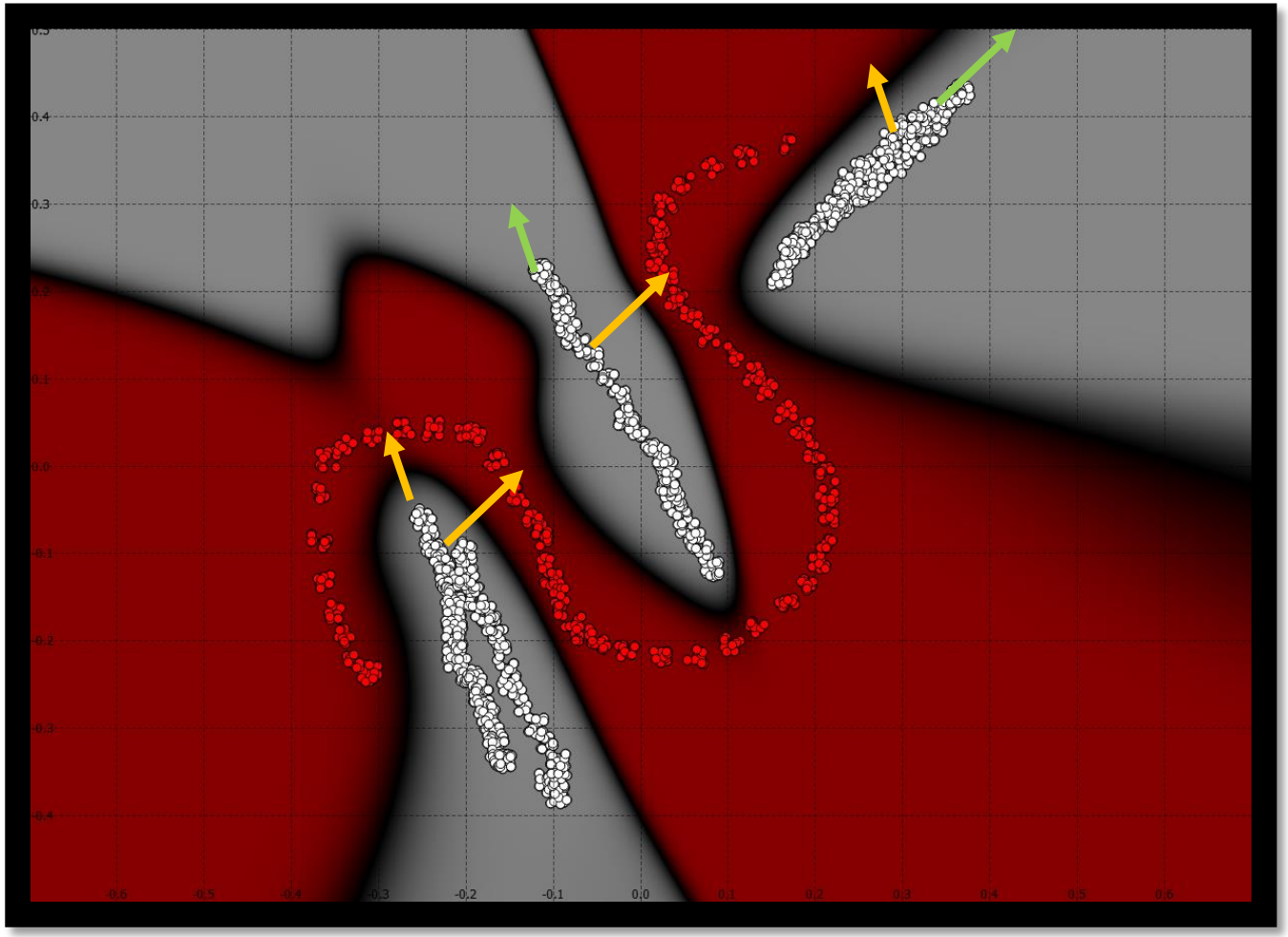
It is as easy if one wants to fool one class for the other.
In the linear classification case (left), one just needs to find the correct perturbation vector.
In nonlinear classification case (right), it becomes easy when the two classes are tightly mingled and we have quasi overfitting across one class.

Which of the two datasets would be easiest to fool?

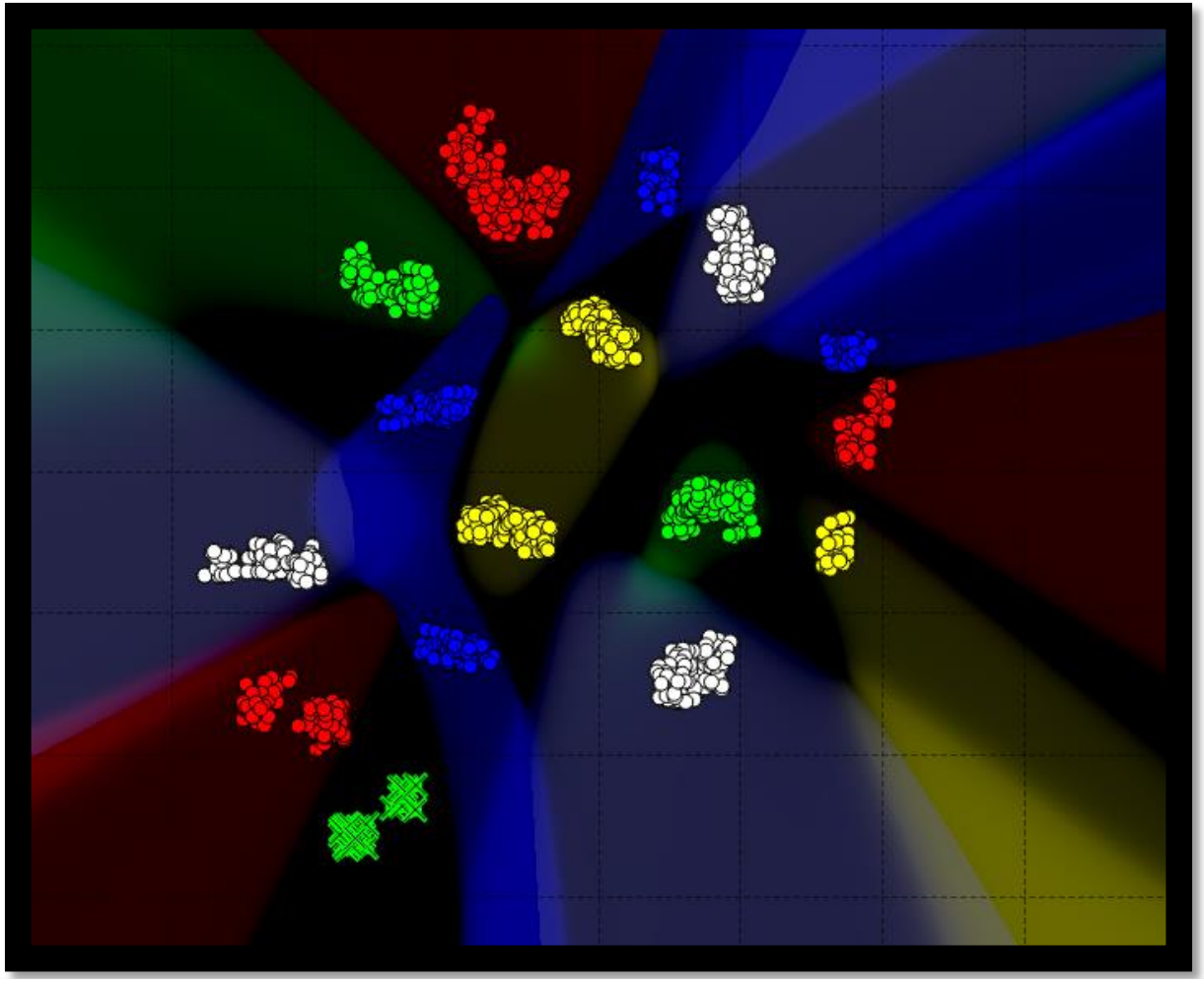




It is not as easy if one wants to fool both classes with a single vector.
In the linear classification case (left), one vector will be insufficient.
In the nonlinear classification case (right), one vector may be sufficient to lead to incorrect classifications of elements of both classes.
The more mingled the classes, the easier it is to fool the NN. It is hence crucial to identify features (subdimensions, subpatterns) distinct for each class and based the classification on these features.



If the curvature of the boundary is very complex, it becomes more difficult to find a single vector that works for all the elements of the classes.



The more classes, the tighter the fit, the easier to generate a perturbation that will lead to misclassification with non-negligible probability on all classes.

How to construct boundaries robust to such perturbations?

1: Build a pre-processing layer to detect « perturbations ».

Akhtar, Naveed, Jian Liu, and Ajmal Mian. "Defense against universal adversarial perturbations." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018

2: Train using adversarial approach – make the network more robust to this through adversarial network

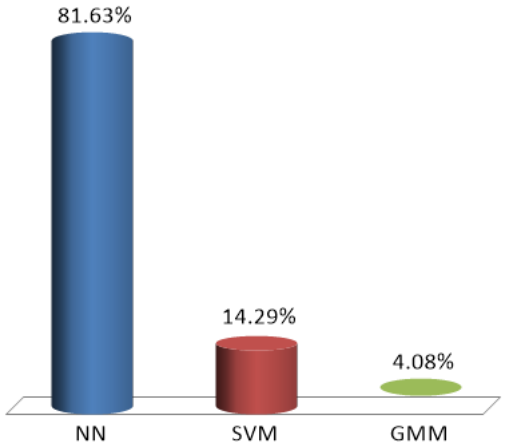
Ali Shafahi, Mahyar Najibi, Zheng Xu, John P. Dickerson, Larry S. Davis, and Tom Goldstein. Universal adversarial training. ArXiv, abs/1811.11304, 2020.

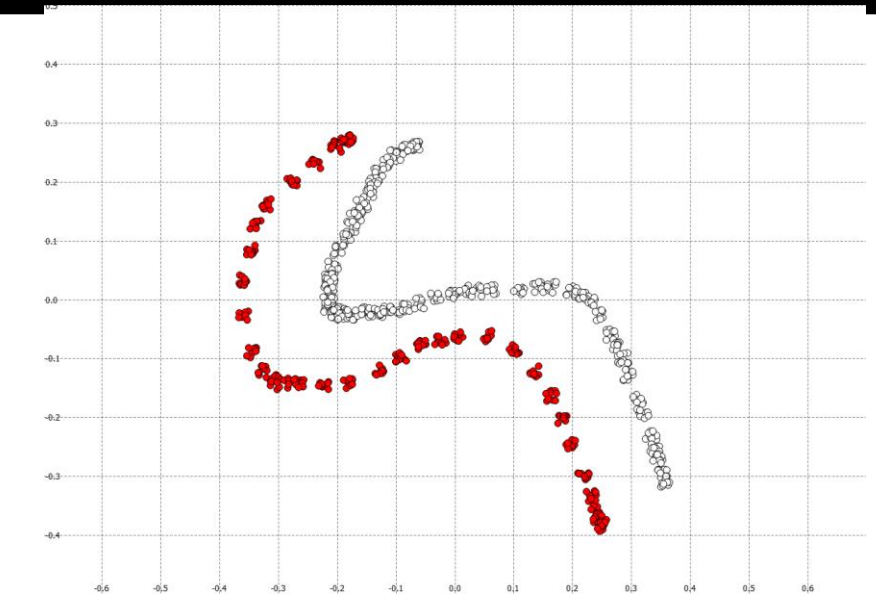
Which of the three classification techniques are most sensitive to fooling?

A. NN
B. SVM
C. GMM

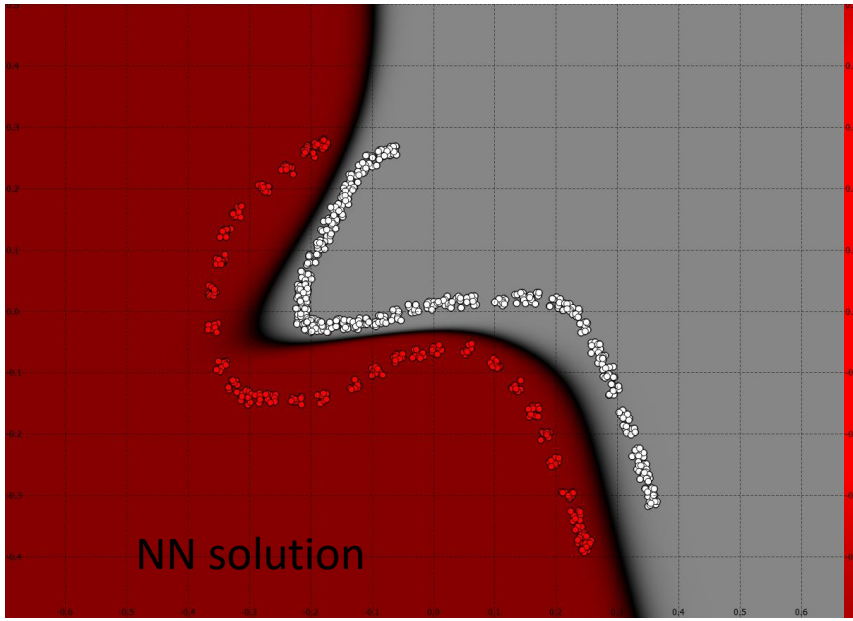
Class label determined by $y \sim \sum_i w_{ij}\,(x+\nu)$, with $\mathrm{Re}\,Lu$

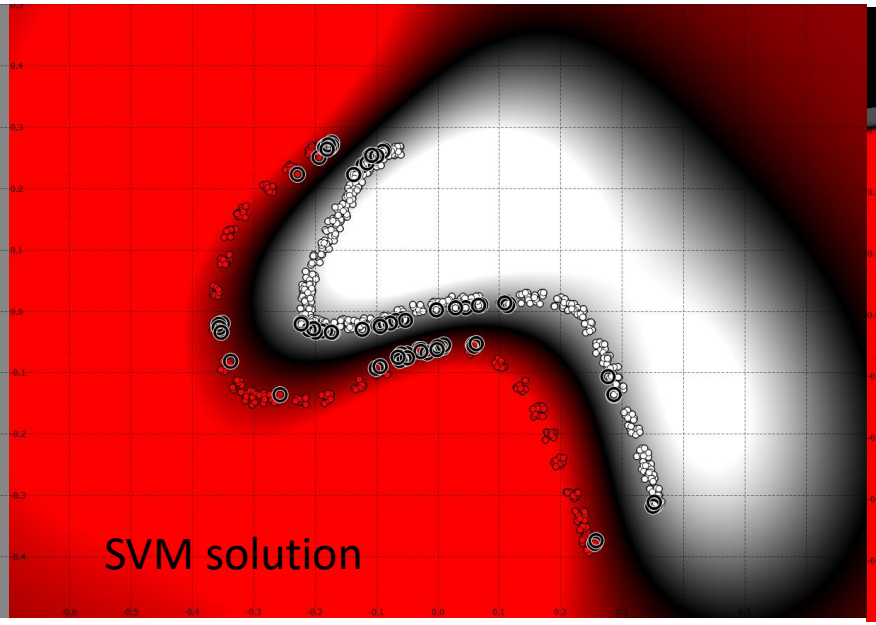
Class label determined by $y \sim \left(\sum_i w_{ij} \cdot k\,(x+\nu)\right)$ k : RBF or Gauss Fct

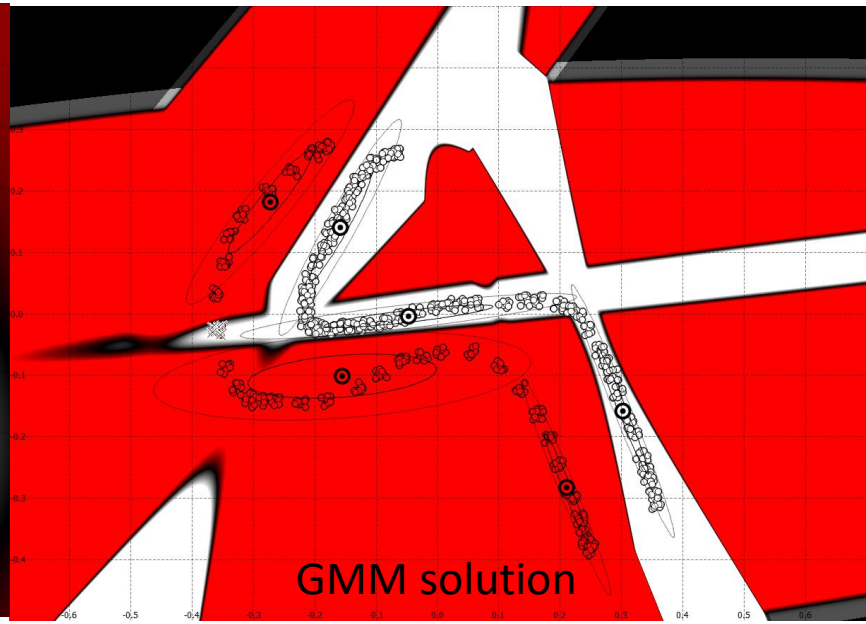




All techniques are as sensitive, as all decisions functions depend on a measure of the distance to the boundary.





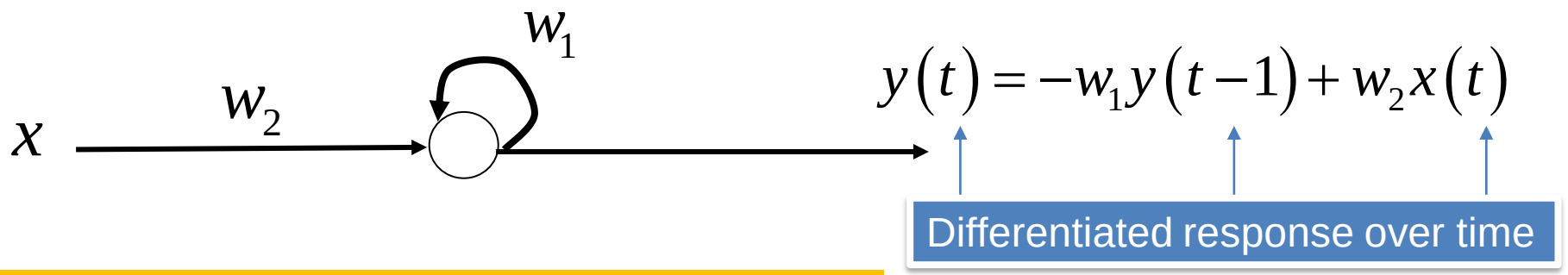


Recurrent Neural Networks

Most biological network have recurrent connections.

This change of direction in the flow of information is interesting, as it can allow:

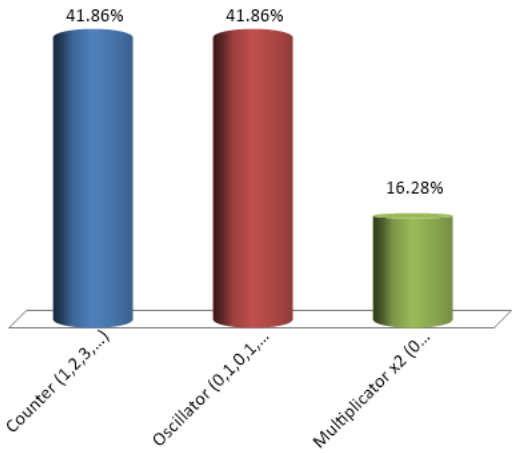
- To keep a memory of the activation of the neuron
- To propagate the information across output neurons

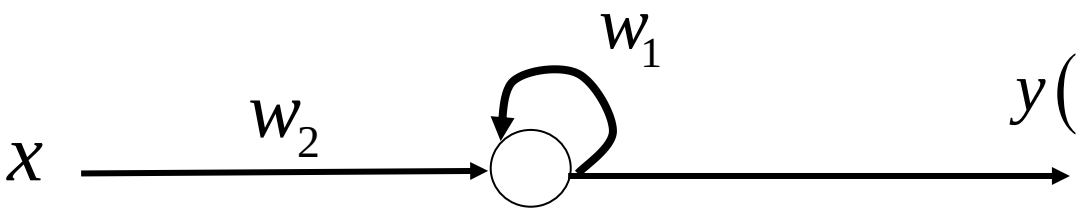


Continuous model: $\dot{y} = -\frac{1}{\tau} y + \int x dt$

What type of function can we embed in a single recurrent neuron?

A. Counter (1,2,3,…)
B. Oscillator (0,1,0,1, …)
C. Multiplicator x2 (0 ; 2)





$$y(t) = -w_1 y(t-1) + w_2 x(t)$$

Binary input 1/0

What type of function can we embed in a single recurrent neuron?

A. Counter (1,2,3,…) $w_1 = -1$ and $w_2 = 1$

B. Oscillator (0,1,0,1, …) $w_1 = w_2 = 1$

C. Multiplicator x2 (0 ; 2) Not possible

Binary input 1/0