Deep learning and Neuroscience

Emlyon school of business

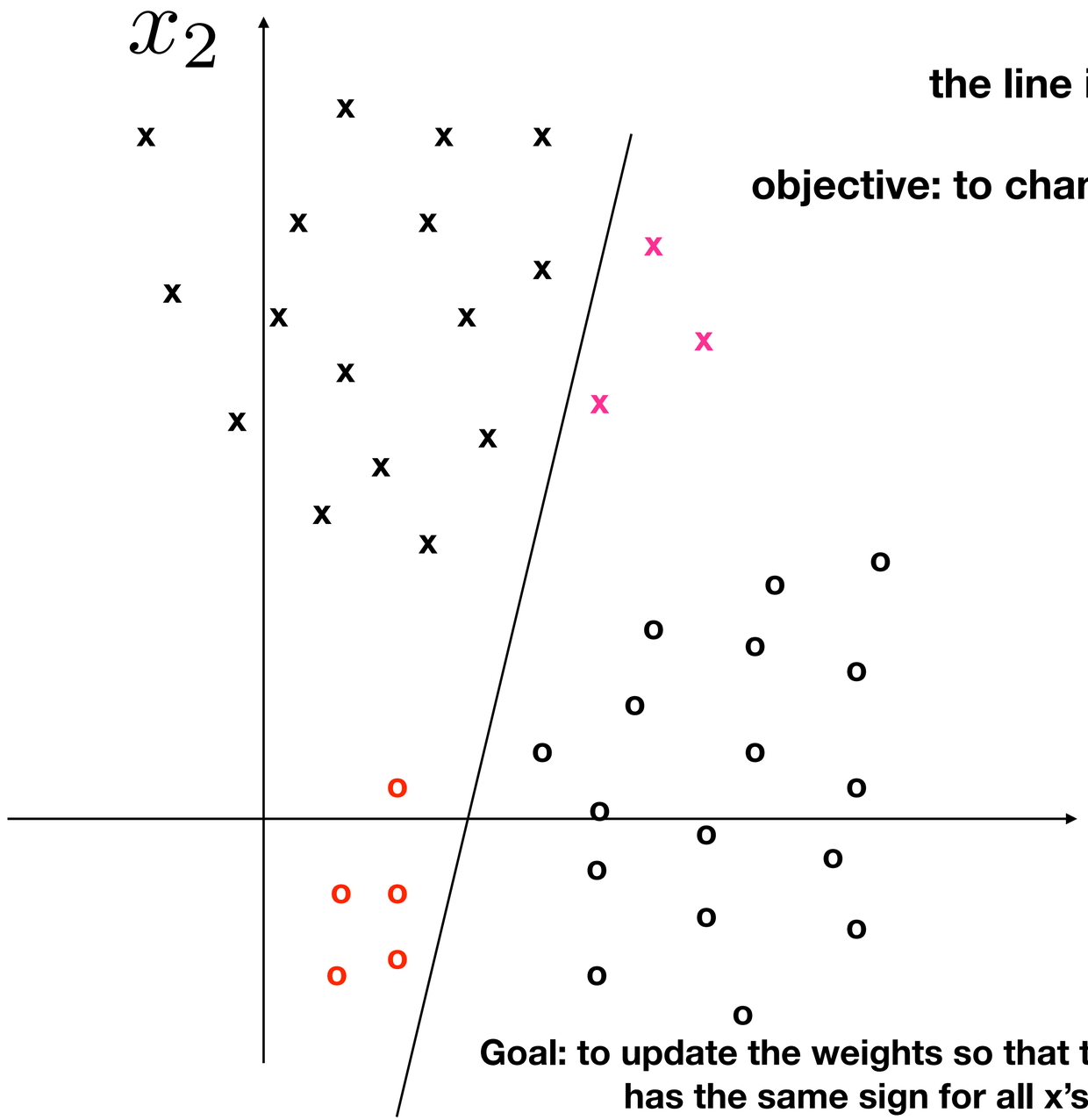
May 2020

Farzad Fathizadeh

Swansea University and MPI for Biological Cybernetics

Course contents

Linear perceptrons, linear regression, logistic regression, deep neural nets and back propagation, convolutional neural nets, recurrent neural nets, large scale statistical inference and signal detection, fast Fourier transform, discussions on geometric dimension reduction, crash course on python and tensorflow.



the line is represented by a vector w

objective: to change w in a way that there is less error.

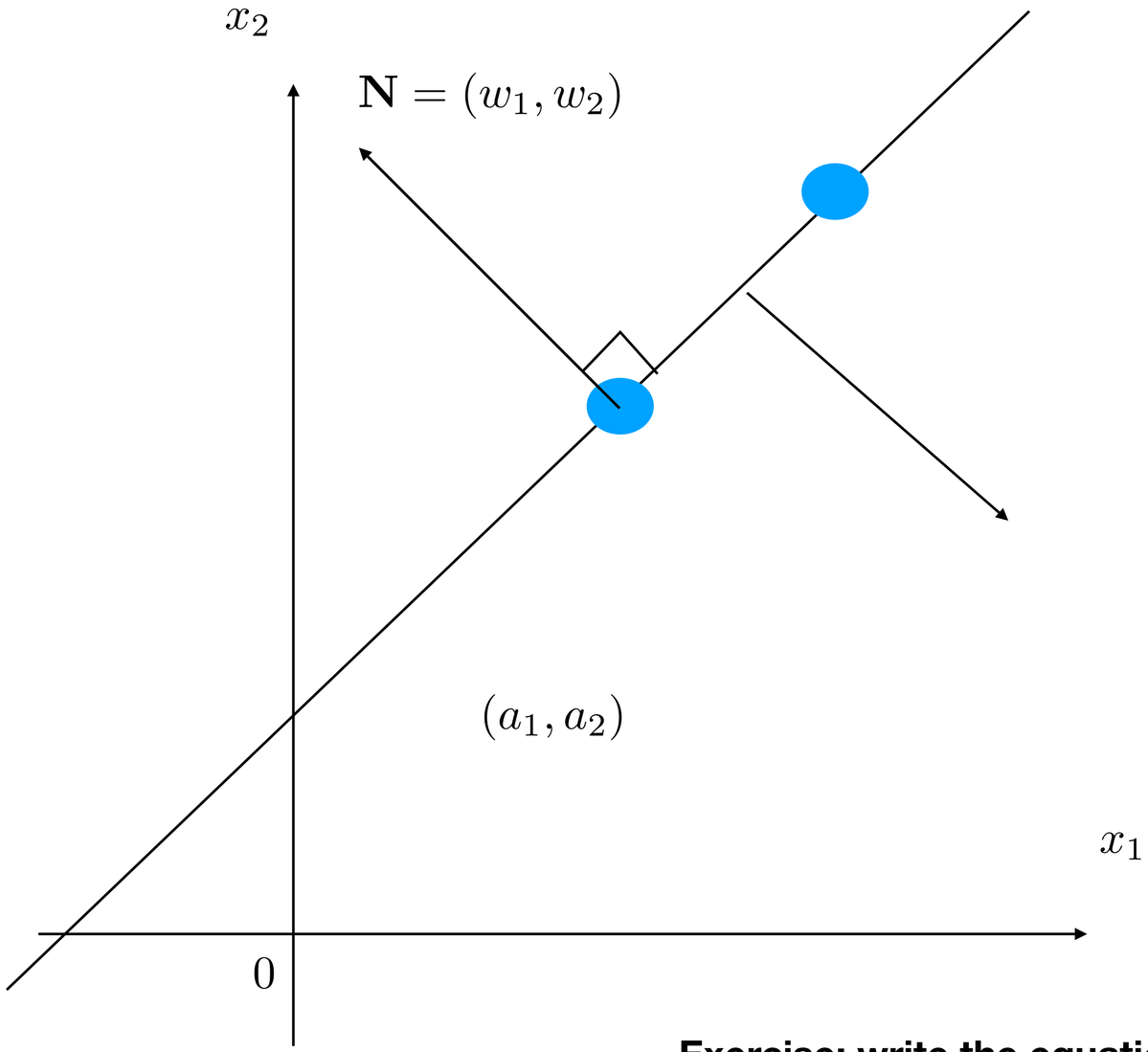
Linear perceptron

red: misclassified points

This is an example of linearly separable data.

$$\mathbf{w} = (w_0, w_1, w_2)$$

Goal: to update the weights so that there is no misclassification: meaning $\mathbf{w} \cdot \mathbf{z}$ has the same sign for all x's and the same sign for all o's



The equation of a line determined by:
two points on the line
or
the slope and one point on the line
or
by one point on the line and the normal vector

y = m x + b

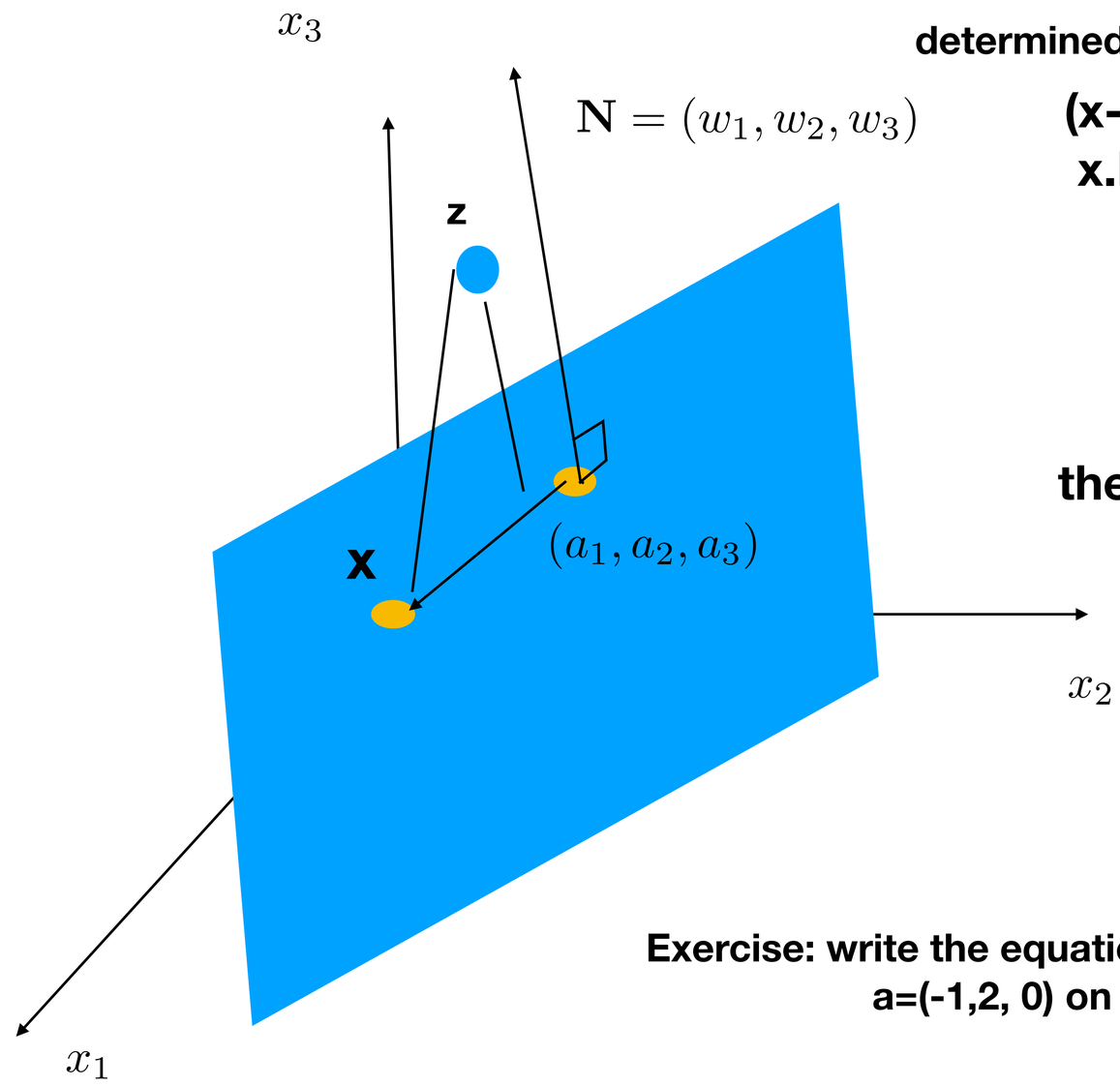
x_2 = m x_1 + b

$$\mathbf{w} = (w_0, w_1, w_2)$$

Exercise: write the equation of the line with normal vector N=(3, 4) with the point (-1,2) on it. Then identity 3 different points on the line.

25 = 16 + 9

5^2 = 4^2 + 3^2



The equation of a plane in 3 dimensional space:

determined by one point on the plane and its normal vector

(x-z).N = + or - the distance = **|N|=1**

x.N - z.N = - w_0 - z.N = w.z **z=(z_0=1, z_1, …, z_n)**

$$\mathbf{x} = (x_1, x_2, x_3)$$

$$\mathbf{a} = (a_1, a_2, a_3)$$

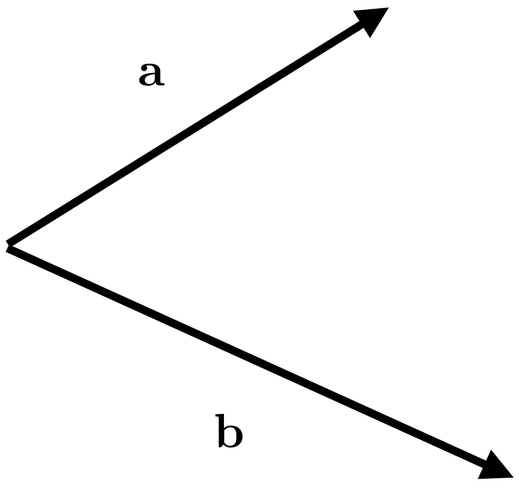
the equation for all x on the surface

$$(\mathbf{x} - \mathbf{a}) \cdot \mathbf{N} = 0$$

orthogonal

Exercise: write the equation of the plane with normal vector N=(3, 4, 1) with the point a=(-1,2, 0) on it. Then identity 3 different points on the plane.

Calculus of vectors

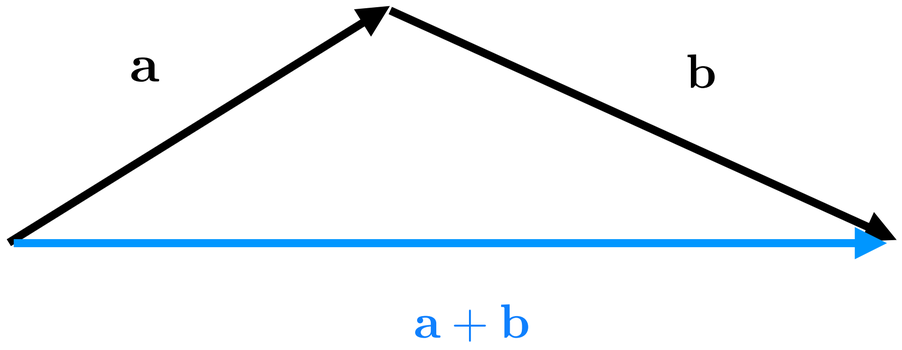


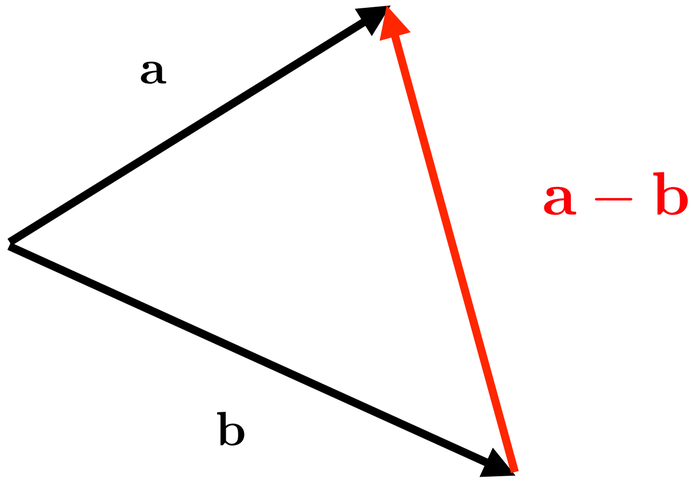
$$\mathbf{a} = (a_1, a_2, ..., a_n)$$

$$\mathbf{b} = (b_1, b_2, ..., b_n)$$

$$\mathbf{a} + \mathbf{b} = (a_1 + b_1, a_2 + b_2, ..., a_n + b_n)$$

$$\mathbf{a} - \mathbf{b} = (a_1 - b_1, a_2 - b_2, ..., a_n - b_n)$$

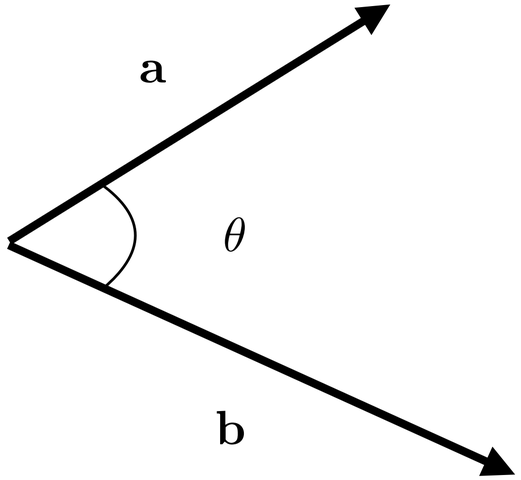




$$\mathbf{a} = (a_1, a_2, ..., a_n)$$ np.array

$$\mathbf{b} = (b_1, b_2, ..., b_n)$$

$$|\mathbf{a}| = (a_1^2 + \cdots + a_n^2)^{1/2}$$ Length



| b/ |b| | =1

$$\mathbf{a} \cdot \mathbf{b} = |a|\,|b|\,\cos(\theta)$$ (Theorem)

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n = \sum_{i=1}^{n} a_i b_i$$

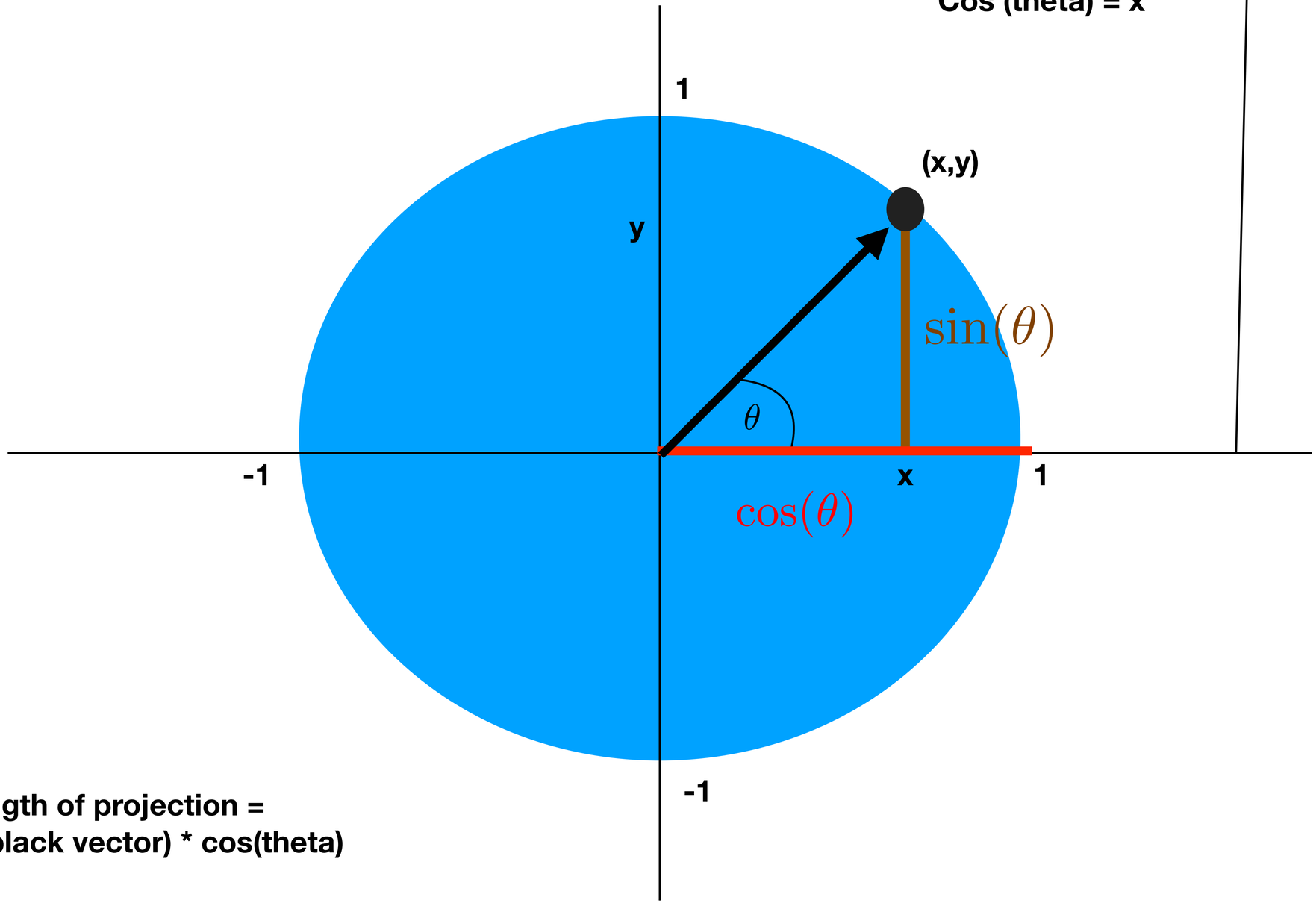
(1, 1), (1, -1)

$$\mathbf{a} \cdot \mathbf{b} = 0$$ means orthogonality because $$\cos(90^\circ) = 0$$

Exercise: write a python code that calculates the angle between two general vectors. Hint: use arccos, cos^{-1}.

Sin and Cos functions

Cos (theta) = x



Length of projection =
Length (black vector) * cos(theta)

$$\mathbf{N} = (w_1, w_2, \ldots, w_n)$$

We always normalize: take the length of the normal vector equal to 1!

$$|N| = 1$$

$$\mathbf{a} = (a_1, a_2, \ldots, a_n)$$

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)$$

$$(\mathbf{x} - \mathbf{a}) \cdot \mathbf{N} = 0$$

$$(x_1 - a_1)w_1 + \cdots + (x_n - a_n)w_n = 0$$

$$w_1x_1 + \cdots + w_nx_n + w_0x_0 = 0$$

$$x_0 = 1$$

The equation of a hyper-plane in dimension n with normal vector N, with the vector (point a) on the hyper-plane

$$w_0x_0 = w_0 = -a_1w_1 - \cdots - a_nw_n$$

The equation of an (n-1)-dimensional plane (hyper-plane) in n-dimensional space is determined by a vector of weights:

$$\mathbf{w} = (w_0, w_1, \ldots, w_n)$$

The points on the hyper-plane are all

$$\mathbf{x} = (x_1, \ldots, x_n)$$

such that

$$w_1 x_1 + \cdots + w_n x_n + w_0 x_0 = 0$$

$$x_0 = 1$$

$$\mathbf{w} = (w_0, w_1, \ldots, w_n)$$
$$\mathbf{x} = (x_0, x_1, \ldots, x_n)$$

$$x_0 = 1$$

$$\mathbf{w} \cdot \mathbf{x} = 0$$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ . \\ . \\ . \\ x_n \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ . \\ . \\ . \\ w_n \end{bmatrix}$$

$$\mathbf{w}^T\mathbf{x} = \begin{bmatrix} w_0 & w_1 & \dots & w_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ . \\ . \\ . \\ x_n \end{bmatrix} = 0$$

Homework 1: implement the following algorithm for linear perceptron with python and try it on data sets that are linearly separable. Also check the performance the non linearly separable data.

Training a linear perceptron: use the misclassified points from the training data to change the weights of the classifying plane to have less and less error.

Simple way (particular for linear perceptron):

initialize w randomly, label points

with y=+1 (correct) and y=-1 (misclassified)

pick a misclassified point and update w:

$$(\mathbf{z}, y)$$

meaning of misclassification

$$sign(\mathbf{w} \cdot \mathbf{z}) \neq y$$

Hint:

(x-z).N = + or - the distance = x.N - z.N = - w_0 - z.N = w.z

$$\mathbf{w} \xrightarrow{\text{replace}} \mathbf{w} + y\,\mathbf{z}$$

Exercise:

error = distance of the misclassified point from the hyper-plane of w = $-y\,\mathbf{w} \cdot \mathbf{z}$

error = distance of the misclassified point from the hyper-plane of updated w =

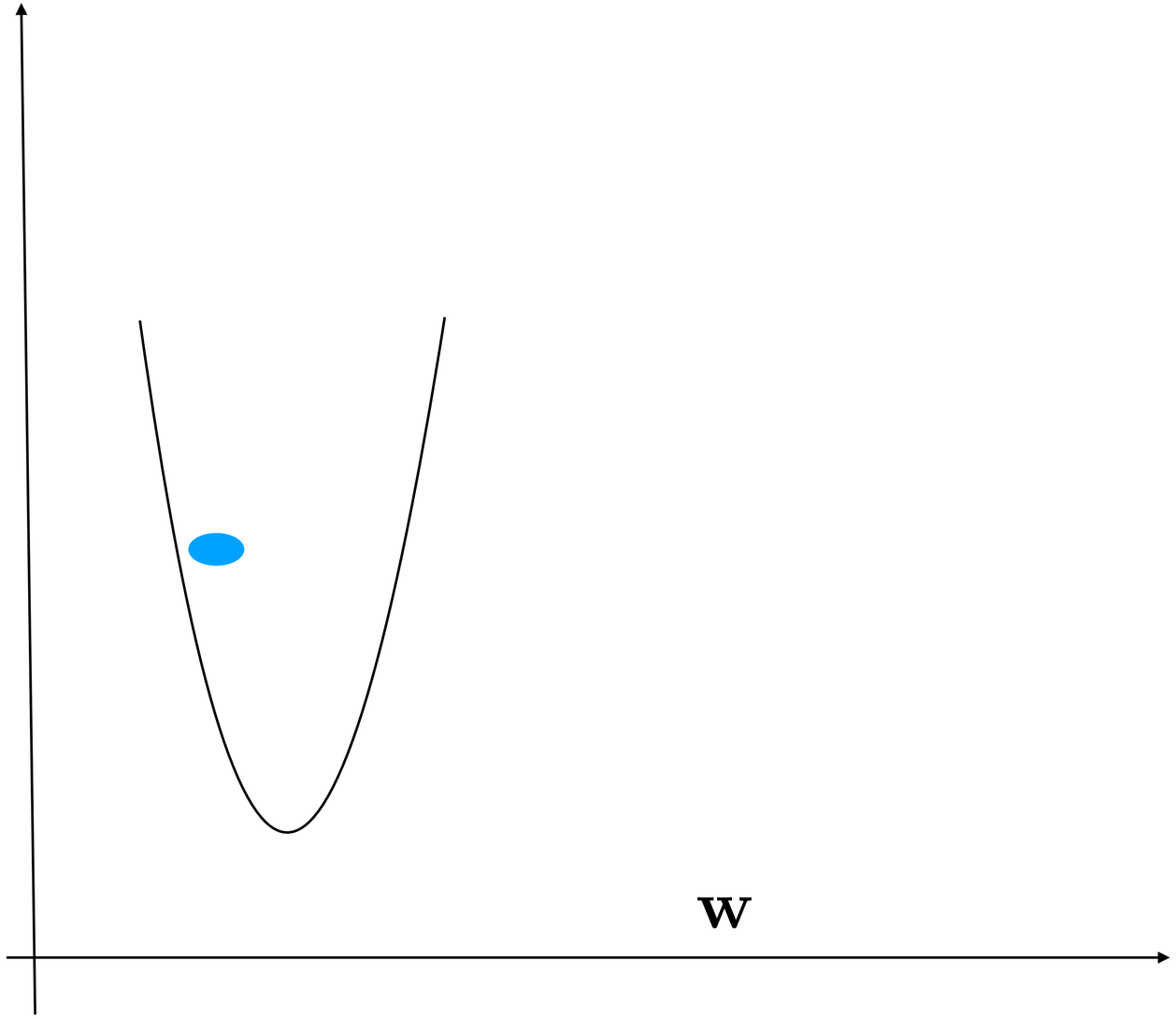
$$-y\,(\mathbf{w} + y\mathbf{z}) \cdot \mathbf{z} = -y\,\mathbf{w} \cdot \mathbf{z} - y^2 \mathbf{z} \cdot \mathbf{z}$$

General way: using gradient

$$sign(s) = \begin{cases} +1, & s > 0 \\ -1, & s < 0 \end{cases}$$

Gradient descent (1-dimensional case)

Error $= f(\mathbf{w})$



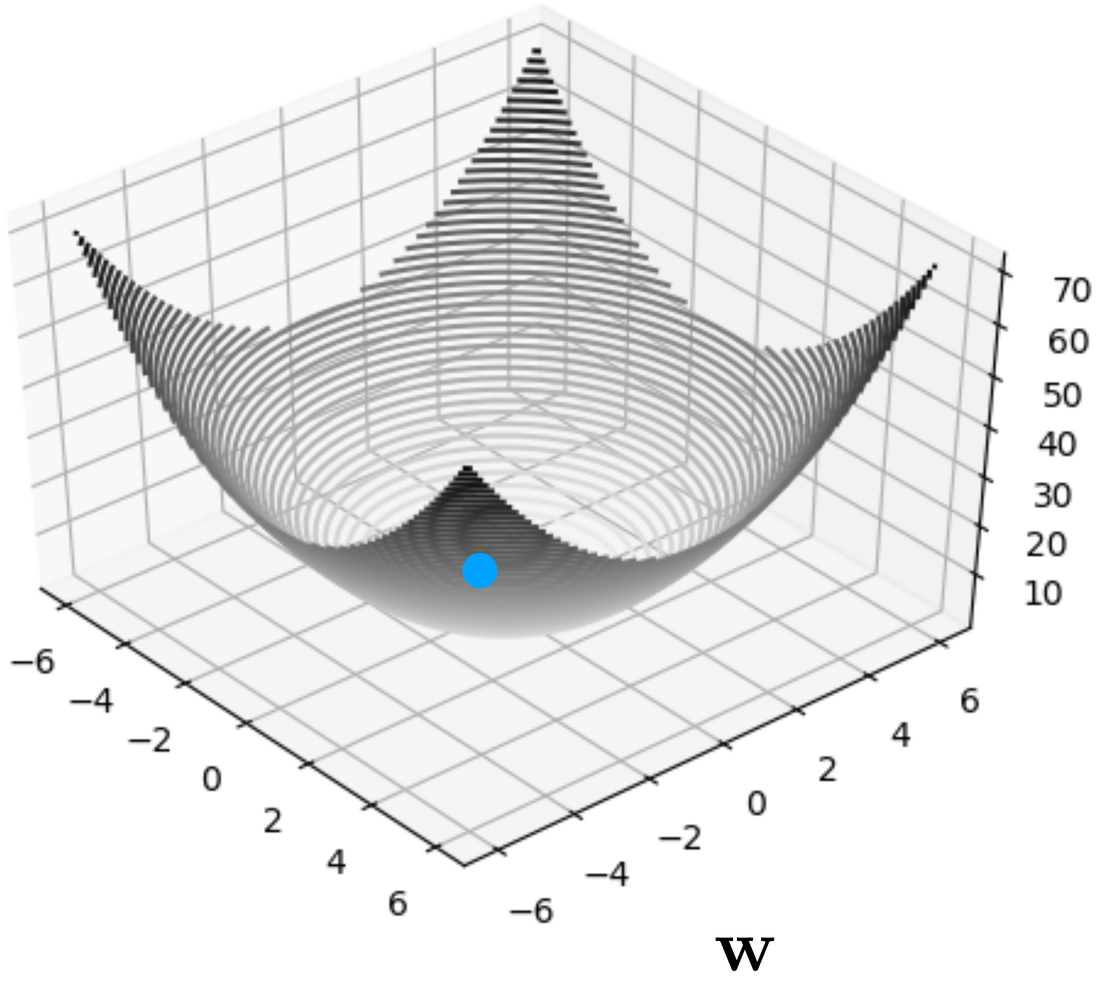
Replace (update)

$$\mathbf{w} \longrightarrow \mathbf{w} - \eta f'(\mathbf{w})$$

η learning rate (positive)

In which direction we have the maximum decrease in the error? in the direction of minus gradient of error

Error = $f(\mathbf{w}) = f(w_0, w_1, \ldots, w_n)$



$$-\nabla f(\mathbf{w})$$

Gradient of f:

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f}{\partial w_0}(\mathbf{w}), \frac{\partial f}{\partial w_1}(\mathbf{w}), \ldots, \frac{\partial f}{\partial w_n}(\mathbf{w}) \right)$$

$$\mathbf{w} \longrightarrow \mathbf{w} - \eta \nabla f(\mathbf{w})$$

$$f(w) = w^k$$

$$f'(w) = \frac{df}{dw}(w) = kw^{k-1}$$

Example of calculating partial derivatives:

$$f(\mathbf{w}) = f(w_0, w_1, w_2) = 5w_0^2 w_1^3 w_2$$

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f}{\partial w_0}, \frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2} \right)$$

$$= \left(10w_0 w_1^3 w_2, 15w_0^2 w_1^2 w_2, 5w_0^2 w_1^3\right)$$

The reason for the gradient descent working

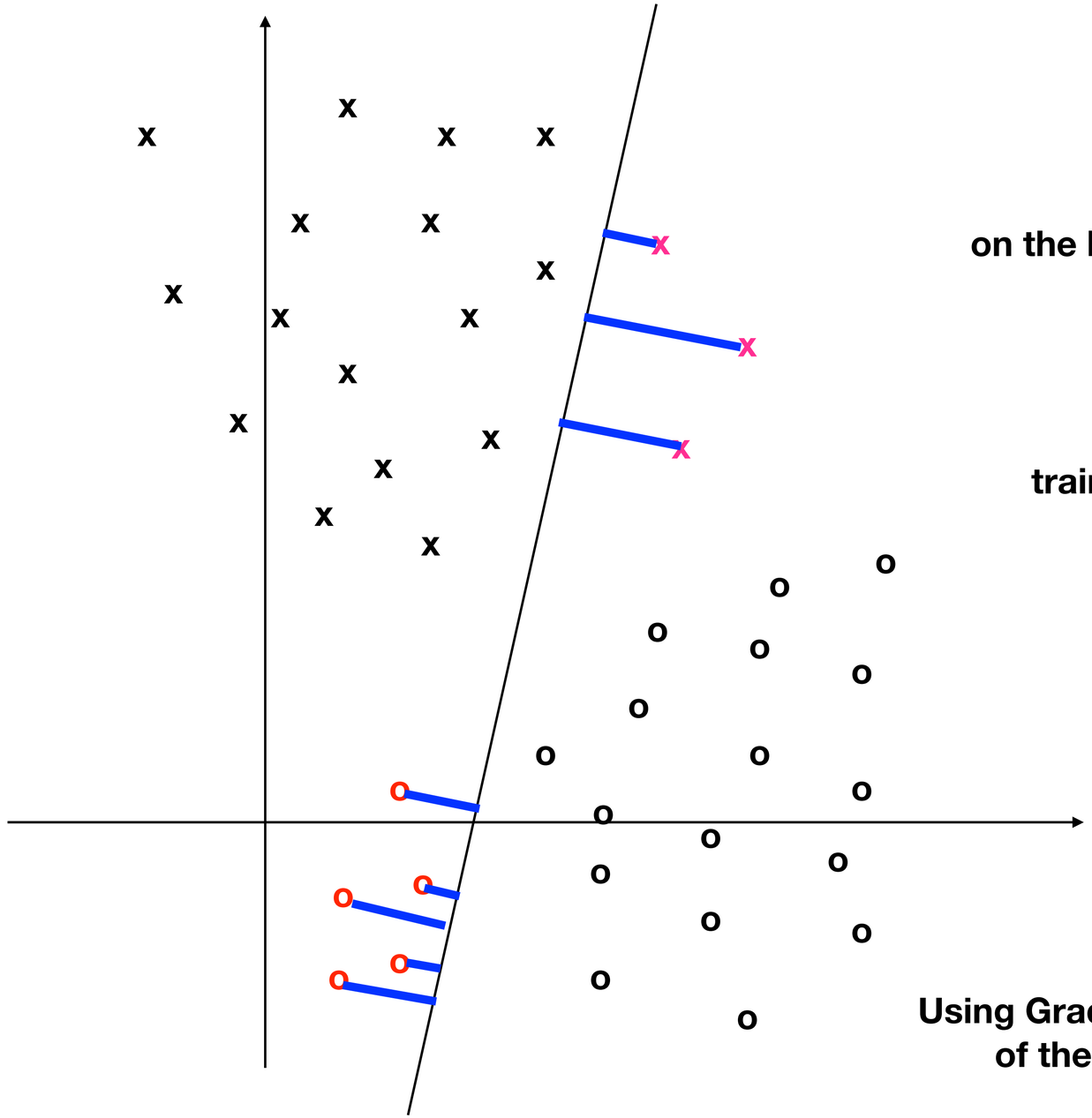
$$f(w+h) \sim f(w) + f'(w)h$$

$$f(\mathbf{w}+\mathbf{h}) \sim f(\mathbf{w}) + \nabla f(\mathbf{w}) \cdot \mathbf{h}$$

$$f\left(\mathbf{w} - \eta \nabla \mathbf{f}(\mathbf{w})\right) \sim f(\mathbf{w}) - \eta \nabla f(\mathbf{w}) \cdot \nabla f(\mathbf{w})$$

$$f(w+h) = \sum_{k=0}^{n} \frac{f^k(w)}{k!} h^k$$

Suggested reading:
Taylor series
in one and several variables



Linear perceptron

on the hyper-plane $\mathbf{w} \cdot \mathbf{x} = 0$

training data: $(\mathbf{z_1}, y_1), (\mathbf{z_1}, y_1), \ldots, (\mathbf{z_N}, y_N)$

$y_i = 1$ or $y_i = -1$

error = sum of the distances of the misclassified points from the line.

Using Gradient descent we decrease this error (by changing the parameters of the line and thereby the line) until there is no misclassified point.

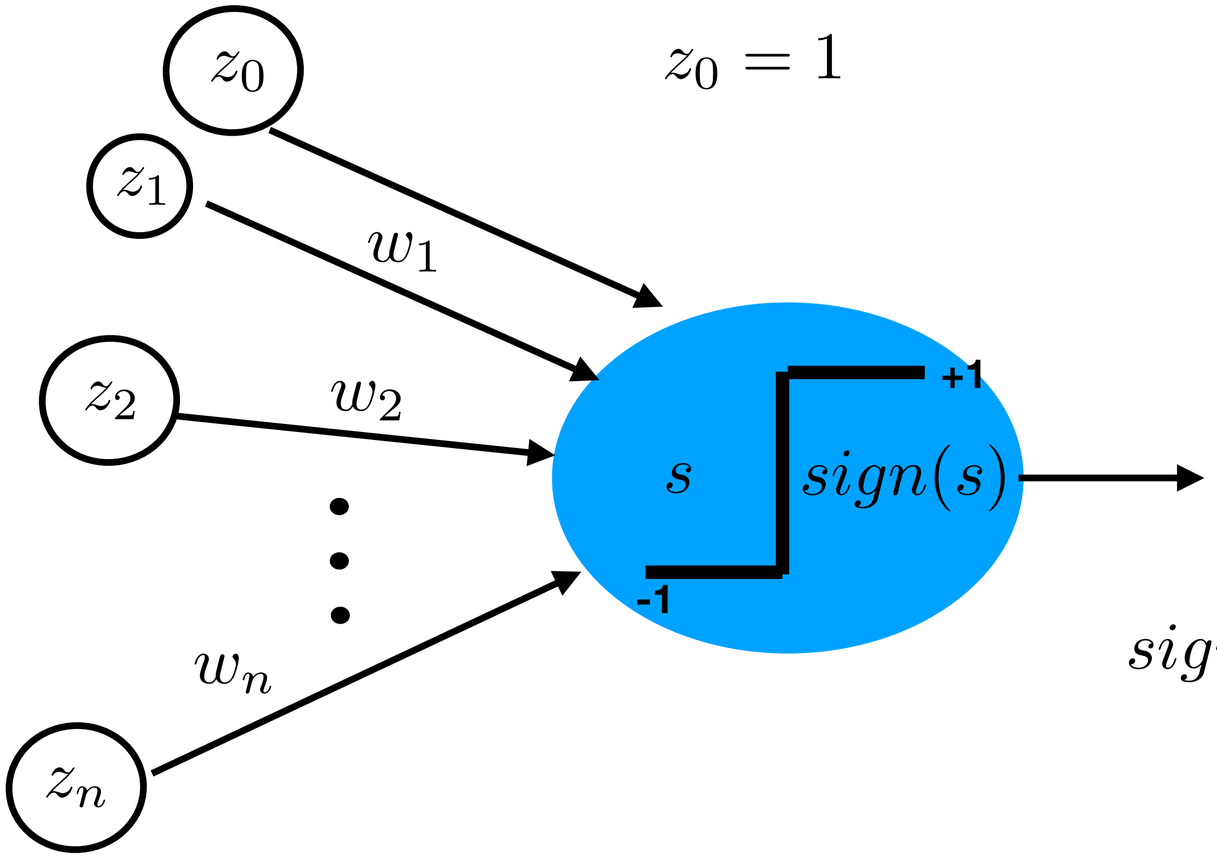
Perceptron (linear classifier)

$$\mathbf{z} = (z_1, z_2, \ldots, z_n)$$

$$z_0 = 1$$

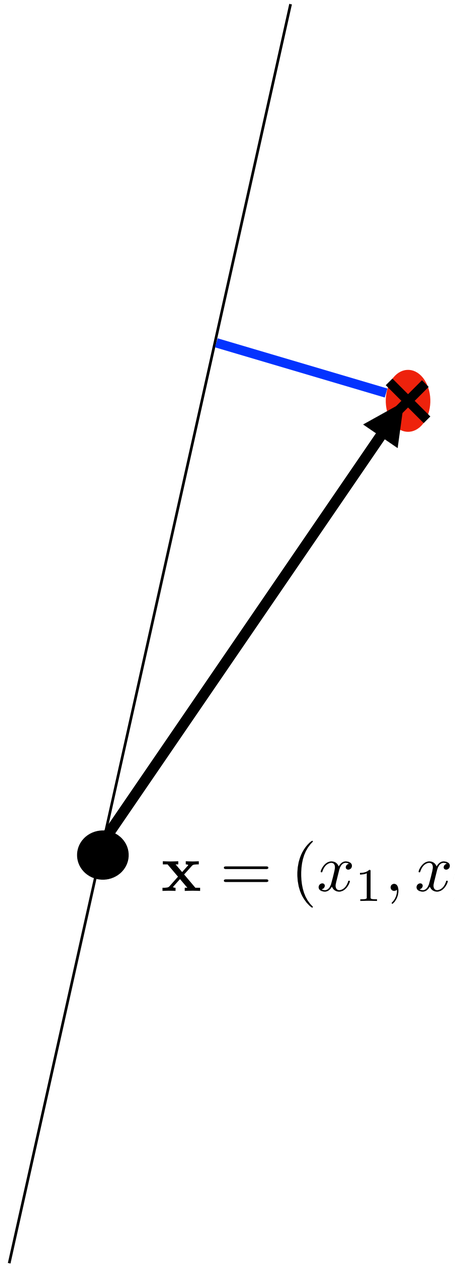
$$h(z) = \quad sign(\mathbf{w} \cdot \mathbf{z})$$

$$s = \sum_{i=1}^{n} w_i z_i + w_0$$



$$sign(s) = \begin{cases} +1, & s > 0 \\ -1, & s < 0 \end{cases}$$

Eventually, this classifier should give the same sign for all x's and the same sign for all o's.



On the hyper-plane:

$$\sum_{i=1}^{n} w_i x_i + w_0 = 0$$

$$\mathbf{N} = (w_1, w_2, \ldots, w_n)$$

$$\mathbf{z} = (z_1, z_2, \ldots, z_n)$$

the distance = $\left|(\mathbf{z} - \mathbf{x}) \cdot \mathbf{N}\right|$ = - $y \left(\sum_{i=1}^{n} z_i w_i + w_0 \right)$

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)$$

Stochastic gradient descent

Pick a misclassified point randomly

Calculate the error

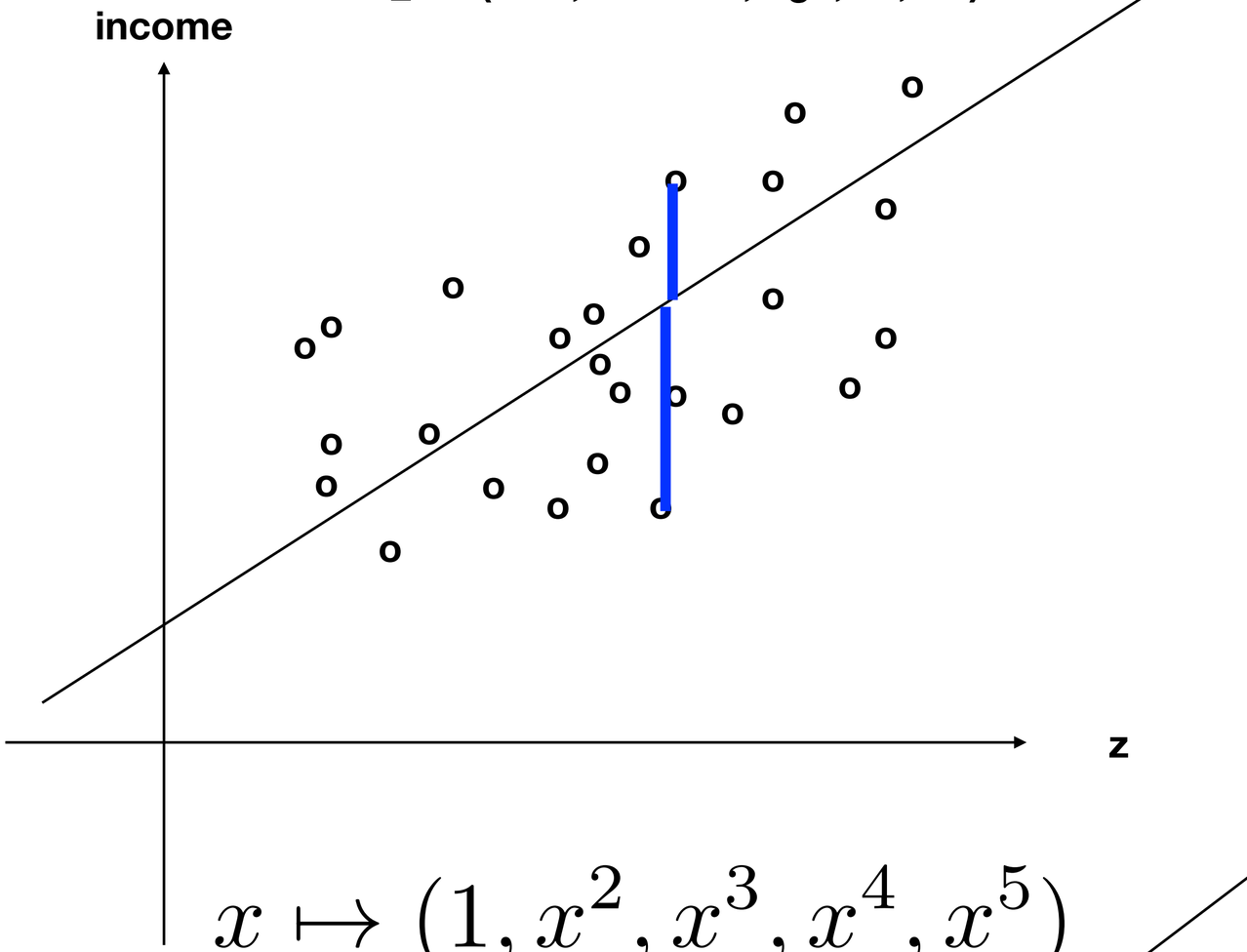
Update the weights using gradient descent to decrease the error (do not forget to normalize the weights)

Continue this process until there is no misclassified point in the training data set

Homework: Write a python code to implement the above algorithm for the linear perceptron and show the performance on a linearly separable data set

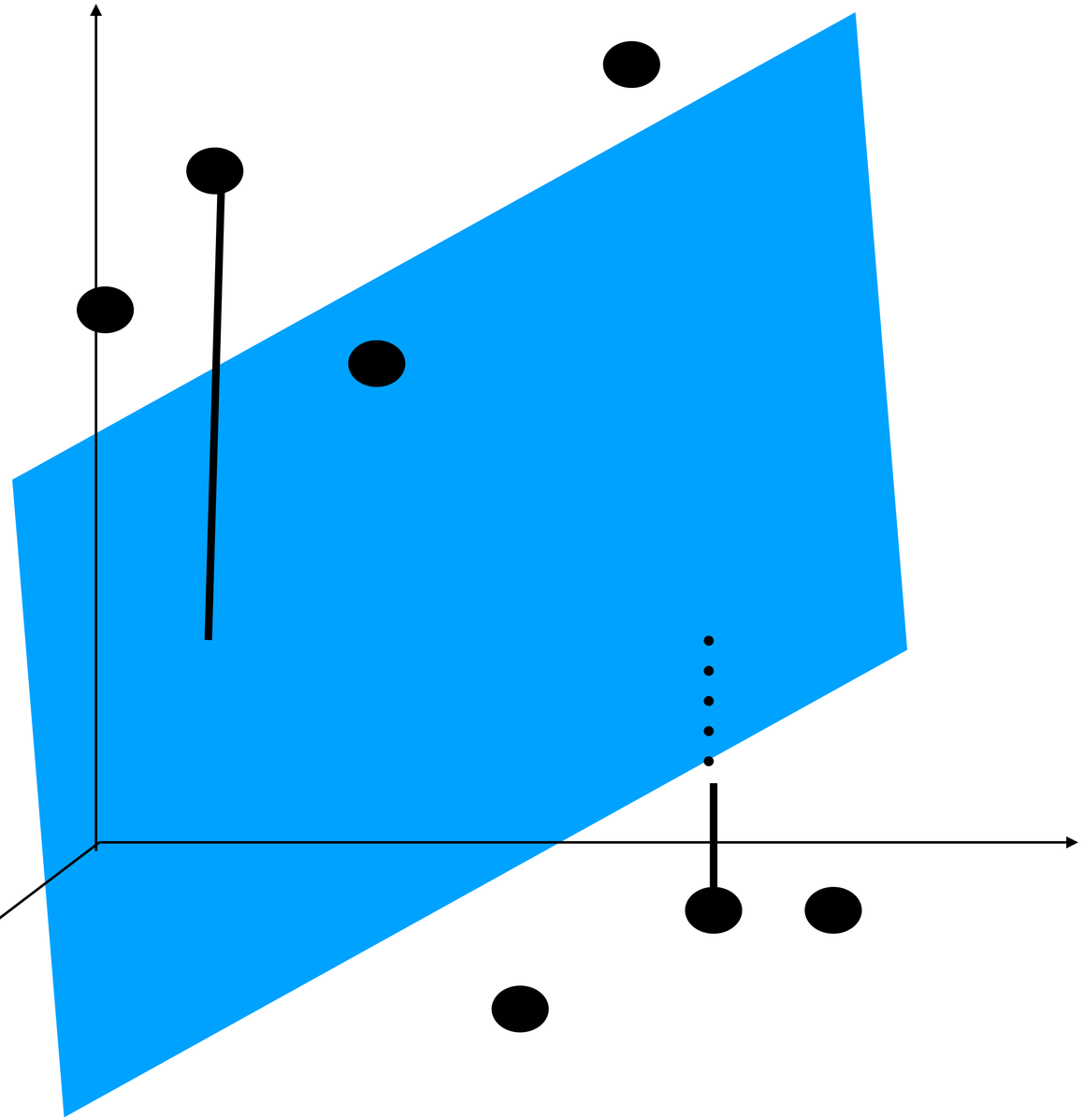
Linear regression: rather than classification, the goal is to fit the data and produce real-valued output

z_n = (GPA, courses, age, IQ, ….)



$$x \mapsto (1, x^2, x^3, x^4, x^5)$$

$$(x_1, x_2) \mapsto (1, x_1, x_1^2, x_2, x_2^2, x_1 x_2)$$



Application of linear regression: prediction

For example to predict the future income of a student based on the grades and courses

Historical data: $(\mathbf{z_1}, y_1), (\mathbf{z_1}, y_1), \ldots, (\mathbf{z_N}, y_N)$

$\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_N}$ **vectors in** $\{1\} \times \mathbb{R}^n$

$y_1, y_2, \ldots, y_N$ **real numbers**

Goal: Find the weights $\mathbf{w} = (w_0, w_1, \ldots, w_n)$

such that the hyper-plane defined by

$$\mathbf{w} \cdot \mathbf{x} = 0 \quad \Longleftrightarrow \quad w_1 x_1 + \cdots + w_n x_n + \; w_0 x_0 = 0$$

$$\mathbf{x} = (x_0, x_1, \ldots, x_n)$$

$$x_0 = 1$$

has the minimum sum of squares of distances for the historical data

$$(\mathbf{z_1}, y_1), (\mathbf{z_1}, y_1), \ldots, (\mathbf{z_N}, y_N)$$

Sum of squares of distances

$$d = \sum_{i=1}^{N} (\mathbf{w} \cdot \mathbf{z_i} - y_i)^2$$

Note: remember that 1 is the first coordinate of each of

$$\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_N}$$

The vector of weights can be found by solving:

$$\nabla d(\mathbf{w}) = 0$$

This is a particular situation!

Goal: Find w in such a way that d is the minimum possible!

our predictor will look like this:

$$h(z) = \mathbf{w} \cdot \mathbf{z} = \sum_{i=0}^{n} w_i z_i$$

Homework: write the linear regression algorithm in python and use it for prediction

$$\mathbf{w} = Z^{\dagger}\mathbf{y}$$

The minimizing vector of weights

shape of the pseudo inverse = (n+1)*N

shape of w = (n+1)*1

$$Z^{\dagger} = \left(Z^{T}Z\right)^{-1}Z^{T}$$

Pseudo-inverse of Z

(n+1)*N

$$Z = \begin{bmatrix} \mathbf{z_1} \\ \mathbf{z_2} \\ . \\ . \\ . \\ \mathbf{z_N} \end{bmatrix}_{N \times (n+1)}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_N \end{bmatrix}_{N \times 1}$$

shape of the Z^T Z= (n+1)*(n+1)

Matrix multiplication

$$C = AB$$

Suggested reading:
matrices from quantum mechanics (Heisenberg)

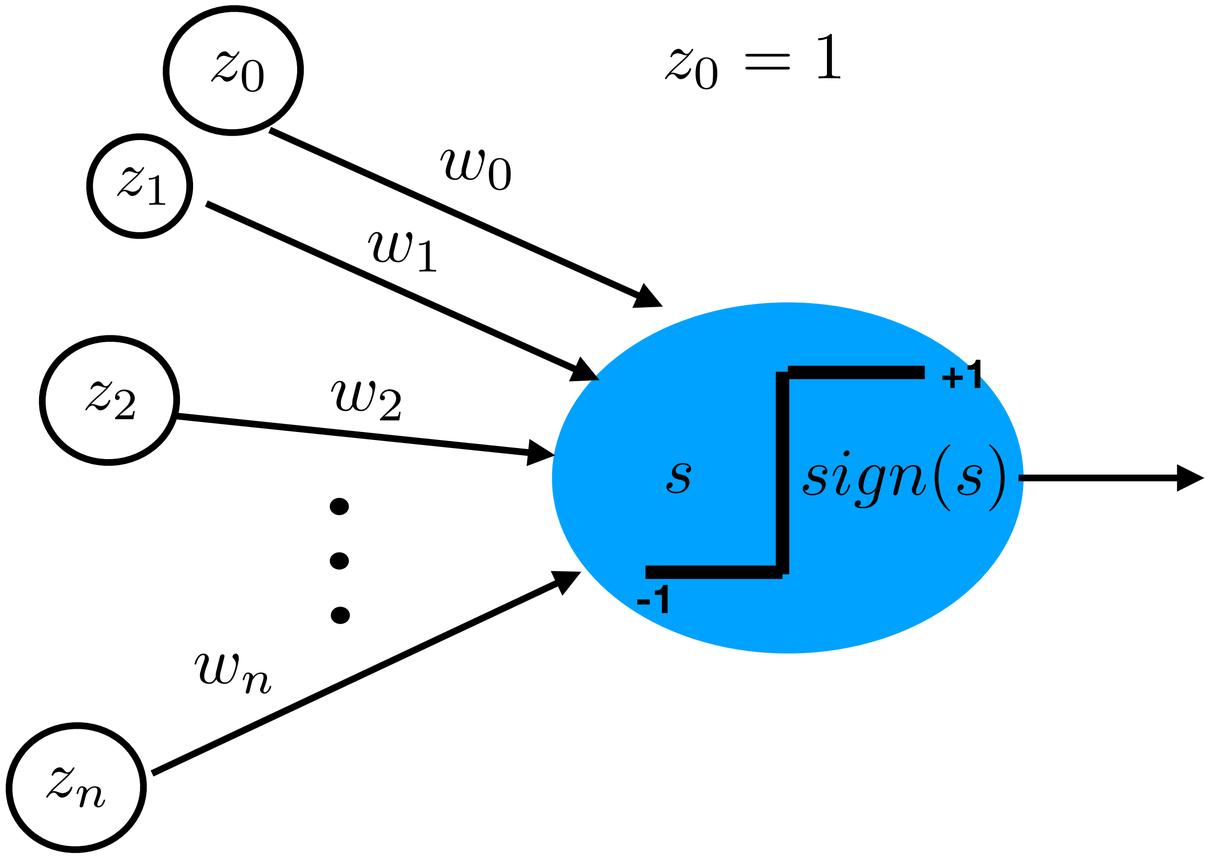
$$C_{m\times n} = A_{m\times k}B_{k\times n}$$

C_{ij} **= the element in row i and column j of the matrix C**

the dot product of of i-th row A with the j-th column of B

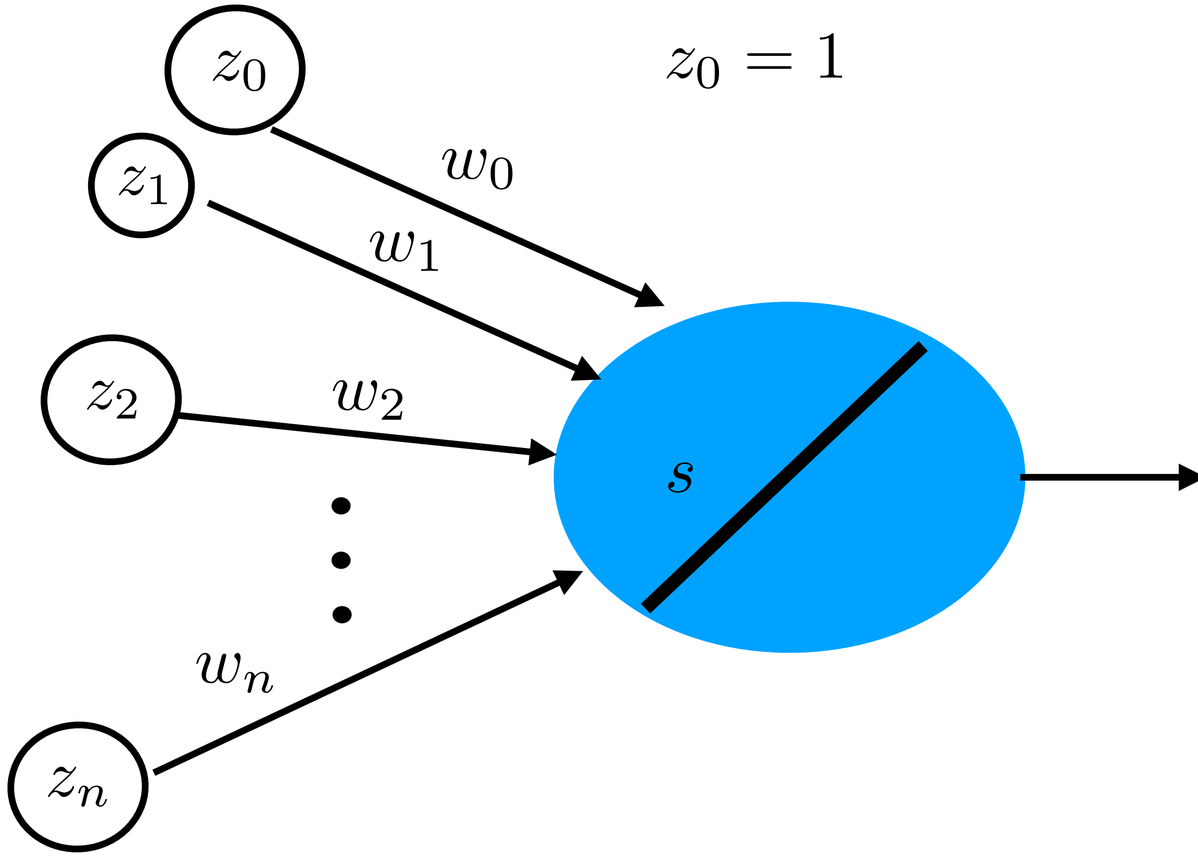
$$\begin{pmatrix} 1 & -1 \\ 4 & 2 \\ 1 & 7 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & -1 \\ 5 & 6 & 1 & 2 \end{pmatrix} = \begin{pmatrix} -4 & -4 & 2 & -3 \\ 14 & 20 & 14 & 0 \\ 36 & 44 & 10 & 13 \end{pmatrix}$$

linear perception: classifier



$$h(z) = sign(\mathbf{w} \cdot \mathbf{z})$$

linear regression: prediction: output is a real number



$$h(z) = \mathbf{w} \cdot \mathbf{z}$$

Statistical inference

$$p = Probability(X|H)$$ **< 0.005**

X = observation

H = an assumption about X

infer: the assumption is wrong!!!!!!!!

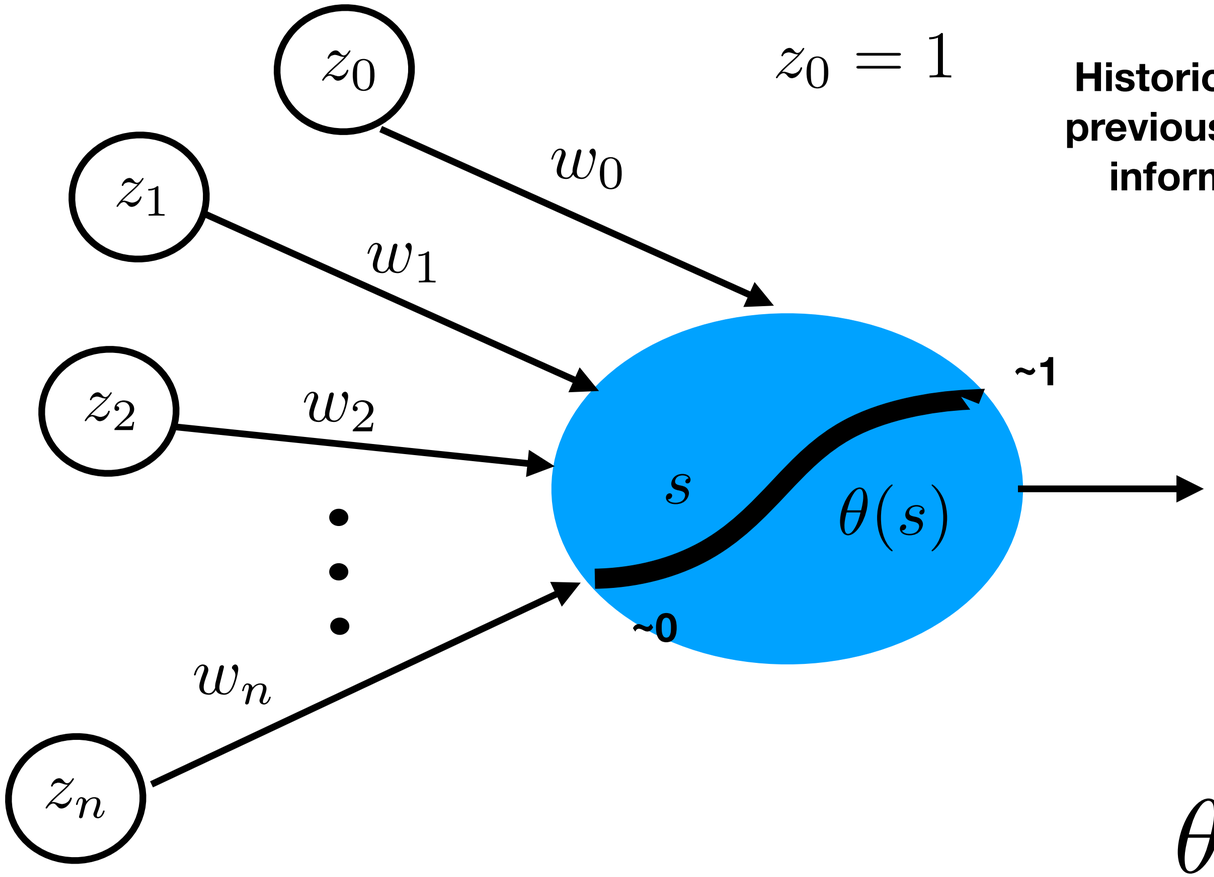
Maximum Likelihood

$$p = prob(X|f_\Theta)$$

Θ **is unknown, and we wish to identify it**

This is done by maximizing p: finding the Theta that maximizes the probability p of observing X

Logistic regression: the goal is to give a probability



$z_0 = 1$

Historical data: previous patient information

$\mathbf{z} = (z_1, z_2, \ldots, z_n)$

y **= +1 or -1**

Goal: $P(y|\tilde{\mathbf{z}}) = \theta(\mathbf{w} \cdot \tilde{\mathbf{z}})$

$\mathbf{w}$ **will be identified using the historical or training data and and maximum likelihood**

$$\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{e^{-s}+1}$$

The logistic function or the sigmoid function

Historical data: $(\mathbf{z_1}, y_1), (\mathbf{z_1}, y_1), \ldots, (\mathbf{z_N}, y_N)$

$\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_N}$ **vectors in** $\{1\} \times \mathbb{R}^n$

$y_1, y_2, \ldots, y_N$ **are +1 or -1**

Exercise:

$$\theta(-s) = 1 - \theta(s)$$

$$P(y|\mathbf{z}) = \begin{cases} \theta(\mathbf{w} \cdot \mathbf{z}), & y = +1 \\ 1 - \theta(\mathbf{w} \cdot \mathbf{z}), & y = -1 \end{cases}$$

$$= \theta(y\mathbf{w} \cdot \mathbf{z})$$

Likelihood of the historical data:

$$\prod_{i=1}^{N} P(y_i|\mathbf{z}_i) = P(y_1|\mathbf{z}_1)P(y_2|\mathbf{z}_2)\cdots P(y_N|\mathbf{z}_N)$$

$$= \prod_{i=1}^{N} \theta(y_i\mathbf{w} \cdot \mathbf{z}_i)$$

Goal: chose the weights (w) such that this likelihood is maximized.

$$\ell = \prod_{i=1}^{N} \theta(y_i \mathbf{w} \cdot \mathbf{z}_i)$$ **Maximize is equivalent to**

$$\ln(\ell)$$ **maximizing this, which is equivalent to**

$$-\ln(\ell)$$ **minimizing this**

$$-\frac{1}{N}\ln(\ell) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(\frac{1}{\theta(y_i \mathbf{w}\cdot\mathbf{z}_i)}\right) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y_i \mathbf{w}\cdot\mathbf{z}_i}\right)$$

risk score;

average of cross entropy errors

"Error of logistic regression" $= \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y_i \mathbf{w}\cdot\mathbf{z}_i}\right)$

Now we can try to minimize the error!

Exercise: ∇ **"Error of logistic regression"** $= \nabla E_L$

$$= -\frac{1}{N}\sum_{i=1}^{N}\frac{y_i \mathbf{z}_i}{1+e^{y_i \mathbf{w}\cdot\mathbf{z}_i}}$$

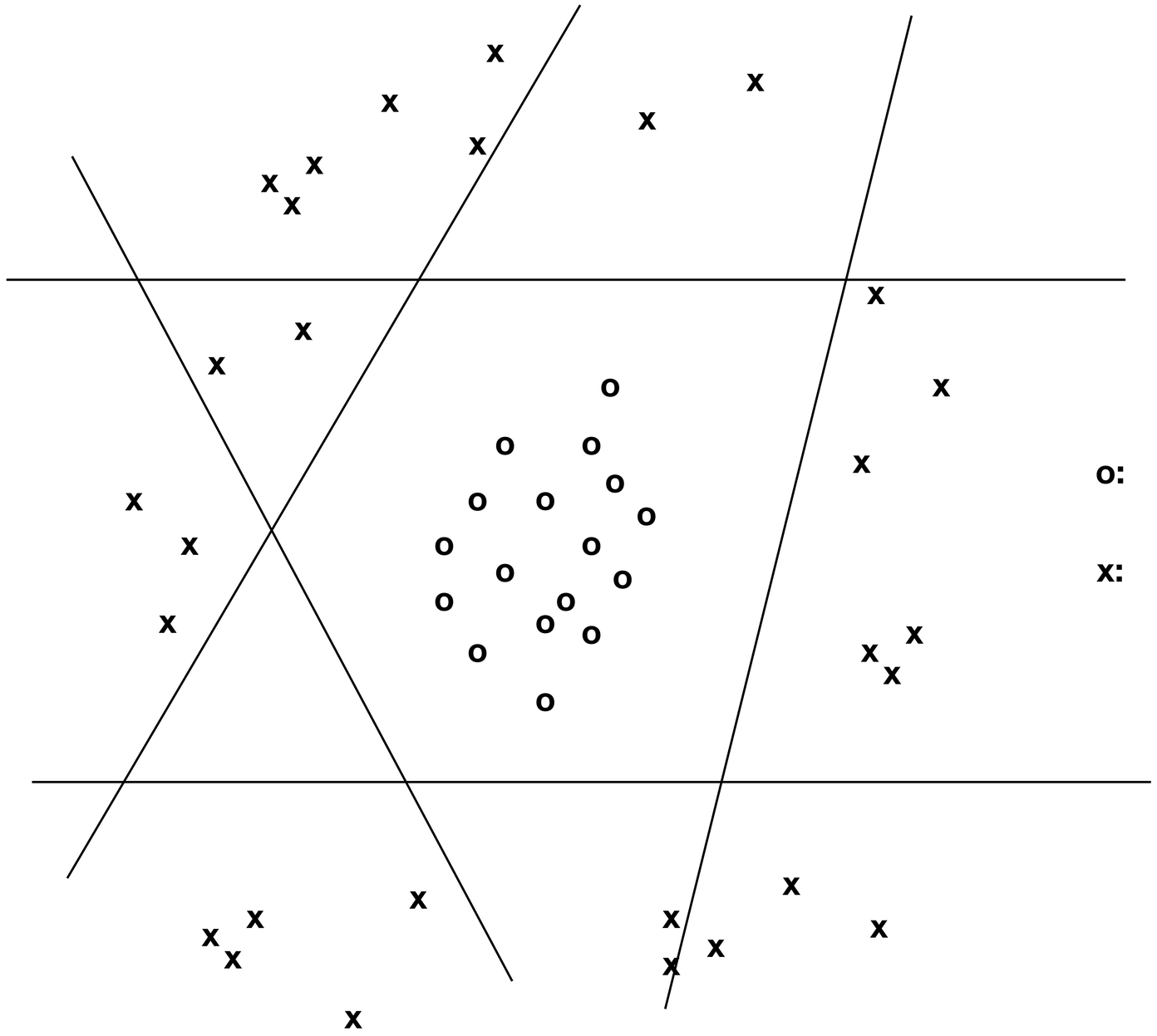
Linear regression algorithm

1. Choose the weights of the vector w randomly

2. compute the "error" $= \frac{1}{N}\sum_{i=1}^{N}\ln\left(1+e^{-y_i \mathbf{w}\cdot\mathbf{z}_i}\right)$

3. Update the weights using the formula $\mathbf{w} \rightarrow \mathbf{w} - \eta\nabla\text{“}error\text{”}$

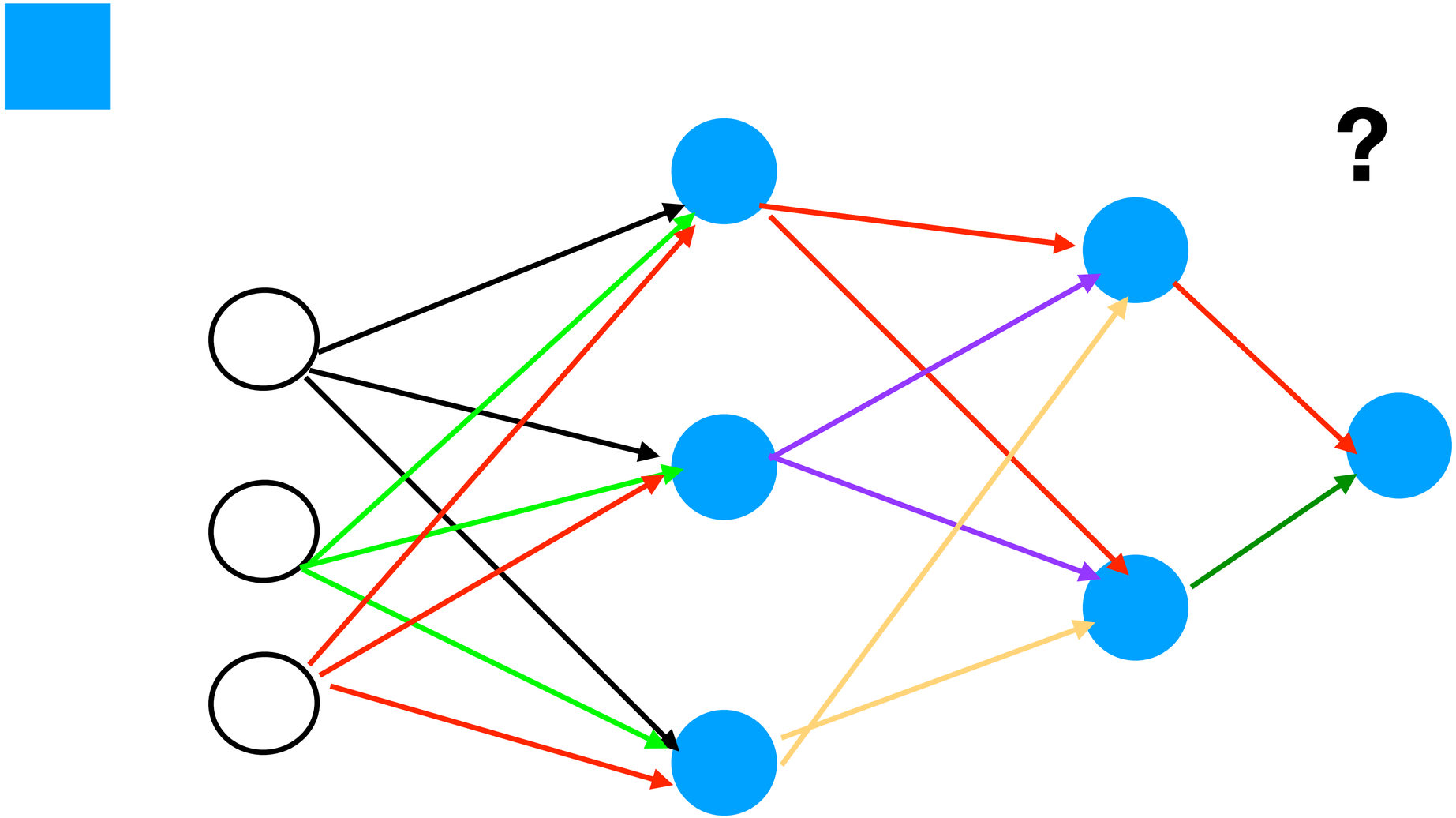
4. Iterate the same method until it is time to stop.

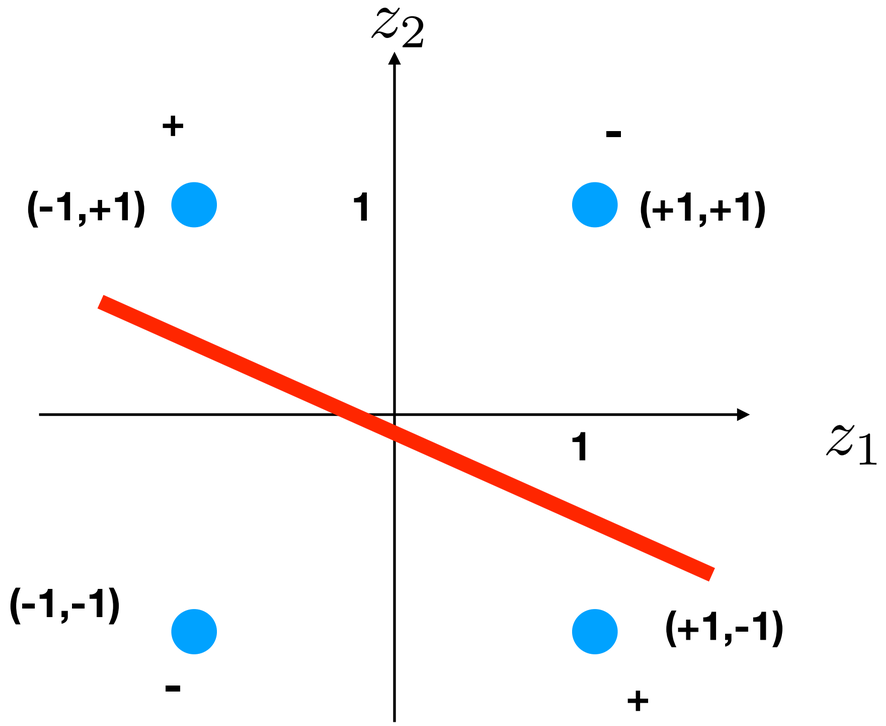


$x_1^2 + x_2^2 = r^2$

o: $x_1^2 + x_2^2 < r^2$

x: $x_1^2 + x_2^2 > r^2$





XOR function is defined on 4 points: (-1, -1), (+1, -1), (+1, +1), (-1, +1)

$$\mathbf{z} = (z_0 = 1, z_1, z_2)$$

Question: is the XOR function a linear perceptron?

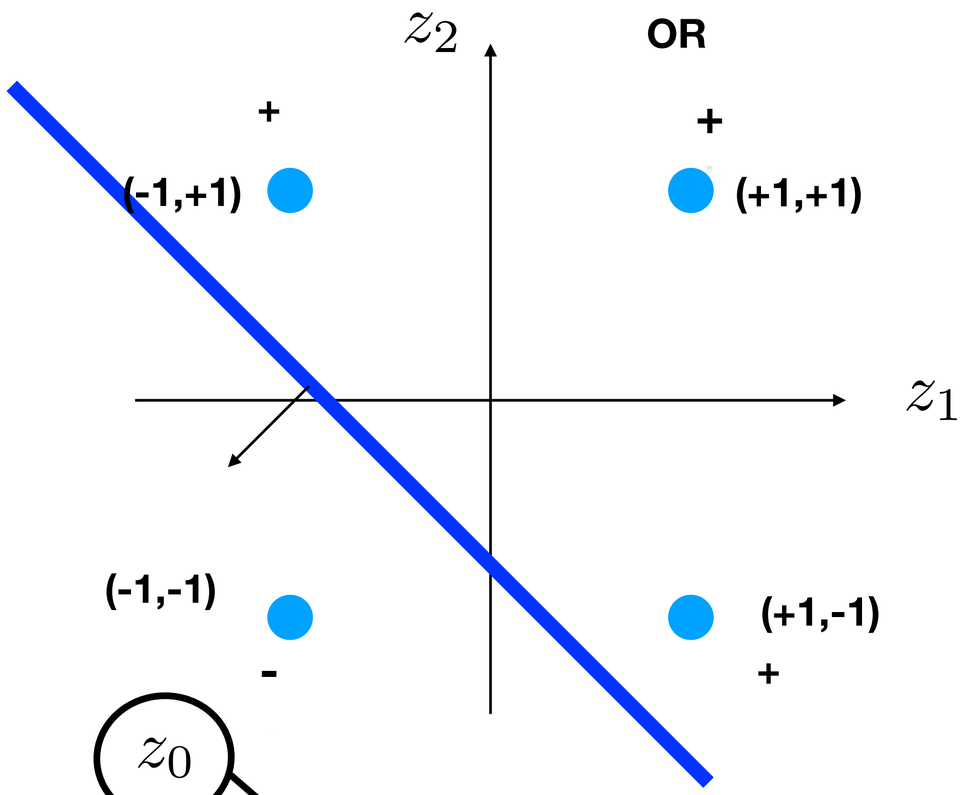
Can we write:

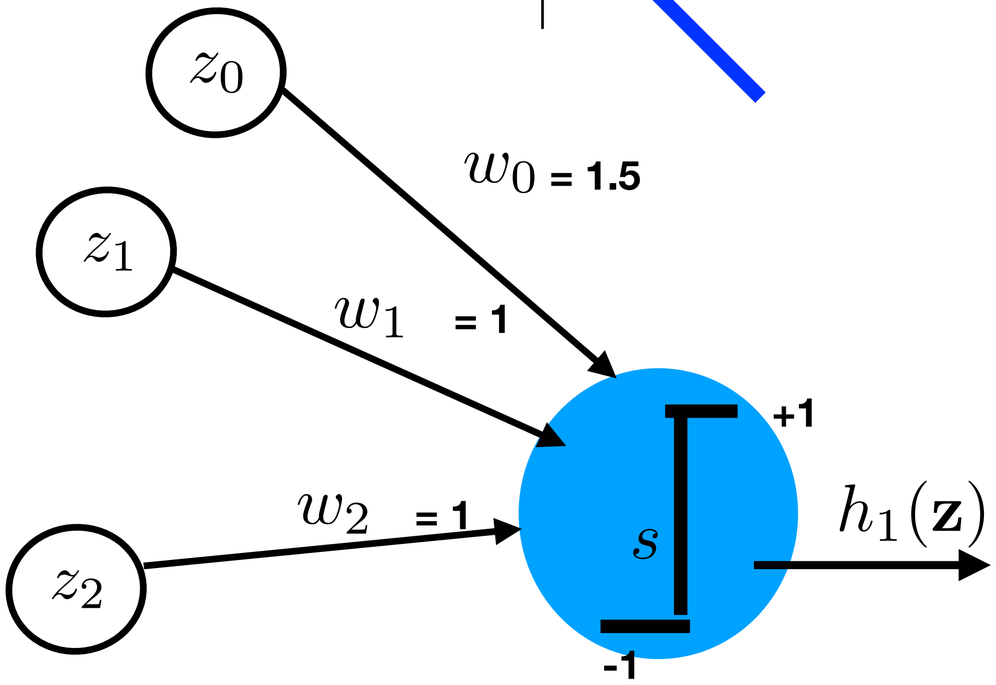
$$XOR(\mathbf{z}) = sing(\mathbf{w} \cdot \mathbf{z}) \quad ?$$

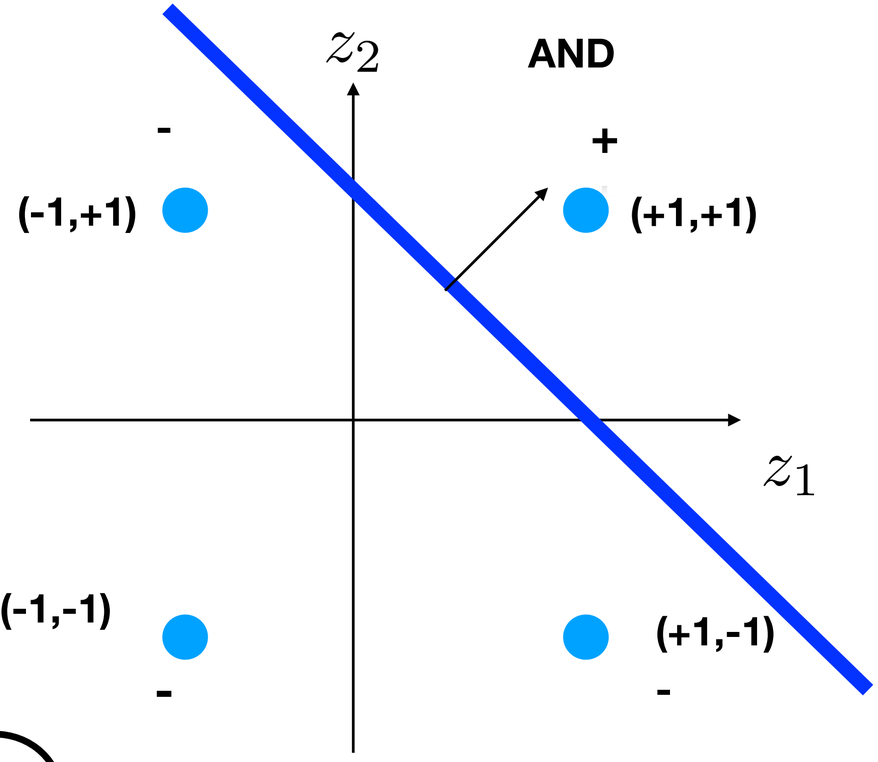
Answer: NO because there is no straight line classifying the + and -'s in our graph.

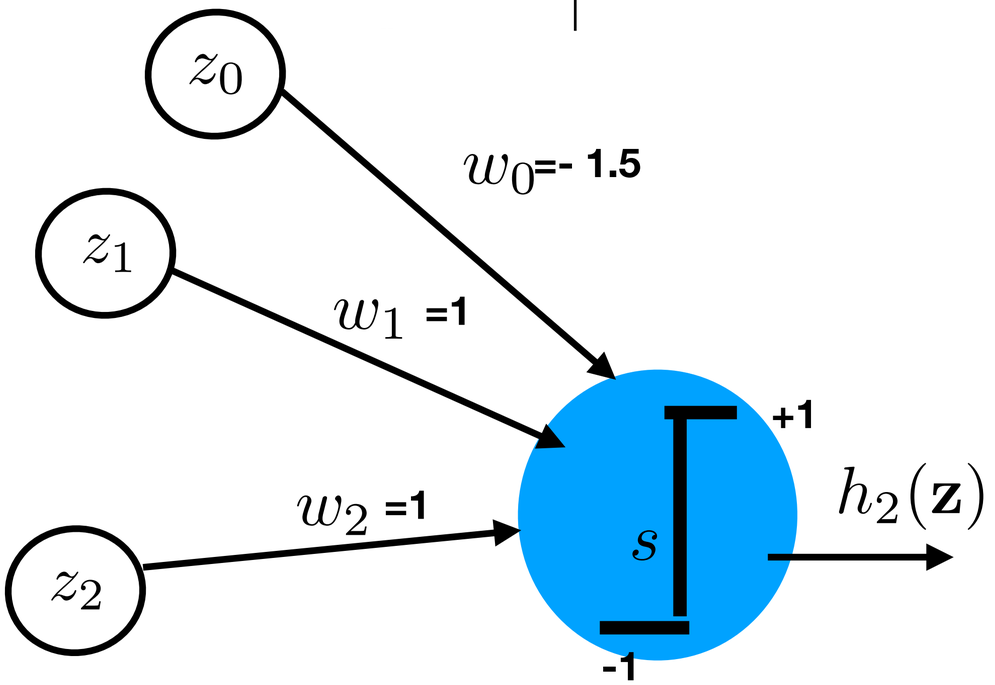
$$XOR(\mathbf{z}) = \begin{cases} +1, & \text{if exactly one coordinate is 1} \quad (z_1 \text{ or } z_2) \\ -1 & \text{otherwise} \end{cases}$$

Question: Can we use a combination of linear perceptrons to construct the XOR function?
Yes, because the AND and OR functions can be written as linear perceptrons.


z_2
OR
+
(-1,+1)
+
(+1,+1)
z_1
(-1,-1)
-
(+1,-1)
+


z_0
w_0 = 1.5
z_1
w_1 = 1
w_2 = 1
z_2
+1
s
-1
$h_1(\mathbf{z})$

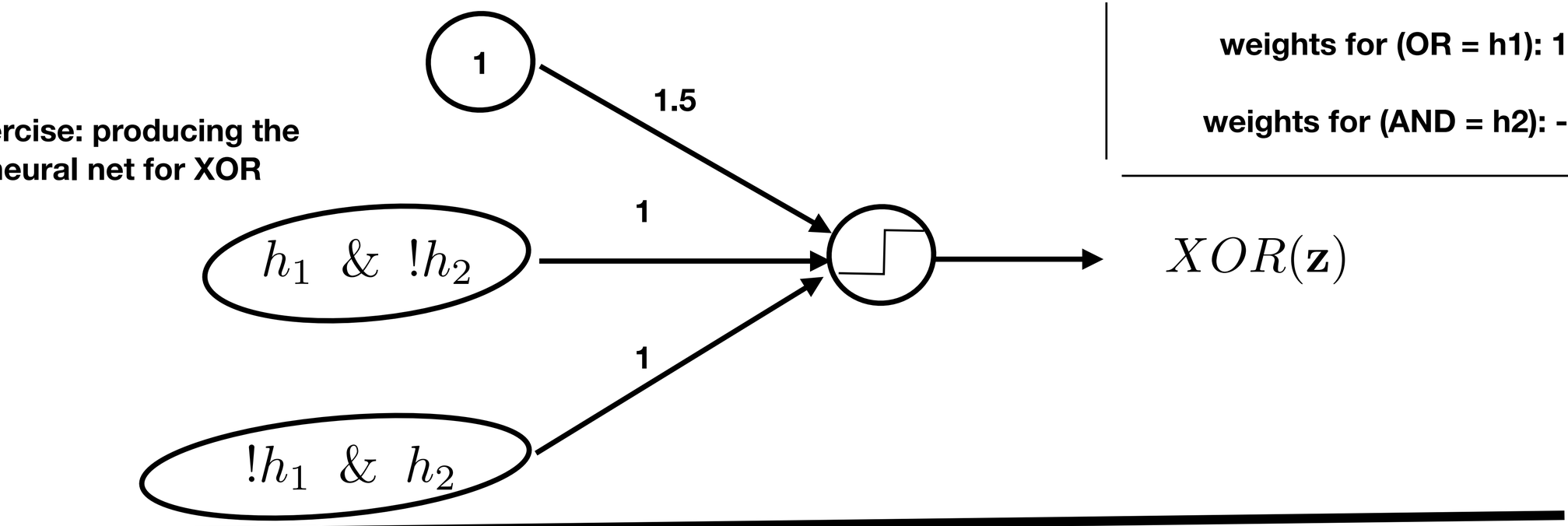

z_2
AND
-
(-1,+1)
+
(+1,+1)
z_1
(-1,-1)
-
(+1,-1)
-

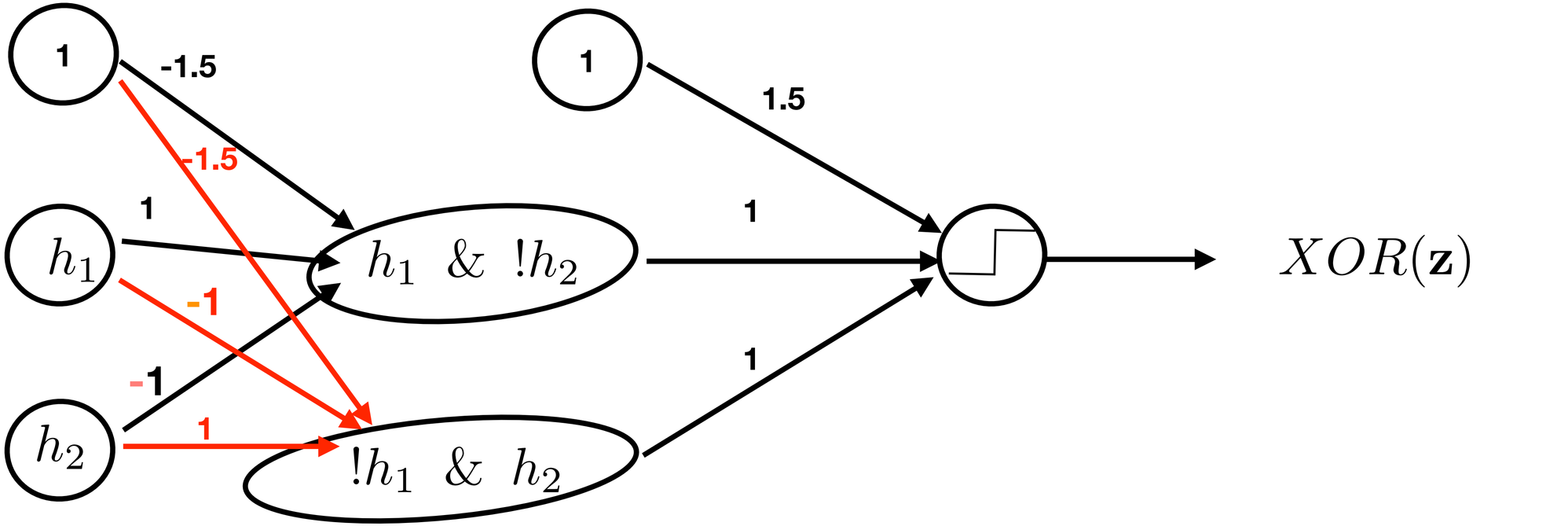

z_0
w_0=- 1.5
z_1
w_1 =1
w_2 =1
z_2
+1
s
-1
$h_2(\mathbf{z})$

Exercise: producing the neural net for XOR

weights for (OR = h1): 1.5, 1, 1

weights for (AND = h2): -1.5, 1, 1





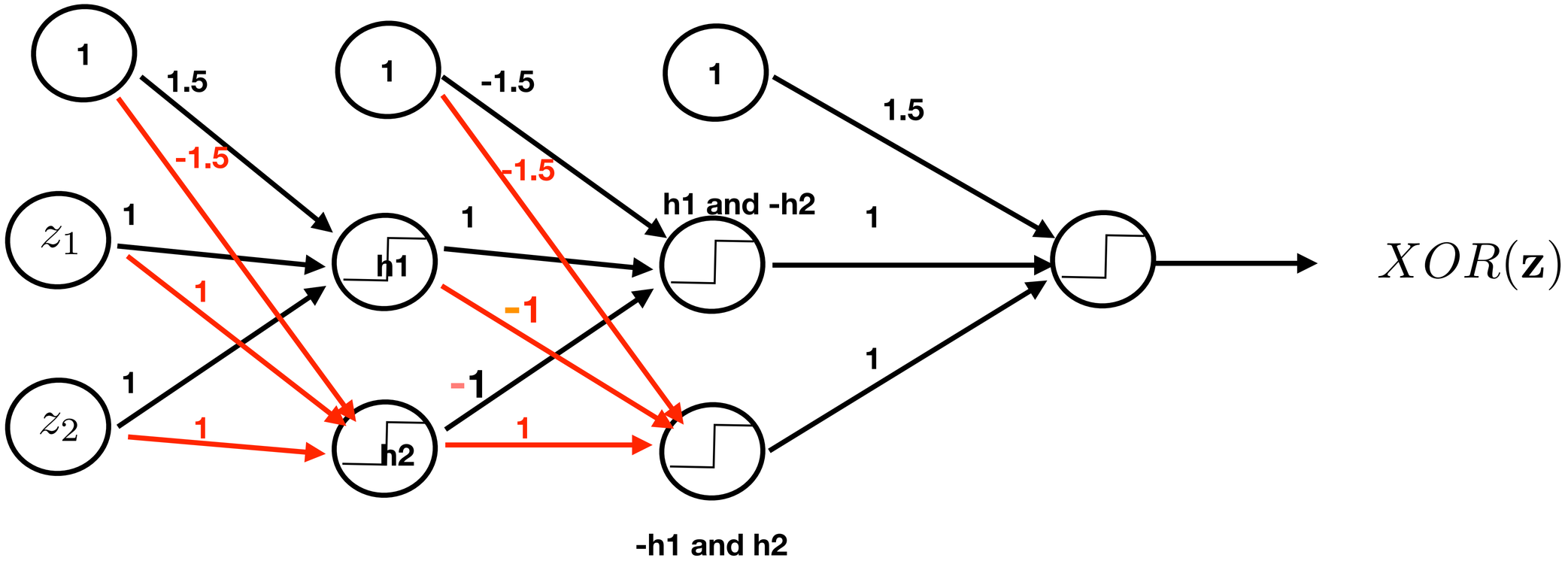
XOR

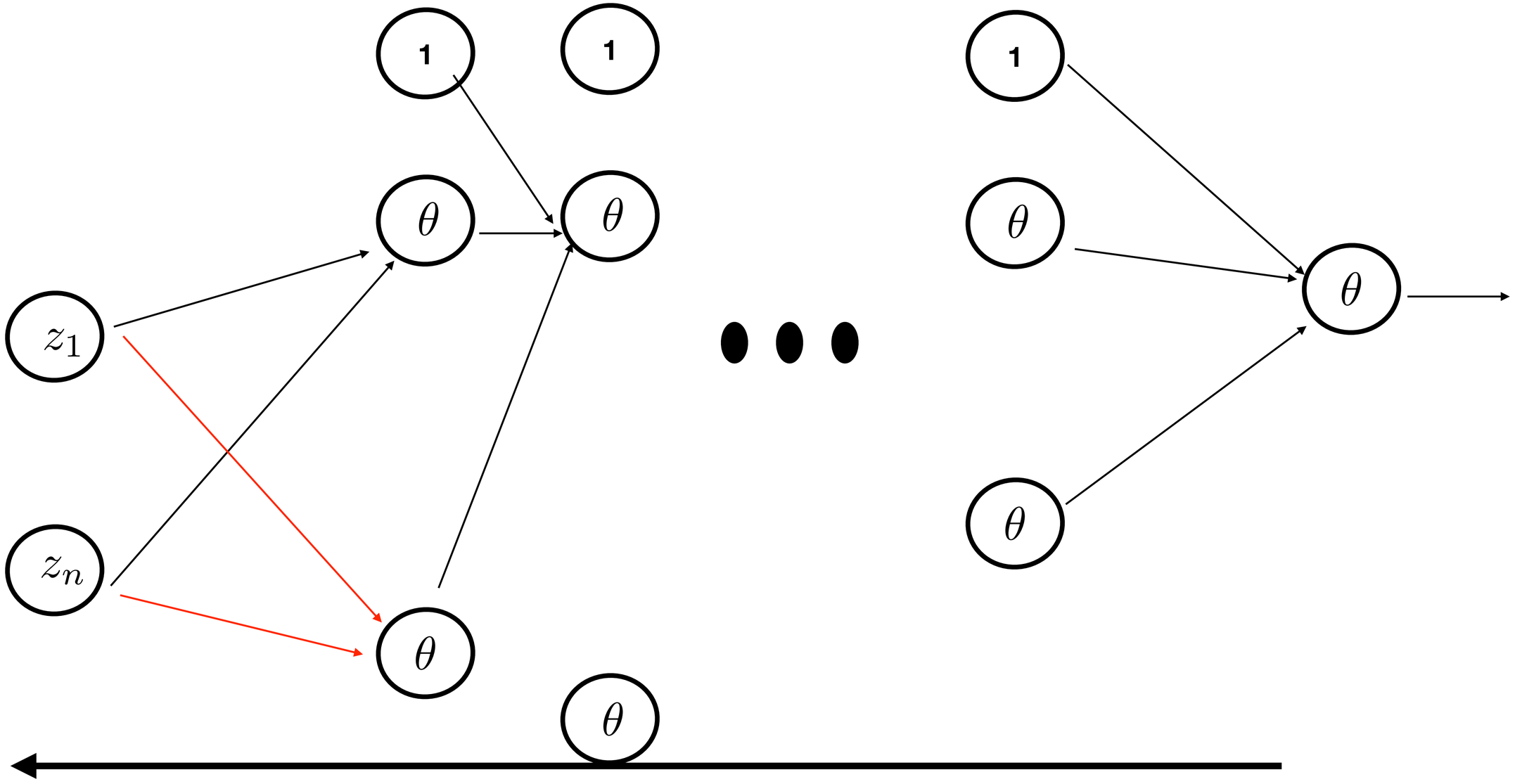
weights for (OR = h1): 1.5, 1, 1

weights for (AND = h2): -1.5, 1, 1

((2=3) or (3=3)) and ((3=4) and 3=3) False = -1

((2=3) and (3=3)) or ((3=4) or 3=3) True= +1





$$1 \leq \ell \leq L$$

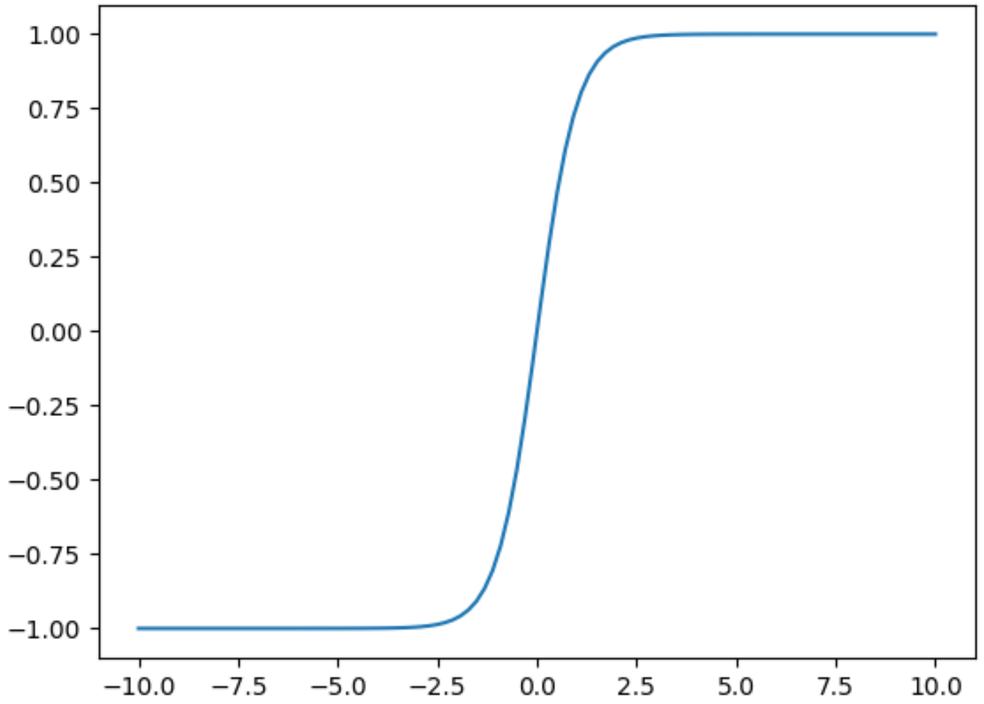
Input layer: layer 0

Hidden layers: 1 to L

(output layer: the last hidden layer)

$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

$$\theta'(s) = 1 - \theta(s)^2$$



Training a neural network

Training data: $(\mathbf{z_1}, y_1), (\mathbf{z_1}, y_1), \ldots, (\mathbf{z_N}, y_N)$

i input of the layer
j output from the layer

weights: $\mathbf{w} = \left(w_{ij}^{(\ell)}\right) \qquad 1 \leq \ell \leq L \qquad 0 \leq i \leq d^{(\ell-1)} \qquad 1 \leq j \leq d^{(\ell)}$

number of neurons in layer \ell

$$z_j^{(\ell)} = \theta(s_j^{(\ell)}) = \theta\left(\sum_{i=0}^{d^{(\ell-1)}} w_{ij}^{(\ell)} z_i^{(\ell-1)}\right)$$

Apply this to a training data to get an output $h(\mathbf{z}) = z_1^{(L)}$

measure the error $e(\mathbf{w}) = e(h(\mathbf{z}), y)$ **for example** $|h(\mathbf{z}) - y|^2$

use gradient descent to decrease the error by updating the weights

Back propagation: a fast algorithm for implementing the gradient descent to neural networks

$$\nabla e(\mathbf{w})$$

$$\frac{\partial e(\mathbf{w})}{\partial w_{ij}^{(\ell)}} = \frac{\partial e(\mathbf{w})}{\partial s_j^{(\ell)}} \frac{\partial s_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} = \frac{\partial e(\mathbf{w})}{\partial s_j^{(\ell)}} z_i^{(\ell-1)} = \delta_j^{(\ell)} z_i^{(\ell-1)}$$

$$\frac{\partial e(\mathbf{w})}{\partial s_j^{(\ell)}} = \delta_j^{(\ell)}$$

In the output layer: $\ell = L, j = 1$

$$\delta_1^{(L)} = \frac{\partial e(\mathbf{w})}{\partial s_1^{(L)}}$$

For example: $e(\mathbf{w}) = (z_1^{(L)} - y)^2$

$$z_1^{(L)} = \theta(s_1^{(L)})$$

Back propagate

$$\delta_i^{(\ell-1)} = \frac{\partial e(\mathbf{w})}{\partial s_i^{(\ell-1)}} =$$

$$= \sum_{j=1}^{d^{(\ell)}} \frac{\partial e(\mathbf{w})}{\partial s_j^{(\ell)}} \frac{\partial s_j^{(\ell)}}{\partial z_i^{(\ell-1)}} \frac{\partial z_i^{(\ell-1)}}{\partial s_i^{(\ell-1)}}$$

$$= \sum_{j=1}^{d^{(\ell)}} \delta_j^{(\ell)} \, w_{ij}^{(\ell)} \, \theta'(s_i^{(\ell-1)})$$

$$\delta_i^{(\ell-1)} = (1 - (z_i^{(\ell-1)})^2) \sum_{j=1}^{d^{(\ell)}} w_{ij}^{(\ell)} \, \delta_j^{(\ell)}$$

Back propagation algorithm for deep forward neural networks

1. Initialize the weights $w_{ij}^{(\ell)}$ at RANDOM

2. Pick a training point $(\mathbf{z}, y)$

3. Go forward in the NN by calculating all $z_j^{(\ell)}$ until you get the output for the chosen trining point

4. Use back propagation to calculate all $\delta_j^{(\ell)}$

5. Update the weights by this formula: $w_{ij}^{(\ell)} \longrightarrow w_{ij}^{(\ell)} - \eta \delta_j^{(\ell)} z_i^{(\ell-1)}$

6. Iterate this method until it is time to stop.

7. Return the final weights.

Chain rule and partial differentiation

Regularization

Why learning is feasible?

$$y = f(\mathbf{z})$$

hypothesis set

learning algorithm finds an element g in $\mathcal{H}$

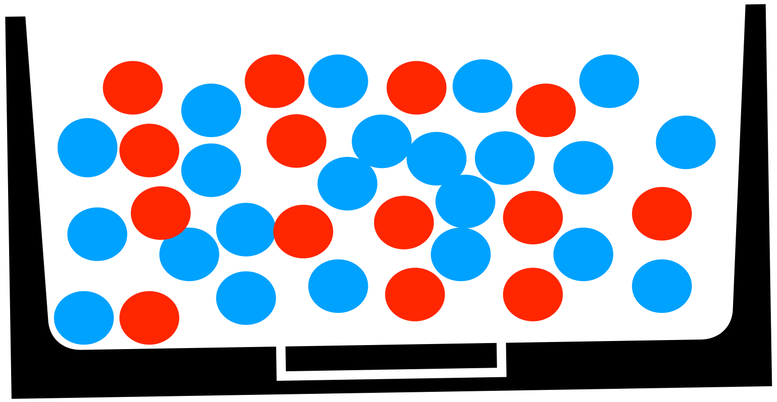
$$g \approx f$$

$$\mu = P(red)$$

$$1 - \mu = P(blue)$$

μ ?

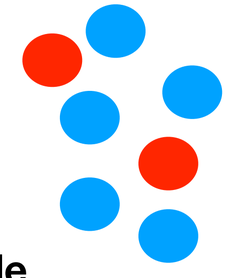
bin with blue and red marbles



sample (training)

ν

proportion of red in the sample



$\nu \longrightarrow \mu$? Probably

$$P(|\nu - \mu| > \epsilon) \leq 2e^{-2\epsilon^2 N}$$

N size of sample

Hoefdding'e inequality

Relation to learning

$h(z) = f(z) \qquad z$

$h(z) \neq f(z) \qquad z$

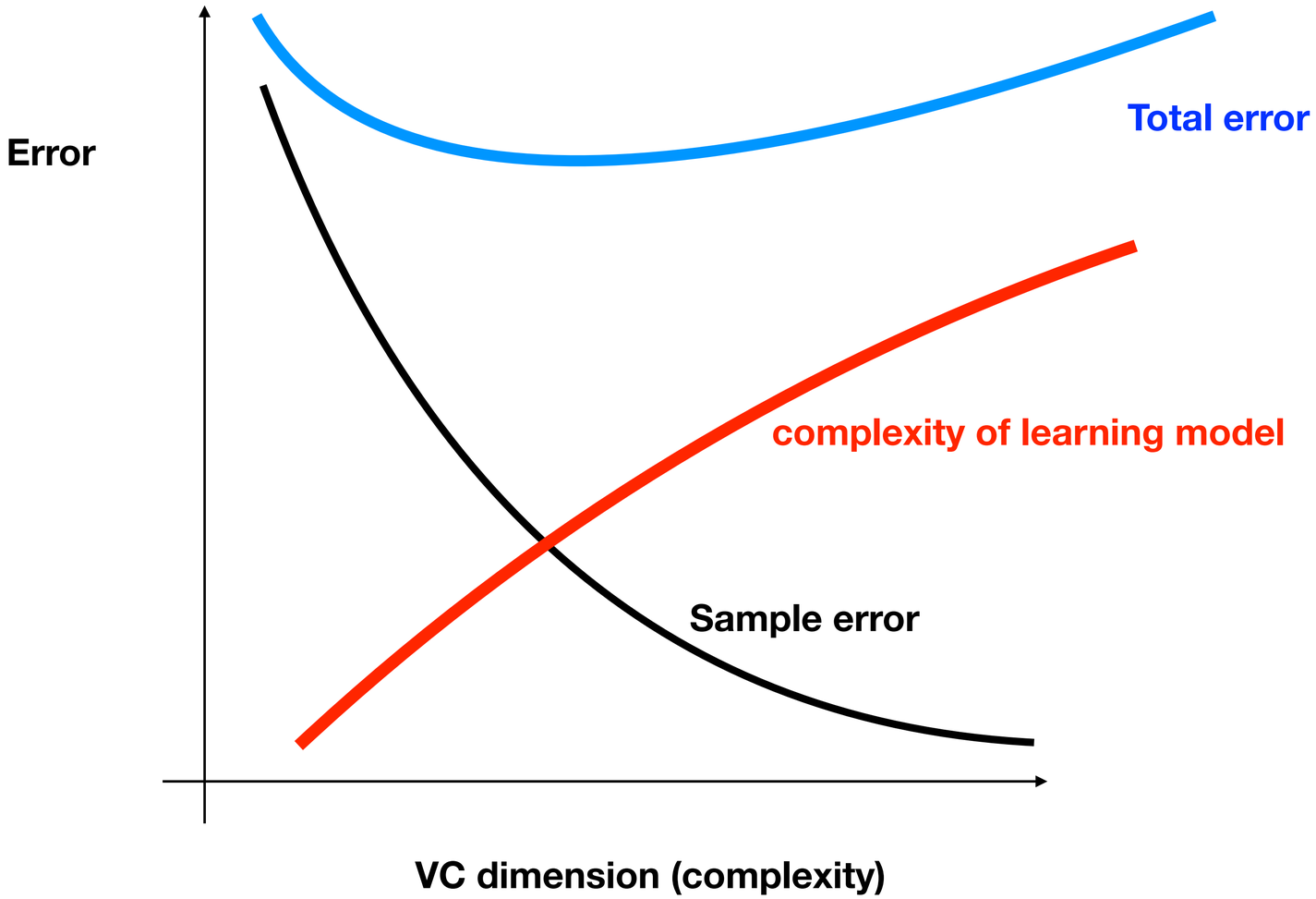
$$\mathcal{H} = \{h_1, h_2, \ldots, h_M\}$$

$$P(|E_{sample}(g) - E_{total}(g)| > \epsilon) \leq 2Me^{-2\epsilon^2 N}$$

Learning well-done if:

$$|E_{sample}(g) - E_{total}(g)| \quad \textbf{small}$$

$$E_{sample}(g) \quad \textbf{small}$$


Error
Total error
complexity of learning model
Sample error
VC dimension (complexity)

$$E_{total} \leq E_{sample} + \Omega$$

Probabilistic inequality for generalization bound

$$\Omega = \Omega(N, \mathcal{H}, \delta)$$

decreases as N increases

increases as VC dimension or complexity of learning model increases

The inequality holds with probability (at least)

$$1 - \delta$$

Bias-Variance tradeoff

$\mathcal{H}$ **large: more chances of having the target function there but takes a long time to get close to it, large total error**

small: better generalization; far from target

$$E_{total}(g^{(D)}) = \mathbb{E}\left((g^{(D)}(z) - f(z))^2\right)$$ **square error summation**

$$\bar{g}(z) = \mathbb{E}_D\left(g^{(D)}(z)\right) \approx \frac{1}{K}\sum_{k=1}^{K} g^{D_k}(z)$$

$D_1, D_2, \ldots, D_k$ **k data sets**

$$\mathbb{E}_D\left((g^{(D)}(z) - f(z))^2\right) = \mathbb{E}_D\left((g^{(D)}(z) - \bar{g}(z))^2\right) + (\bar{g}(z) - f(z))^2$$

= variance(z) + bias (z)

Variance by definition: how much we might vary!

how biased away from the target function we are!

$$\mathbb{E}_D\left(E_{out}(g^{(D)})\right) = \quad \textbf{bias + variance}$$

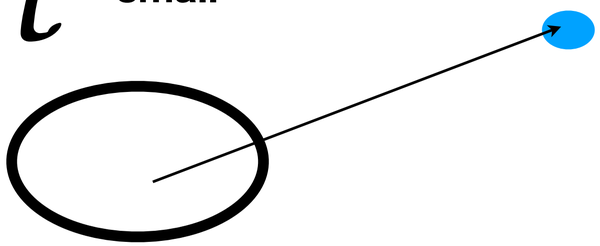
The choice of complexity should be determined by the available data.

big bias

small variance

target f

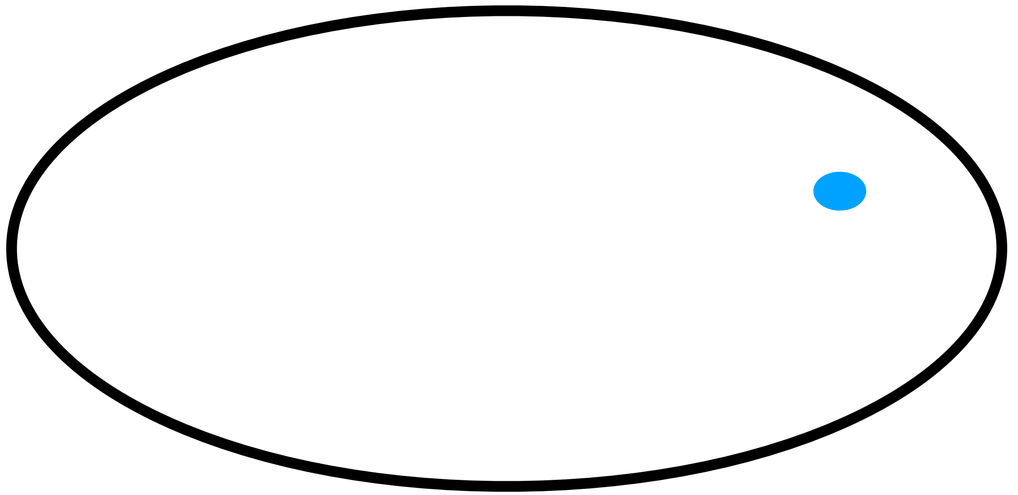
$\mathcal{H}$ small



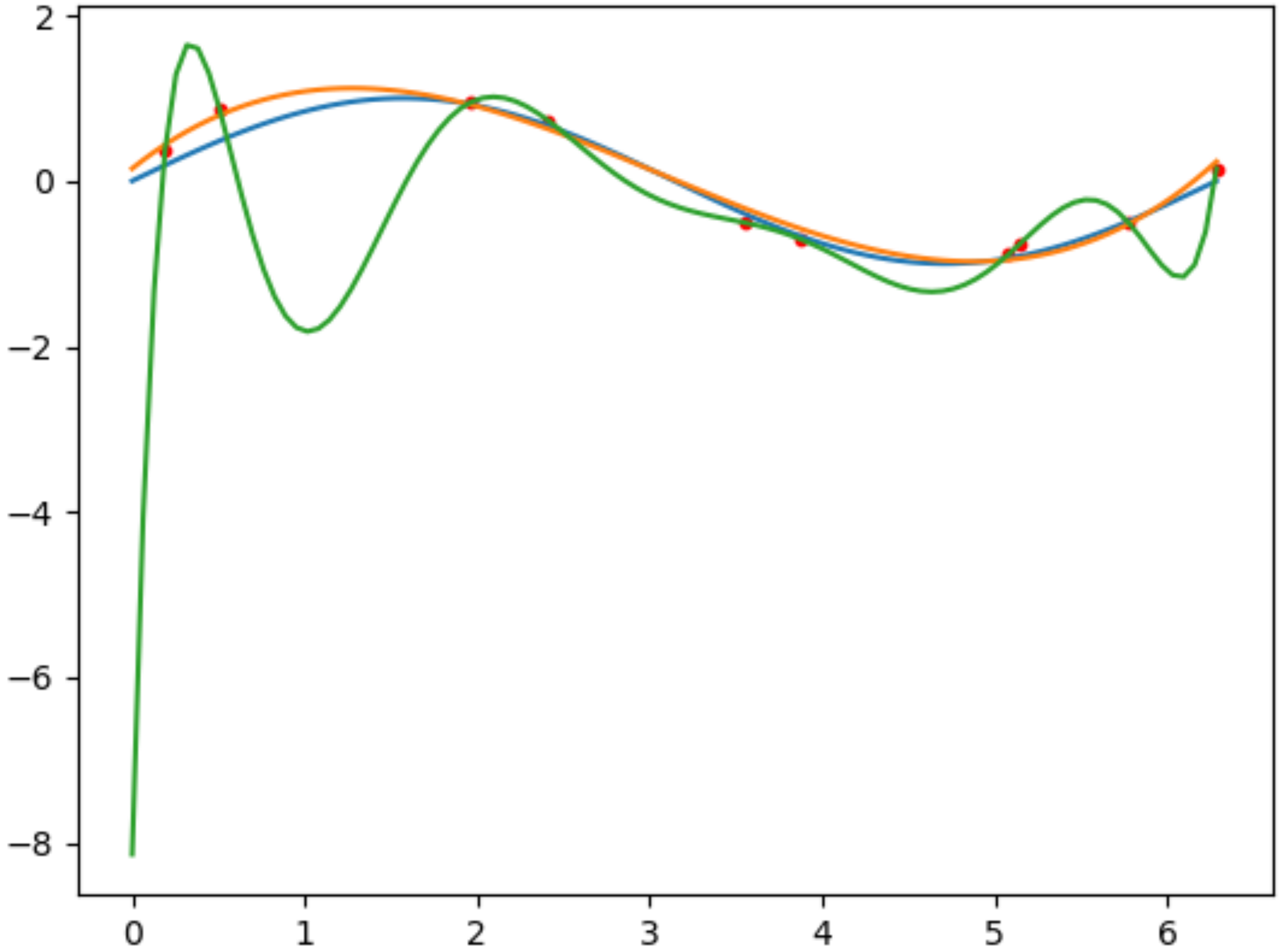
small bias

large variance

$\mathcal{H}$ large



overfitting (and underfitting)



Remedy for overfitting: regularization

minimizing under constraints on the weights

$$\mathbf{w} \cdot \mathbf{w} = \sum_{i=0}^{n} w_i^2 \leq C$$

Suggested reading:

Lagrange multiplier method

$$E_{extended} = E_{sample} + \frac{\lambda}{N} \mathbf{w} \cdot \mathbf{w}$$

Gradient Descent, and validation

Convolutional neural networks (CNN)

Convolution, pooling, etc

$$x * w(0) = \sum_{i=-\infty}^{\infty} x(-i)w(i)$$

$$x * w = w * x$$

$$(x * w)(t) = \sum_{i} x(t-i)w(i)$$

$$(I * K)(m,n) = \sum_{i,j} I(m-i, n-j)K(i,j)$$

I = Image

K = kernel

2 3 4 -1

1 2 -1 1

3 -2 0 1

Convolution

*1 *1

*0 *0

Convolved feature

6 9 2

6 -1 0

average pooling

21/3 10/3

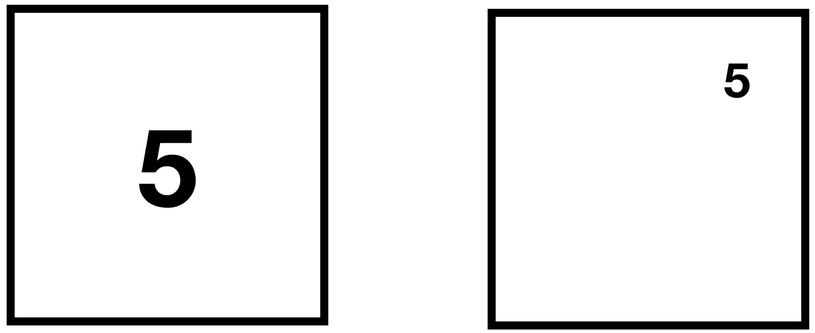
A convolutional neural network is network that has at least one convolutional layer in its hidden layers

Convolutional layer:

1. **Convolution**
2. **Activation function (nonlinear): detection**
3. **Pooling**

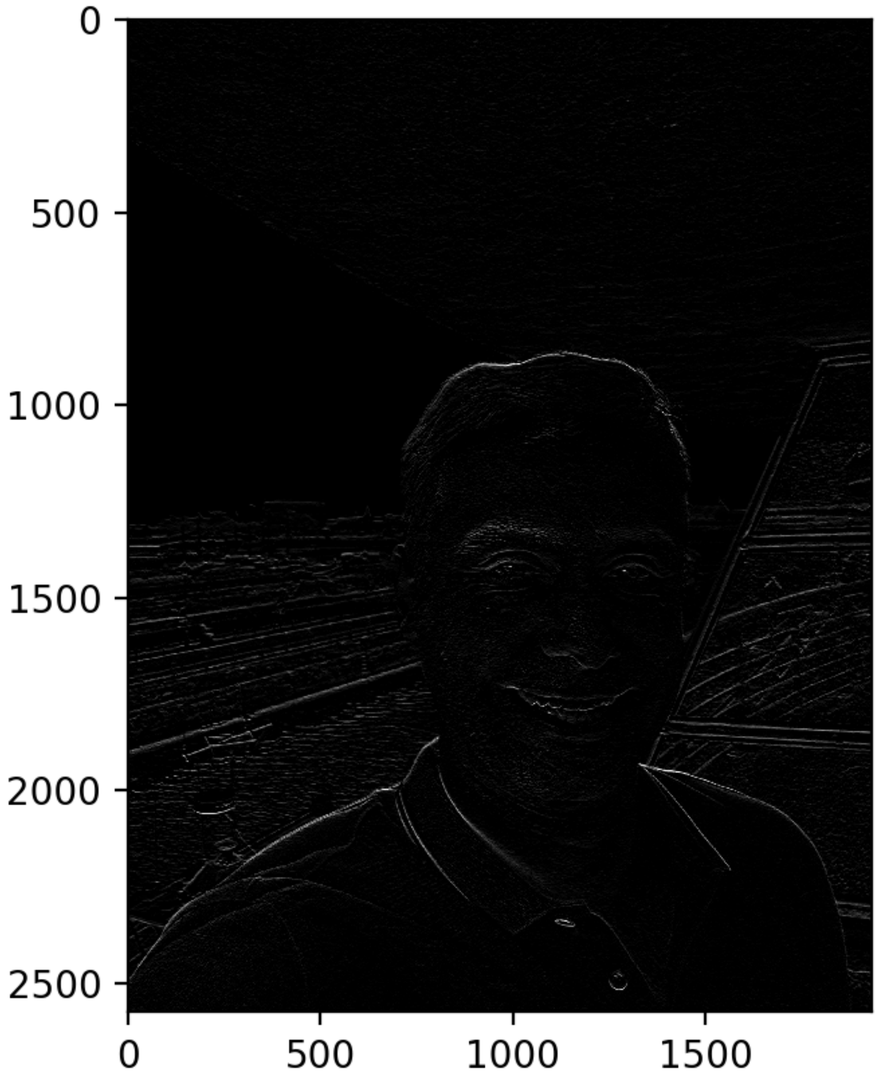
Pooling: Calculating a summary statistics of the output of stage 2 such as mean, max, norm, etc.

e.g. maximum of a rectangular neighborhood
advantage: makes the training less sensitive to translations, therefore digits located in different places can be detected

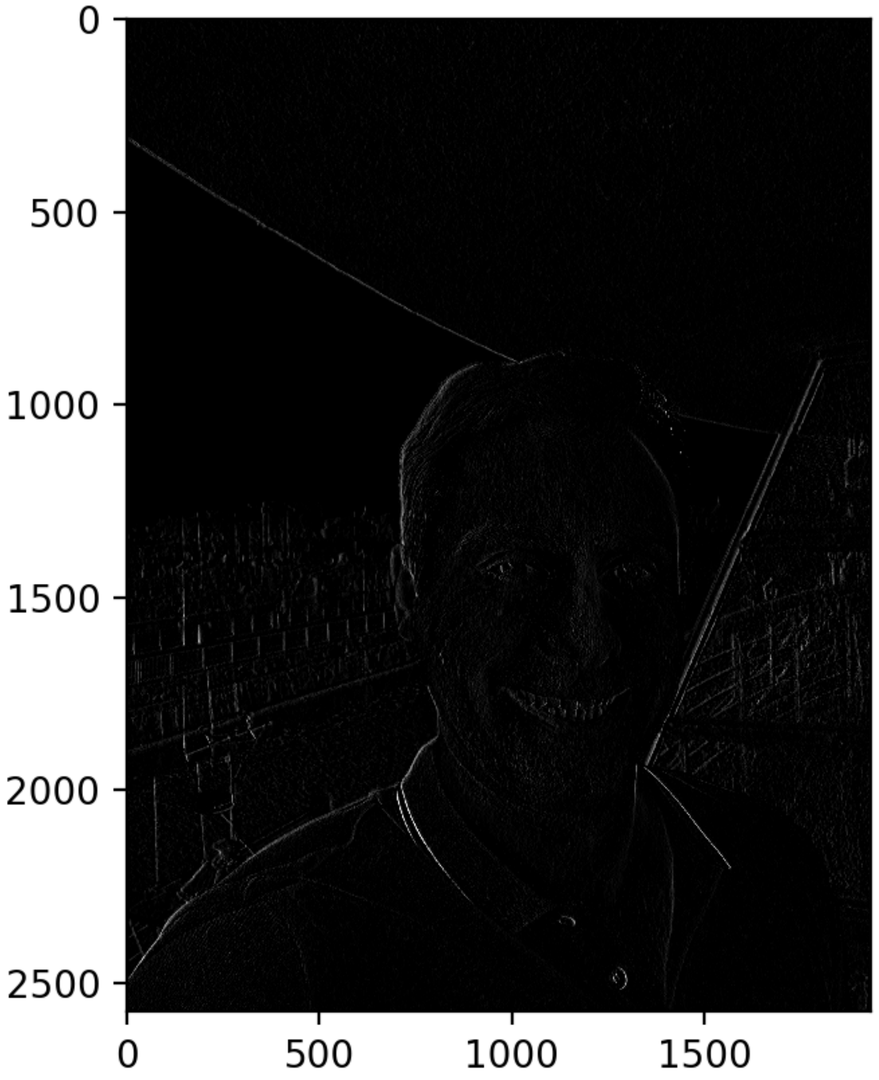


*1 *1

*0 *0



1, 1, 1
0, 0, 0
-1, -1, -1



1, 0, -1
1,0, -1
1, 0, -1

Recurrent neural networks (RNN)

Sequential data: language processing, sentences, …

The order of the data is important.

Predicting the forthcoming data.

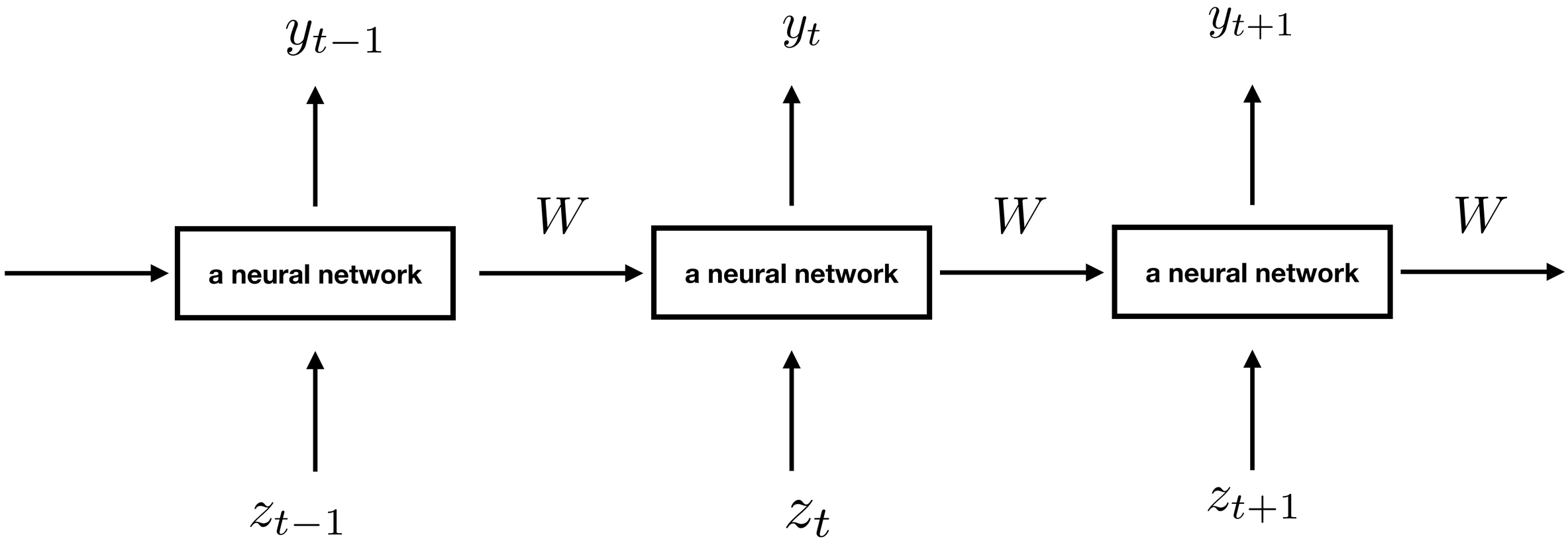
Time series analysis: prediction of stock prices

Dealing with issues such as: length of input data changes

Need a number of neural networks connected to each other:
The connection is made by connecting their hidden layers

Parameter sharing:
Same W between the hidden layers

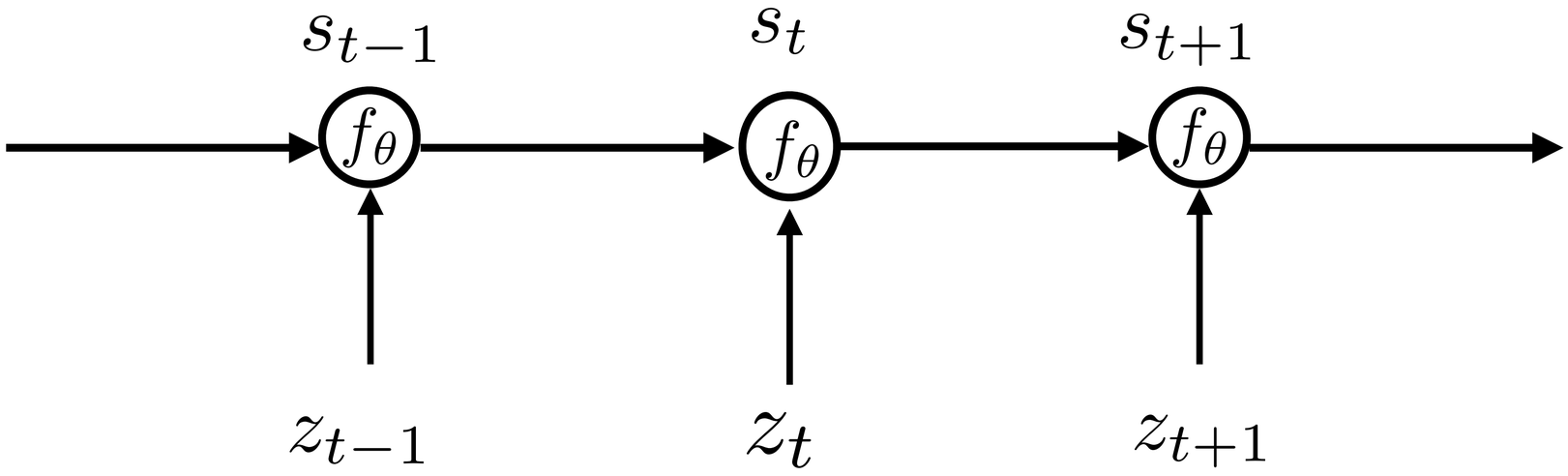
$$a_t = b + W s_{t-1} + U x_t$$
$$s_t = \tanh(a_t)$$
$$y_t = c + V s_t$$
$$p_t = softmax(y_t)$$



$$a_t = b + W s_{t-1} + U x_t$$
$$s_t = \tanh(a_t)$$
$$y_t = c + V s_t$$
$$p_t = softmax(y_t)$$

Dynamical system

$$s_t = f_\theta(s_{t-1}, z_t)$$



$$(3, 2, 1) = 3(1, 0, 0) + 2(0, 1, 0) + 1(0, 0, 1)$$

$$e_1 = (1, 0, 0)$$
$$e_2 = (0, 1, 0)$$
$$e_3 = (0, 0, 1)$$

$$\mathbf{x} = (x_1, x_2, x_3)$$
$$= (\mathbf{x} \cdot e_1)e_1 + (\mathbf{x} \cdot e_2)e_2 + (\mathbf{x} \cdot e_3)e_3$$

$$e_n(x) = e^{2\pi inx} = \cos(2\pi nx) + i\sin(2\pi nx)$$

base functions

$$n = 0, \pm 1, \pm 2, \ldots$$

$$i^2 = -1$$

$$(x, y) = x + iy$$

$$\bar{(x, y)} = x - iy$$

$$< f, g >= \quad f(x) \cdot g(x) = \int_0^1 f(x)\bar{g}(x)\, dx$$

f and g are periodic functions of period 1

$$f(x) = \sum_{n=-\infty}^{\infty} < f, e_n > e_n(x)$$

n runs over all integers and period of f is 1!!!

$$\hat{f}(n) =< f, e_n >$$

n-th Fourier coefficient

$$\hat{f}(\xi) = \int_{\mathbb{R}^m} f(x) e^{-2\pi i \xi \cdot x} \, dx$$

Fourier transform of a function defined on all real numbers (not periodic)

$$f(x) = \int_{\mathbb{R}^m} \hat{f}(\xi) e^{2\pi i \xi \cdot x} \, d\xi$$

Fourier inversion formula: reconstruction from Fourier transform

$$(f * g)(x) = \int f(x-t) g(t) \, dt$$

$$\widehat{(f * g)}(x) = \hat{f}(x) \, \hat{g}(x)$$

Fast Fourier transform

$$f(0), f(1), \ldots, f(N-1)$$

$$\hat{f}(k) = \sum_{n=0}^{N-1} f(n) e^{i(-2\pi kn/N)}$$

$$f(n) = \sum_{k=0}^{N-1} \hat{f}(k) e^{i(2\pi kn/N)}$$

$$e^{ix} = \cos(x) + i\sin(x)$$

$$\cos(x) = \frac{1}{2}(\exp(-ix) + \exp(ix))$$

$$\sin(x) = \frac{\exp(ix) - \exp(-ix)}{2i}$$

k: **-(N-1)/2 , …, -2, -1, 0, 1, 2, …, (N-1)/2**

assuming N is odd

Fast Fourier transform
2 dimensional

$$f(0,0), f(1,0), \ldots, f(N_1-1,0) \qquad N_1 \times N_2$$

$$f(0,1), f(1,1), \ldots, f(N_1-1,1)$$

$$\bullet \quad \bullet \quad \bullet$$

$$f(0,N_2-1), f(1,N_2-1), \ldots, f(N_1-1,N_2-1)$$

$$\hat{f}(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} f(n_1,n_2) e^{i(-2\pi k_1 n_1/N_1)} e^{i(-2\pi k_2 n_2/N_2)}$$

$$f(n_1,n_2) = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} \hat{f}(k_1,k_2) e^{i(2\pi k_1 n_1/N_1)} e^{i(2\pi k_2 n_2/N_2)}$$

Signal detection in electrophysiological measurements

Multi-modal time series of neural activities are accompanied by noise and our goal has been to detect the signals and spikes in such series by way of analyzing their intrinsic statistical properties.

Our approach has been able to **sort the spikes** indicating a difference in their formation, and to **detect faint signals**.

Our method has the power to deal **efficiently** with correlated time series and **correlated noise**.

Statistical inference

Consider an **assumption H** about a sample X.

Calculate the probability that X would be observed given that H is true:

$$p = P(X|H)$$

Choose a *significance level* (a small number) such as:

$$\alpha = 0.05$$

If $p \leq \alpha$ **infer** that the hypothesis **H** should be **rejected**.

Example of statistical inference

X : a poisson distribution such as the number of accidents on a street in one week with mean 5

$$P(X = i) = e^{-\lambda}\,\frac{\lambda^i}{i!}, \qquad i = 0, 1, 2, \ldots$$

Hypothesis **H**: $\lambda = 5$

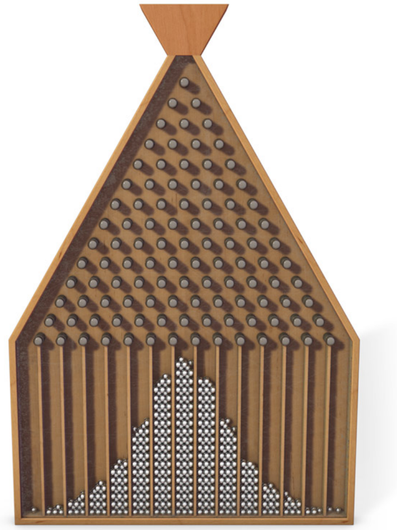
$$P(X = 10|H) = 0.0181328$$

So based on one week that incidentally there were 10 accidents one will **reject the correct hypothesis**.

Multiple comparisons

How to make statistical inference about a hypothesis from many samples ?

Fundamental idea: To exploit **Bell shaped** distributions that form in nature prevalently when the outcome of an event is the result of the **independent** contribution of many effects with **the same distribution**.



Galton board

De Moivre-Laplace theorem (CLT)

$$f(x) = (1/2\pi)\, e^{-x^2/2}$$

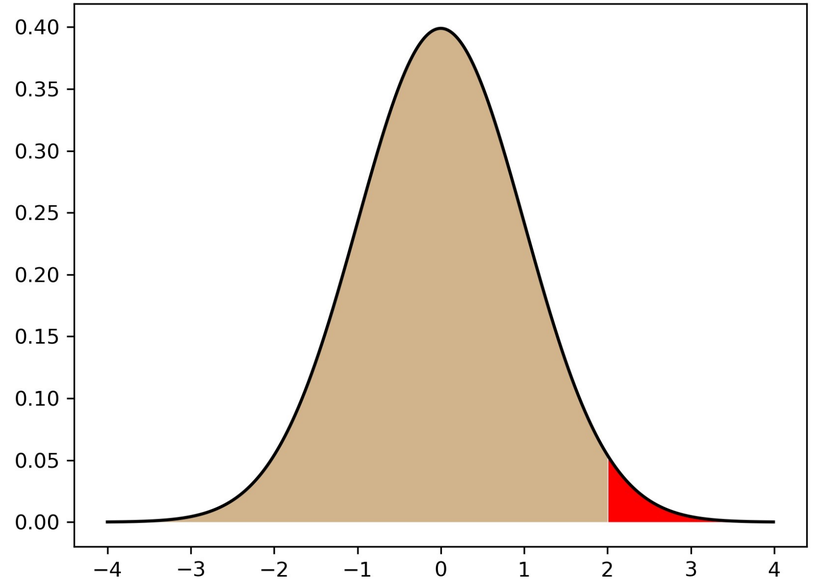
$$P(\mathcal{N}(0,1) > 2) = 0.022$$

$X_1, \ldots, X_m$

$Y_1, \ldots, Y_m$ defined as:

$Y_j = 1$ if **j**-th inference rejects **H** at significance α

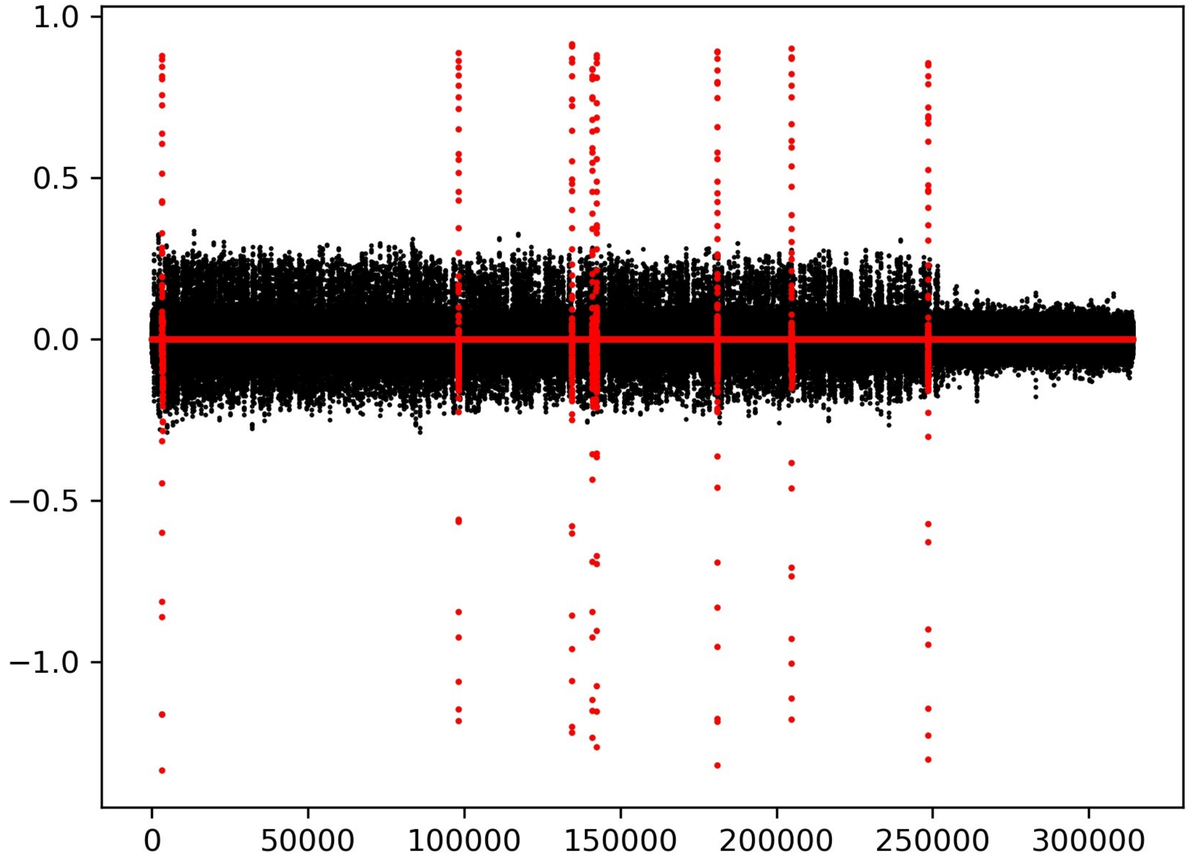
$Y_j = 0$ otherwise



$$HC_{m,\alpha} = \frac{(Y_1 + \cdots + Y_m)/m - \alpha}{\sqrt{\alpha(1-\alpha)}/\sqrt{m}}$$ approaches $\mathcal{N}(0,1)$ as $m \to \infty$

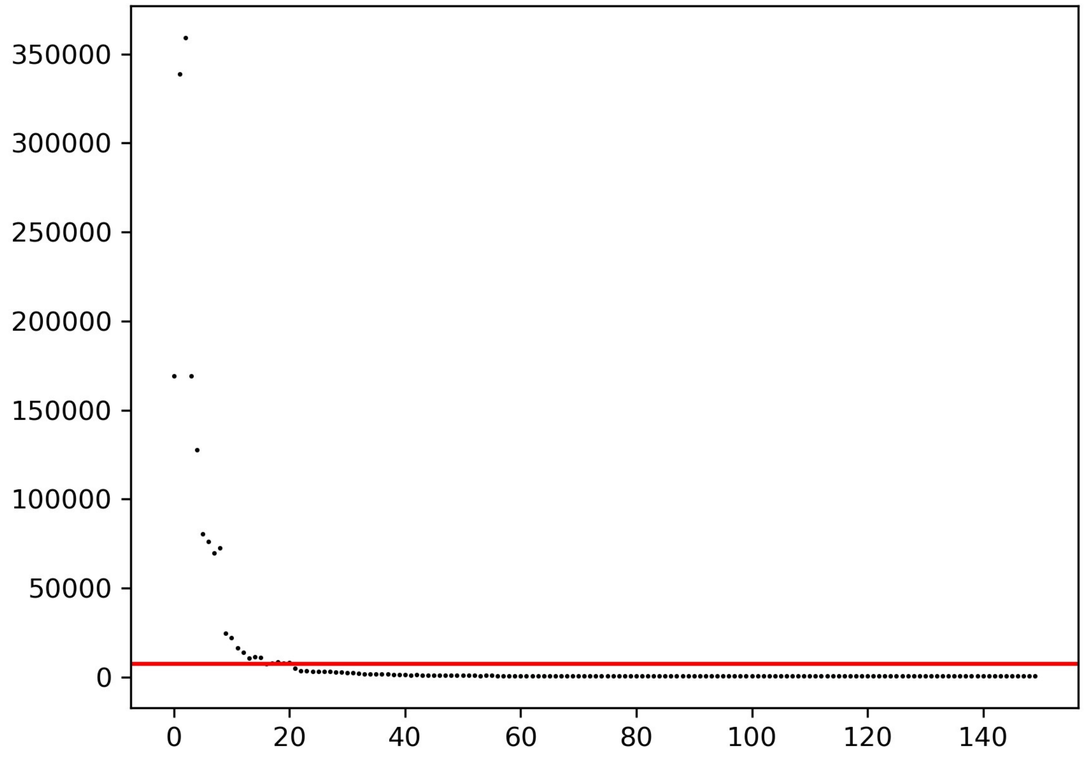
$HC_m = \max HC_{m,\alpha}$ **Tukey**: reject **H** if $HC_m > 2$

Detection of electrophysiological data spikes with higher criticism

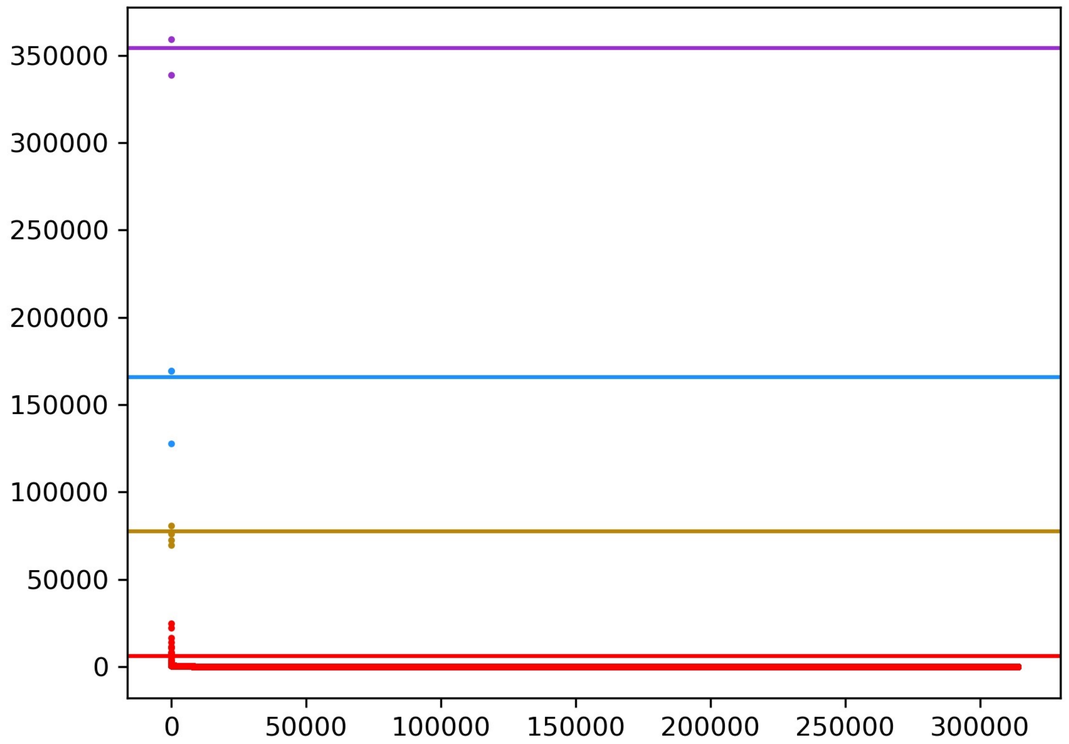


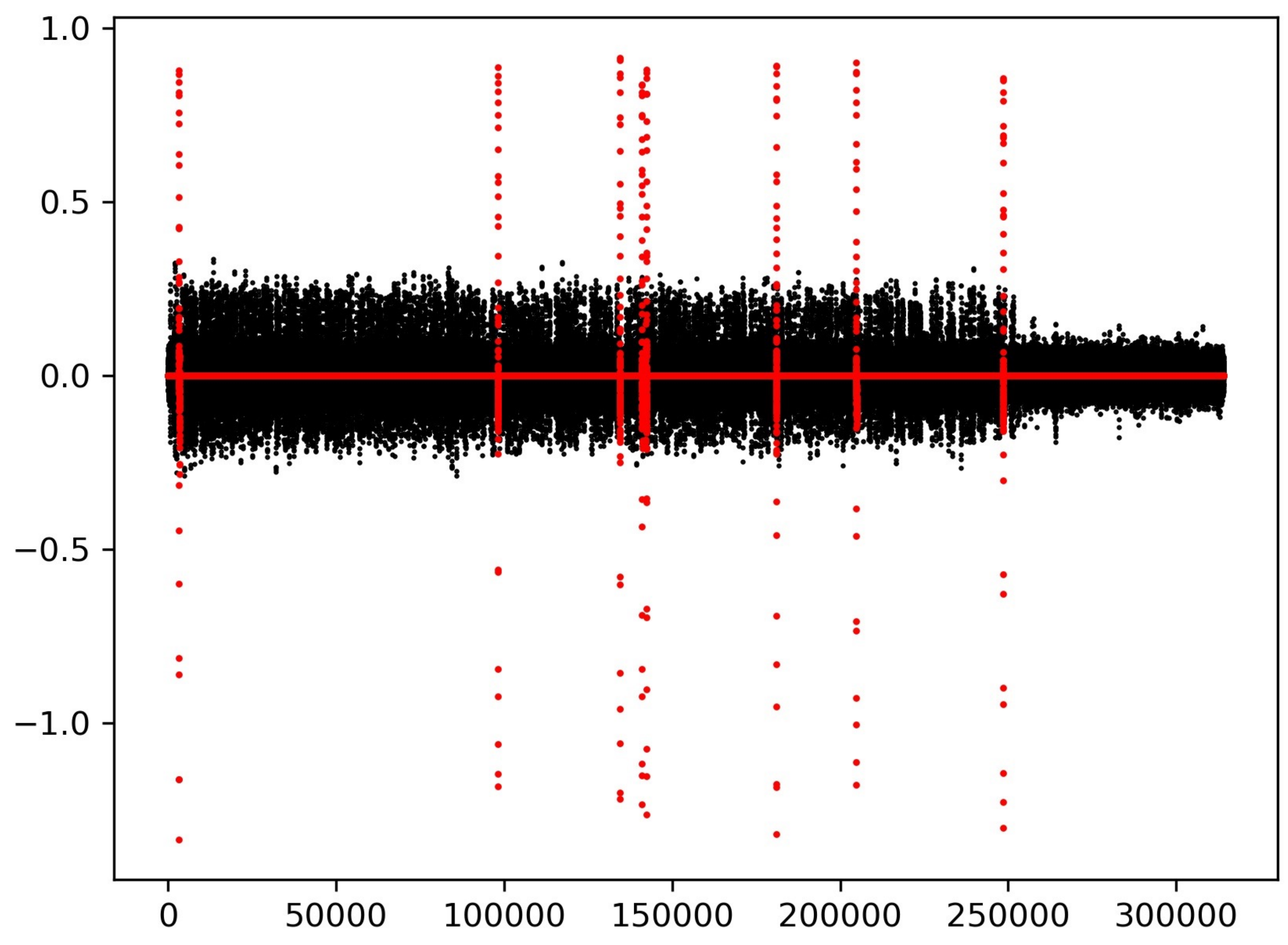
Farzad Fathizadeh, Ekaterina Mitricheva, Rui Kimura, Nikos Logothetis, Hamid R. Noori

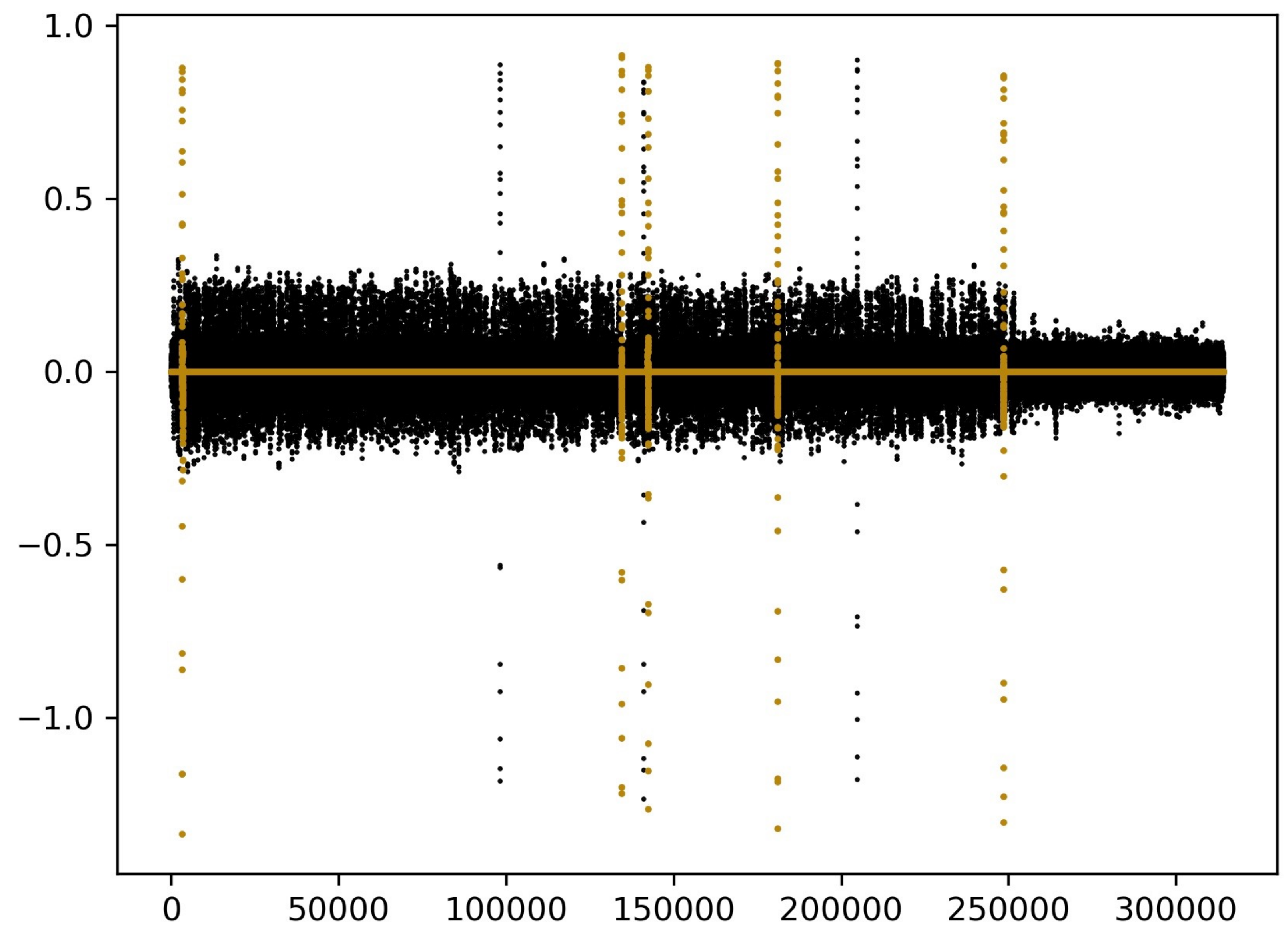
HC quantities associated with the electrophysiological time series

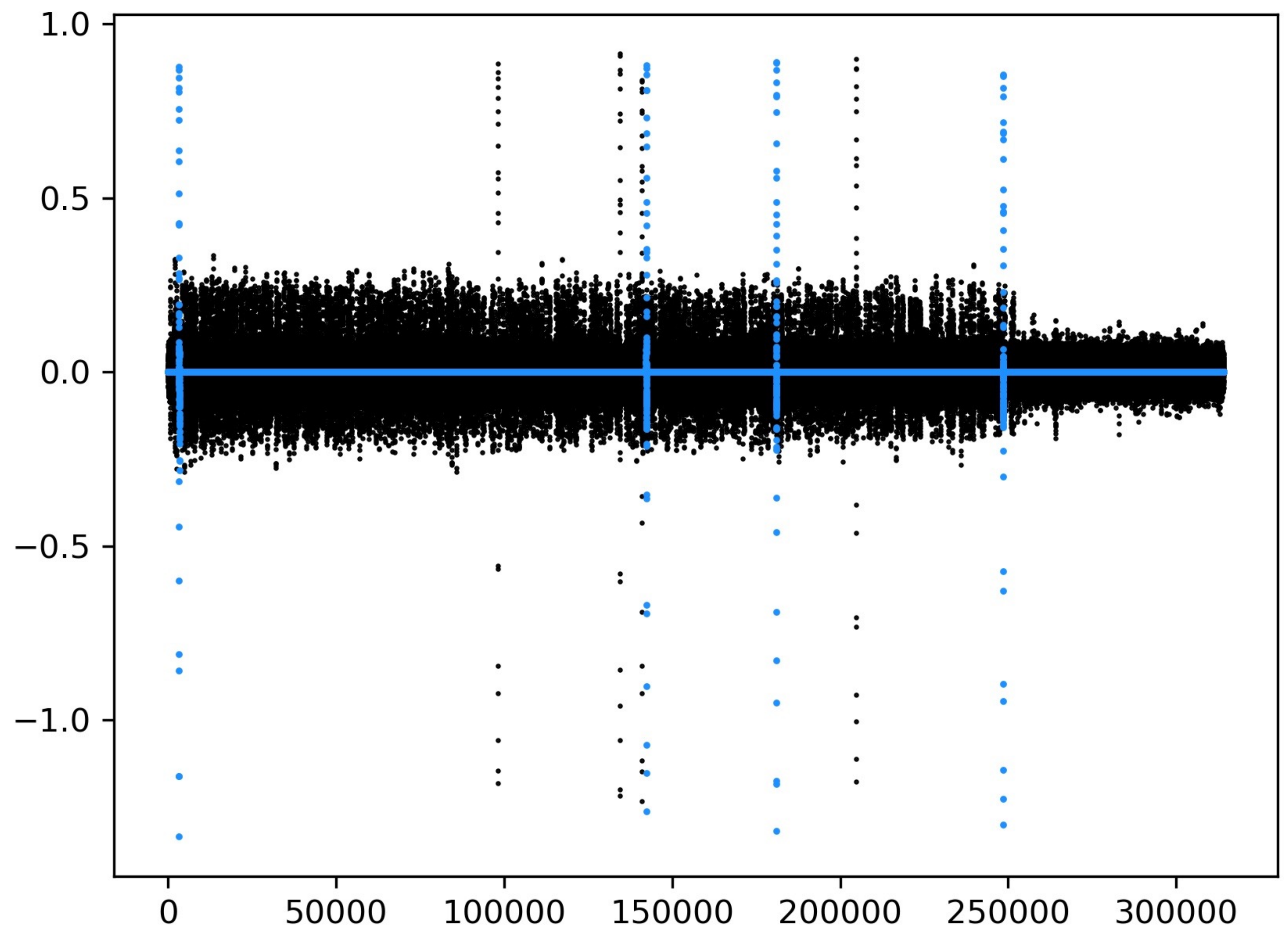


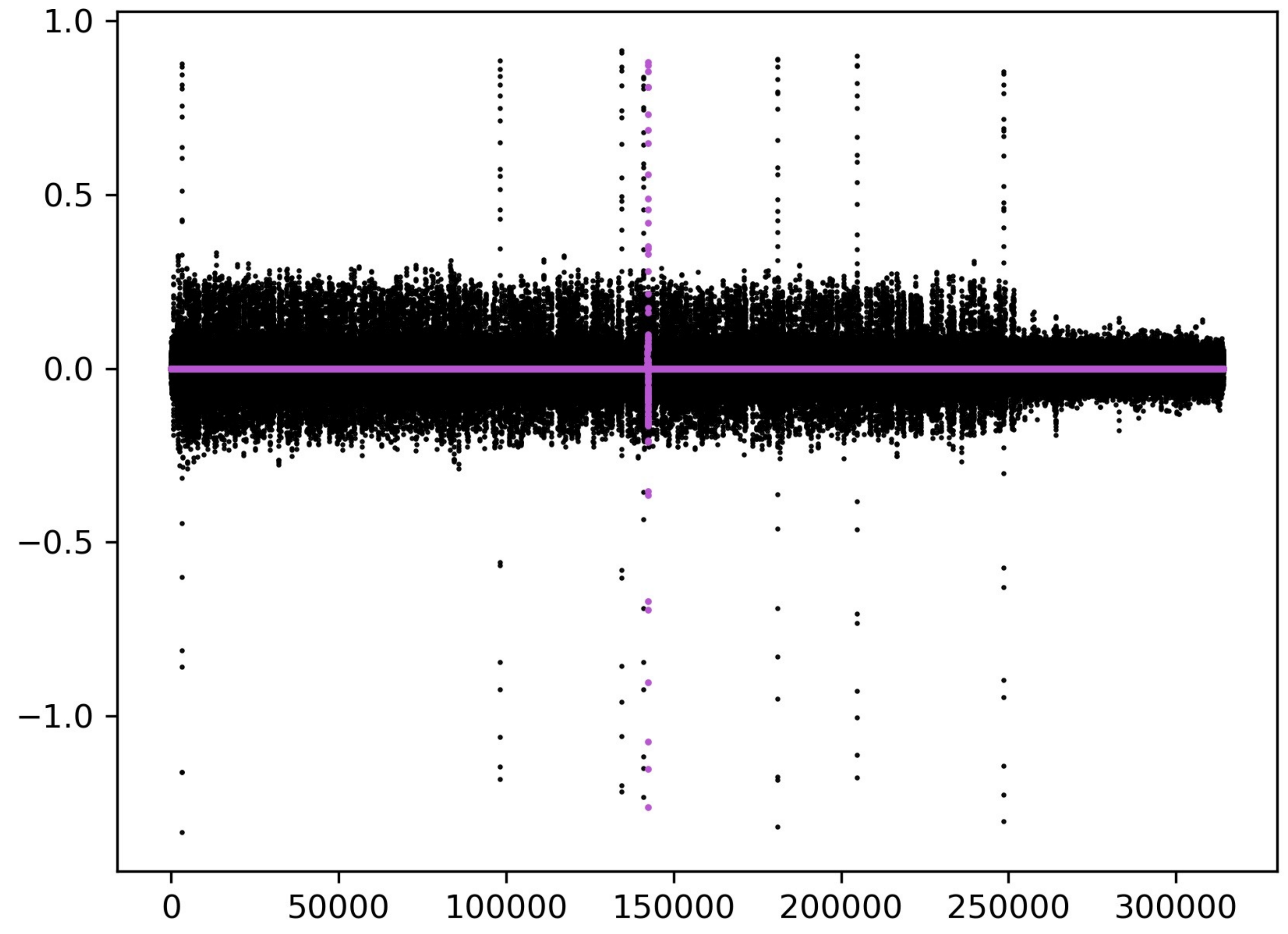
HC quantities clustered and thresholds associated with the clusters











Spike sorting

Farzad Fathizadeh, Ekaterina Mitricheva, Rui Kimura, Nikos Logothetis, Hamid R. Noori

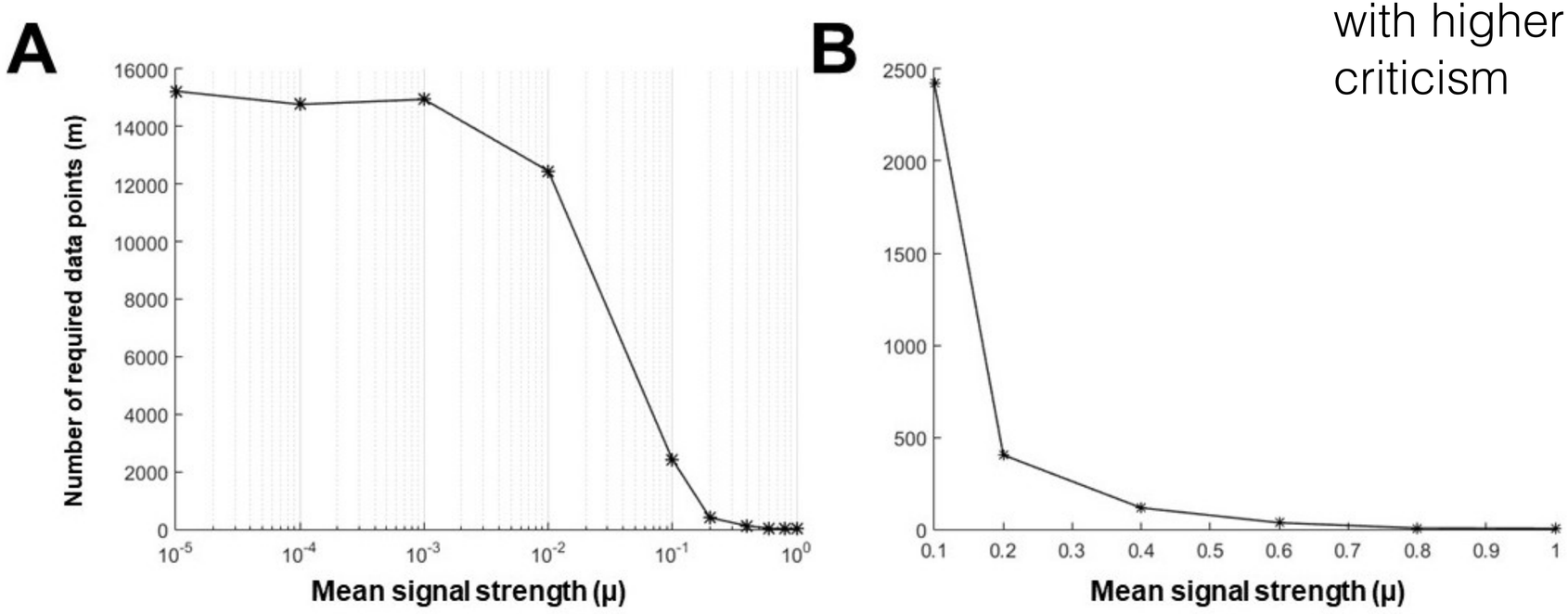
How many data points do we need to detect weak signals ?

$$X \sim \mathcal{N}(\mu, 1)$$

To detect small non-zero

$$\mu$$

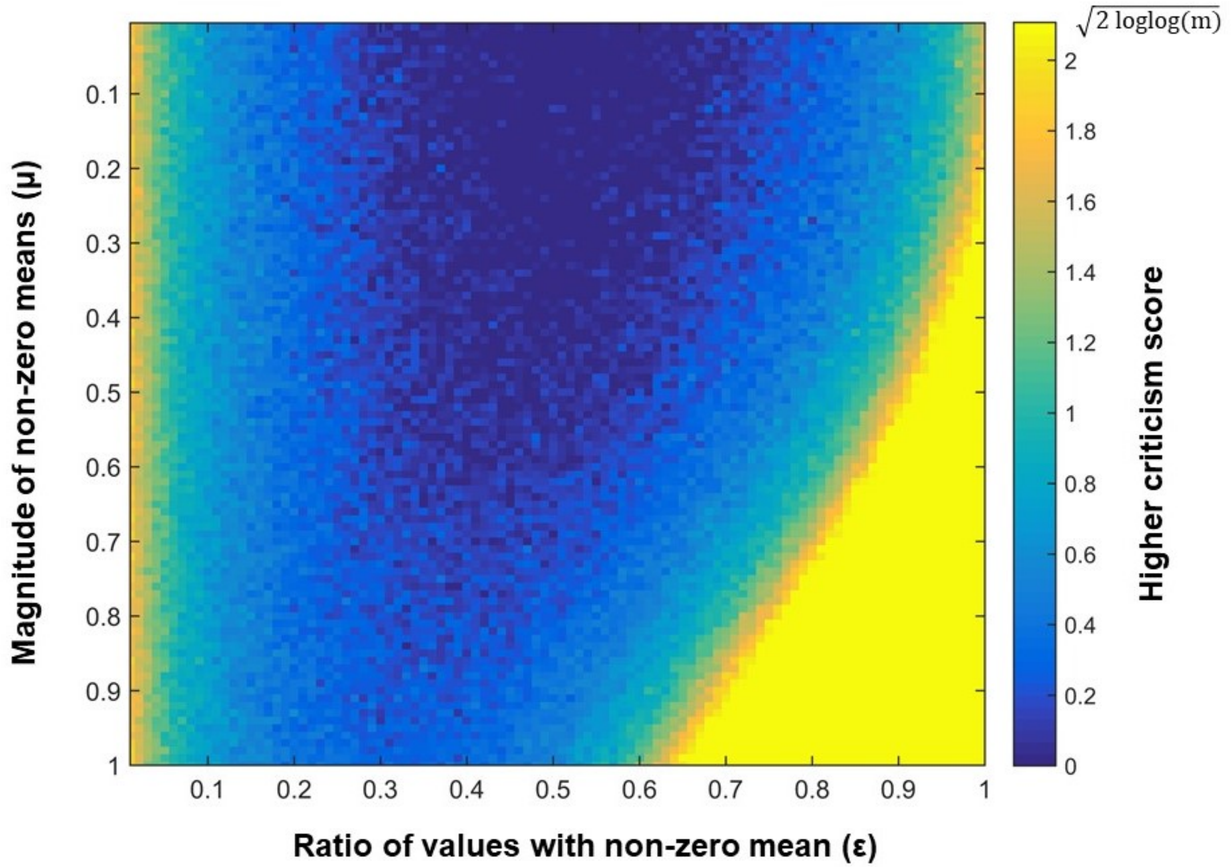
with higher criticism



Farzad Fathizadeh, Ekaterina Mitricheva, Rui Kimura, Nikos Logothetis, Hamid R. Noori

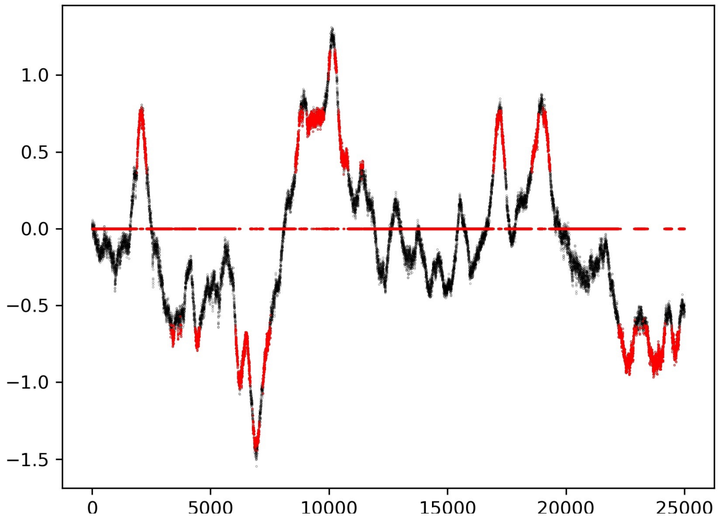
How many data points do we need to detect weak sparse signals ?

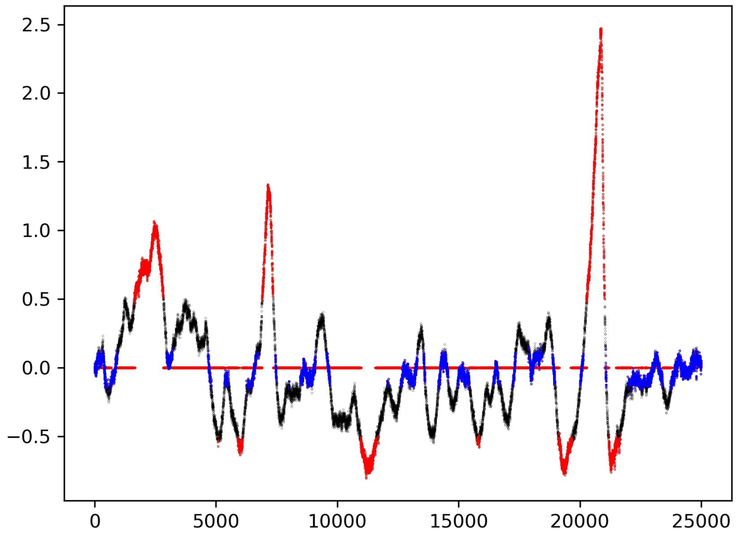
$$X \sim \varepsilon\,\mathcal{N}(\mu, 1) + (1-\varepsilon)\,\mathcal{N}(0, 1)$$

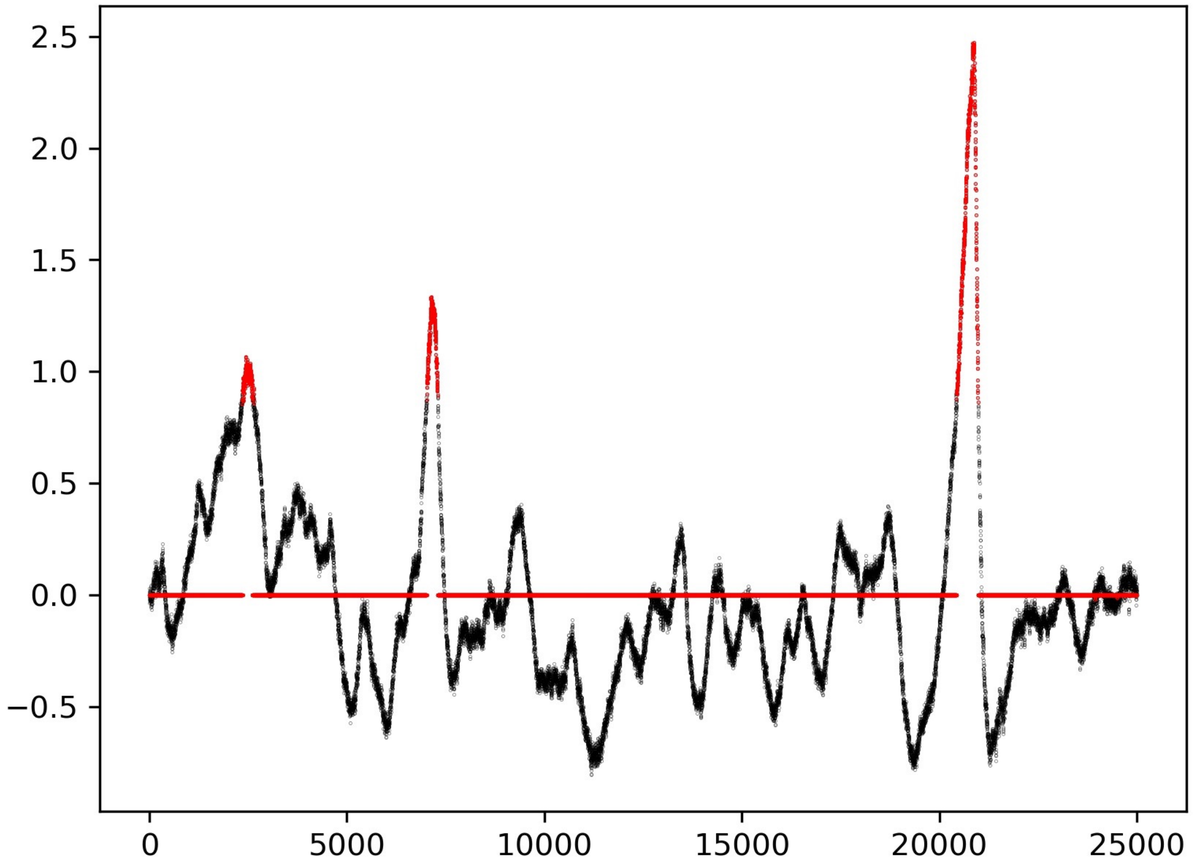


Farzad Fathizadeh, Ekaterina Mitricheva, Rui Kimura, Nikos Logothetis, Hamid R. Noori

Multi-unit activity data peaks detected with higher criticism







Farzad Fathizadeh, Ekaterina Mitricheva, Rui Kimura, Nikos Logothetis, Hamid R. Noori

Multi-modal correlated time series and correlated noise

Note: HC is not sensitive to shuffling of coordinates, hence ignores the correlation

How to make the performance of higher criticism better in this situation ?

The covariance matrix of the data is a positive definite matrix: apply matrices using linear algebra to turn this matrix to identity (in sparse cases), or to decompose this matrix to the product of simpler matrices. Then use the HC on the result of application of such matrices to the coordinates.

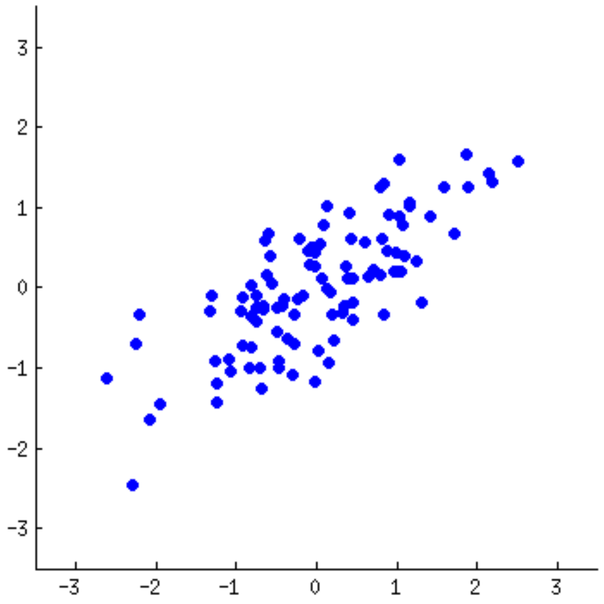
Analysis of movement and behavioral pattern recognition

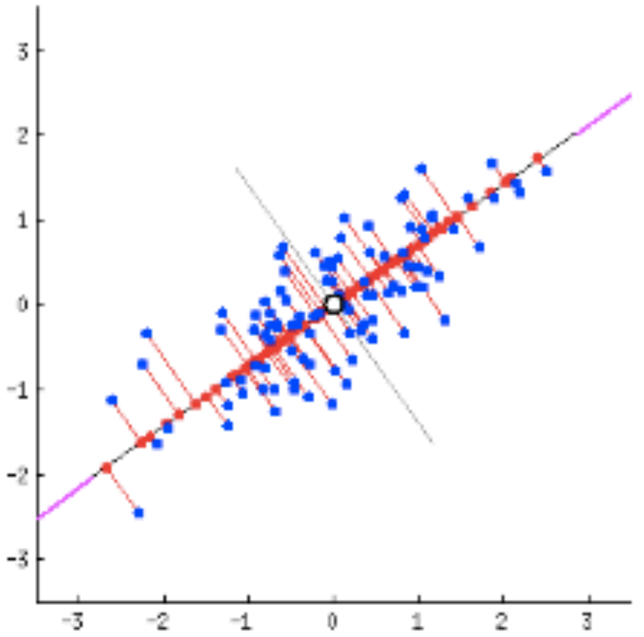
We analyze the data from the smart housing facilities to understand stochastic properties of the animal behavior in their social life and to find relations between their behavior and biological states related to health, disease, gene modification, hormonal imbalances, etc.

Geometric learning will be useful for encoding the correlations especially in the movements and for seeing patterns of social behavior in the geometric properties of the underlying manifold (non-linear structure).

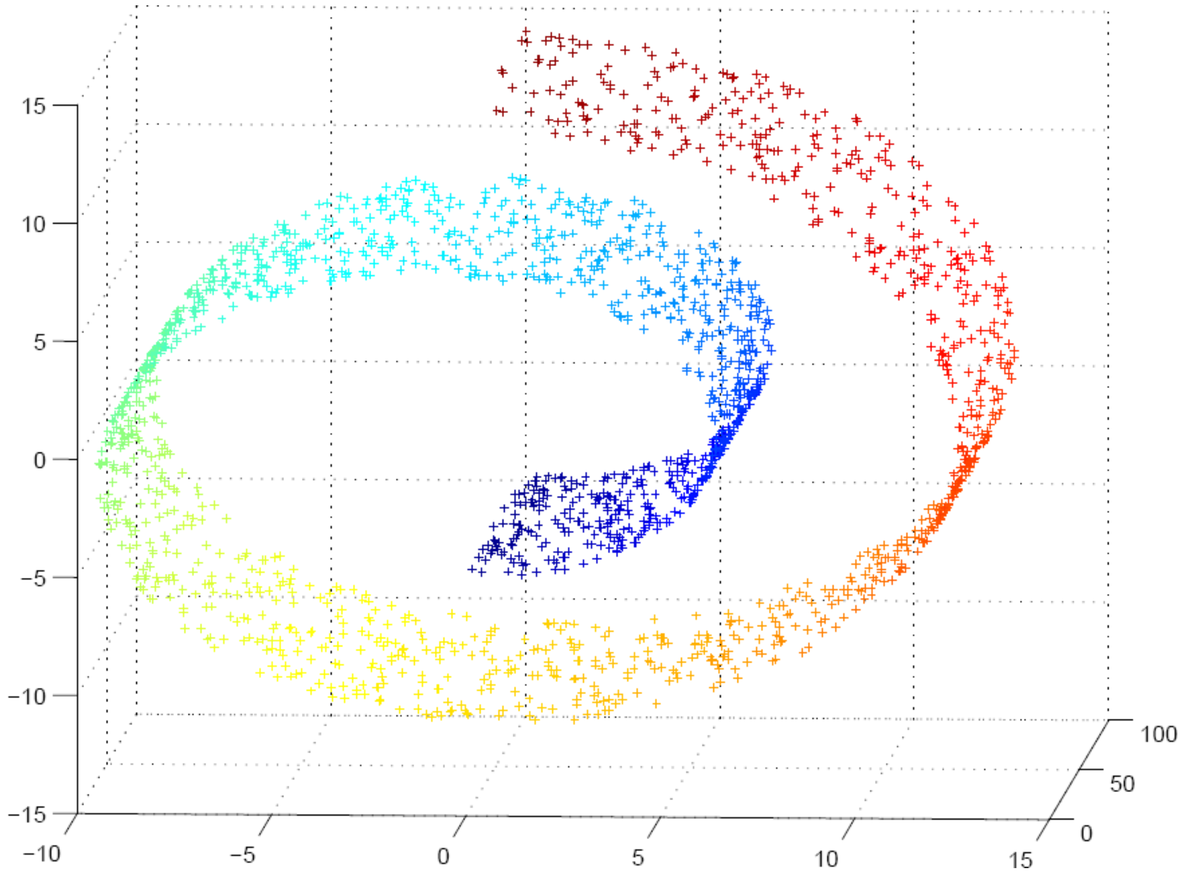
Deep learning will be used to characterize animal behavior based on finger/hand motion, body statue, gait, facial mimics and complex interactive processes such as touching and grooming. Subsequently, we will study correlations between characteristic behavior and biological time-series (such as body temperature, cerebral activity, metabolism, and blood flow) as well as relations with the underlying neural and genetic mechanisms, and predict statistically disease-onset and dynamics.

Principal component analysis





Geometric learning: correlations preserved in the geometry of the data while dimension reduction is achieved



Geometric learning based on heat kernel and Laplacian

- M. Belkin, P. Niyogi, Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering, in
"Advances in Neural Information Processing Systems 14" pp.585–591, MIT Press, 2001.

- M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural Comput. 15 (6) (2003) 1373–1396.

M. Belkin, P. Niyogi, Towards a theoretical foundation for Laplacian-based manifold methods, J. Comput. System Sci. 74 (2008), no. 8, 1289–1308.

Instataneous time frequency analysis

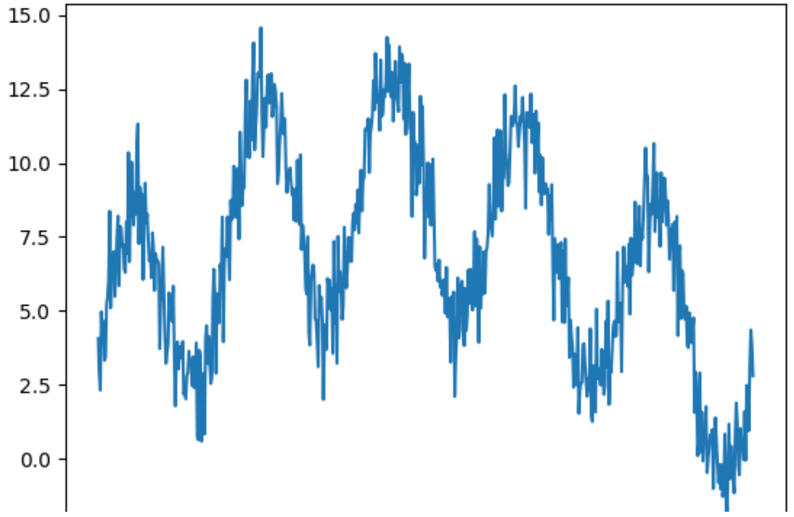
Fourier analysis:

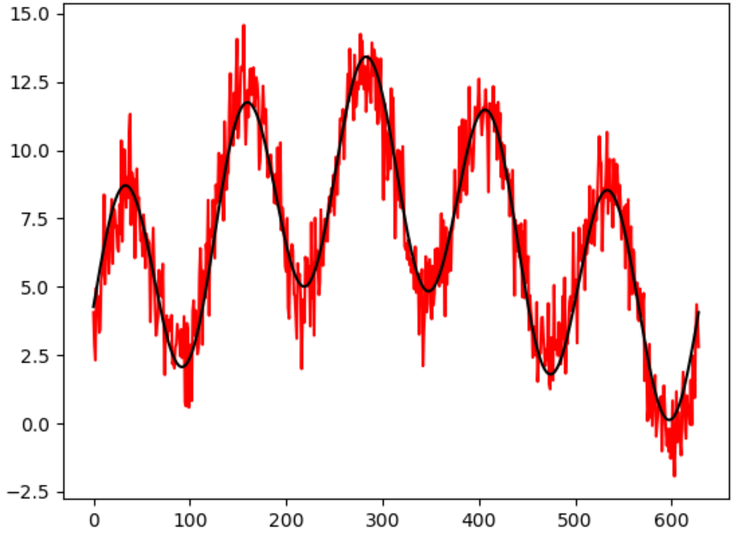
$$x(t) = \sum_j a_j e^{i\omega_j t} = \sum_j a_j \left(\cos(\omega_j t) + i \sin(\omega_j t)\right)$$

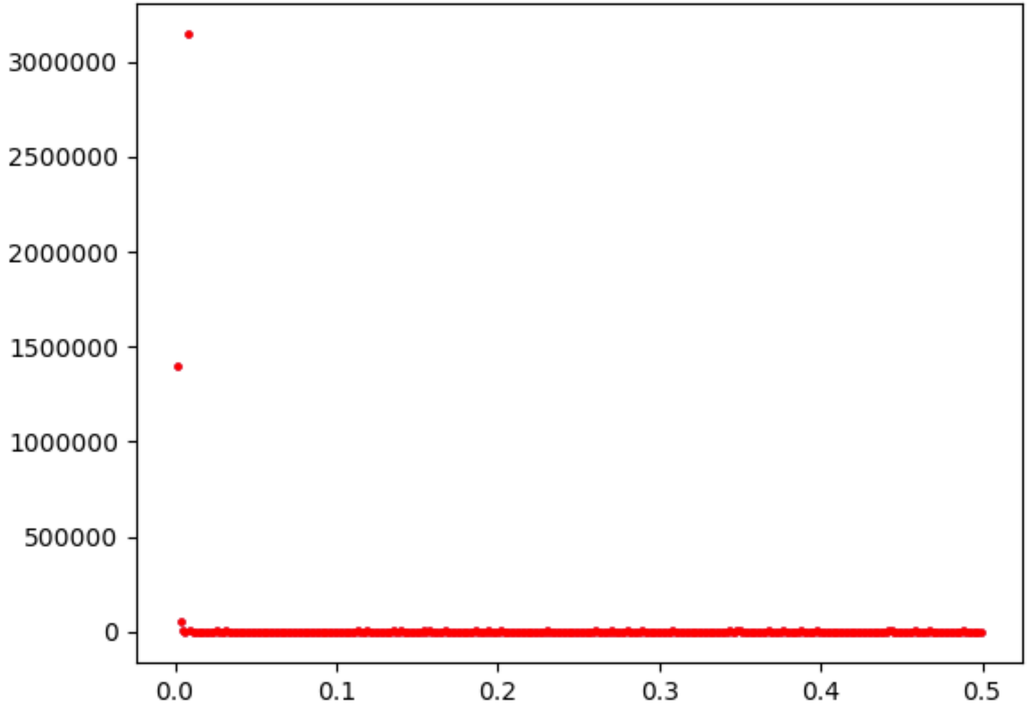
Hilbert spectral analysis:

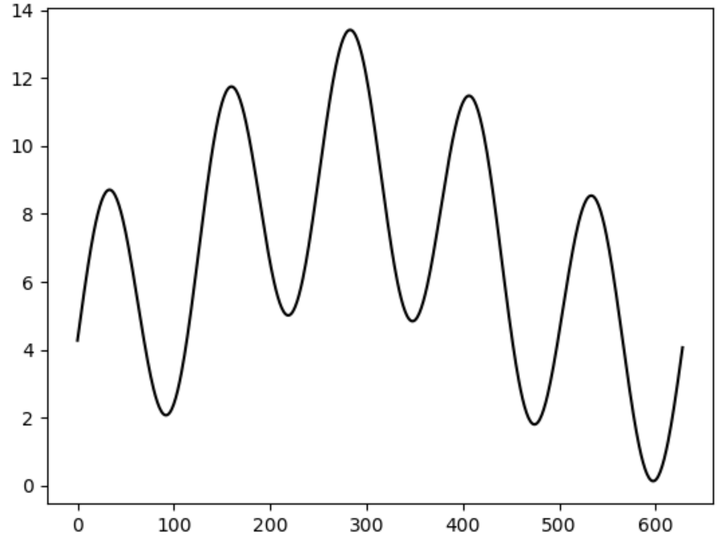
$$x^*(t) = P.V. \int \frac{x(\tau)}{t-\tau}\, d\tau$$

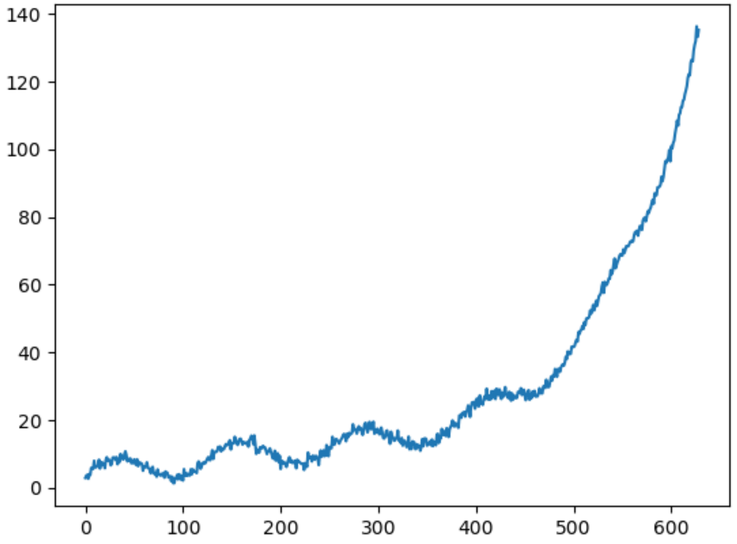
$$x(t) = \sum_j a_j(t) e^{i\omega_j(t)} = \sum_j a_j(t) \left(\cos(\omega_j(t)) + i \sin(\omega_j(t))\right)$$

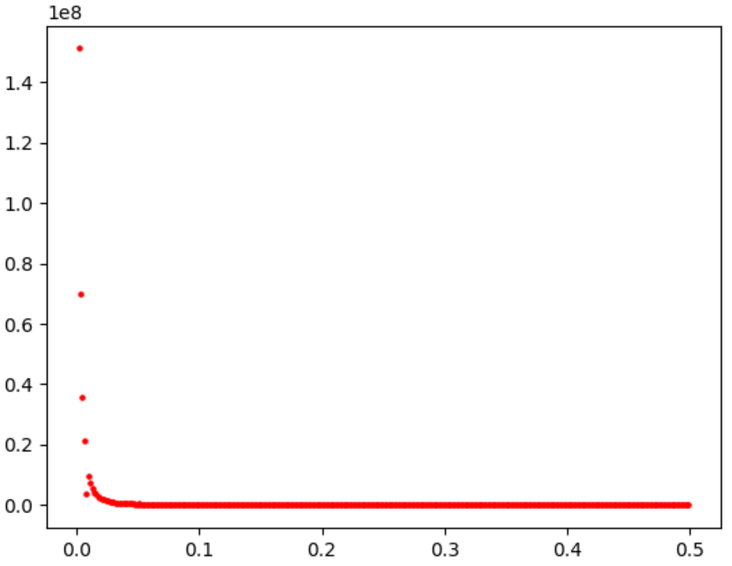

15.0
12.5
10.0
7.5
5.0
2.5
0.0

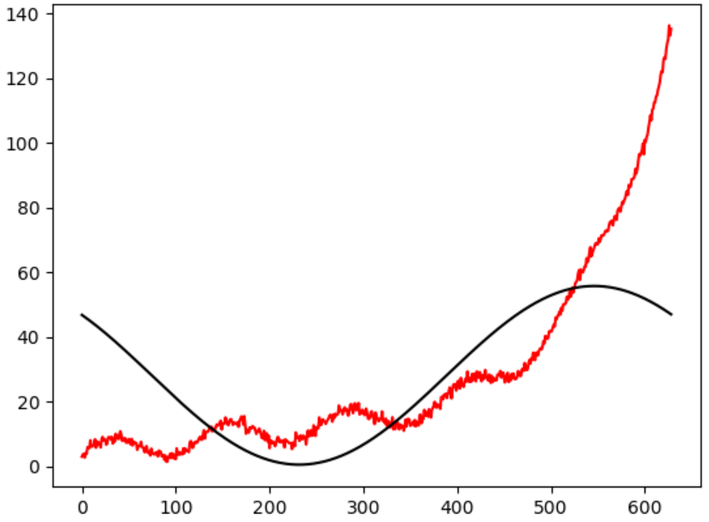

15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
0
100
200
300
400
500
600

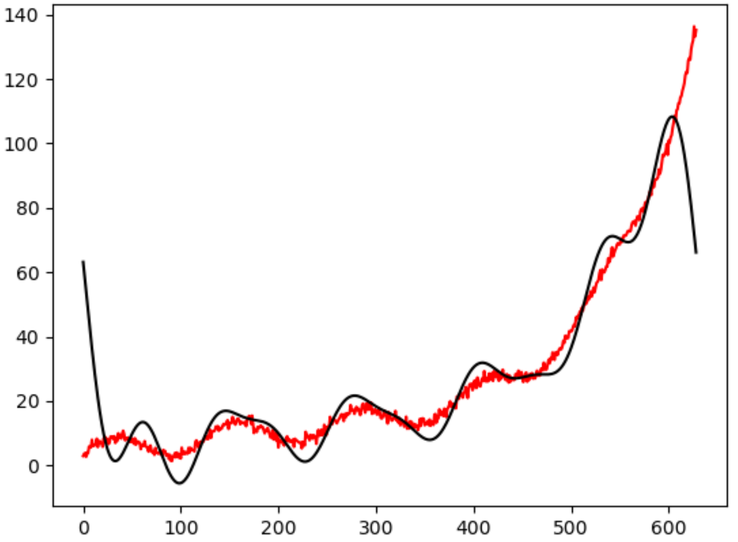

3000000
2500000
2000000
1500000
1000000
500000
0
0.0
0.1
0.2
0.3
0.4
0.5

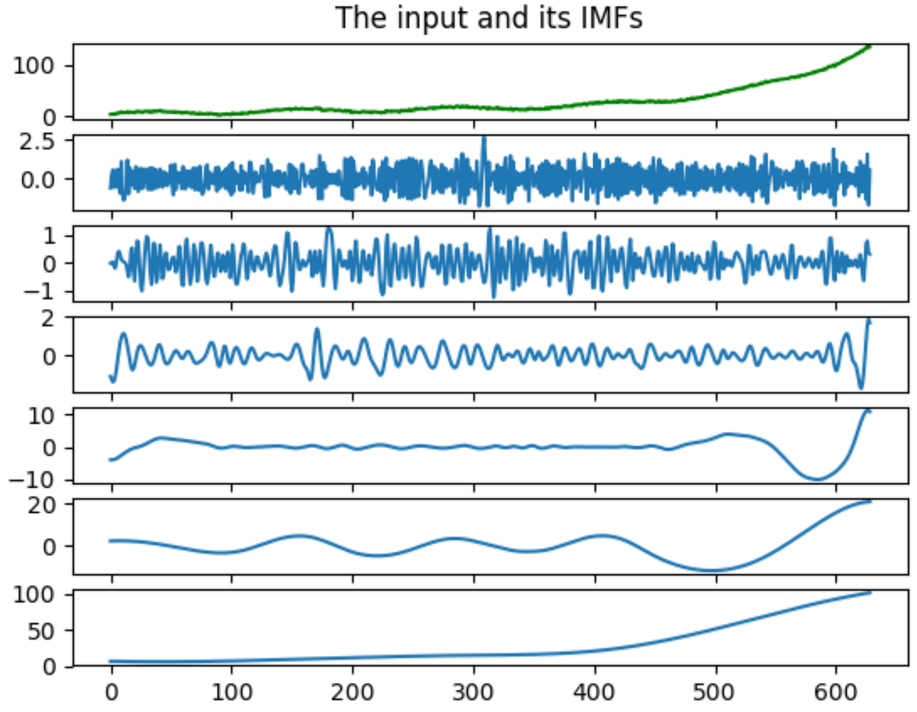

14
12
10
8
6
4
2
0
0
100
200
300
400
500
600

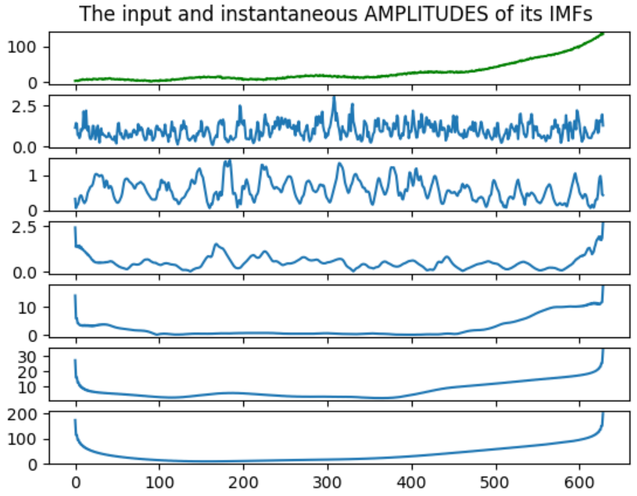

140
120
100
80
60
40
20
0
0
100
200
300
400
500
600

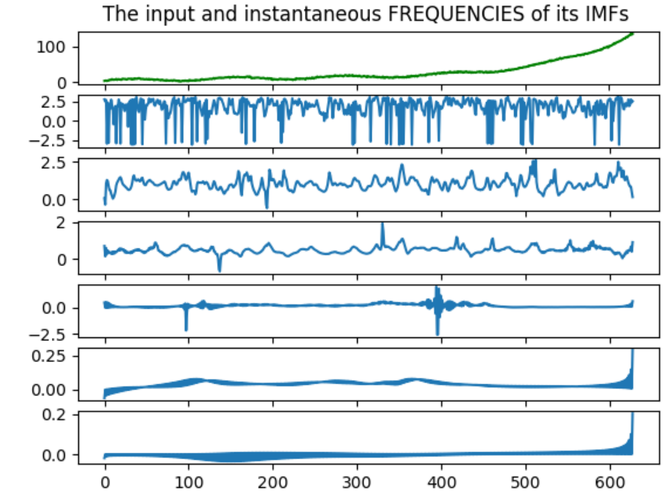

1e8
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
0.4
0.5

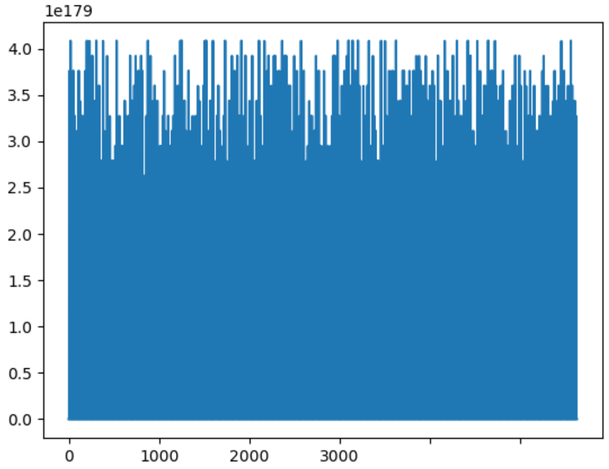

140
120
100
80
60
40
20
0
0
100
200
300
400
500
600

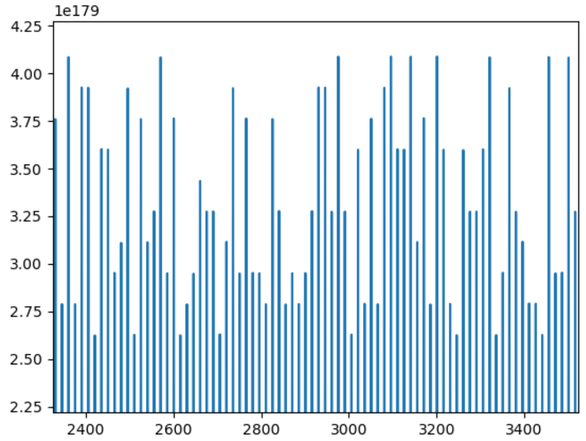

140
120
100
80
60
40
20
0
0
100
200
300
400
500
600

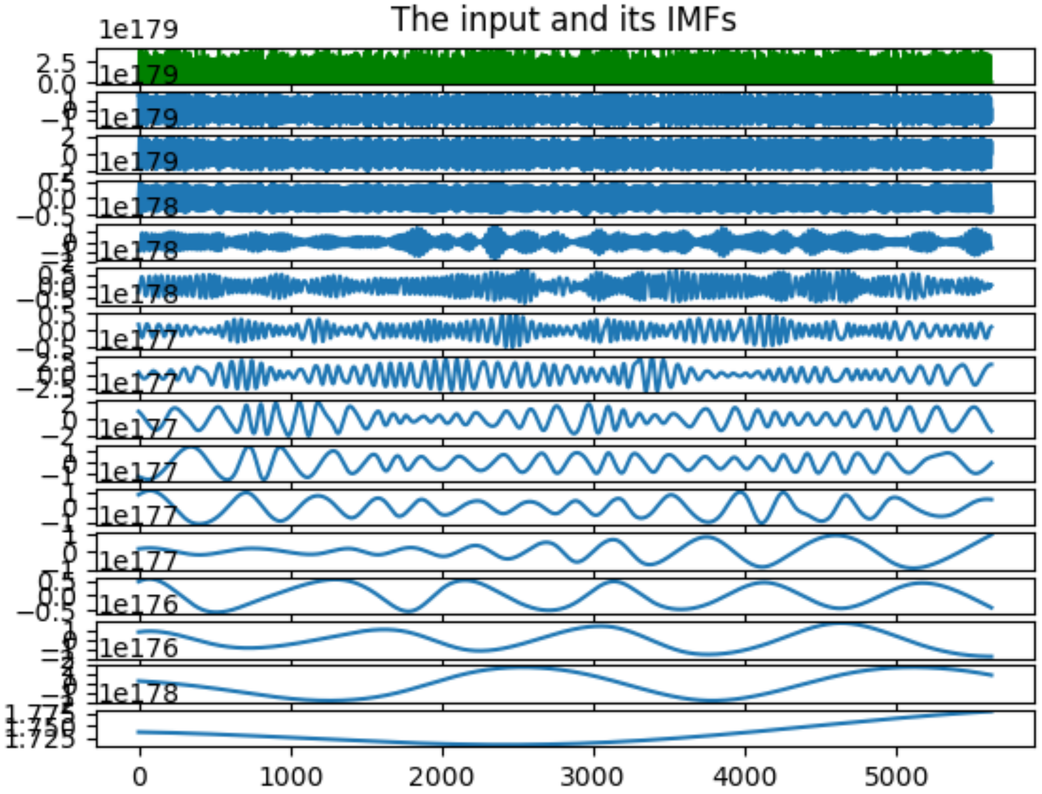

The input and its IMFs

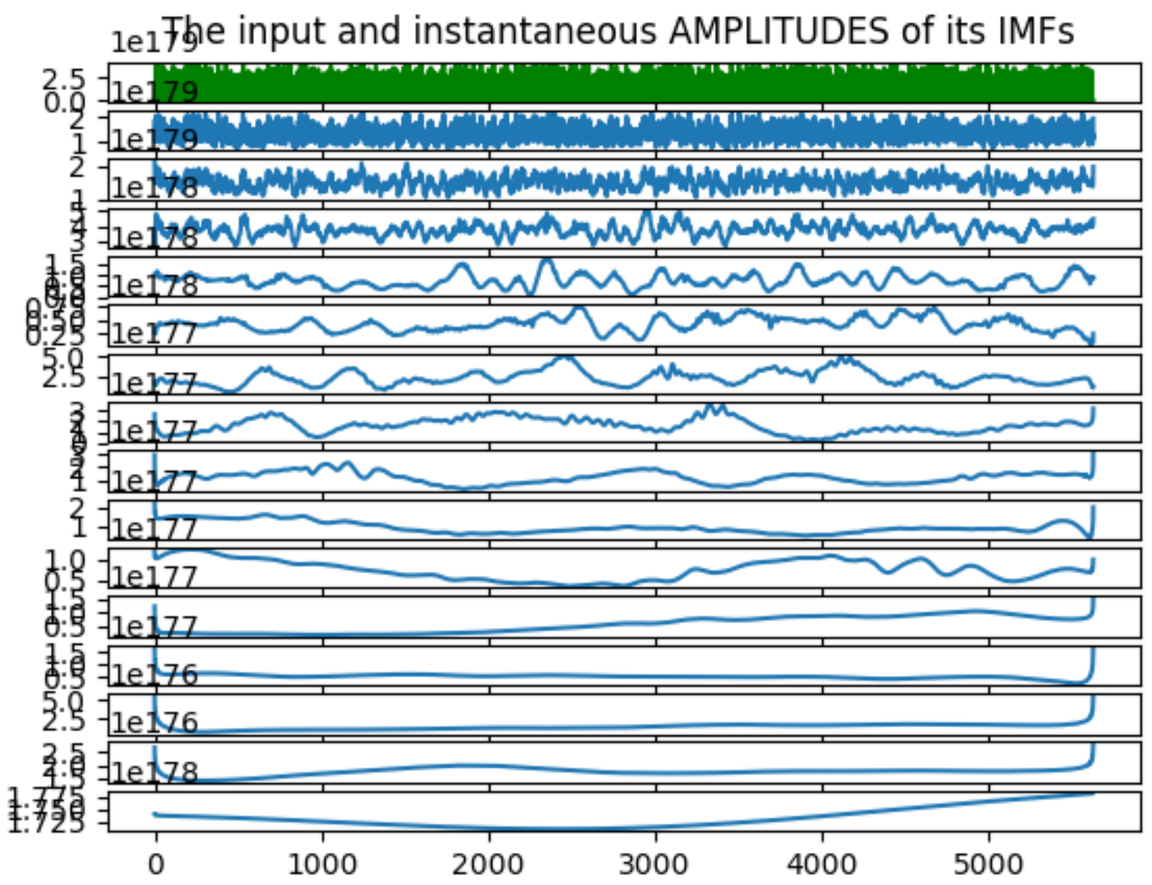

The input and instantaneous AMPLITUDES of its IMFs

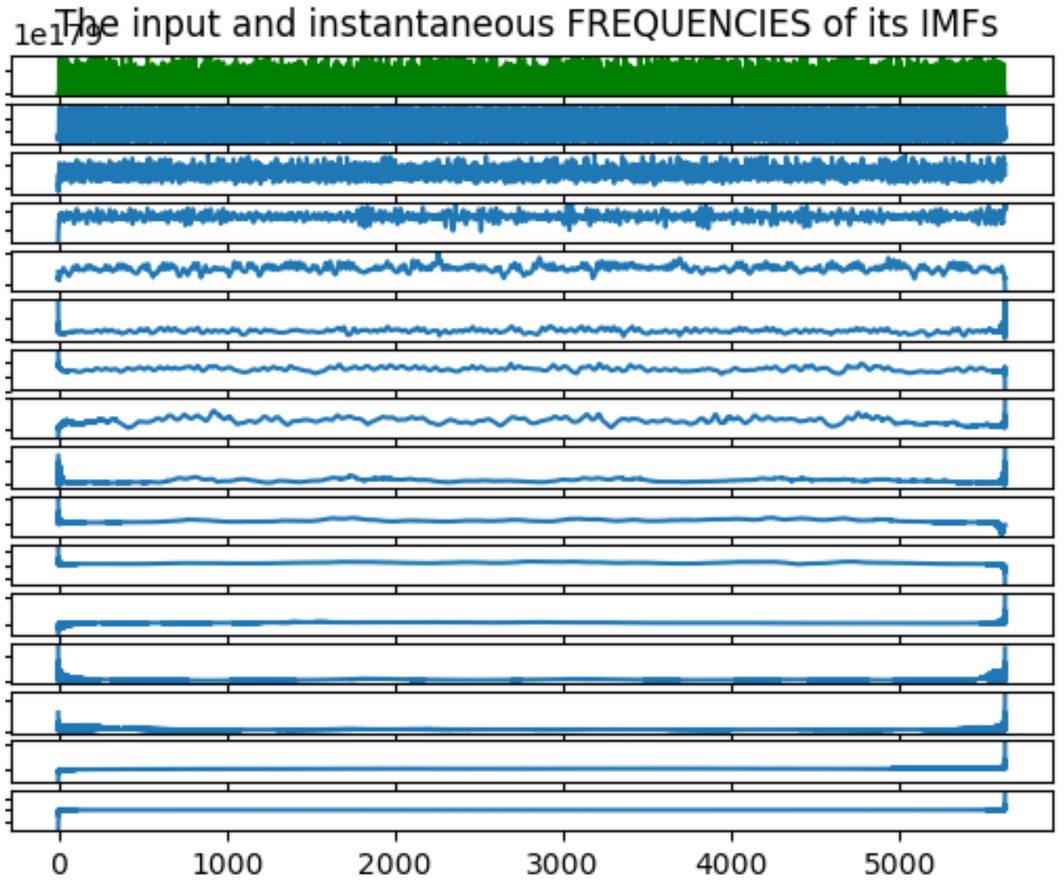

The input and instantaneous FREQUENCIES of its IMFs


1e179
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1000
2000
3000


1e179
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2400
2600
2800
3000
3200
3400


The input and its IMFs
1e179
1e179
1e179
1e179
1e178
1e178
1e178
1e177
1e177
1e177
1e177
1e177
1e176
1e176
1e178
0
1000
2000
3000
4000
5000


The input and instantaneous AMPLITUDES of its IMFs
1e179
1e179
1e179
1e178
1e178
1e178
1e177
1e177
1e177
1e177
1e177
1e177
1e176
1e176
1e178
0
1000
2000
3000
4000
5000


The input and instantaneous FREQUENCIES of its IMFs
1e179
0
1000
2000
3000
4000
5000