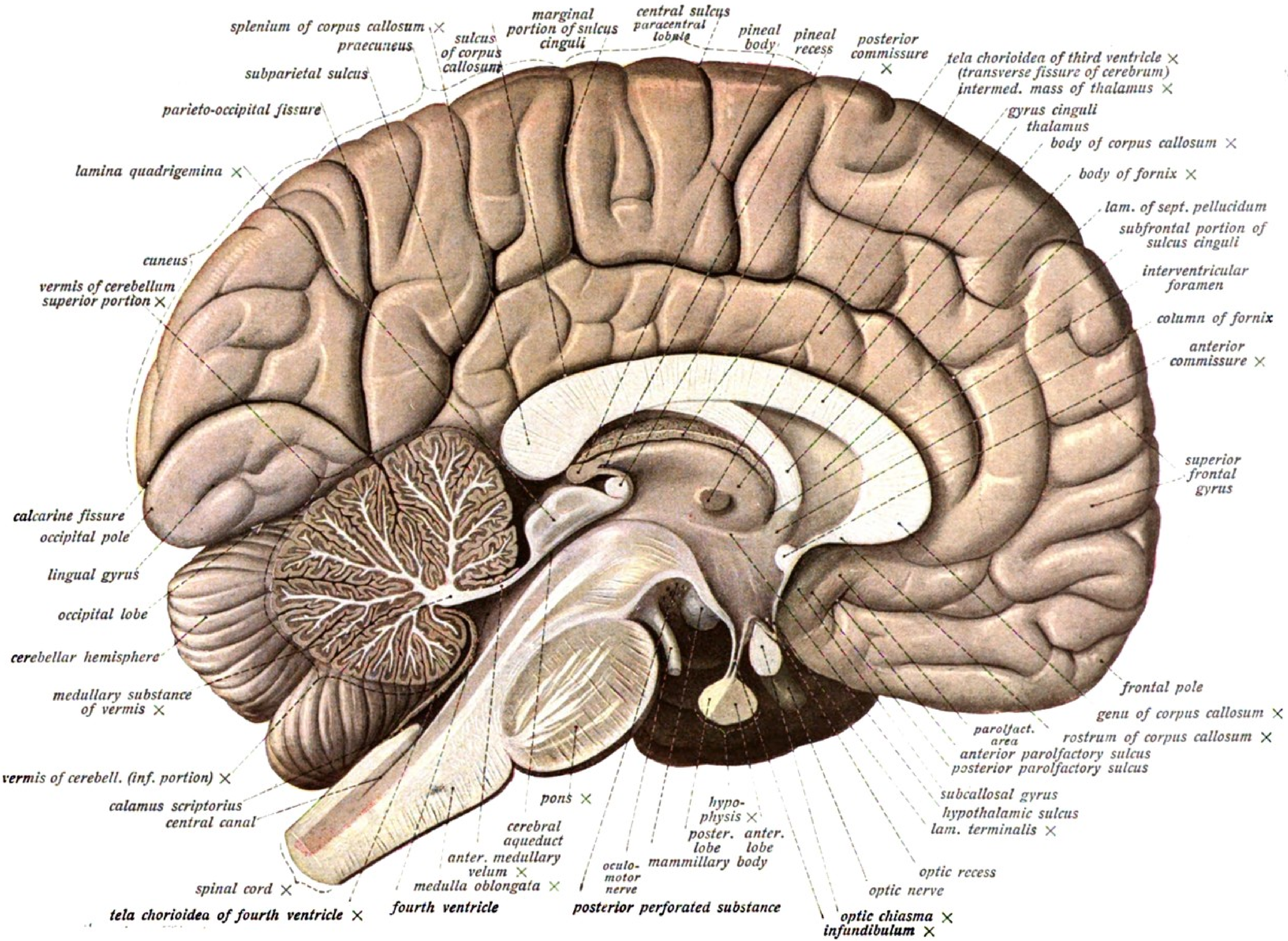
P08820 Introduction to Machine Learning

Week 5:
Neural Networks

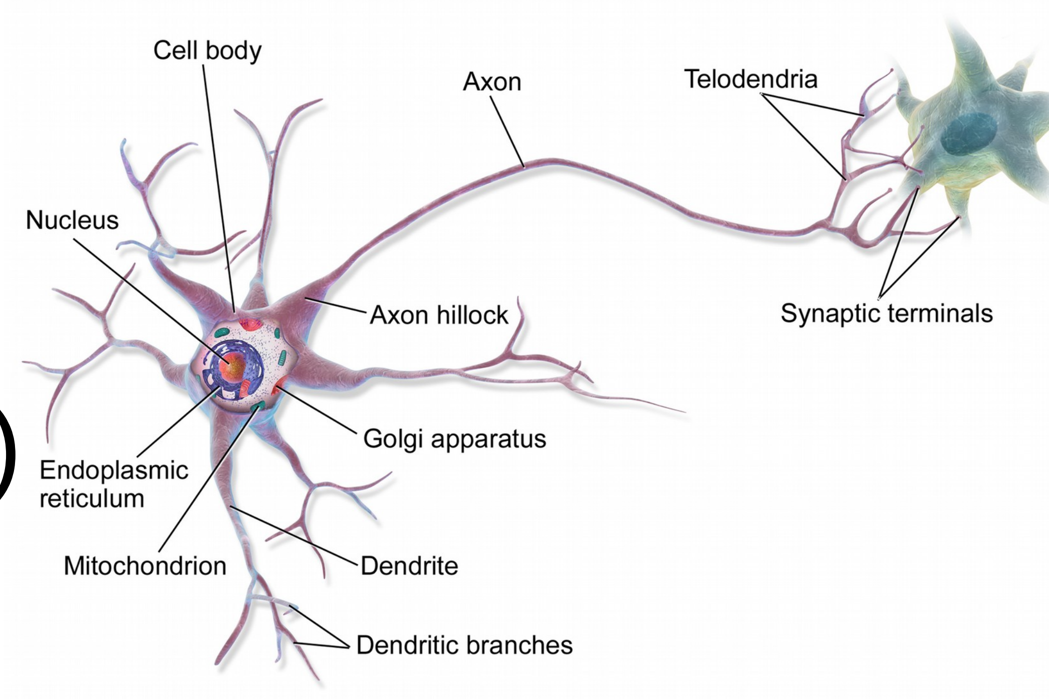
Matthias Rolf

mrolf@brookes.ac.uk


splenium of corpus callosum ×
praecuneus
subparietal sulcus
parieto-occipital fissure
lamina quadrigemina ×
cuneus
vermis of cerebellum
superior portion ×
calcarine fissure
occipital pole
lingual gyrus
occipital lobe
cerebellar hemisphere
medullary substance
of vermis ×
vermis of cerebell. (inf. portion) ×
calamus scriptorius
central canal
spinal cord ×
tela chorioidea of fourth ventricle ×
sulcus
of corpus
callosum
marginal
portion of sulcus
cinguli
central sulcus
paracentral
lobule
pineal
body
pineal
recess
posterior
commissure
×
tela chorioidea of third ventricle ×
(transverse fissure of cerebrum)
intermed. mass of thalamus ×
gyrus cinguli
thalamus
body of corpus callosum ×
body of fornix ×
lam. of sept. pellucidum
subfrontal portion of
sulcus cinguli
interventricular
foramen
column of fornix
anterior
commissure ×
superior
frontal
gyrus
frontal pole
genu of corpus callosum ×
parolfact.
area
rostrum of corpus callosum ×
anterior parolfactory sulcus
posterior parolfactory sulcus
subcallosal gyrus
hypothalamic sulcus
lam. terminalis ×
optic recess
optic nerve
optic chiasma ×
infundibulum ×
pons ×
cerebral
aqueduct
anter. medullary
velum ×
medulla oblongata ×
fourth ventricle
oculo-
motor
nerve
posterior perforated substance
hypo-
physis ×
poster.
lobe
anter.
lobe
mammillary body

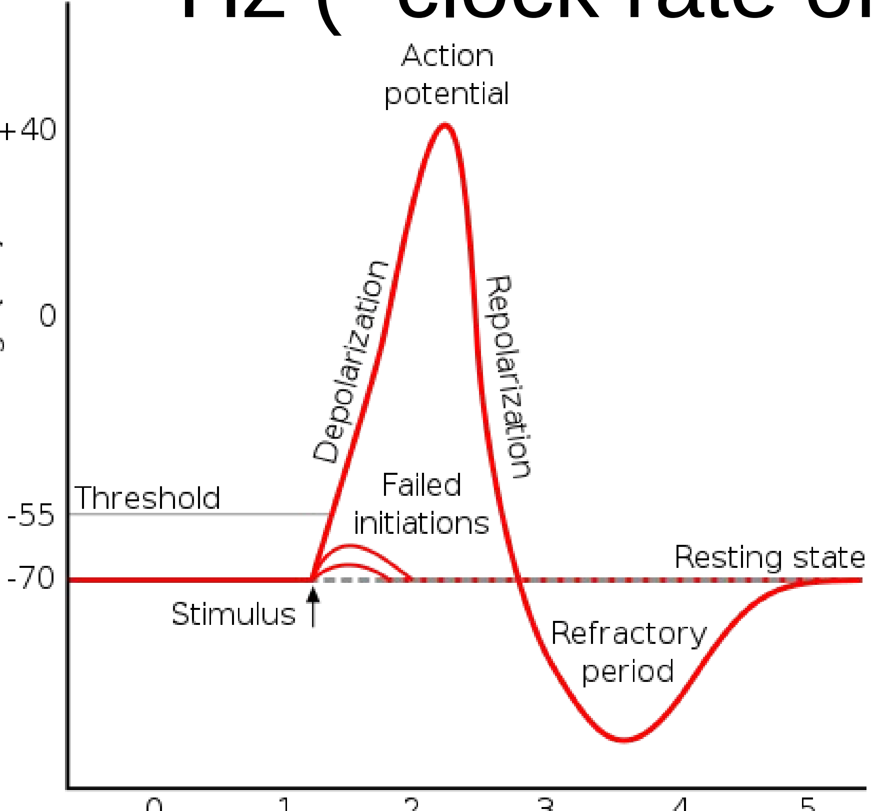
Neurons

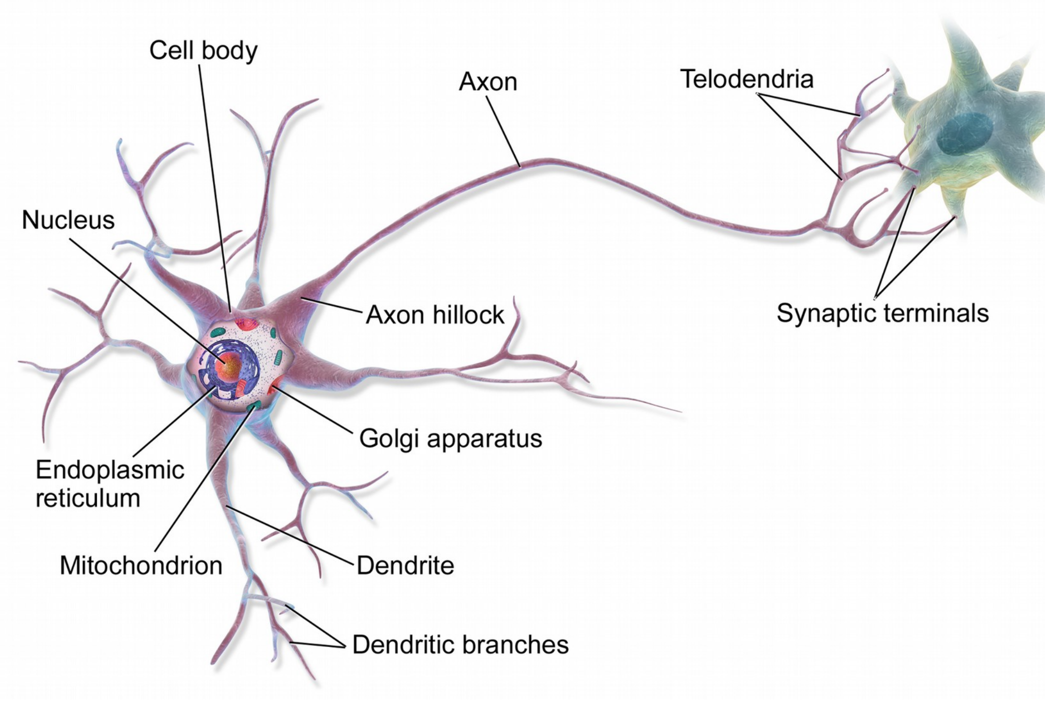
- Relatively simple information processing units – but ca 100 billion
- Collect electric charge through dendrites
- Release spike of electric charge as a result through axon (“nerve fiber”)



Neurons

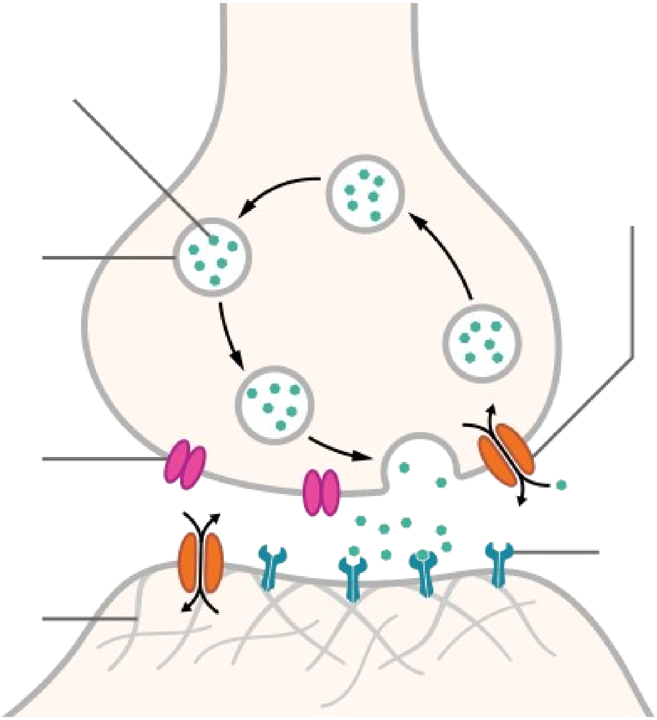
- A spike (action potential) needs a threshold influx
- Refractory time limits spike frequency to 200 Hz (~clock rate of the brain)

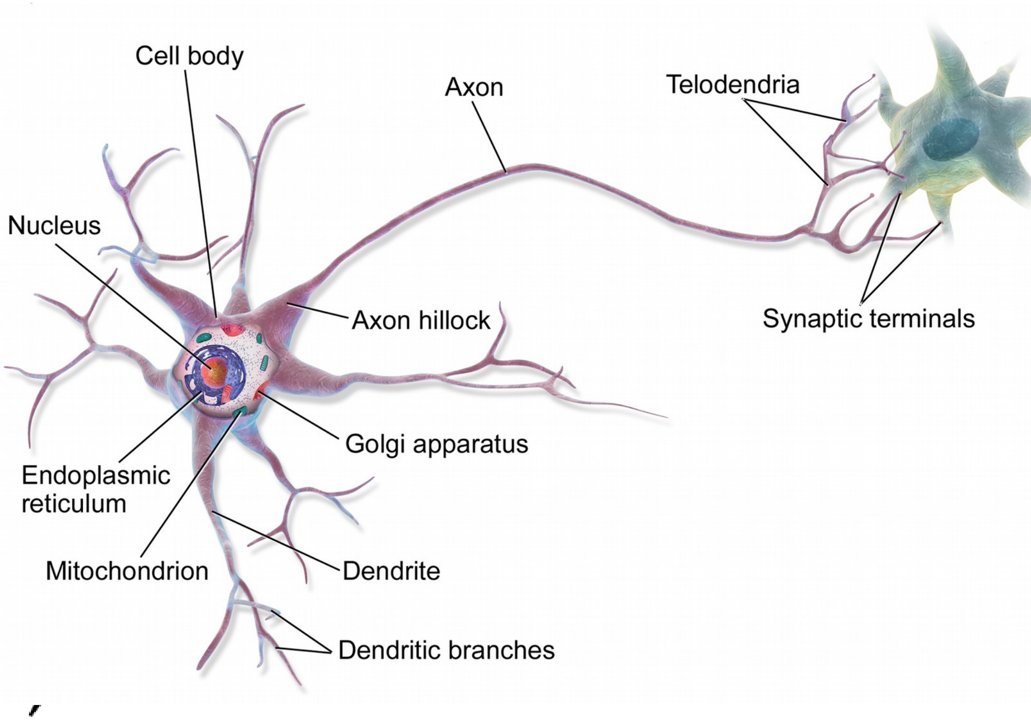




Synapses

- A synapse allows the flow of charge from one neuron to another
- Can “excite” or “inhibit” action potentials
- Synapses change their strength with “experience” (and even grow newly) – “synaptic plasticity”
- Result: changed computation (learning!)

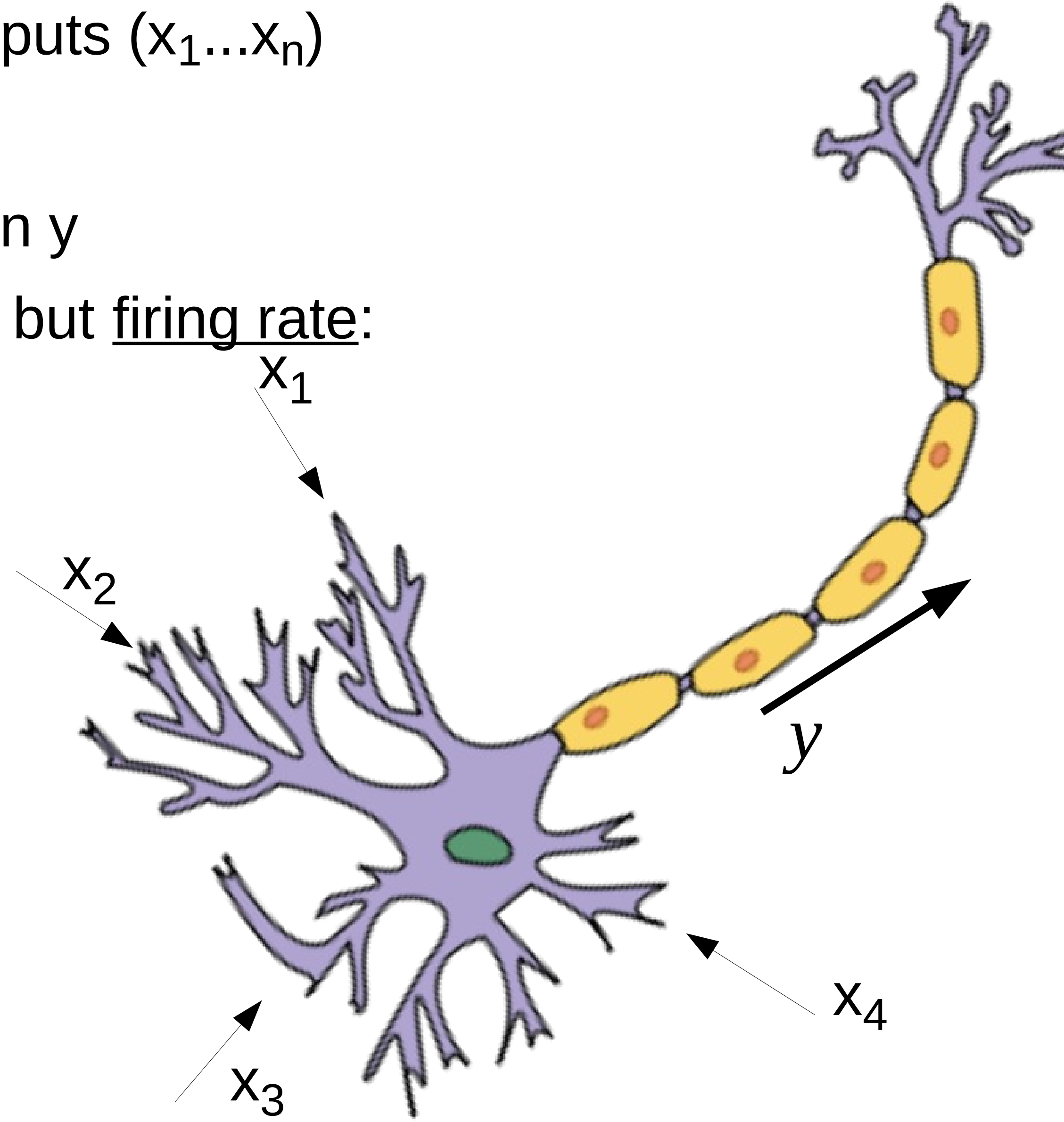




A Simple Model

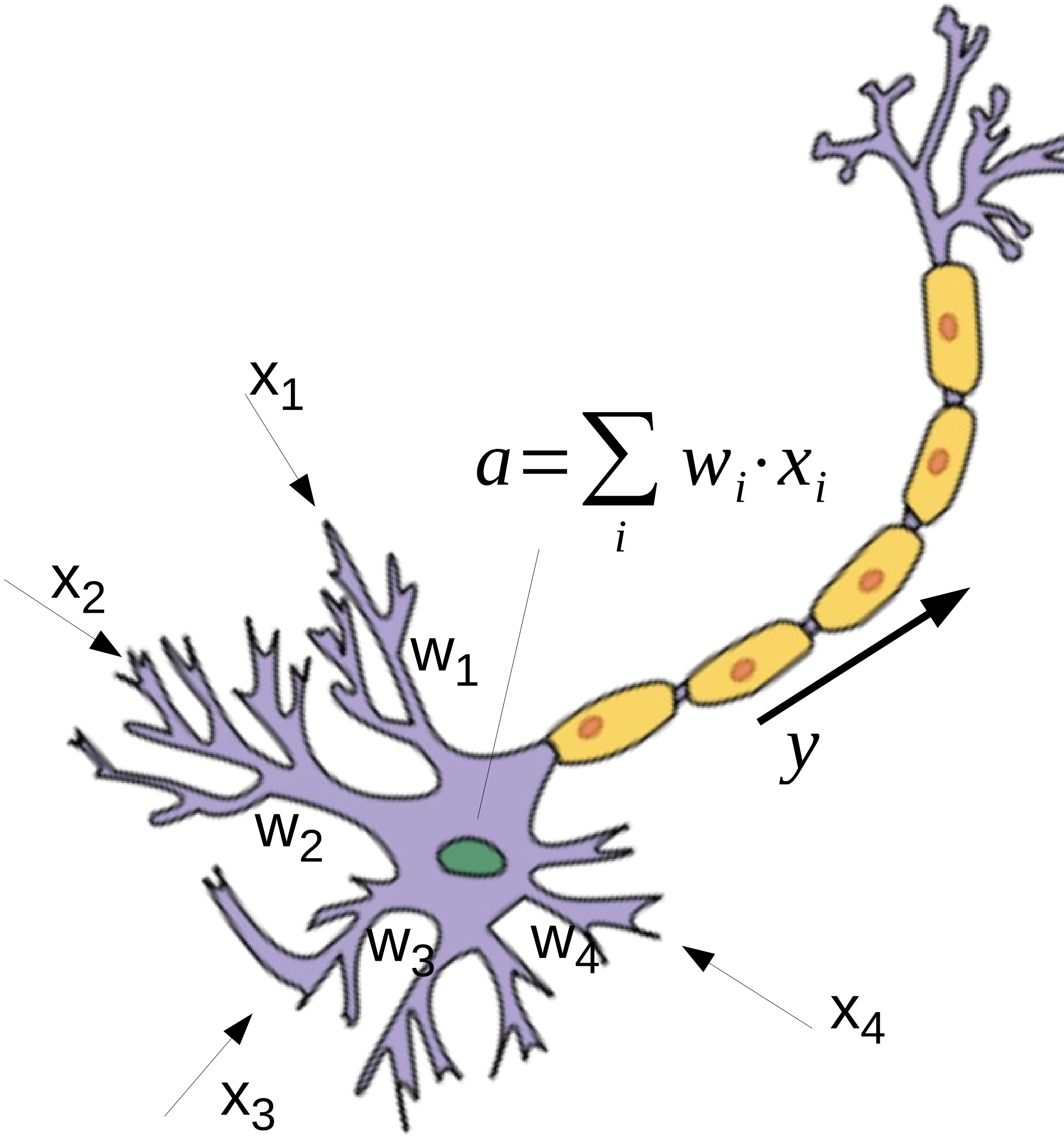
- A list of input neurons with outputs ($x_1 \ldots x_n$)

  → Inputs to this neuron
- Output activation of this neuron y
- Do not code individual spikes, but firing rate:
  - Zero = no action potentials at all
  - One = max. firing frequency



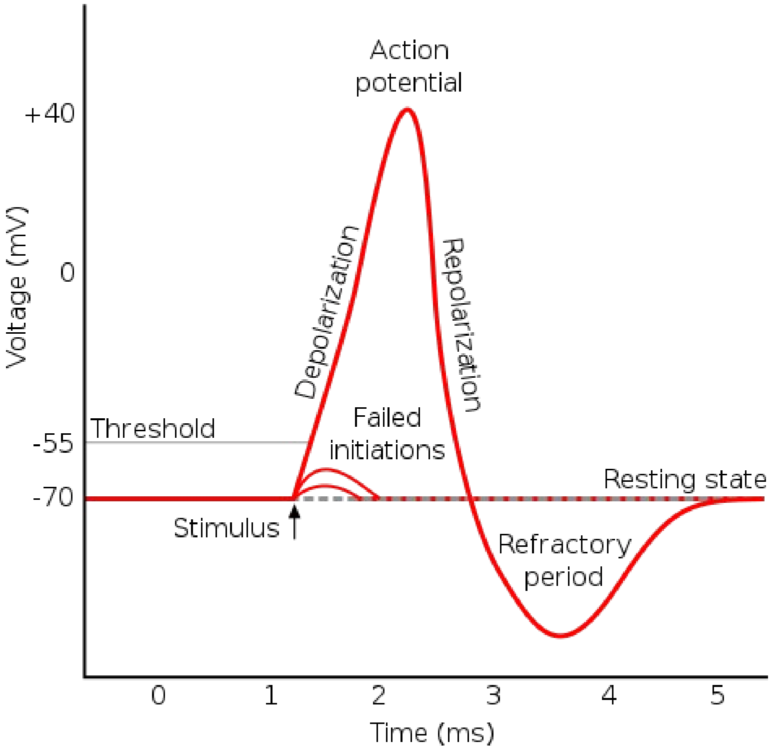
A Simple Model

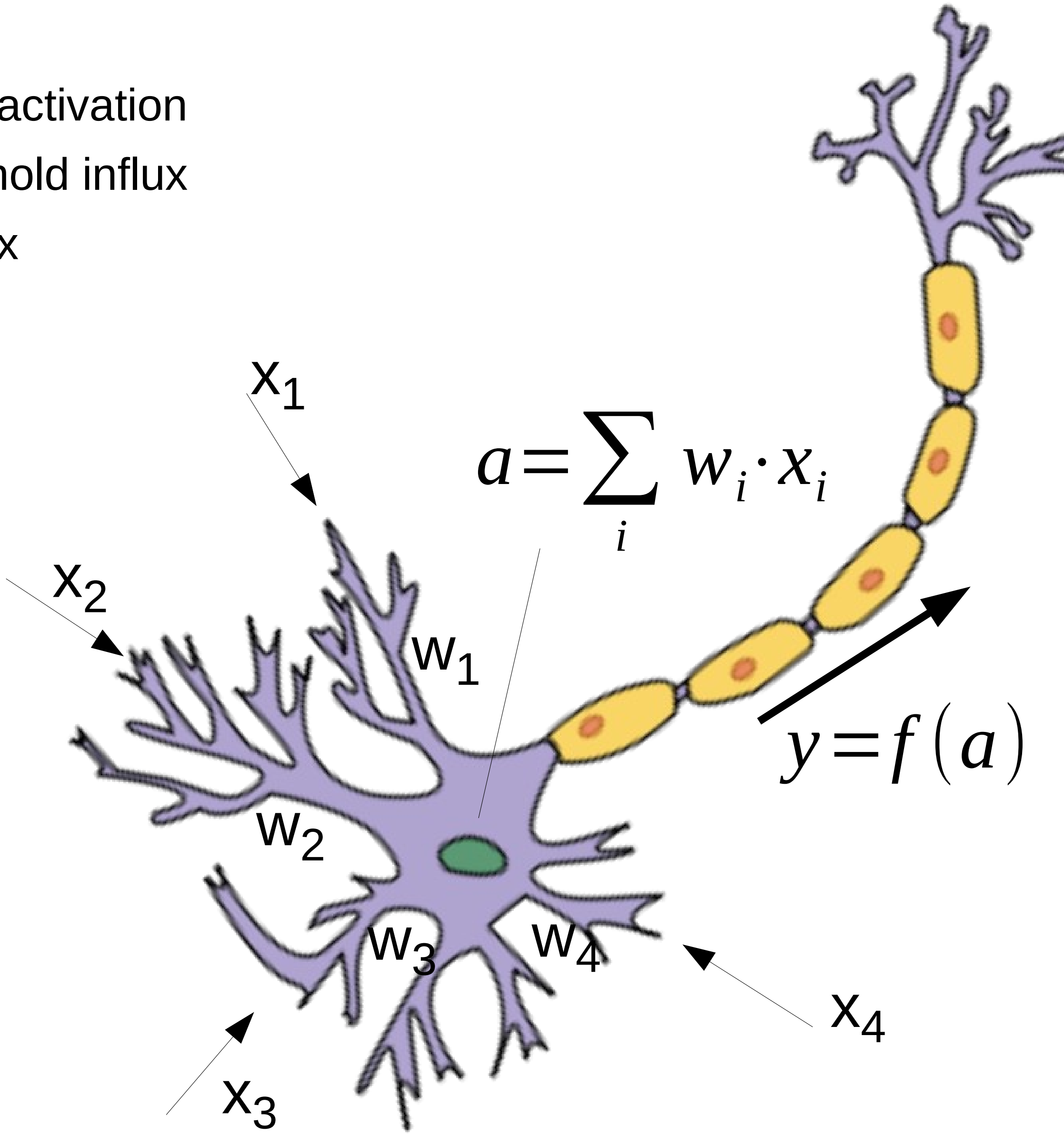
- Synaptic weights ($w_1 ... w_n$):
  - Positive w = excitatory
  - Negative w = inhibitory
  - Large = strong effect
  - Zero = no effect
- Inflow from neuron i
  - Modulated input $w_i \cdot a_i$
- Net activation a
  - Sum of inflows



A Simple Model

- Activation function f
  - Determines output firing rate from net activation
  - Low to no output (0.0) for below threshold influx
  - Saturated firing rate (1.0) for high influx
  - Monotonically increasing in between
- Several candidates (design choice)

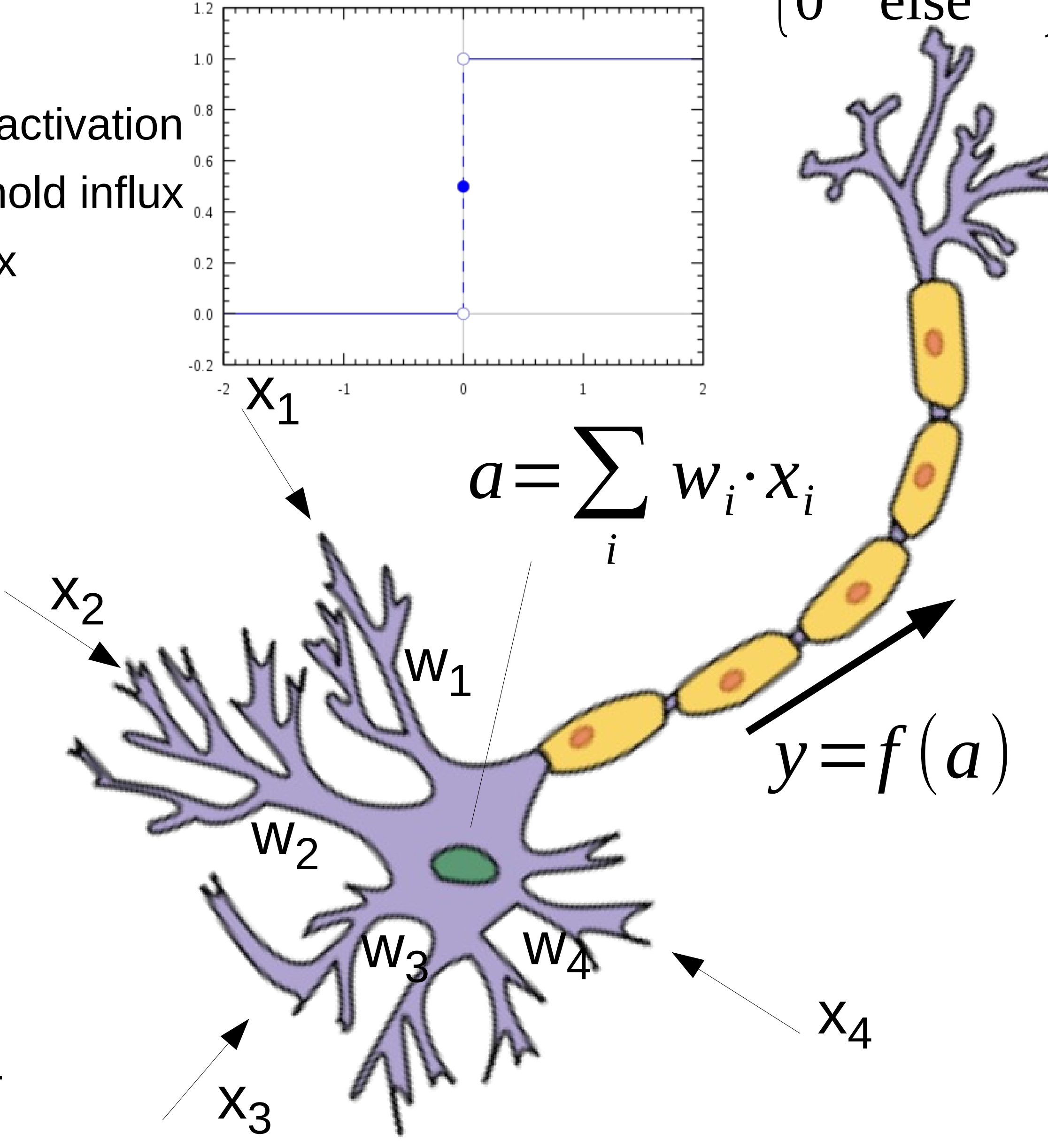




A Simple Model

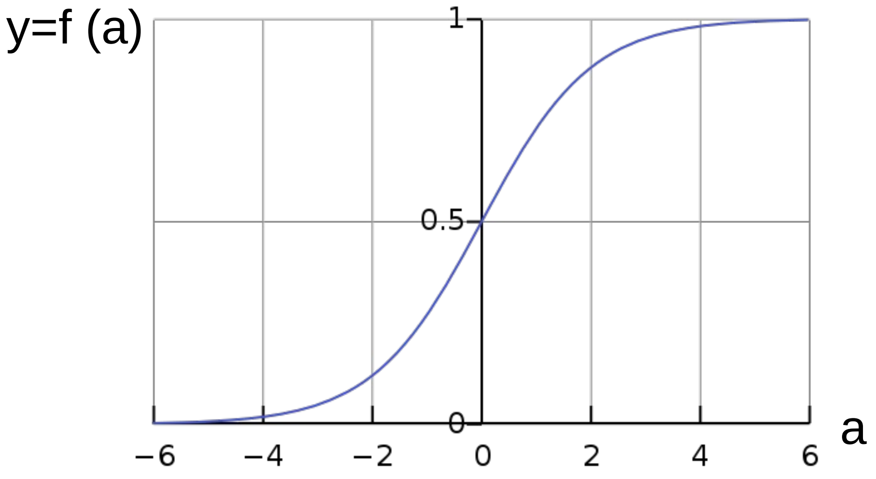
- Activation function f
  - Determines output firing rate from net activation
  - Low to no output (0.0) for below threshold influx
  - Saturated firing rate (1.0) for high influx
  - Monotonically increasing in between
- Several candidates (design choice)

Step function: $f(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{else} \end{cases}$



$$a = \sum_i w_i \cdot x_i$$

$$y = f(a)$$



Sigmoid function: $f(a) = \frac{1}{1+e^{-a}}$

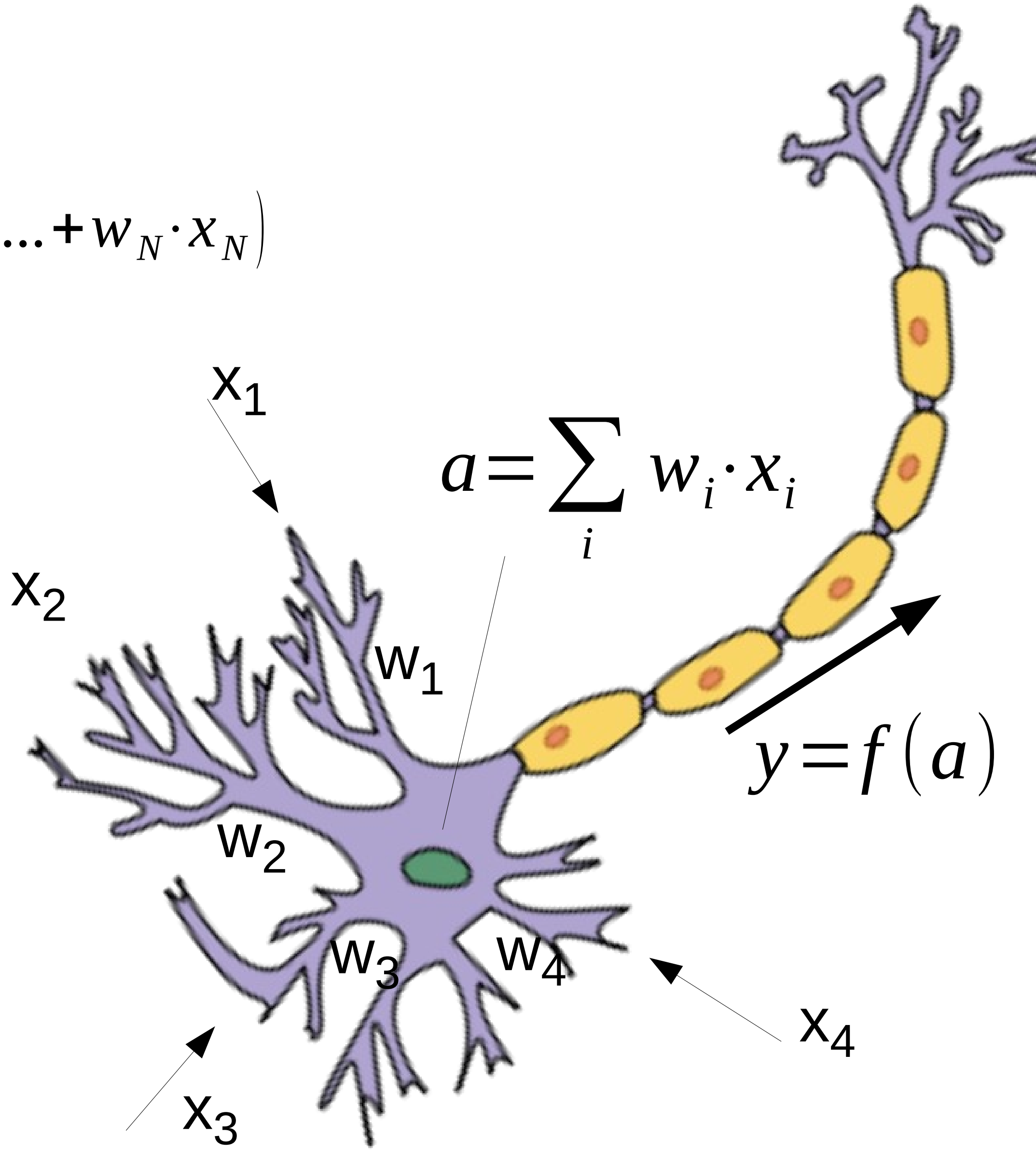
A Simple Model

- In summary:

$$y=f\left(\sum_{i=0}^{N} w_i\cdot x_i\right)=f\left(w_0\cdot x_0+\ldots+w_N\cdot x_N\right)$$

- In vector notion:

$$y=f\left(\begin{pmatrix}w_0\\ w_1\\ \vdots\\ w_2\end{pmatrix}^T\cdot\begin{pmatrix}1\\ x_1\\ \vdots\\ x_2\end{pmatrix}\right)=f\left(\vec{w}^T\cdot\vec{x}\right)$$

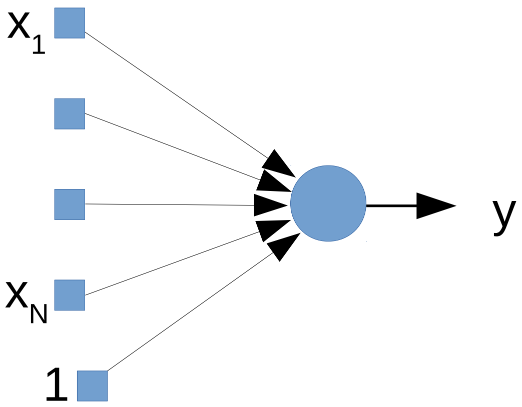


Perceptron

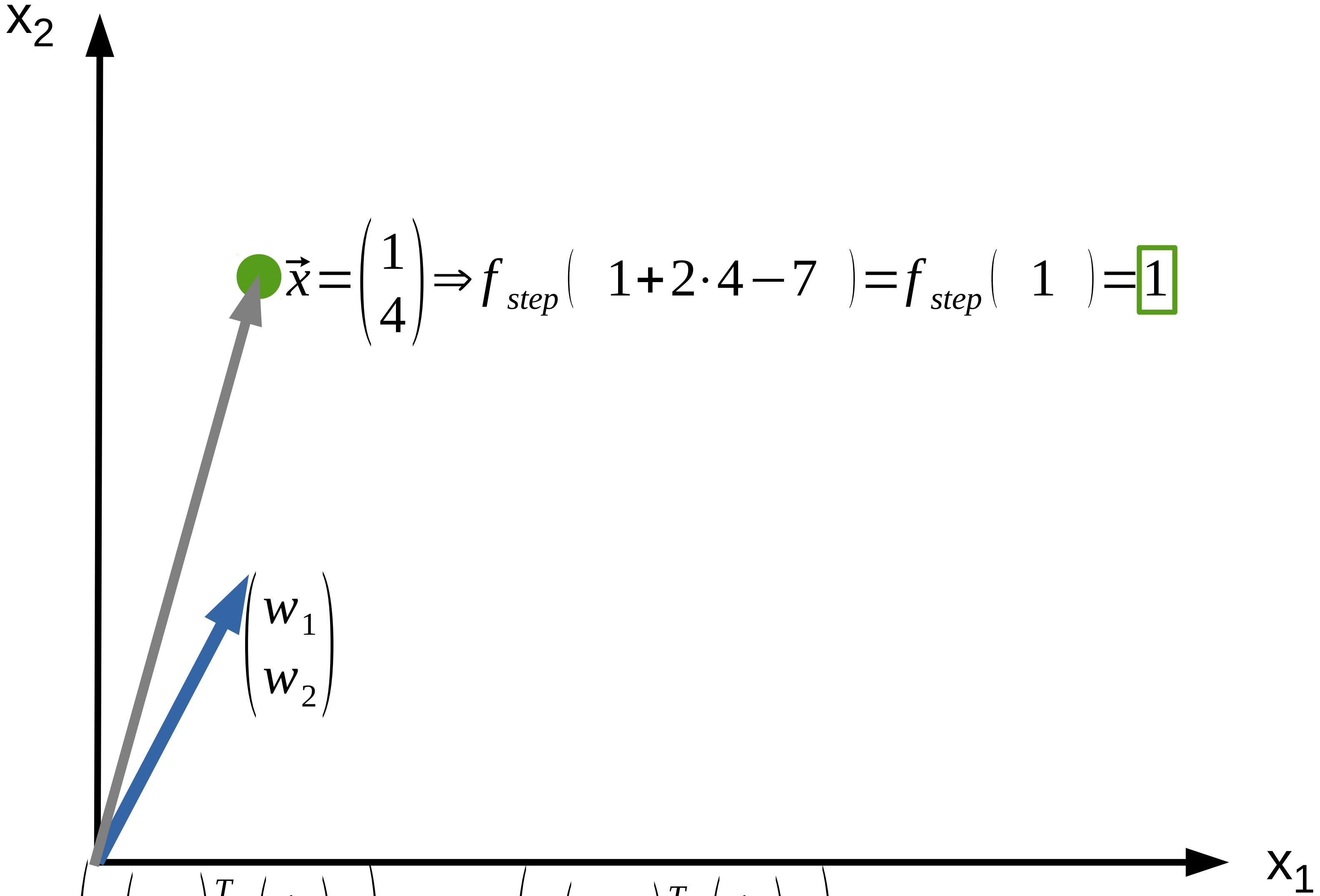
- Invented by Frank Rosenblatt in 1957
- A single neuron with step function
- Supervised learning rule!

$$y = f\left(\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_2 \end{pmatrix}^T \cdot \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_2 \end{pmatrix} \right) = f\left(\vec{w}^T \cdot \vec{x} \right)$$

Step function: $f(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{else} \end{cases}$



Perceptron Decisions

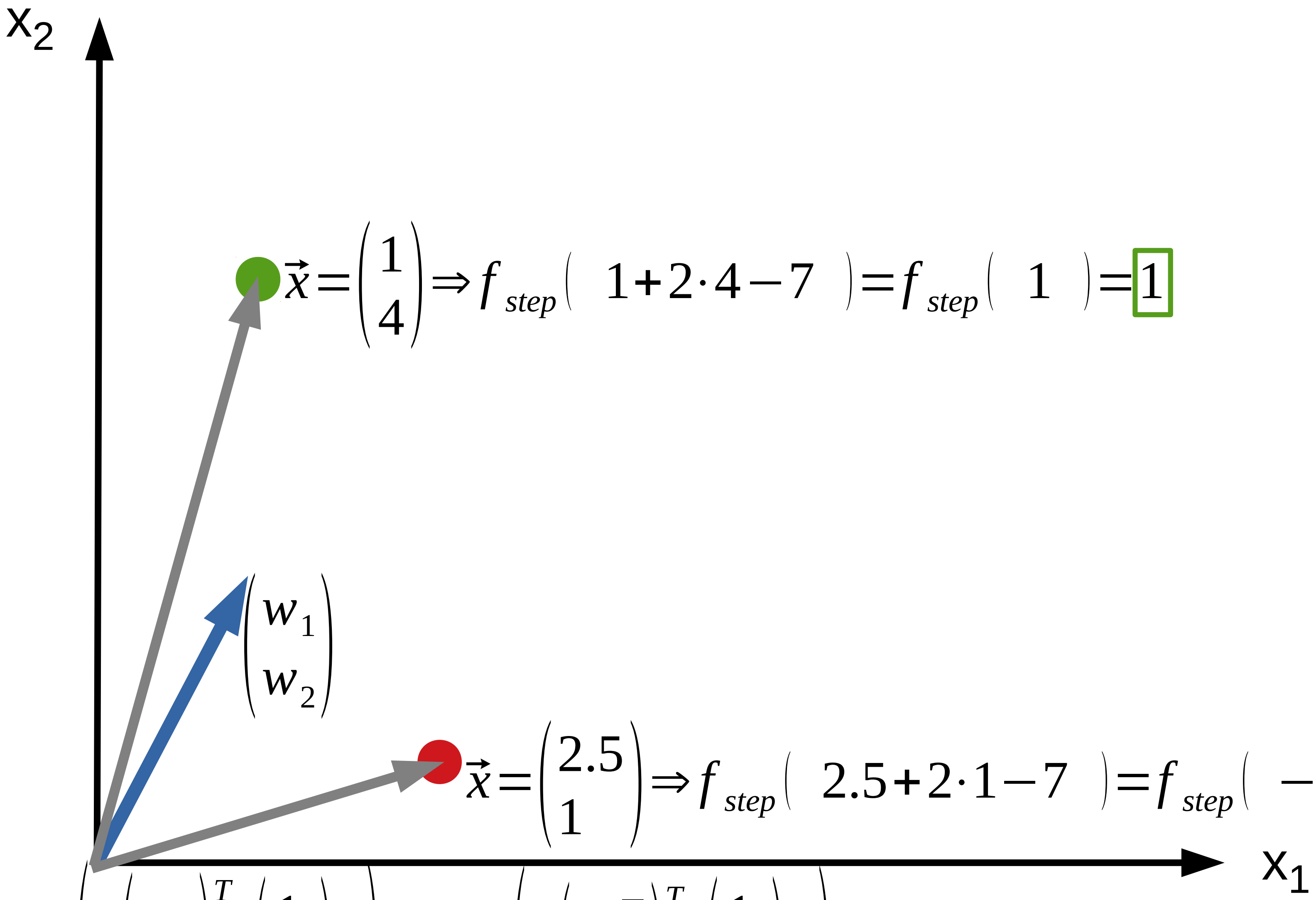


$$\vec{x} = \begin{pmatrix}1\\4\end{pmatrix} \Rightarrow f_{step}(\;1+2\cdot4-7\;) = f_{step}(\;1\;) = \boxed{1}$$

$$\begin{pmatrix}w_1\\w_2\end{pmatrix}$$

$$f_{step}\left(\begin{pmatrix}w_0\\w_1\\w_2\end{pmatrix}^T \cdot \begin{pmatrix}1\\x_1\\x_2\end{pmatrix}\right) = f_{step}\left(\begin{pmatrix}-7\\1\\2\end{pmatrix}^T \cdot \begin{pmatrix}1\\x_1\\x_2\end{pmatrix}\right) = f_{step}(\;x_1+2\cdot x_2-7\;)$$

Perceptron Decisions



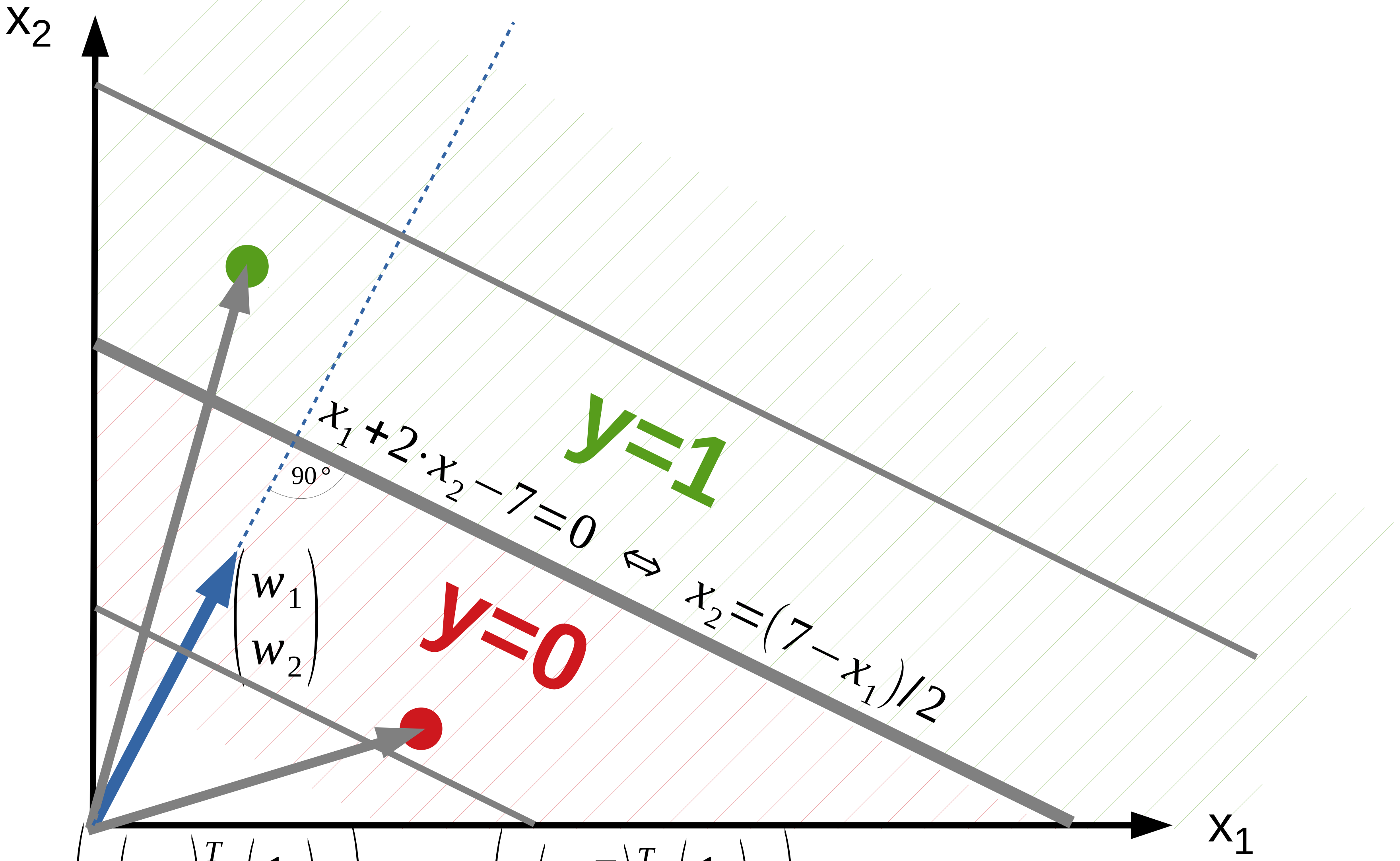
$$\vec{x}=\begin{pmatrix}1\\4\end{pmatrix}\Rightarrow f_{step}(\;1+2\cdot4-7\;)=f_{step}(\;1\;)=\boxed{1}$$

$$\begin{pmatrix}w_1\\w_2\end{pmatrix}$$

$$\vec{x}=\begin{pmatrix}2.5\\1\end{pmatrix}\Rightarrow f_{step}(\;2.5+2\cdot1-7\;)=f_{step}(\;-2.5\;)=\boxed{0}$$

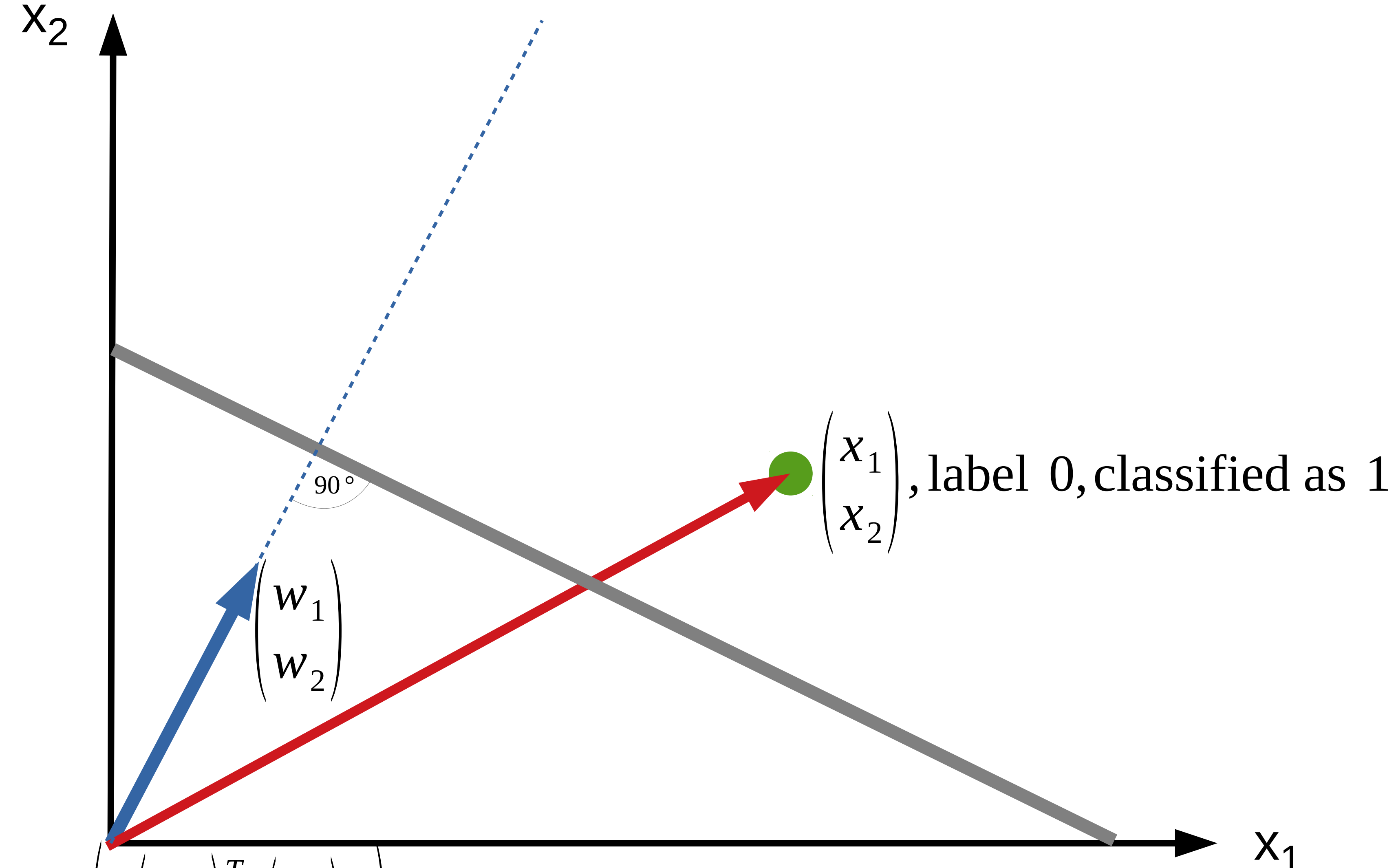
$$f_{step}\left(\begin{pmatrix}w_0\\w_1\\w_2\end{pmatrix}^T\cdot\begin{pmatrix}1\\x_1\\x_2\end{pmatrix}\right)=f_{step}\left(\begin{pmatrix}-7\\1\\2\end{pmatrix}^T\cdot\begin{pmatrix}1\\x_1\\x_2\end{pmatrix}\right)=f_{step}(\;x_1+2\cdot x_2-7\;)$$

Perceptron Decisions


x_2
x_1
$x_1+2\cdot x_2-7=0 \Leftrightarrow x_2=(7-x_1)/2$
$y=1$
$y=0$
$90°$
$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$

$$f_{step}\left(\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}^T \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}\right)=f_{step}\left(\begin{pmatrix} -7 \\ 1 \\ 2 \end{pmatrix}^T \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}\right)=f_{step}\left(x_1+2\cdot x_2-7\right)$$

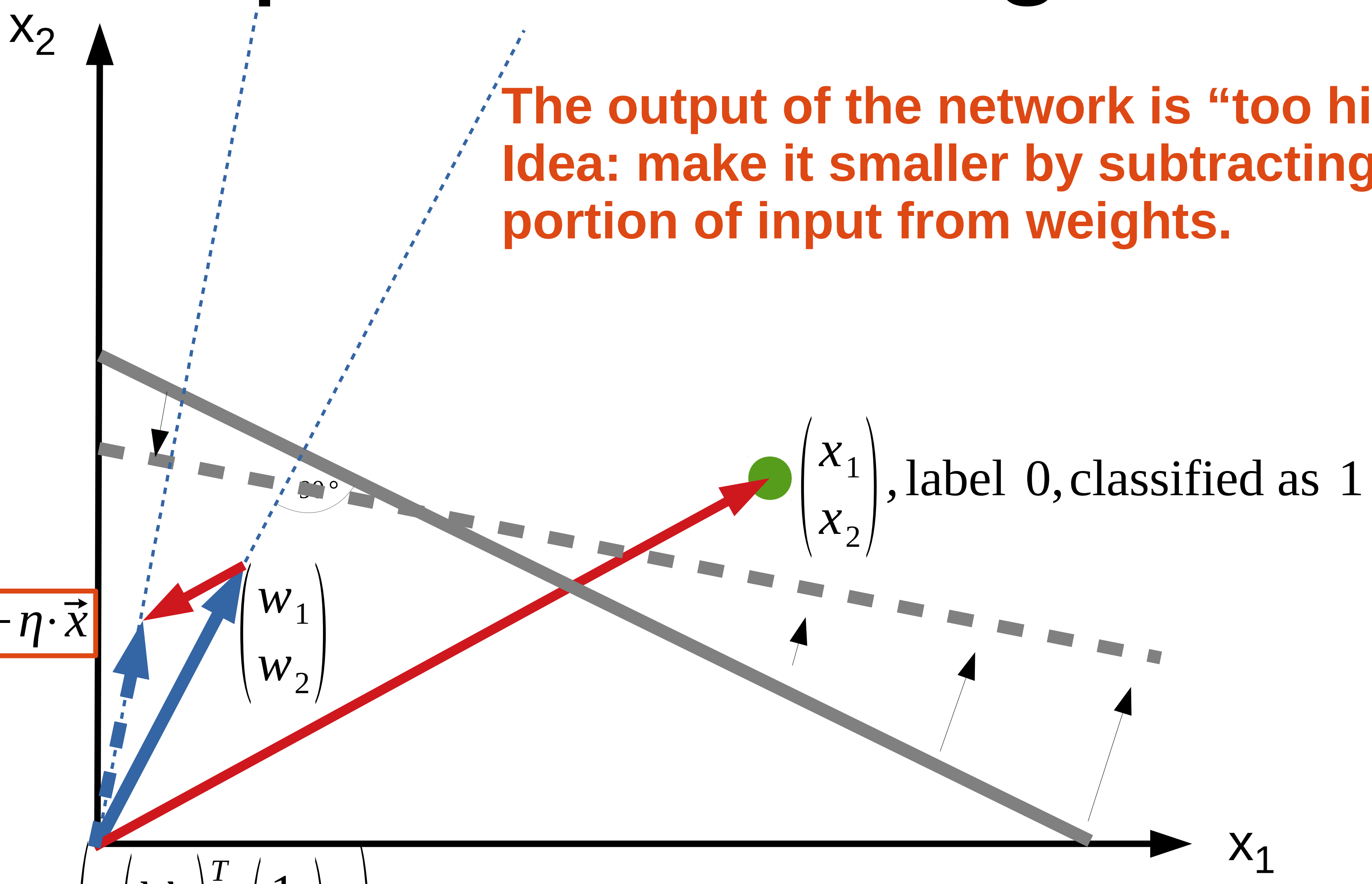
Perceptron Learning Rule


x_2
x_1
90 °
$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$
$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, label 0, classified as 1

$$f_{step}\left(\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}^T \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}\right)$$

Perceptron Learning Rule

The output of the network is “too high”. Idea: make it smaller by subtracting portion of input from weights.



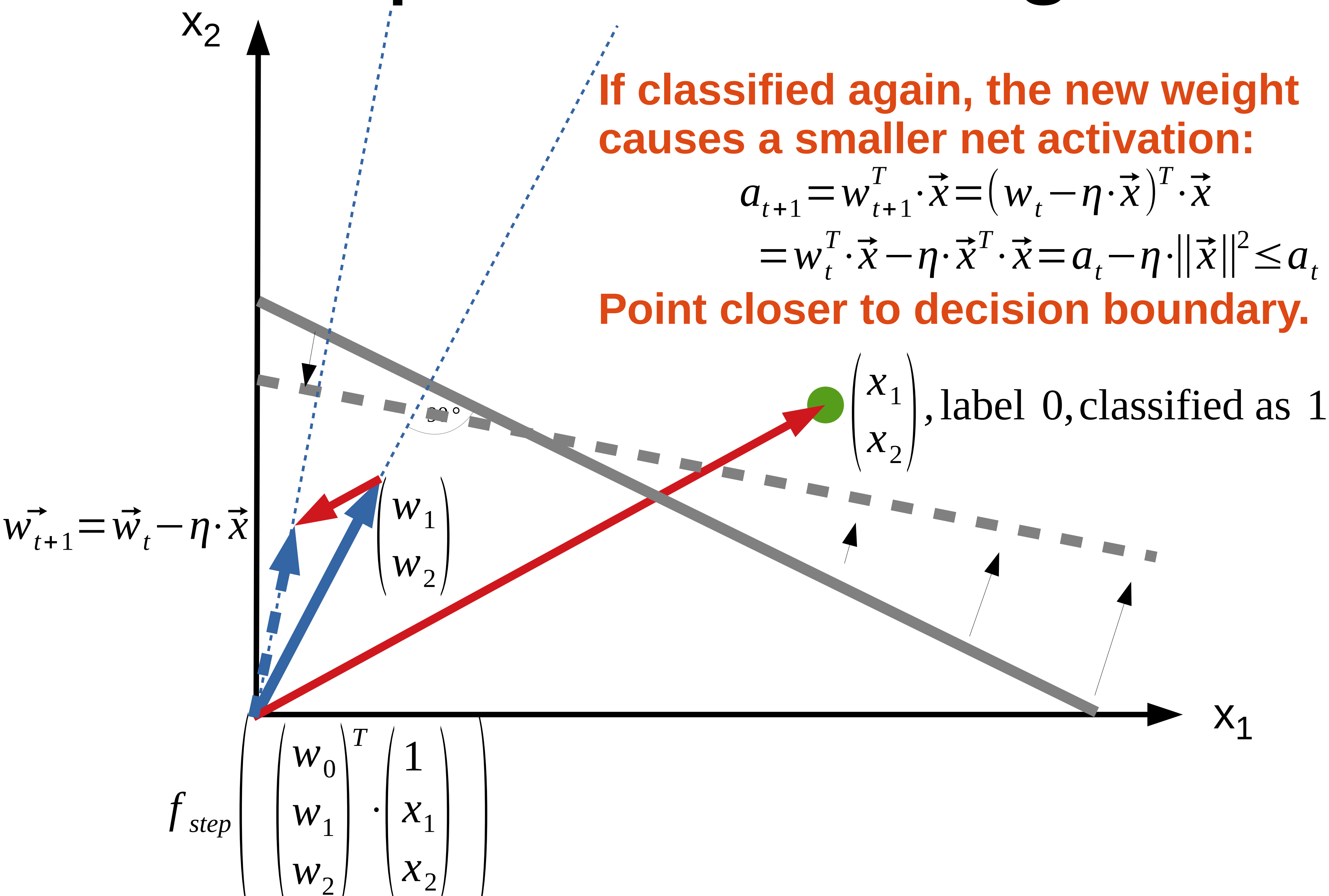
$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, label 0, classified as 1

$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$

$$\vec{w}_{t+1} = \vec{w}_t - \eta \cdot \vec{x}$$

$$f_{step}\left(\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}^T \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \right)$$

Perceptron Learning Rule



x_2
x_1
$\vec{w}_{t+1} = \vec{w}_t - \eta\cdot\vec{x}$
$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$
$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, label 0, classified as 1
90°
$f_{step}\left(\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}^T \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}\right)$

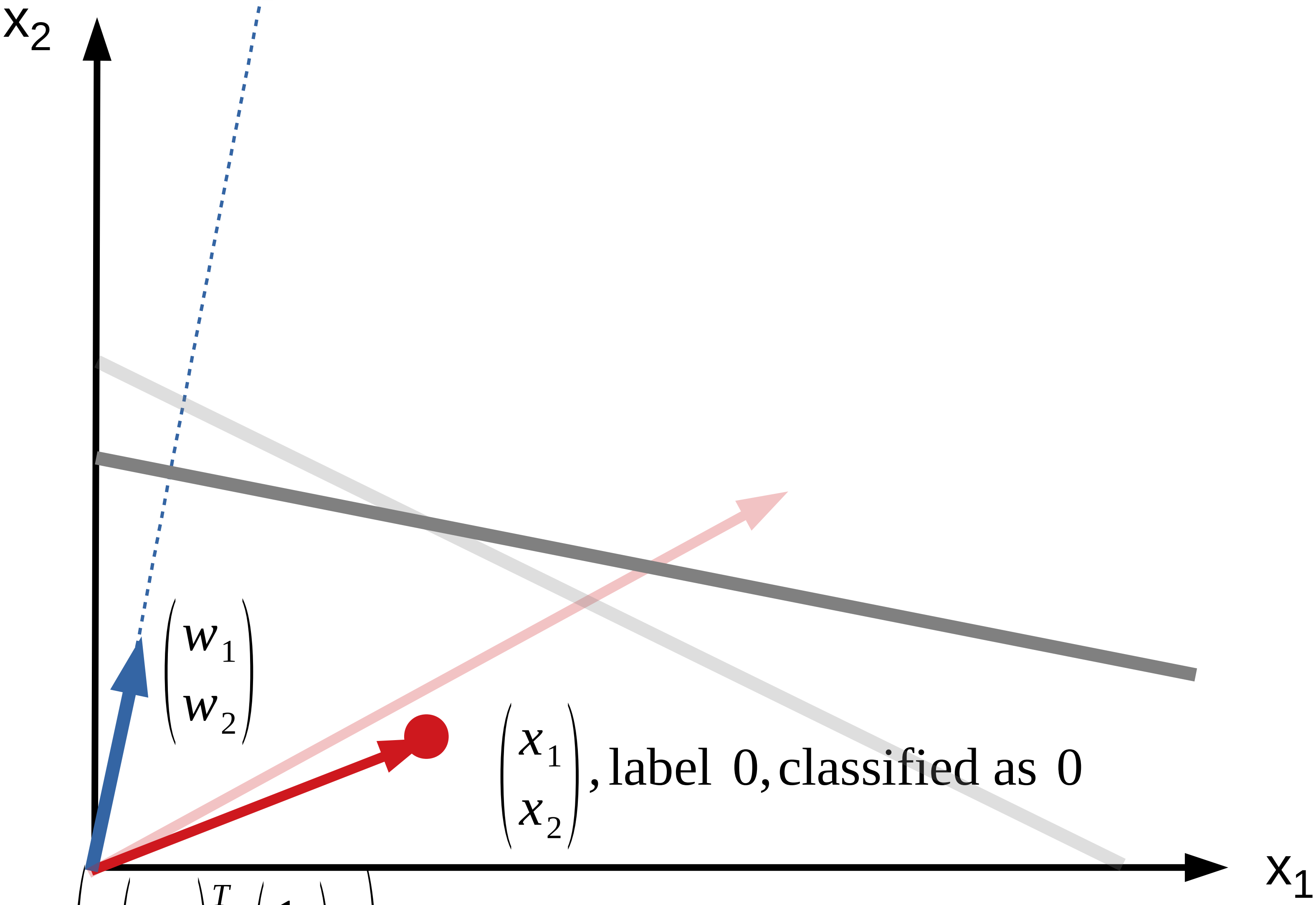
If classified again, the new weight causes a smaller net activation:

$$a_{t+1} = w_{t+1}^T \cdot \vec{x} = (w_t - \eta\cdot\vec{x})^T \cdot \vec{x}$$

$$= w_t^T \cdot \vec{x} - \eta\cdot\vec{x}^T \cdot \vec{x} = a_t - \eta\cdot\|\vec{x}\|^2 \le a_t$$

Point closer to decision boundary.

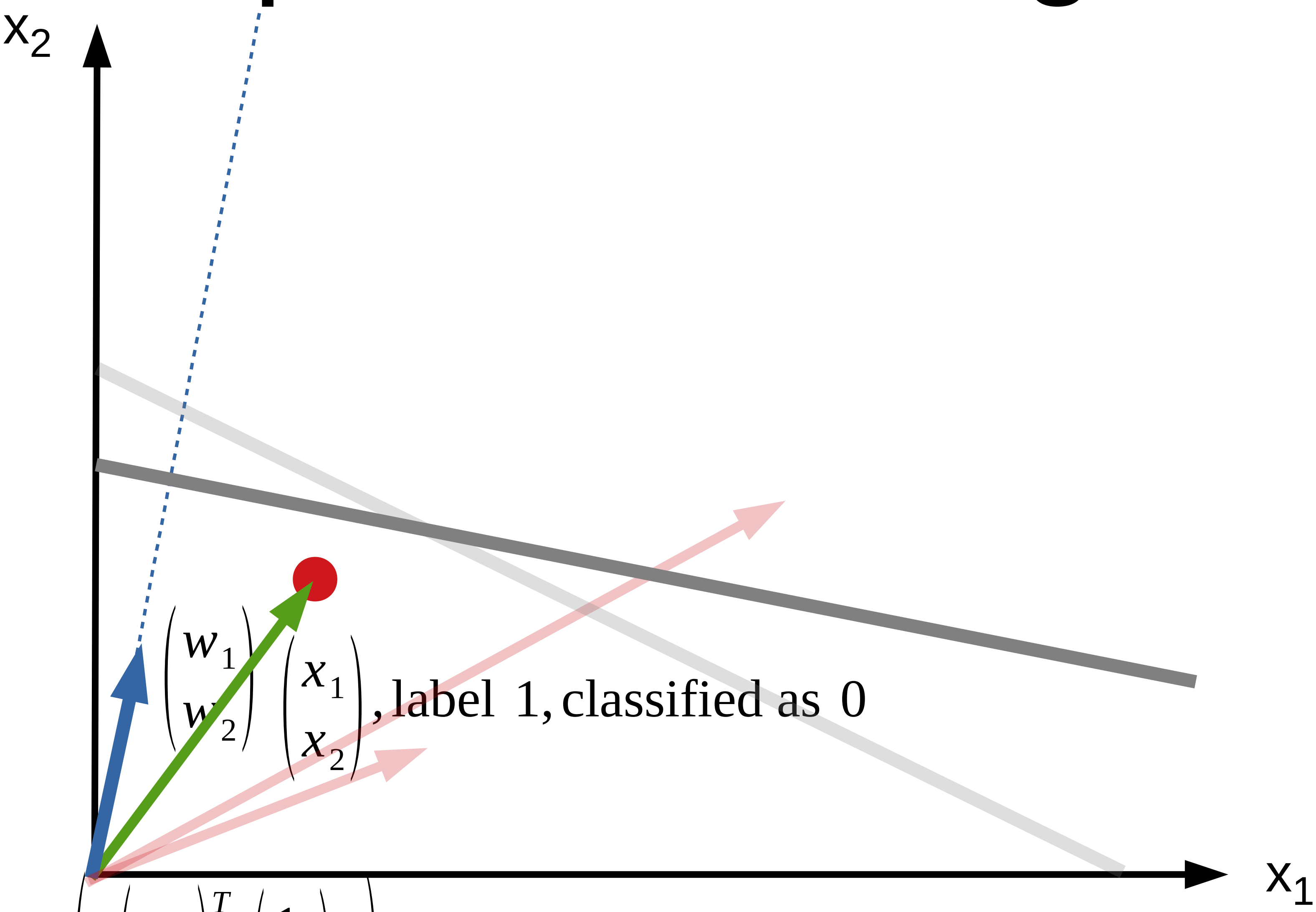
Perceptron Learning Rule



x_2
$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$
$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, label 0, classified as 0
x_1

$$f_{step}\left(\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}^T \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}\right)$$

Perceptron Learning Rule



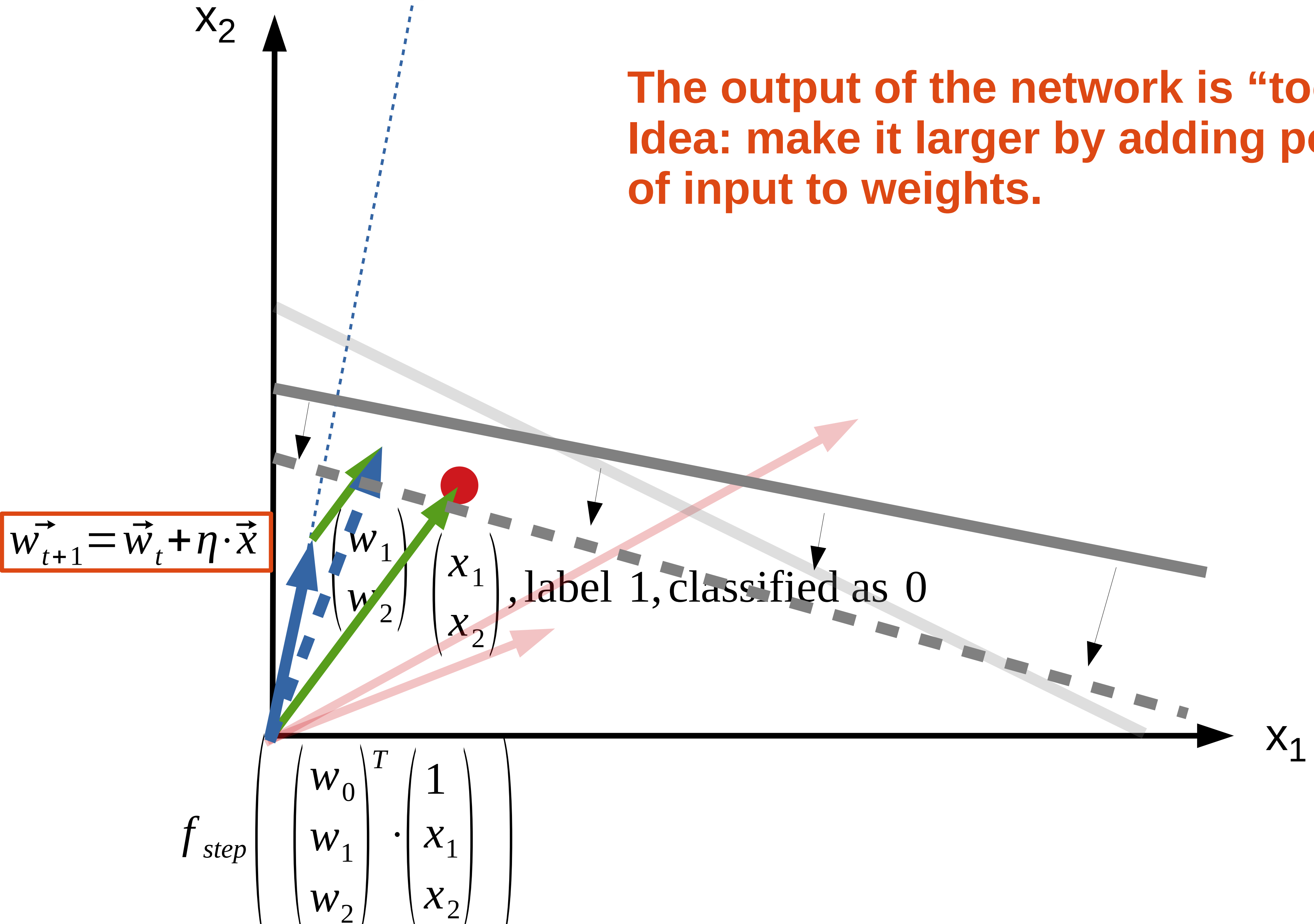
$\begin{pmatrix}w_1\\w_2\end{pmatrix}$

$\begin{pmatrix}x_1\\x_2\end{pmatrix}$, label 1, classified as 0

$$f_{step}\left(\begin{pmatrix}w_0\\w_1\\w_2\end{pmatrix}^T \cdot \begin{pmatrix}1\\x_1\\x_2\end{pmatrix}\right)$$

Perceptron Learning Rule

The output of the network is “too low”. Idea: make it larger by adding portion of input to weights.



$$\boxed{\vec{w}_{t+1}=\vec{w}_t+\eta\cdot\vec{x}}$$

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \text{label } 1, \text{classified as } 0$$

$$f_{step}\left(\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}^T \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}\right)$$

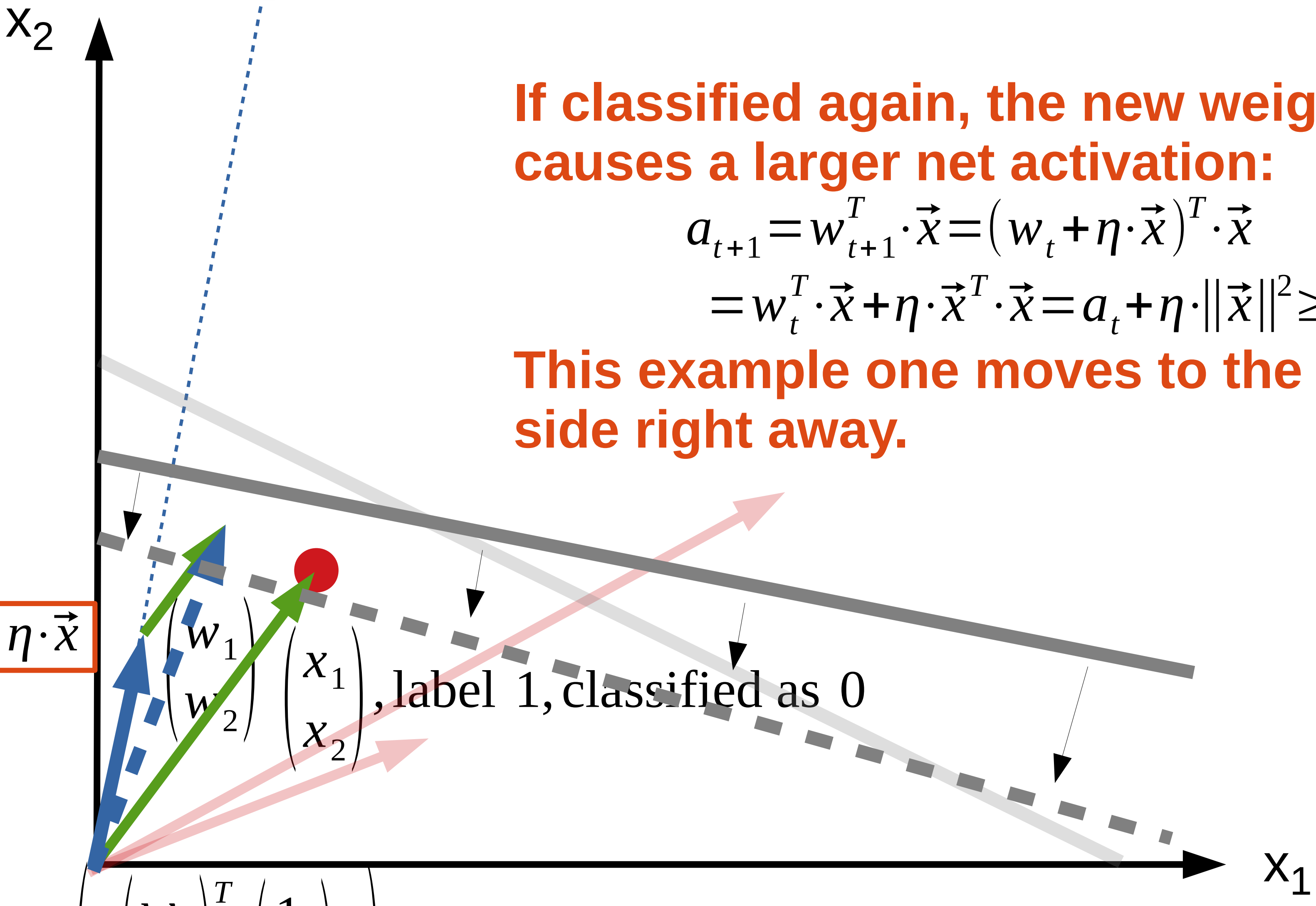
Perceptron Learning Rule

If classified again, the new weight causes a larger net activation:

$$a_{t+1} = w_{t+1}^T \cdot \vec{x} = \left(w_t + \eta \cdot \vec{x}\right)^T \cdot \vec{x}$$

$$= w_t^T \cdot \vec{x} + \eta \cdot \vec{x}^T \cdot \vec{x} = a_t + \eta \cdot \|\vec{x}\|^2 \geq a_t$$

This example one moves to the correct side right away.



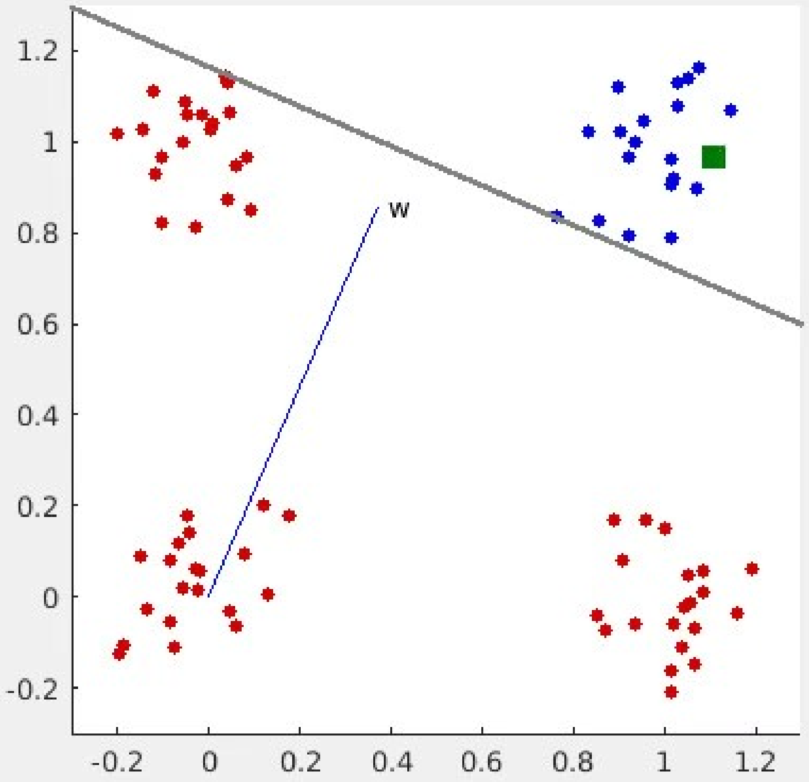
$$\boxed{\vec{w}_{t+1} = \vec{w}_t + \eta \cdot \vec{x}}$$

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \text{label } 1, \text{classified as } 0$$

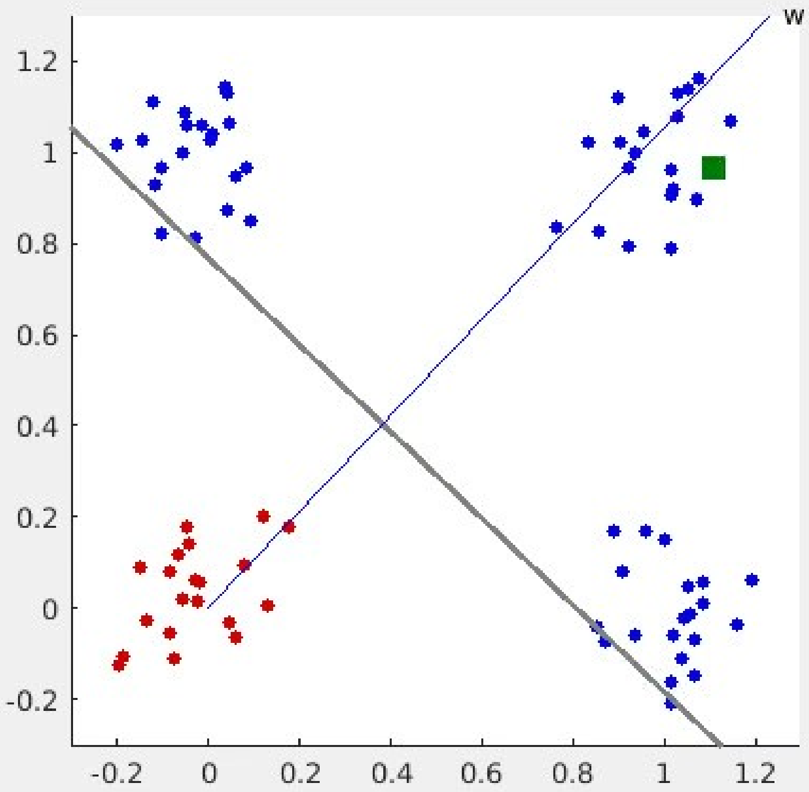
$$f_{step}\left(\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}^T \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \right)$$

Learning Logical Functions

AND



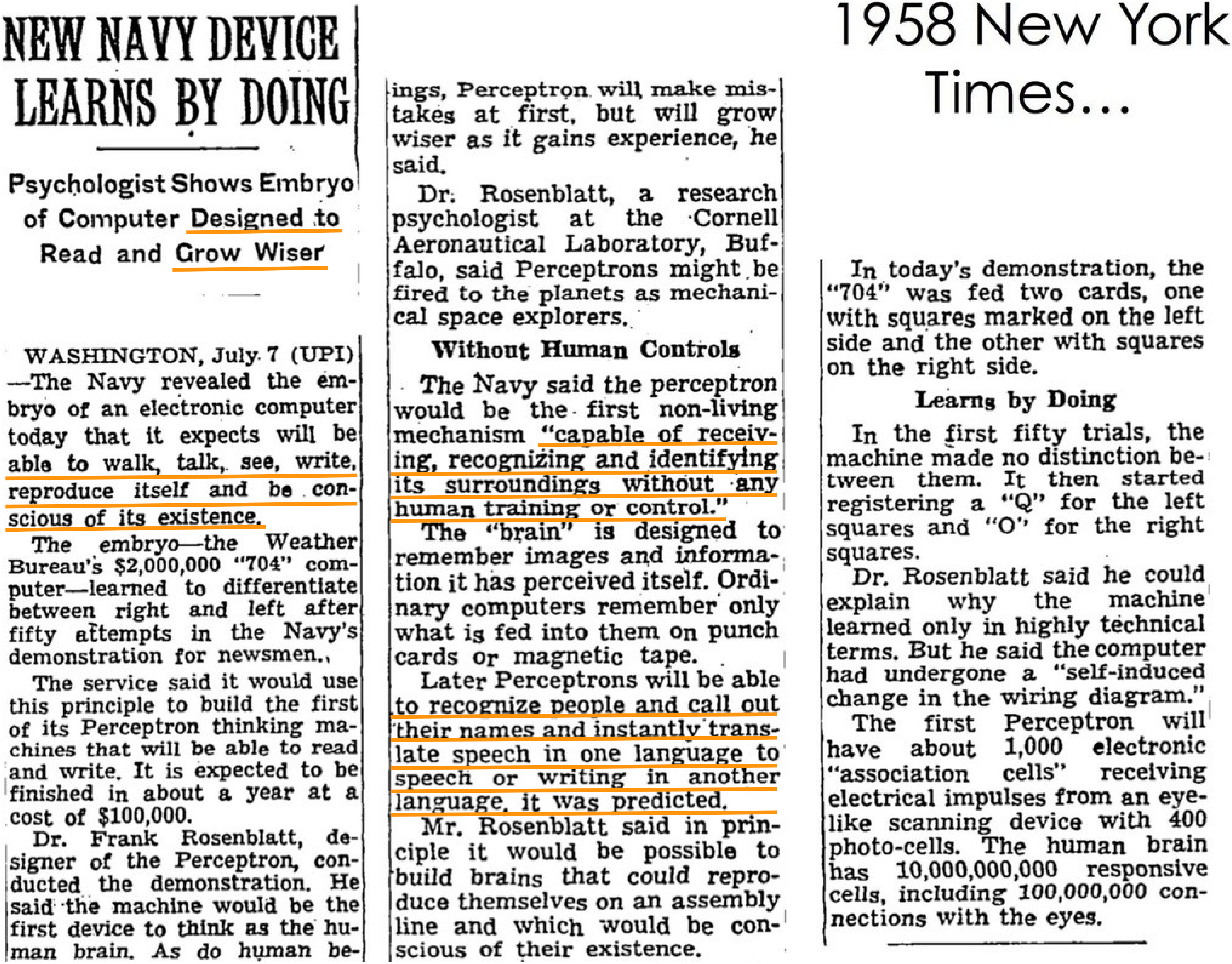
OR



It was by no means previously clear that “neurons” could be “logical”, and even learn to behave so.

This causes a great hype in cognitive science around “connectionism”, where neural networks were used as main metaphor to explain all intelligence

1958 New York Times...



NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

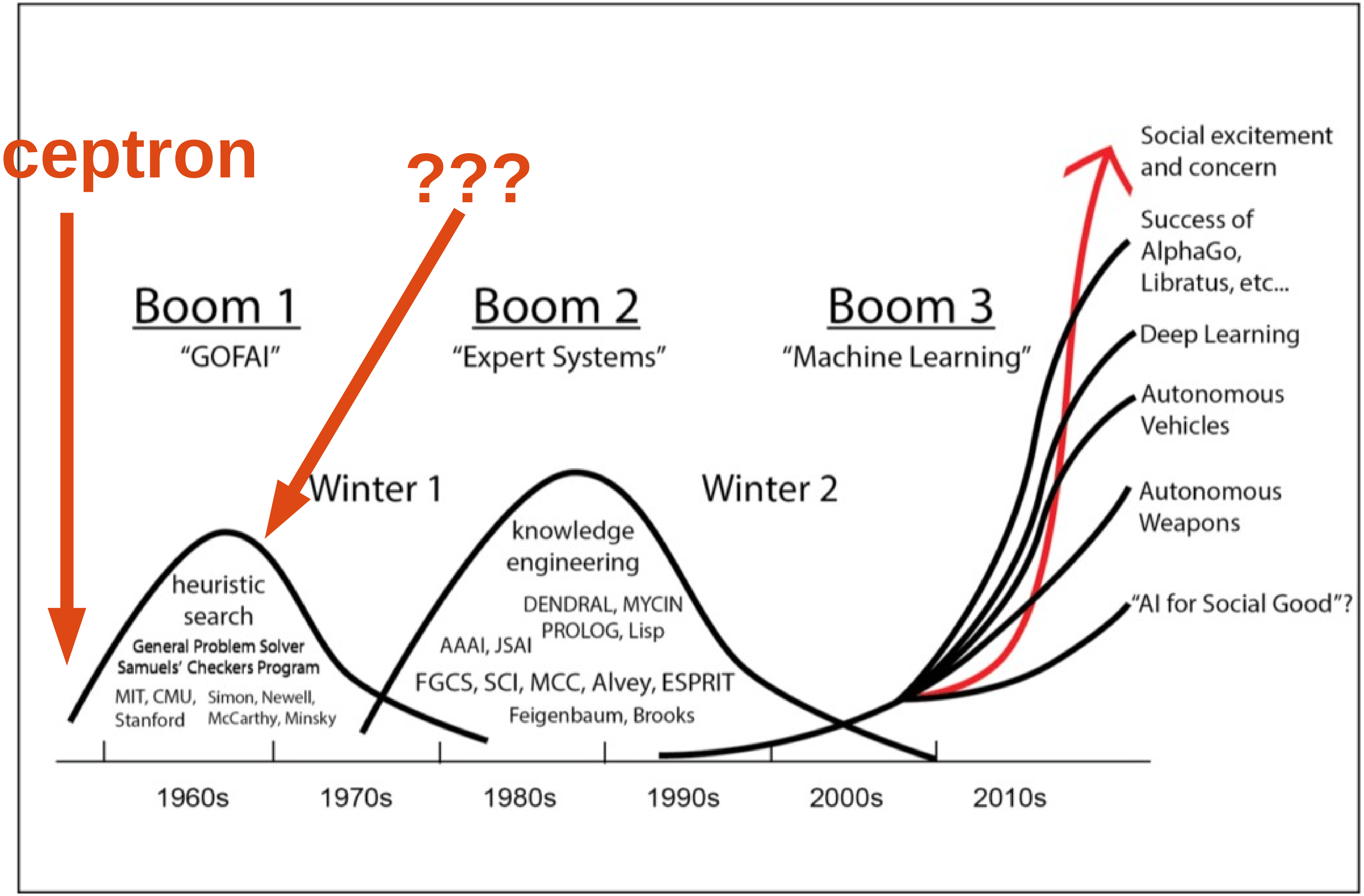
In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

Perceptron

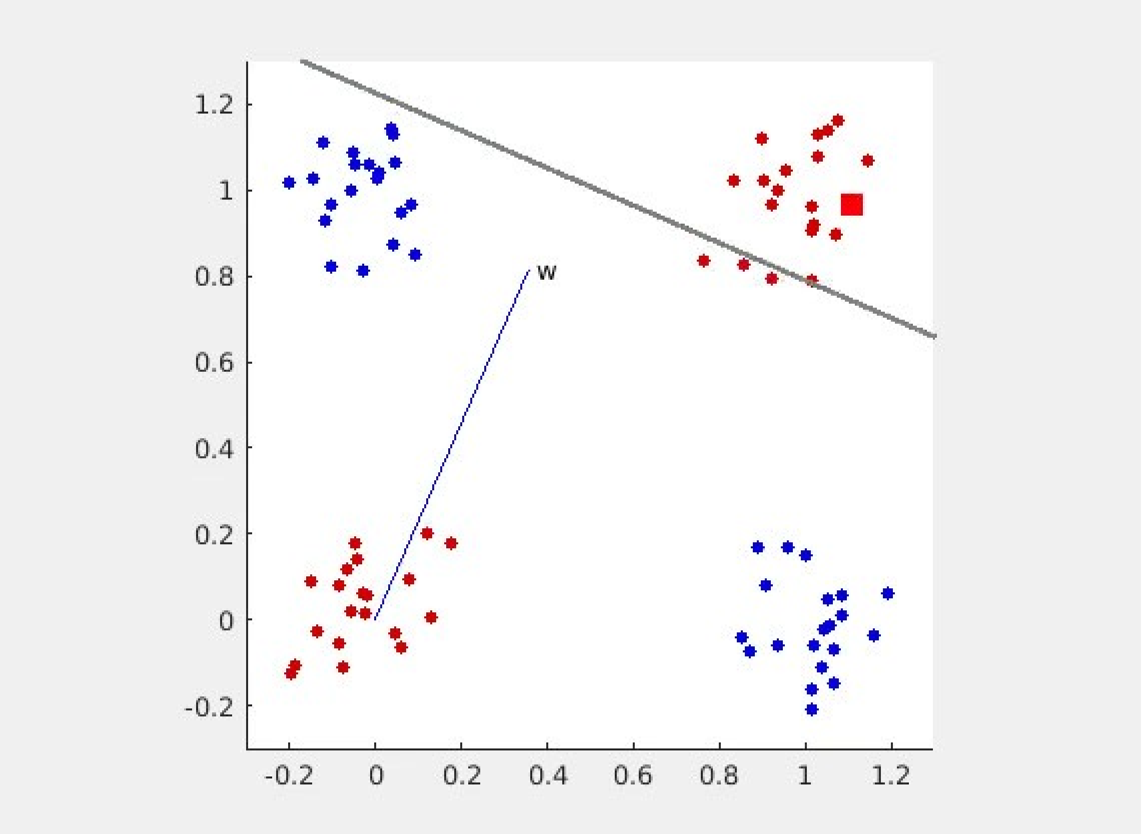
???


Boom 1
"GOFAI"
Boom 2
"Expert Systems"
Boom 3
"Machine Learning"
Winter 1
Winter 2
heuristic search
General Problem Solver
Samuels' Checkers Program
MIT, CMU, Stanford
Simon, Newell, McCarthy, Minsky
knowledge engineering
DENDRAL, MYCIN
PROLOG, Lisp
AAAI, JSAI
FGCS, SCI, MCC, Alvey, ESPRIT
Feigenbaum, Brooks
Social excitement and concern
Success of AlphaGo, Libratus, etc...
Deep Learning
Autonomous Vehicles
Autonomous Weapons
"AI for Social Good"?
1960s
1970s
1980s
1990s
2000s
2010s

The XOR Shock

- In 1969 Marvin Minsky and Seymour Papert (famous AI pioneers) published this example
- XOR =

  Exclusive “OR”



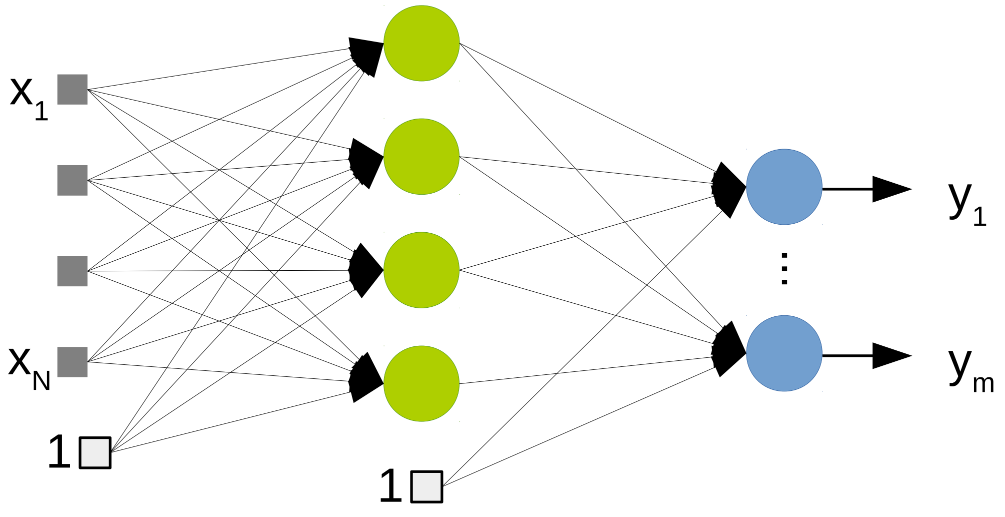
The XOR Shock

- Single neurons are fundamentally unable to describe an XOR function
- Its classes are not linearly separable

  → No possible straight line could separate them
- We need something non-linear…
- Features, maybe?
- Neural features?

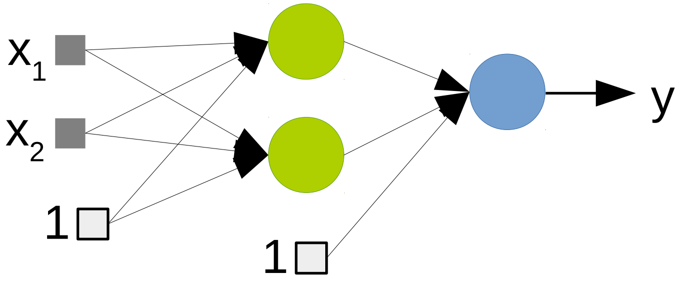
Multi-layer perceptron

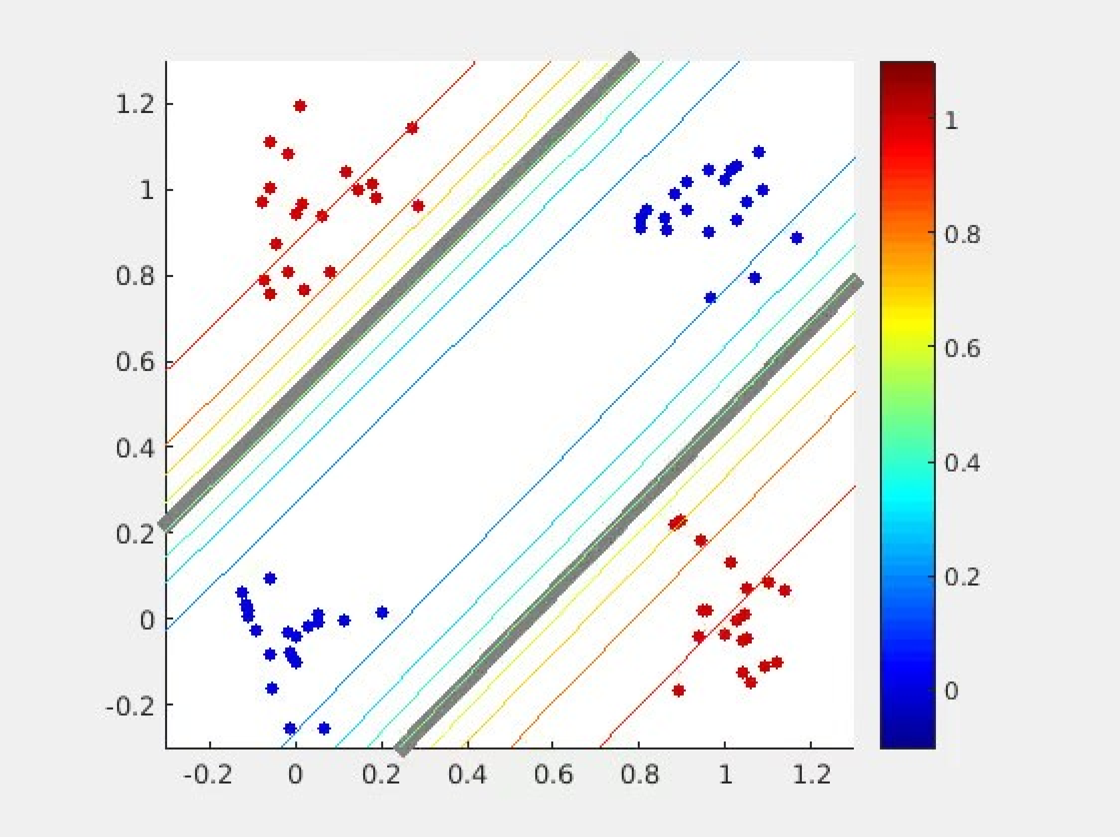
- Modern neural networks are layered
- Layers of neurons provide non-linear features for the next ones



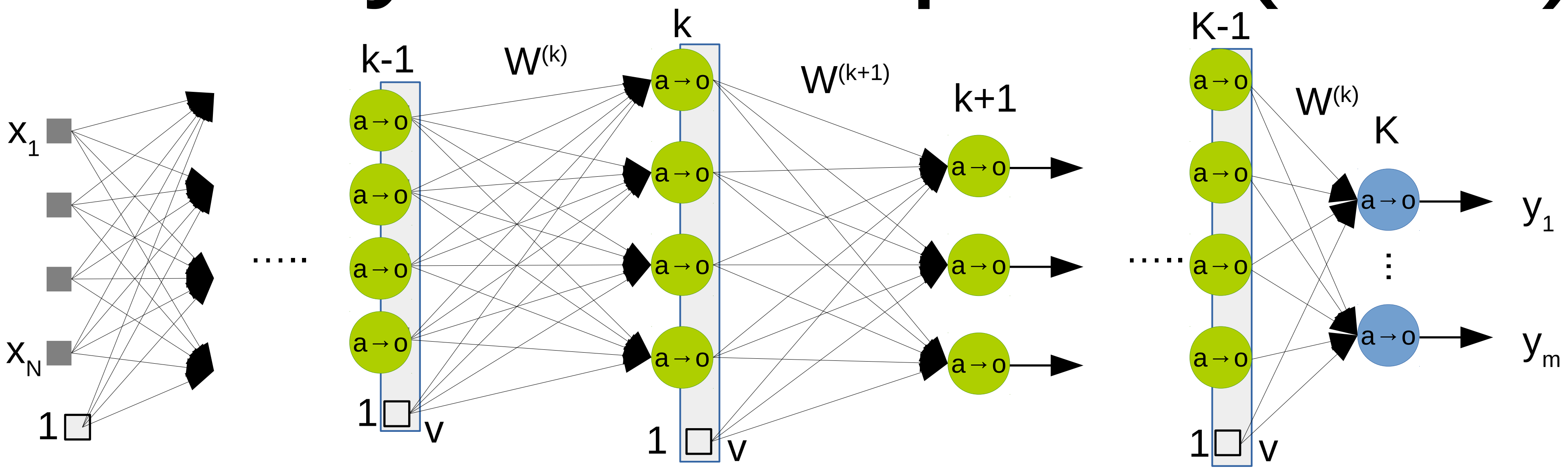
Multi-layer perceptron

- A simple two layer solution to XOR:
- One hidden neuron each for one red cluster
- “or” in the output layer



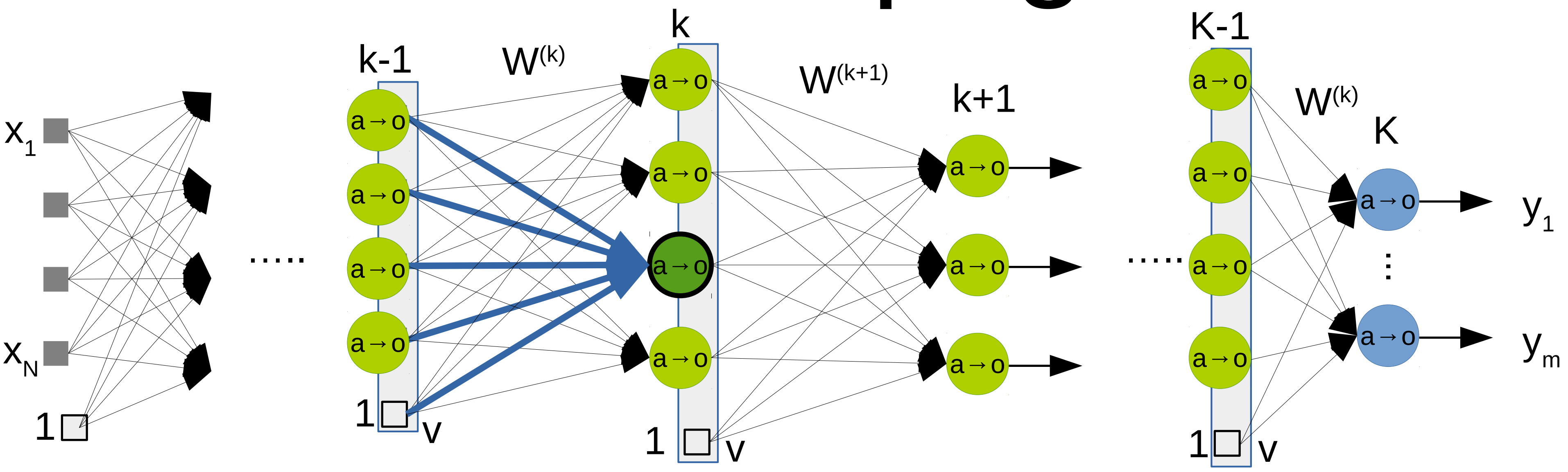


Multi-layer Perceptron (MLP)



- Arbitrary number of hidden layers
- An MLP is a universal approximator
  - Can in principle approximate any continuous function (even with just one hidden layer)
  - Remember: polynomials are universal approximators, too
- Its activations are computed from left to right
  - “Forward propagation” of values
  - As a architecture this is called “feed-forward neural network”

Forward Propagation



Every neuron acts like a perceptron

Multi-layered perceptron with K layers

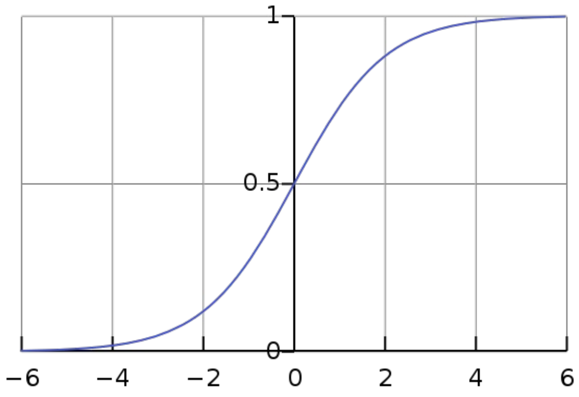
$N^{(k)}$ number of neurons in layer k $(\rightarrow N^{(0)}=n, N^{(K)}=m)$

Synaptic weights from previous neuron j to layer k's neuron i: $w_{ij}^{(k)}$

Net activation i in layer k: $a_i^{(k)} = \sum_{j=1}^{N^{(k-1)+1}} w_{ij}^{(k)} \cdot v_j^{(k-1)}$

Output activation: $o_i^{(k)} = f\left(a_i^{(k)}\right)$

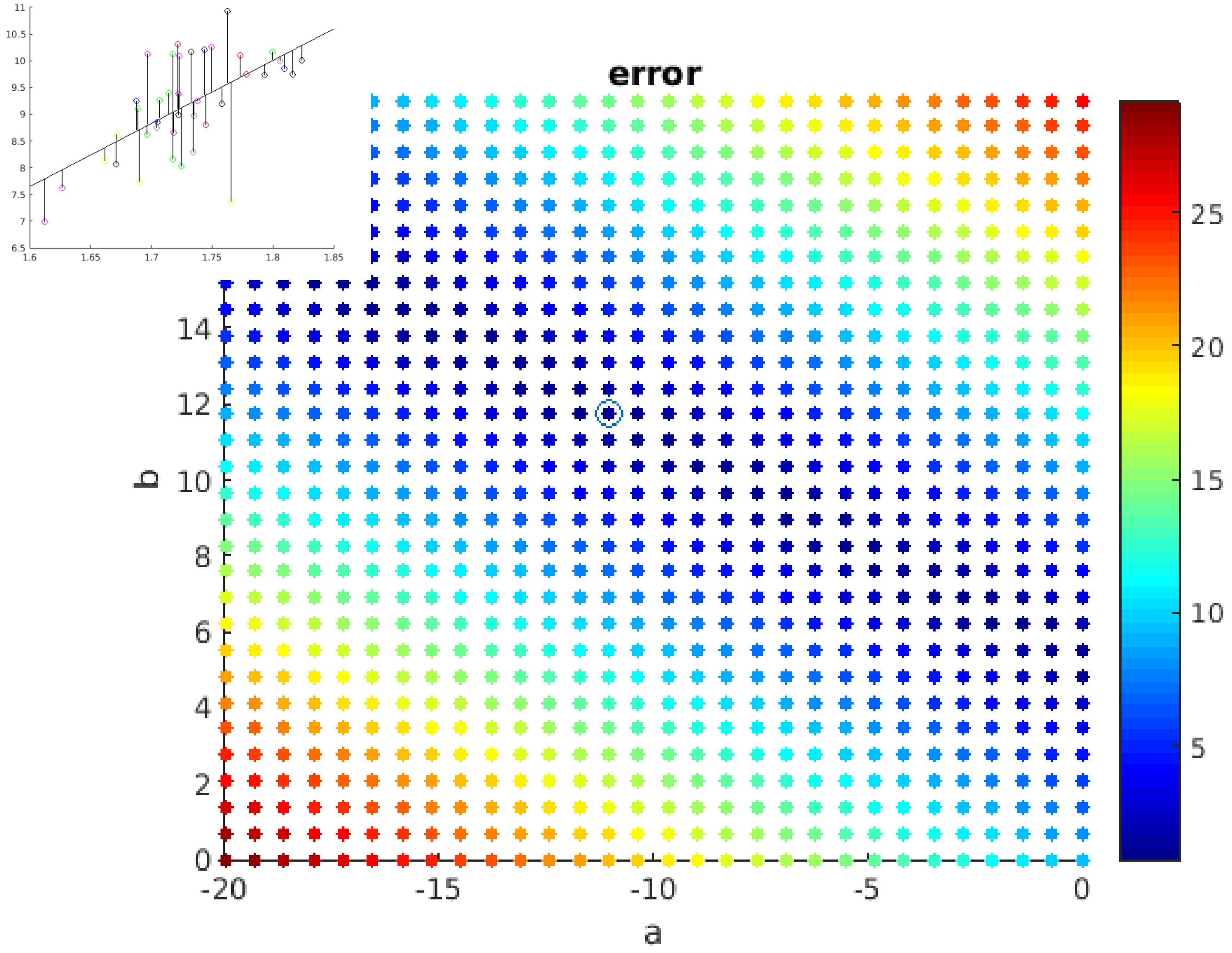
f is the sigmoid function



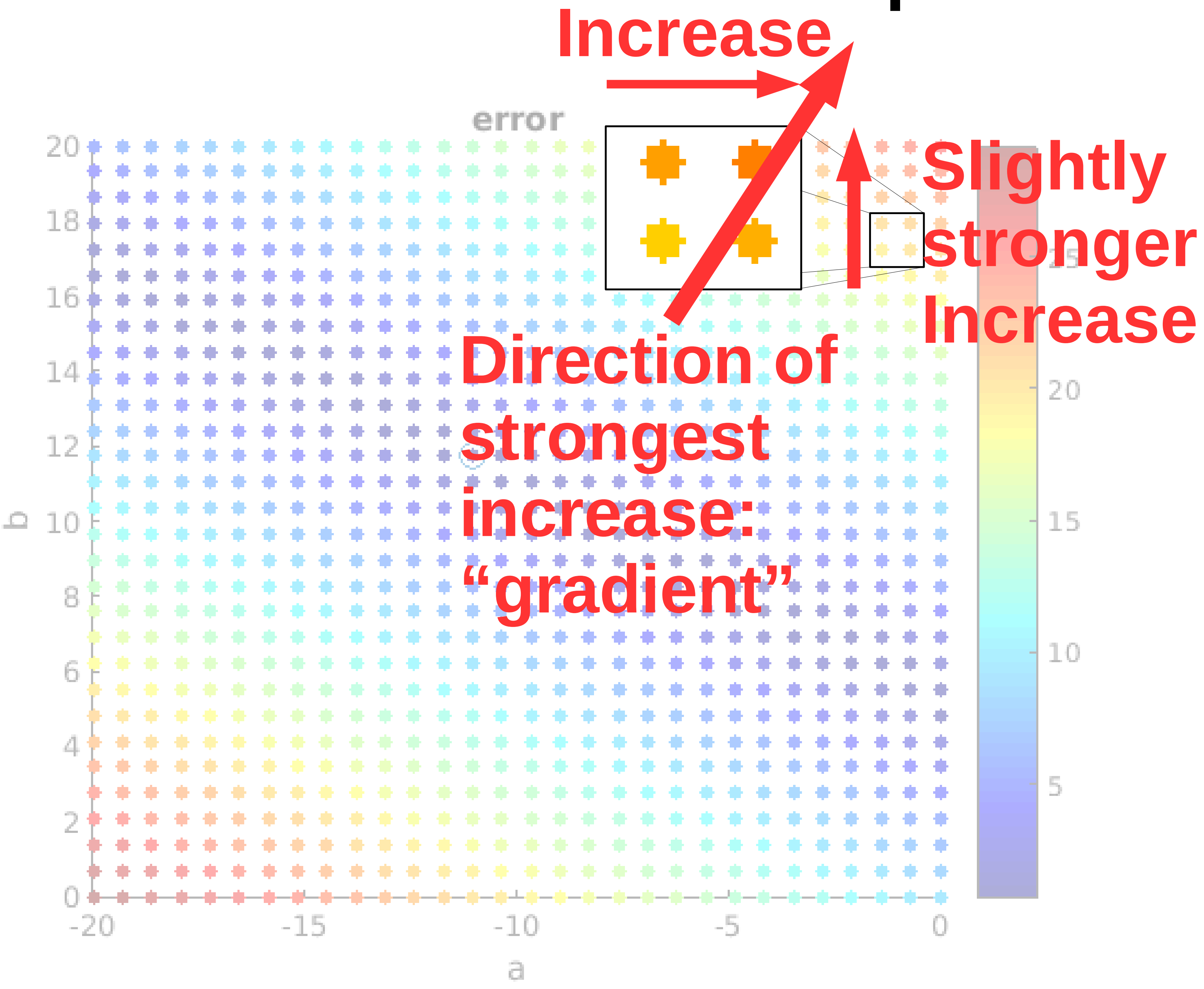
Learning

- How to optimize the parameters?

Error Landscape



Error Landscape

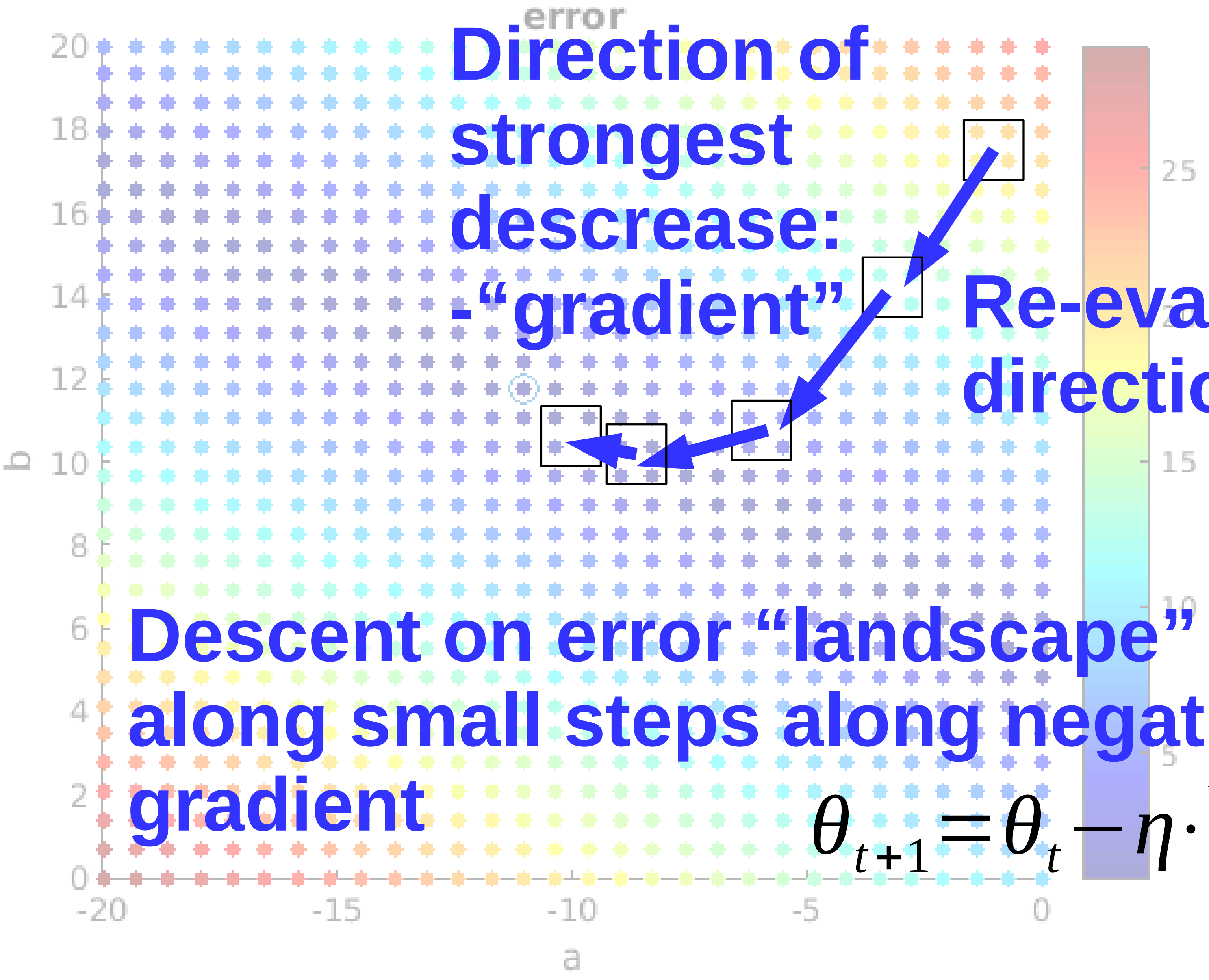


Increase

Slightly stronger Increase

Direction of strongest increase: “gradient”

Error Gradient descent



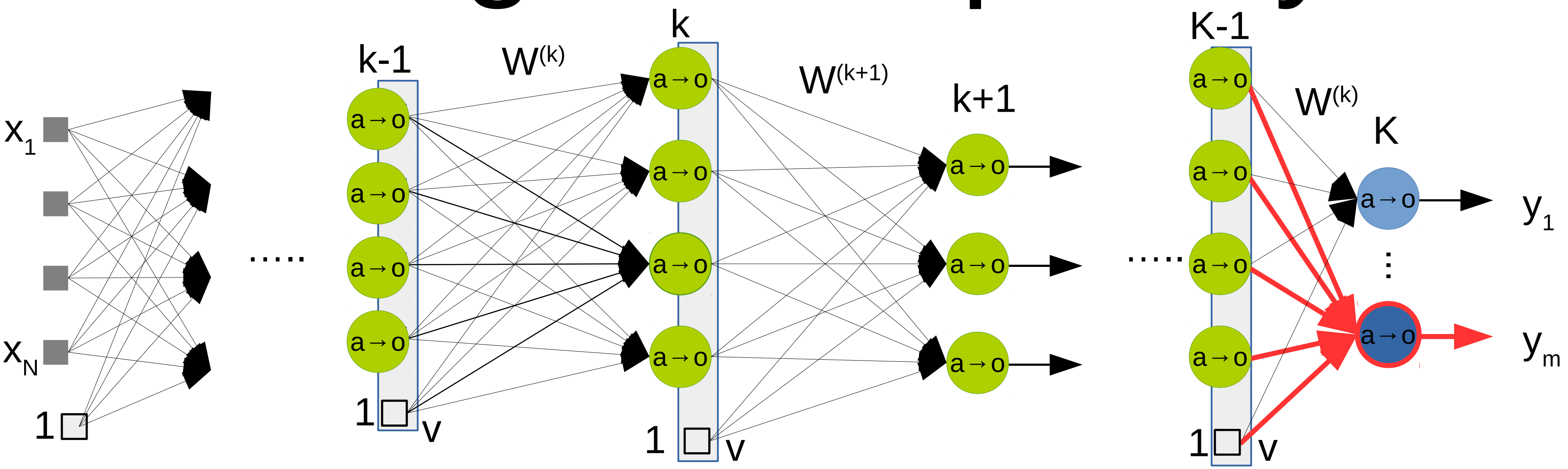
Direction of strongest descrease: -“gradient”

Re-evaluate direction

Descent on error “landscape” along small steps along negative gradient

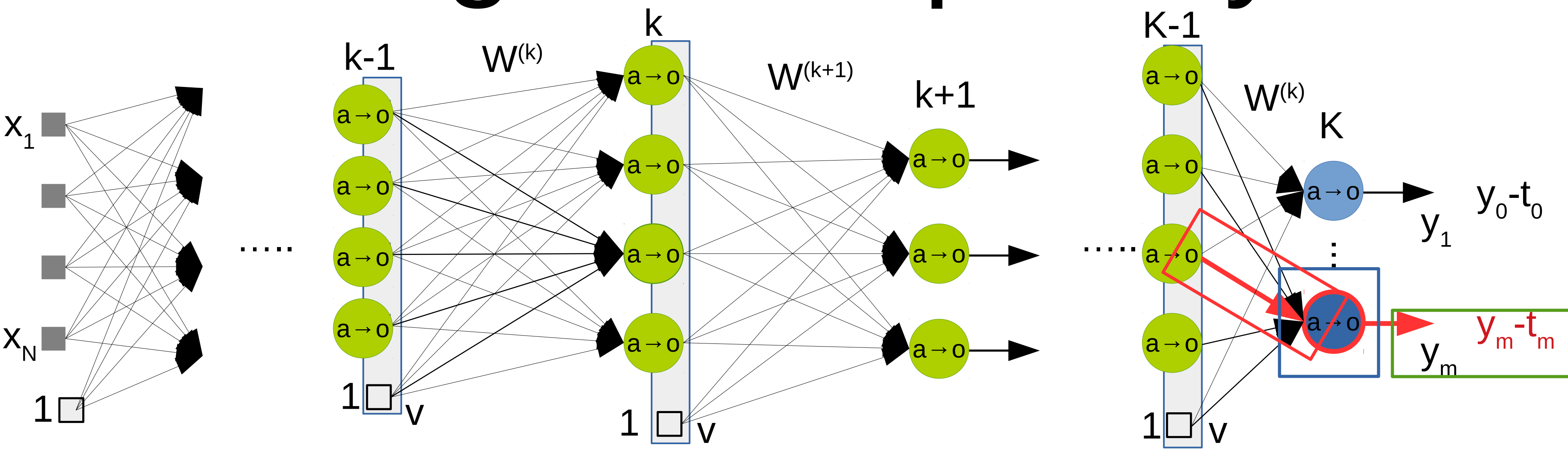
$$\theta_{t+1} = \theta_t - \eta \cdot \nabla E(\theta_t)$$

Training the output layer



- The last layer can be trained almost like linear regression
- There is a target output available
- Gradient descent!

Training the output layer



Supervision: target ouput $\vec{t}$

$$E = \|\vec{y} - \vec{t}\|^2 = (\vec{y} - \vec{t})^T \cdot (\vec{y} - \vec{t})$$

$$\vec{y} = \vec{o} = f\left(W^{(K)} \cdot \vec{v}^{(K-1)}\right)$$

Chain rule gives the derivative for a single weight:

$$\frac{\partial E}{\partial w_{ij}^{(K)}} = \boxed{\frac{\partial E}{\partial o_i^{(K)}}} \cdot \boxed{\frac{\partial o_i^{(K)}}{\partial a_i^{(K)}}} \cdot \boxed{\frac{\partial a_i^{(K)}}{\partial w_i^{(K)}}}$$

$$= [o_i^{(K)} - t_i] \cdot [o_i^{(K)} \cdot (1 - o_i^{(K)})] \cdot v_j$$

How much does wiggling w wiggle E?

=

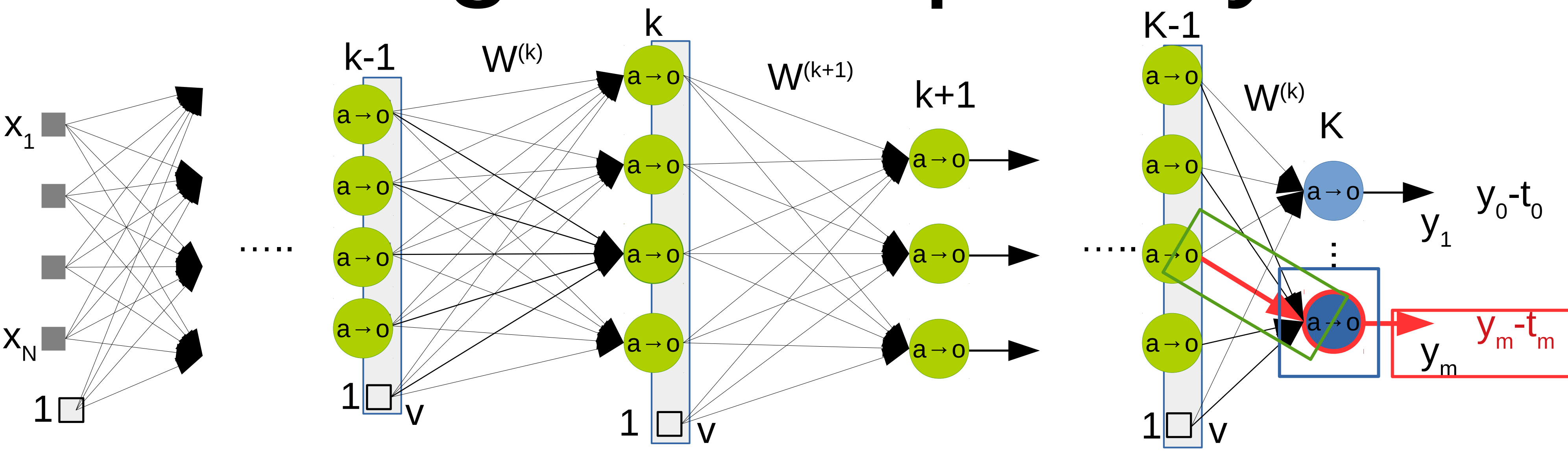
How much does wiggling w wiggle a?

How much does wiggling a wiggle o?

How much does wiggling 0 wiggle E?

Gradient descent update: $w_{ij}^{(K)} \leftarrow w_{ij}^{(K)} - \eta \cdot \frac{\partial E}{\partial w_{ij}^{(K)}}$

Training the output layer



Supervision: target ouput $\vec{t}$

$$E=\|\vec{y}-\vec{t}\|^2=(\vec{y}-\vec{t})^T\cdot(\vec{y}-\vec{t})$$

$$\vec{y}=\vec{o}=f(W^{(K)}\cdot\vec{v}^{(K-1)})$$

Chain rule gives the derivative for a single weight:

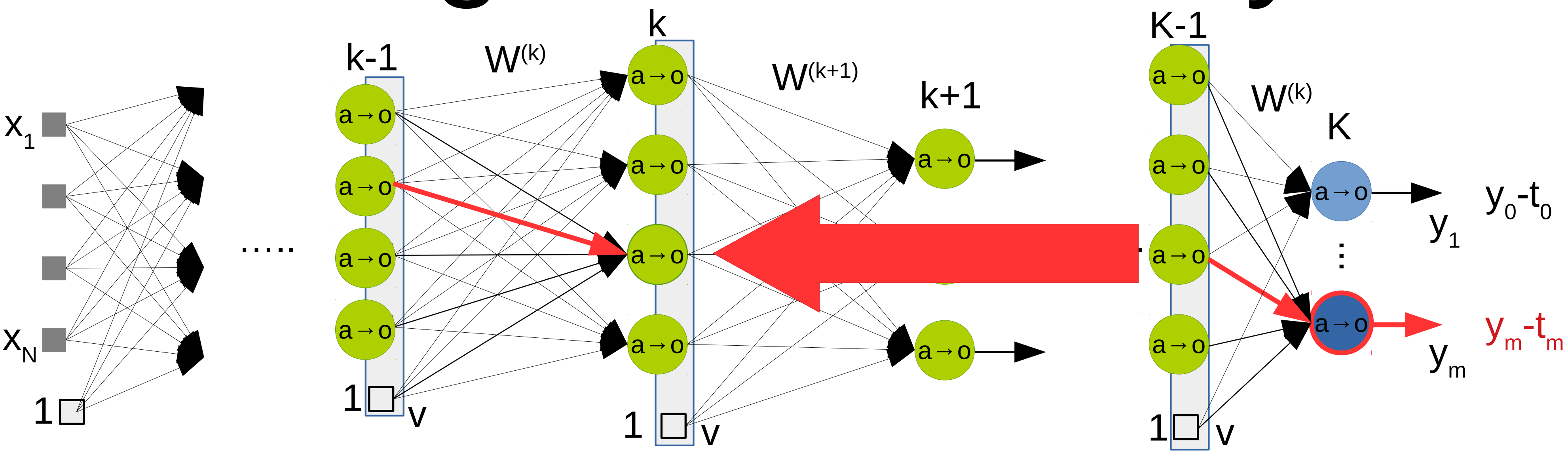
$$\frac{\partial E}{\partial w_{ij}^{(K)}}=\boxed{\frac{\partial E}{\partial o_i^{(K)}}}\cdot\frac{\partial o_i^{(K)}}{\partial a_i^{(K)}}\cdot\boxed{\frac{\partial a_i^{(K)}}{\partial w_i^{(K)}}}$$

$$=\boxed{[o_i^{(K)}-t_i]}\cdot[o_i^{(K)}\cdot(1-o_i^{(K)})]\cdot\boxed{v_j}$$

This is like the previous perceptron:
Change = error x input

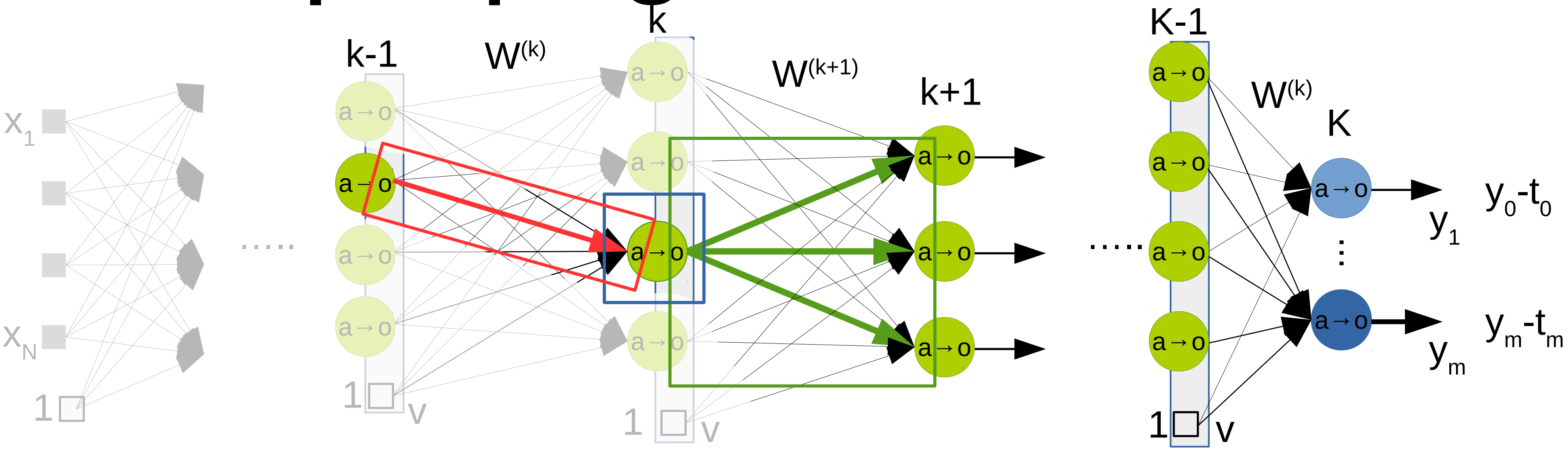
Gradient descent update: $w_{ij}^{(K)}\leftarrow w_{ij}^{(K)}-\eta\cdot\frac{\partial E}{\partial w_{ij}^{(K)}}$

Training the hidden layer



- How to get the gradient for the hidden layers?
- No direct target output for the neurons available!
- Approach: propagate error backwards

Backpropagation of errors



Chain rule gives the derivative for a single weight (same as before):

$$\frac{\partial E}{\partial w_{ij}^{(k)}} = \boxed{\frac{\partial E}{\partial o_i^{(k)}}} \cdot \boxed{\frac{\partial o_i^{(k)}}{\partial a_i^{(k)}}} \cdot \boxed{\frac{\partial a_i^{(k)}}{\partial w_i^{(k)}}}$$

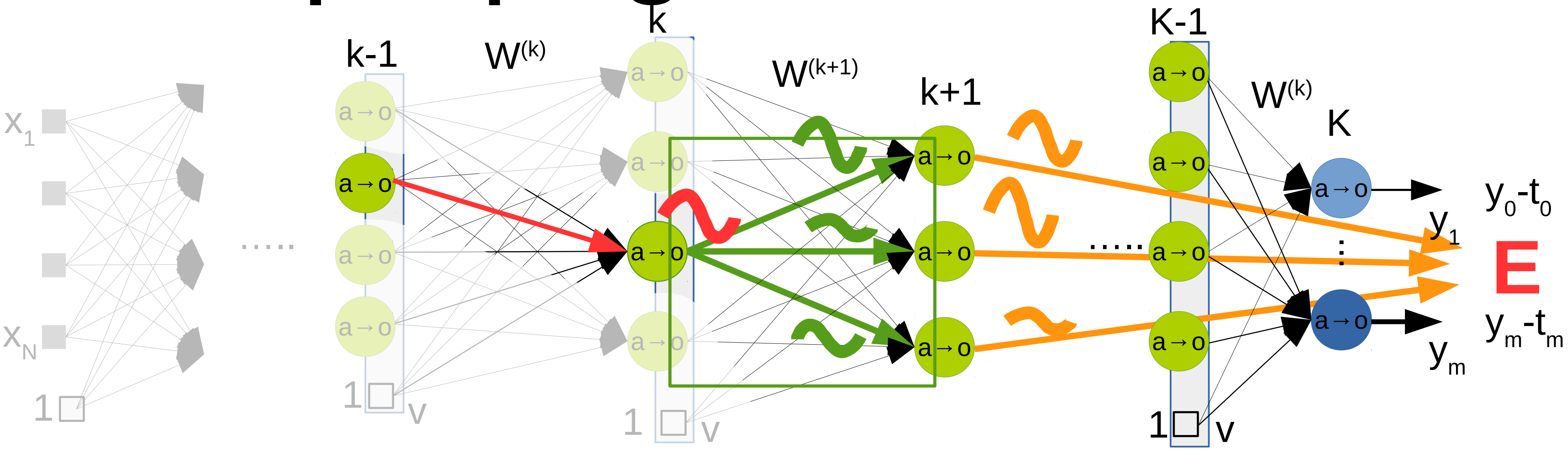
$$= \boxed{[?????]} \cdot \boxed{[o_i^{(k)} \cdot (1 - o_i^{(k)})]} \cdot \boxed{v_j}$$

Gradient descent: $w_{ij}^{(k)} \leftarrow w_{ij}^{(k)} - \eta \cdot \frac{\partial E}{\partial w_{ij}^{(k)}}$

The 2nd and 3rd term are exactly the same as for the output layer.

The 1st term has to be determined through recursion through the green weights.

Backpropagation of errors



Chain rule gives the derivative for a single weight (same as before):

$$\frac{\partial E}{\partial w_{ij}^{(k)}} = \boxed{\frac{\partial E}{\partial o_i^{(k)}}} \cdot \frac{\partial o_i^{(k)}}{\partial a_i^{(k)}} \cdot \frac{\partial a_i^{(k)}}{\partial w_i^{(k)}}$$

$$= \boxed{[?????]} \cdot [o_i^{(k)} \cdot (1 - o_i^{(k)})] \cdot v_j$$

Gradient descent: $w_{ij}^{(k)} \leftarrow w_{ij}^{(k)} - \eta \cdot \frac{\partial E}{\partial w_{ij}^{(k)}}$

How much does wiggling o wiggle E?

=

How does wiggling o wiggle a_1 wiggle E?

+

How does wiggling o wiggle a_2 wiggle E?

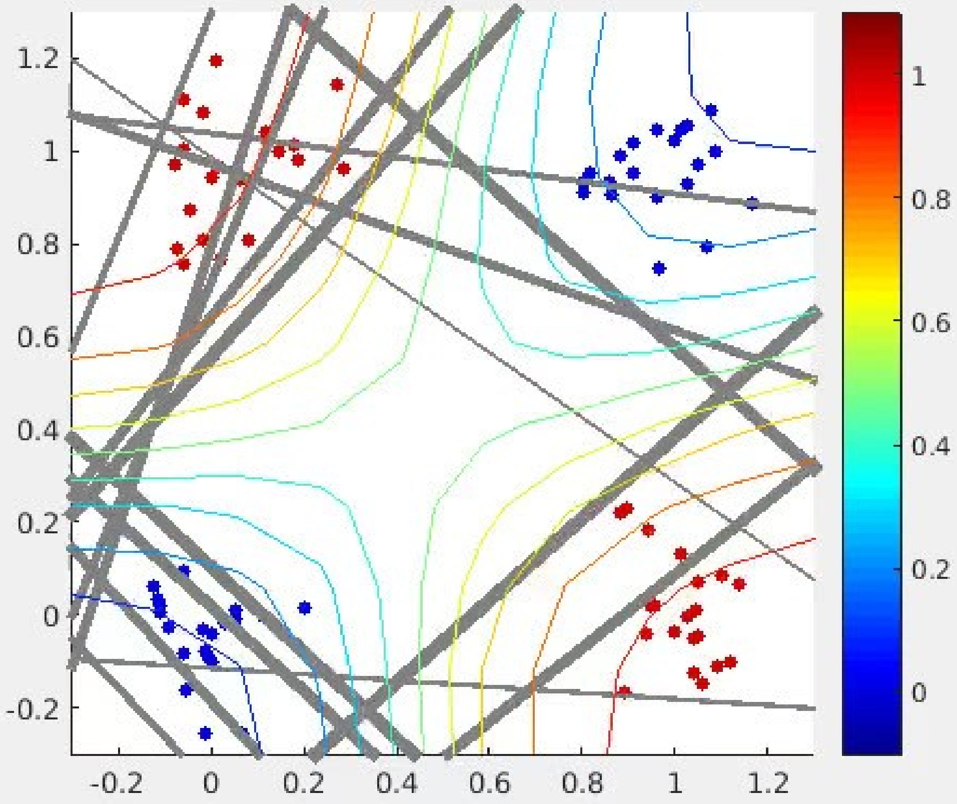
+

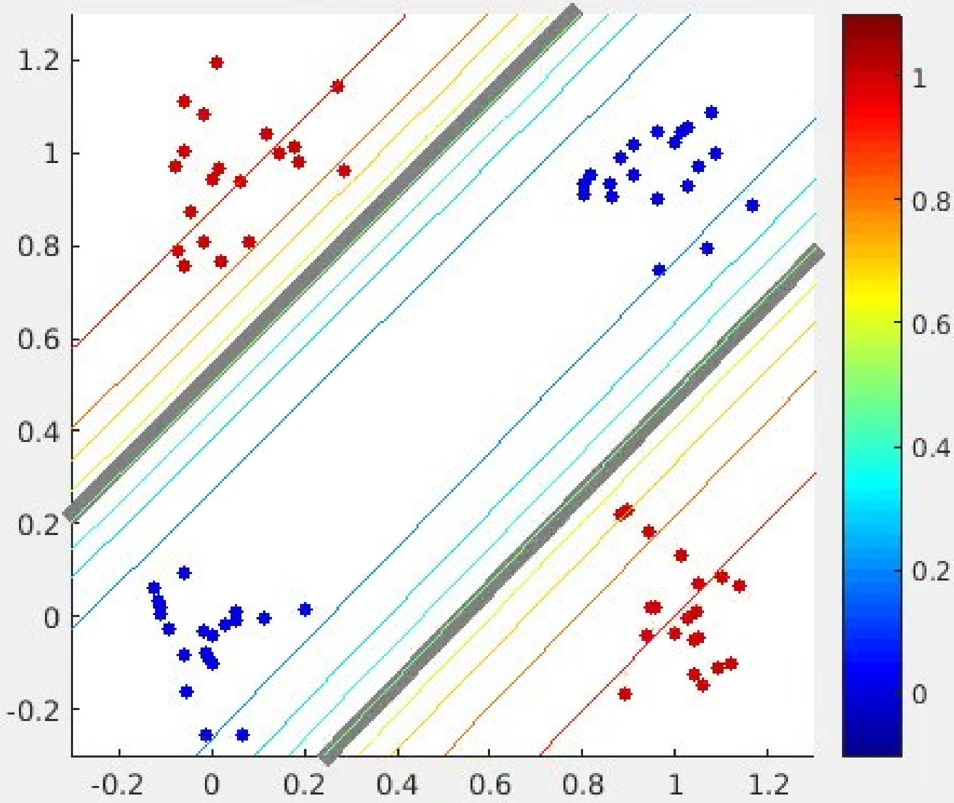
...

$$\frac{\partial E}{\partial o_i^{(k)}} = \sum_{l=1}^{N^{(k+1)}} \boxed{\frac{\partial E}{\partial a_l^{(k+1)}}} \cdot \boxed{\frac{\partial a_l^{(k+1)}}{\partial o_i^{(k)}}} = \sum_{l=1}^{N^{(k+1)}} \boxed{\frac{\partial E}{\partial a_l^{(k+1)}}} \cdot \boxed{w_{li}^{(k)}}$$

The orange term was computed in the previous layer! Recursion complete!

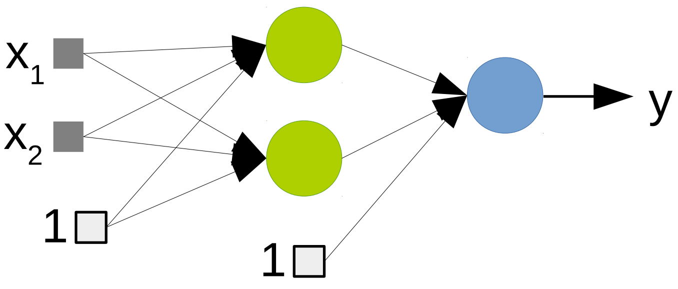
XOR Revisited

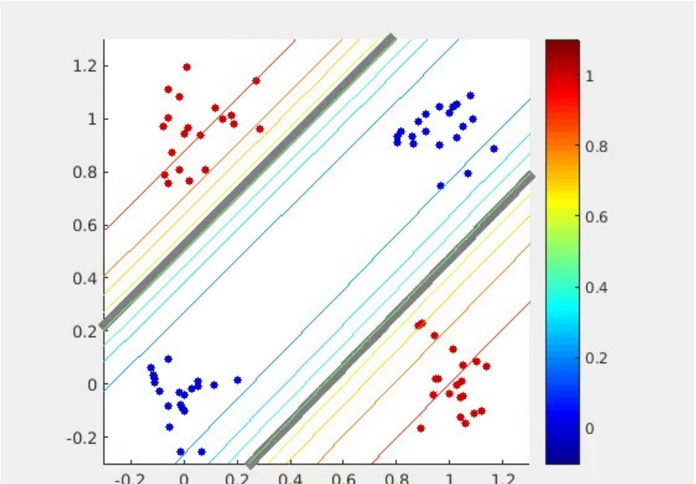




Random Initialization

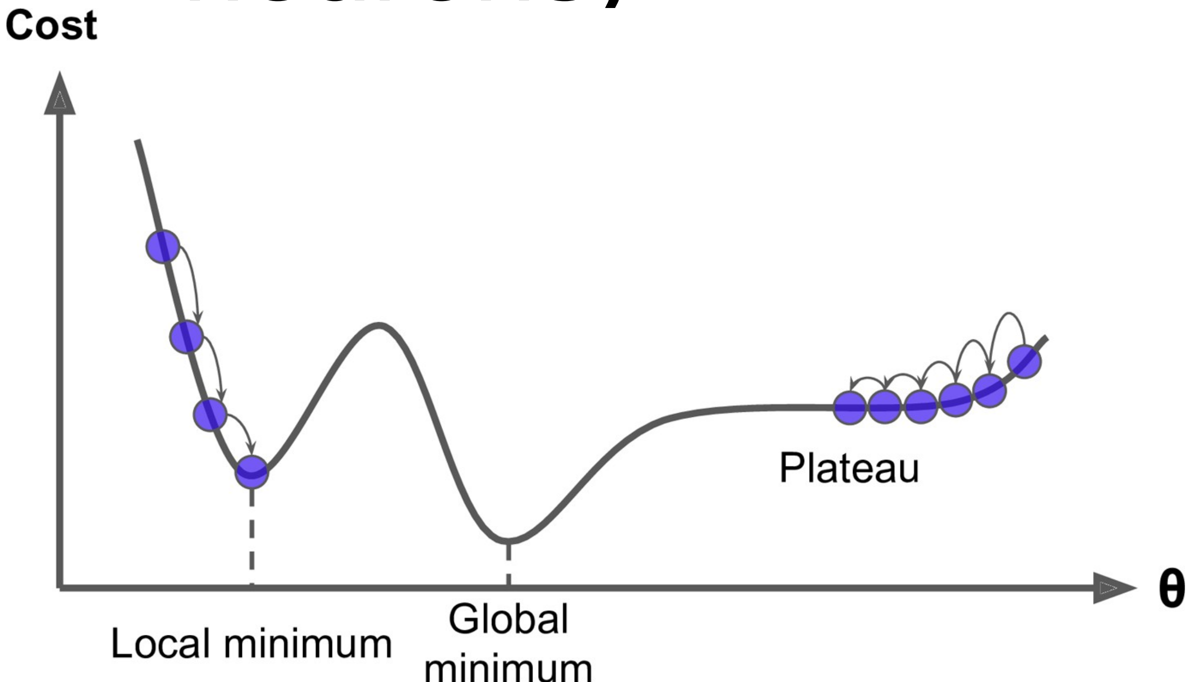
- Neural network weights cannot be initialized as zero
- Random initialization necessary!
- Same initialization would make all features the same → same gradient → always the same features
- This makes MLP a stochastic algorithm that needs to be run several times to estimate how good it is!

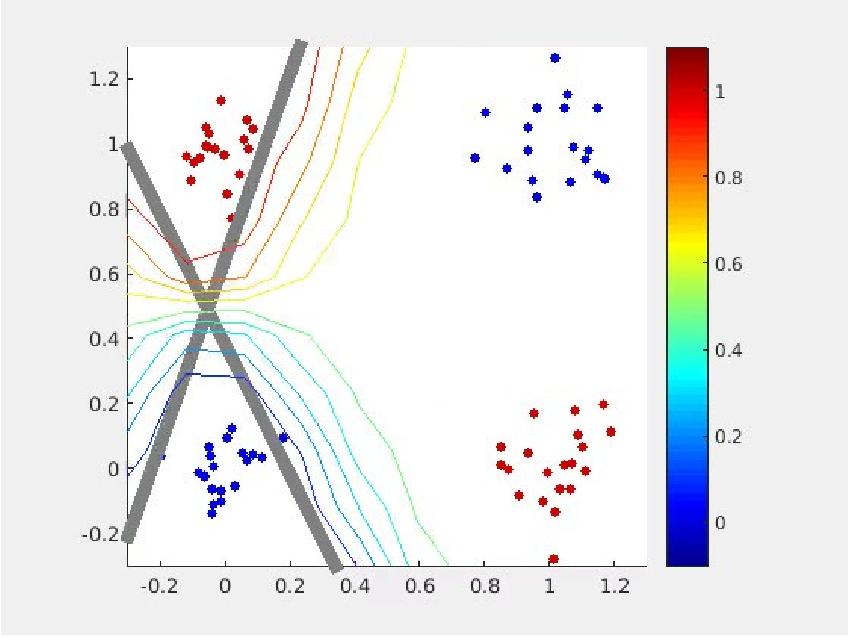




Local Optima

- MLP do have local optima!
- They also have several equal global optima (by switching neurons)





Practical

- The following code allows to hand-code a neural network by choosing weights

```
library(nnet)

# bias,x1,x2 -> first hidden, then second hidden; bias,h1,h2 -> output
W <- c(-3,10,0,-3,0,10,-2,2,2)
nn <- nnet(matrix(c(0,0),nrow=1, ncol=2), matrix(c(0), nrow=1, ncol=1), size = 2, maxit = 1)
# the following line hand-sets the weights
# nnet can be trained automatically by providing real data above and removing the following line
nn$wts <- W

x <- (0:20) * 0.05
y <- x
xy <- expand.grid(x, y)
z <- predict(nn, xy)
z <- matrix(z,nrow=21,ncol=21,byrow=TRUE)
contour(x,y,z)
```

- Run the code and try to understand the neuron’s output
- Choose the weights differently so that the neural network implements a XOR function of the inputs

Practical

- Start preparing for the coursework interim assessment (week 6)
- Inspect loadMNIST.R script uploaded on moodle