Overview of the Fundamentals of Deep Learning Models

Benoit Liquet

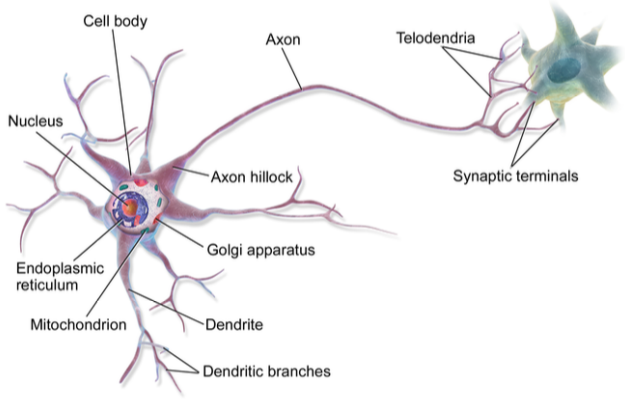
LMAP, University of Pau et Pays de L'Adour (France)

13-15 February 2026

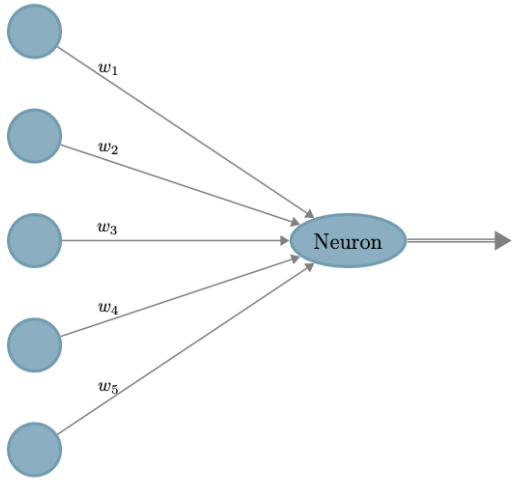
Session: Nuts & Bolts of AI in Neurosurgery (Theoretical and Practical Session)

Ghasem Azemi and Eric Suero Molina

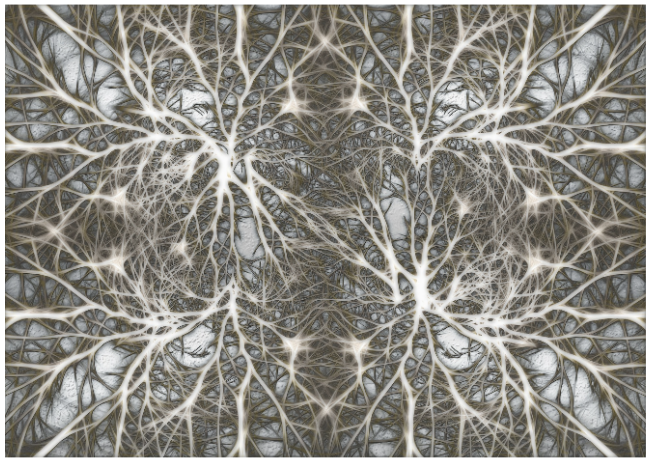
Biological vs. Artificial Neurons: The Building Blocks of Neural Networks



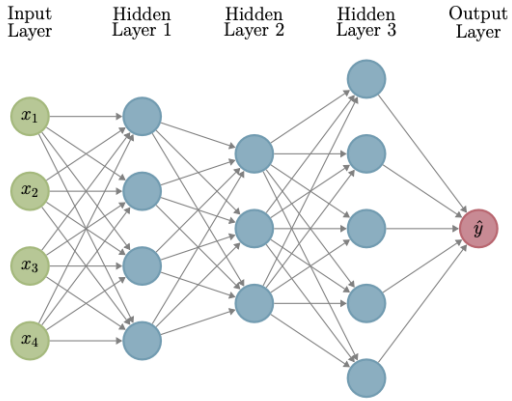
(a)



(b)

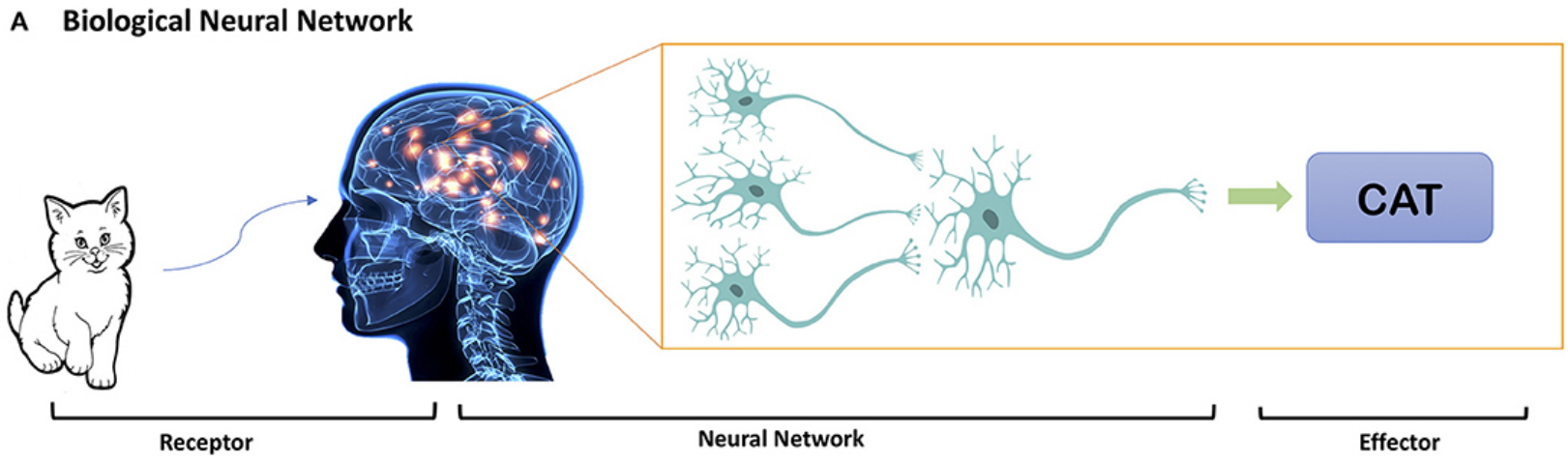


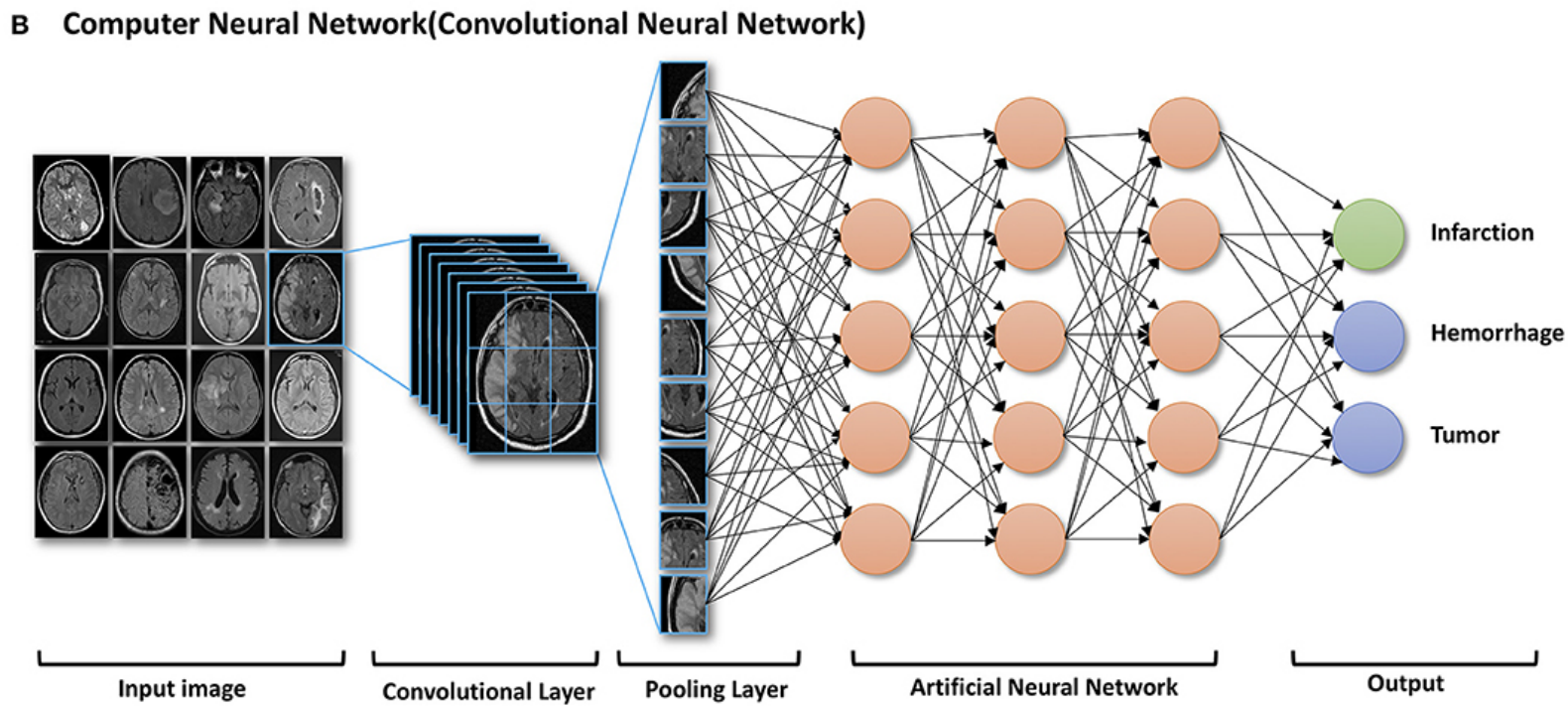
(c)



(d)

Brain Inspiration: CNNs for Medical Imaging

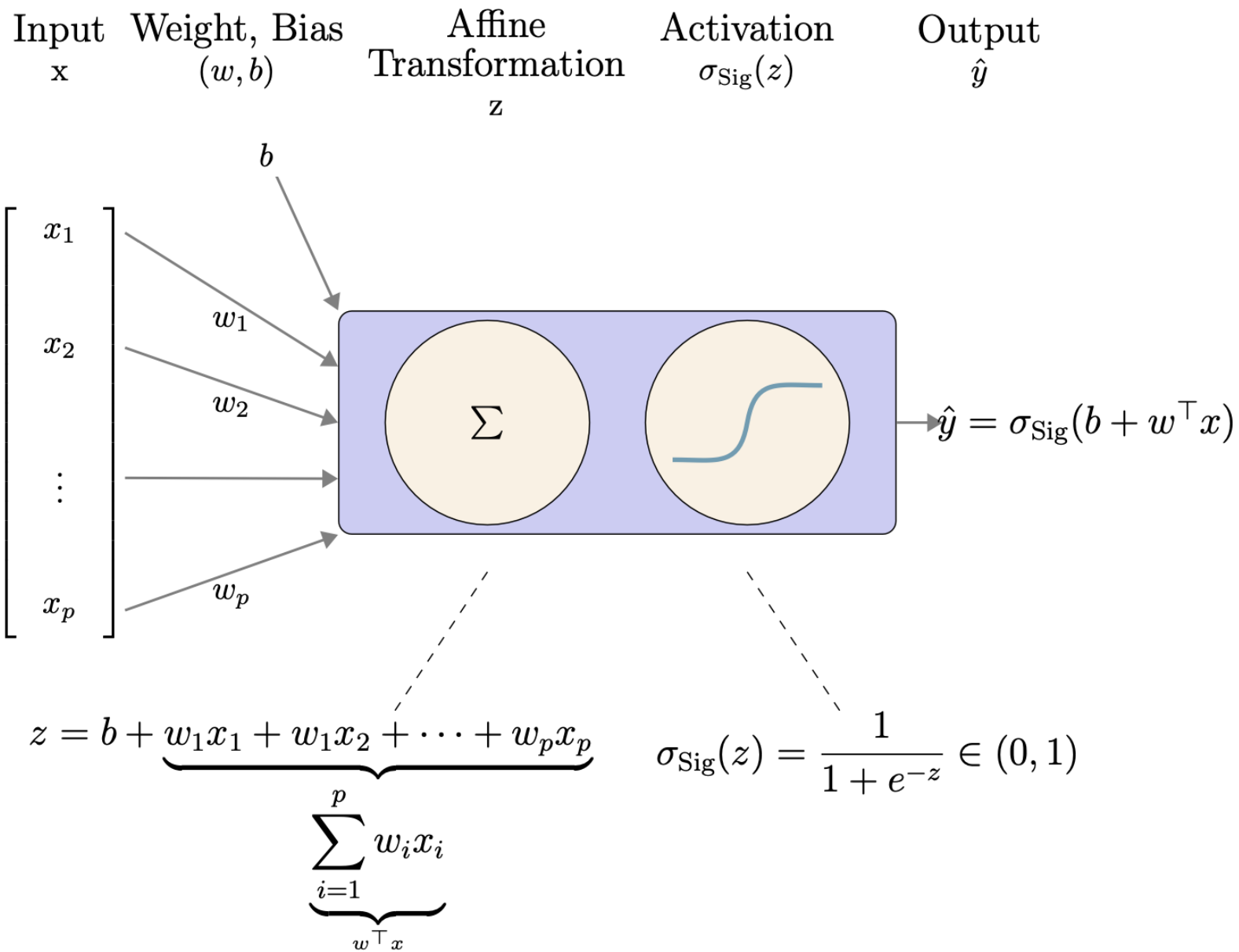




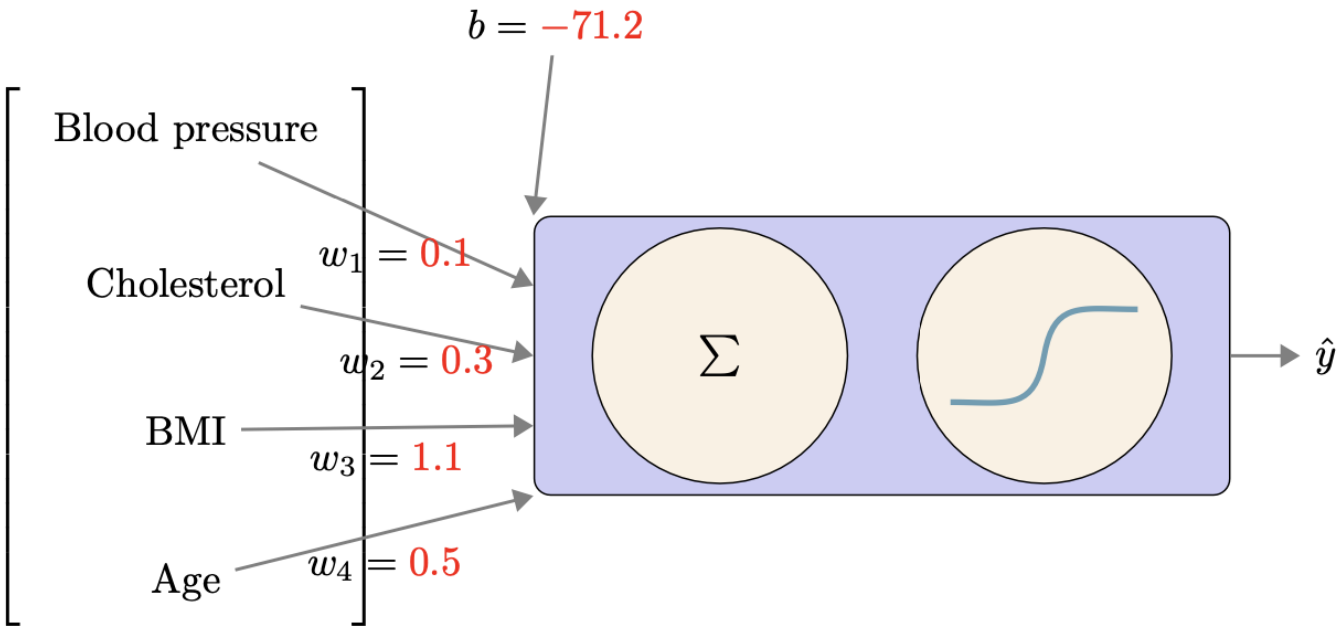
Question 1

How confident are you with the math or algebra behind deep learning models?

Artificial neuron: Sigmoid model (logistic model)



Toy example: predict stroke



Patient 1

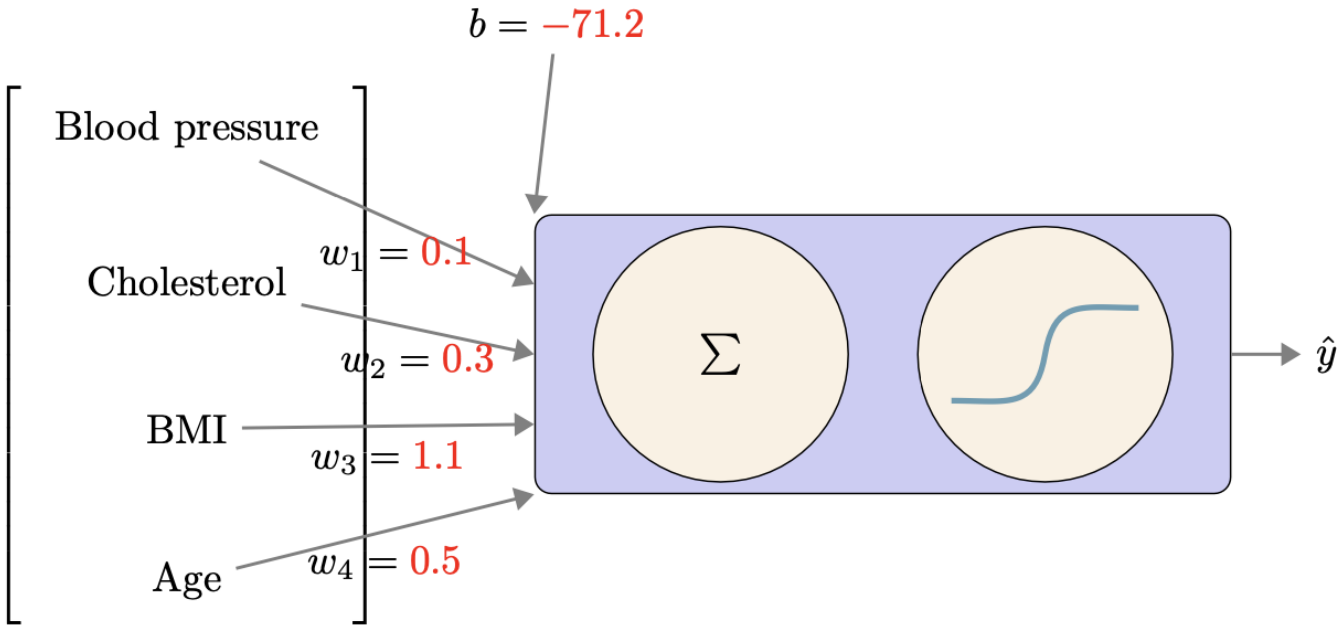
$$x = \begin{bmatrix} 130 \\ 6 \\ 28 \\ 55 \end{bmatrix}$$

$$z = \underbrace{-71.2}_{b} + \underbrace{0.1}_{w_1} \cdot \overbrace{130}^{x_1} + \underbrace{0.3}_{w_2} \cdot \overbrace{6}^{x_2} + \underbrace{1.1}_{w_3} \cdot \overbrace{28}^{x_3} + \underbrace{0.5}_{w_4} \cdot \overbrace{55}^{x_4}$$

$$= 1.9$$

$$\Rightarrow \quad \widehat{y} = \sigma_{\text{Sig}}(z) = \frac{1}{1 - e^{-1.9}} = \underbrace{0.71}_{\textbf{Probability of getting a stroke}}$$

Toy example: predict stroke



Patient 2

$$x = \begin{bmatrix} 130 \\ 5.5 \\ 23 \\ 56 \end{bmatrix}$$

$$z = \underbrace{-71.2}_{b} + \underbrace{0.1}_{w_1} \cdot \overbrace{130}^{x_1} + \underbrace{0.3}_{w_2} \cdot \overbrace{5.5}^{x_2} + \underbrace{1.1}_{w_3} \cdot \overbrace{23}^{x_3} + \underbrace{0.5}_{w_4} \cdot \overbrace{56}^{x_4}$$

$$= -3.25$$

$$\Rightarrow \quad \widehat{y} = \sigma_{\text{Sig}}(z) = \frac{1}{1 - e^{-(-3.25)}} = \underbrace{0.04}_{\textbf{Probability of getting a stroke}}$$

Sigmoid model as a binary classifier

A classifier can be constructed via a *decision rule* based on a threshold τ:

$$\widehat{\mathscr{Y}} = \begin{cases} 0 \text{ (negative)}, & \text{if} \quad \widehat{y} \leq \tau \\ 1 \text{ (positive)}, & \text{if} \quad \widehat{y} > \tau. \end{cases}$$

In many cases one selects the threshold at $\tau = 0.5$.

Sigmoid model as a binary classifier

A classifier can be constructed via a *decision rule* based on a threshold τ:

$$\widehat{\mathscr{Y}} = \begin{cases} 0 \text{ (negative)}, & \text{if} \quad \widehat{y} \leq \tau \\ 1 \text{ (positive)}, & \text{if} \quad \widehat{y} > \tau. \end{cases}$$

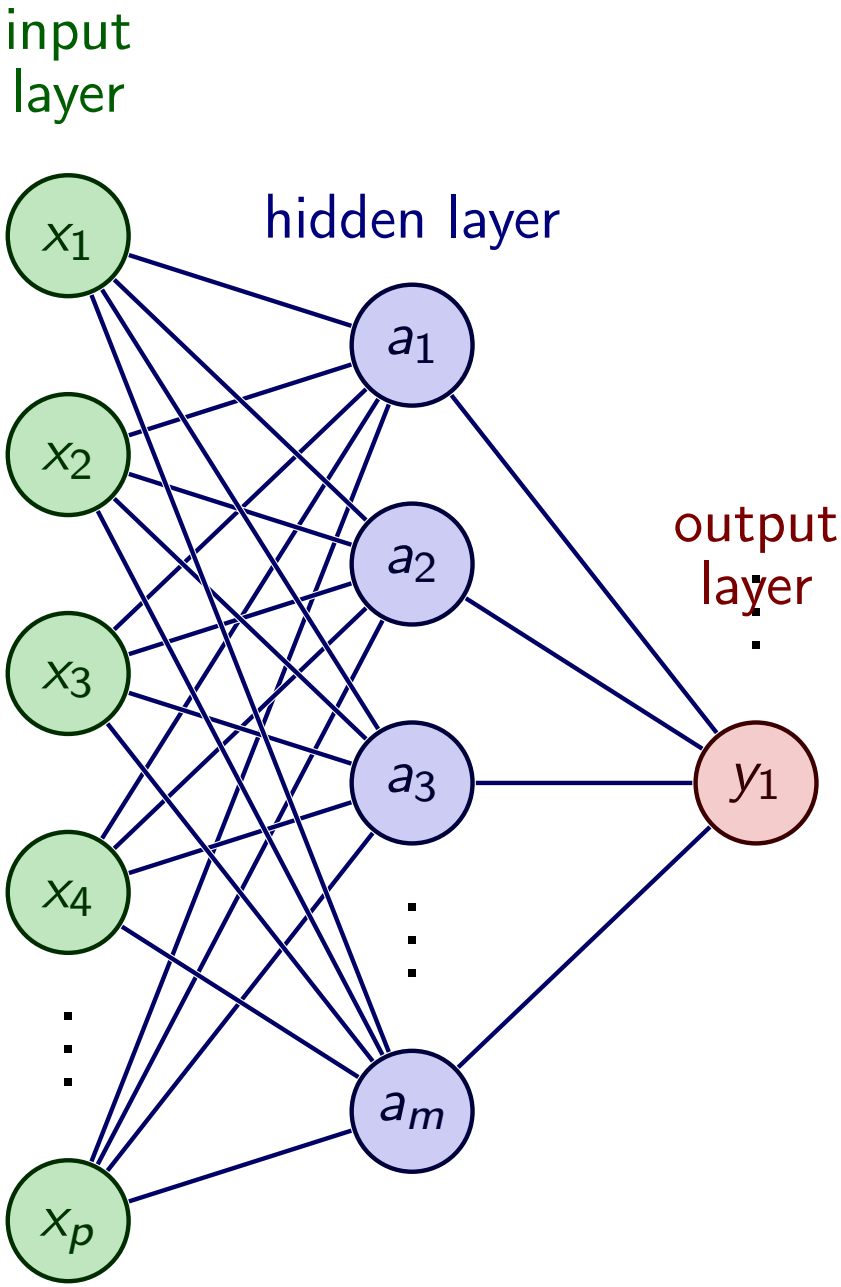
In many cases one selects the threshold at $\tau = 0.5$.

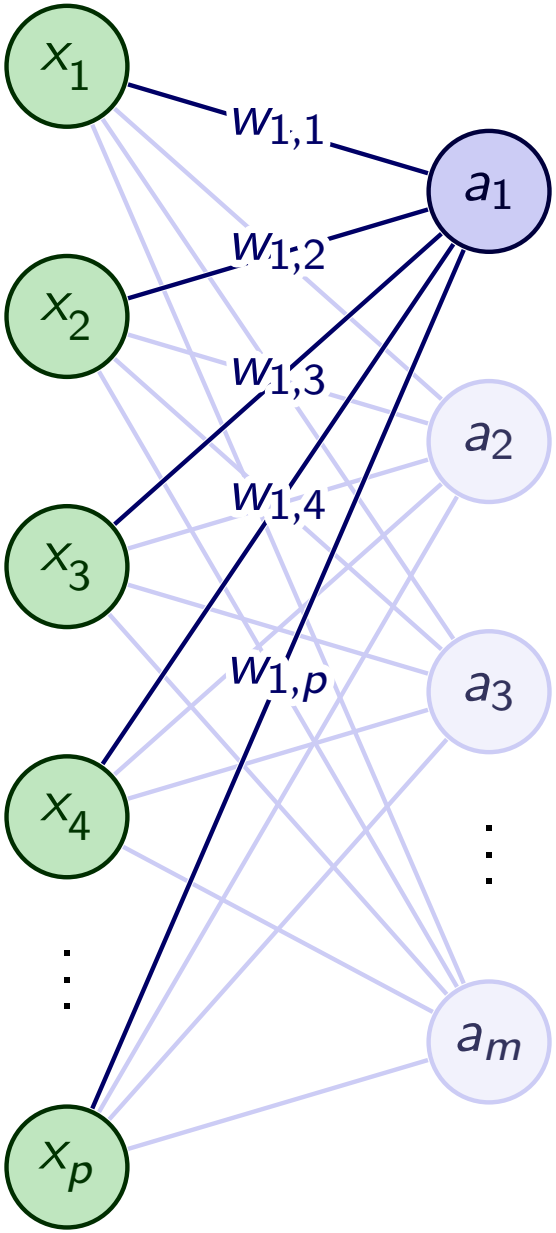
- Patient 1: $\widehat{y} = 0.71 > 0.5 \quad \Rightarrow \quad \widehat{\mathscr{Y}} = 1$ **(Positive)**
- Patient 2: $\widehat{y} = 0.04 \leq 0.5 \quad \Rightarrow \quad \widehat{\mathscr{Y}} = 0$ **(Negative)**

Question 2

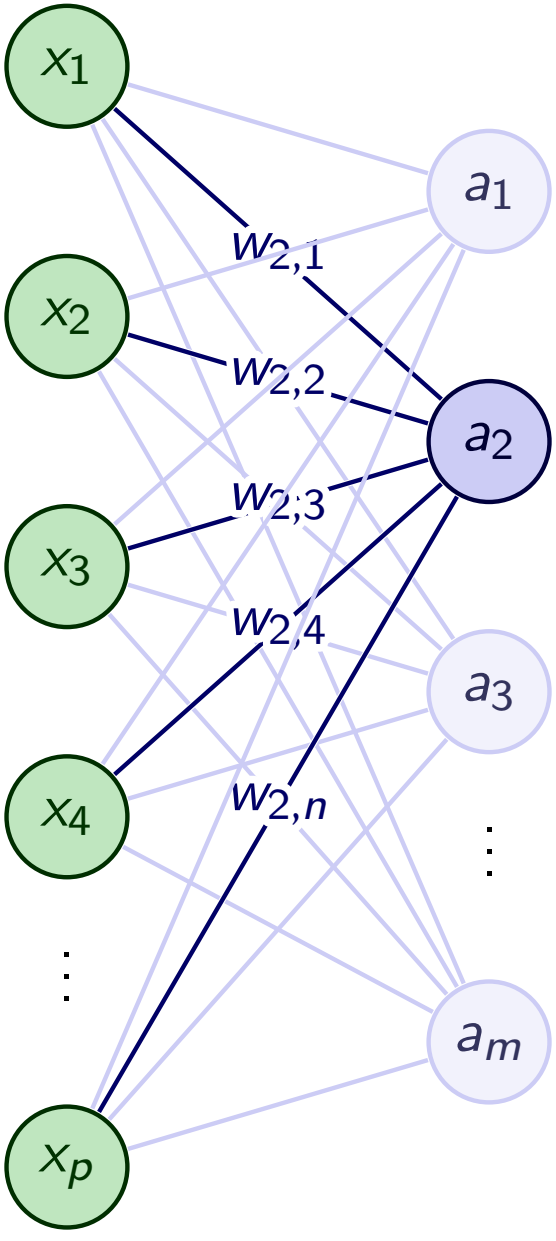
Will changing the threshold after applying the sigmoid function affect the model's performance?

Single hidden layer

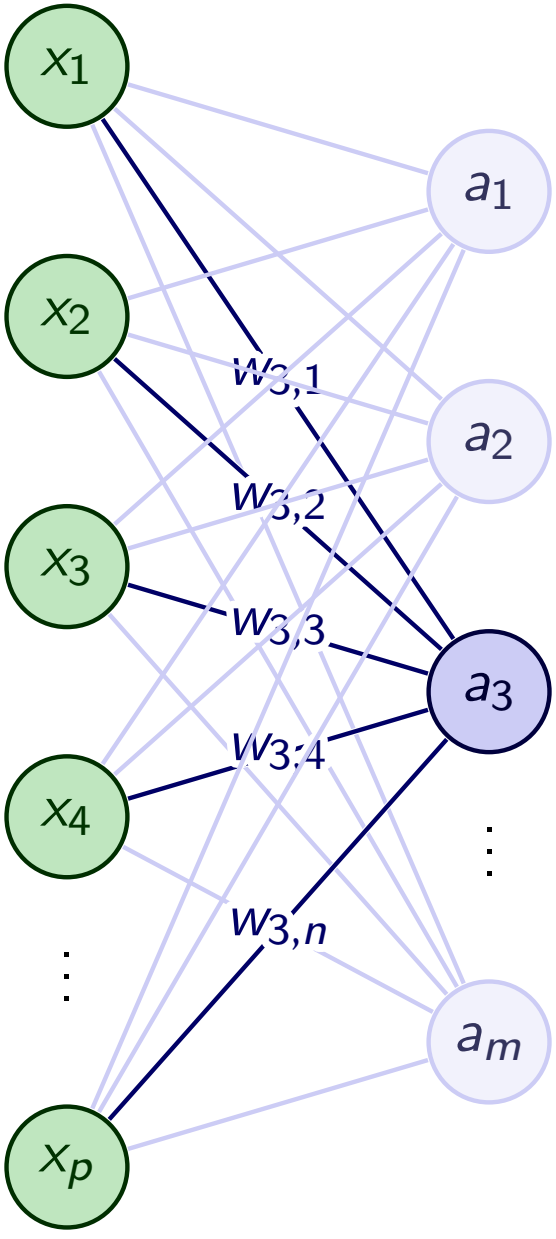




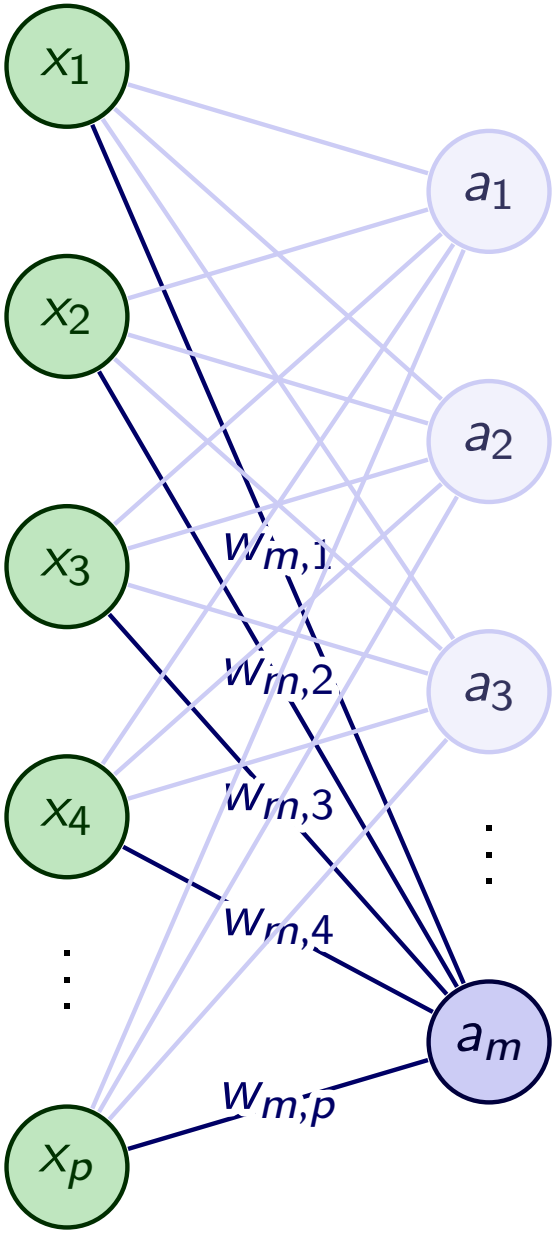
$$= \sigma\left(\underbrace{w_{1,1}x_1 + w_{1,2}x_2 + \ldots + w_{1,p}x_n + b_1}_{z_1}\right) = \sigma(z_1)$$



$$a_2 = \sigma\left(\underbrace{w_{2,1}x_1 + w_{2,2}x_2 + \ldots + w_{2,p}x_n + b_2}_{z_2}\right)$$
$$= \sigma(z_2)$$

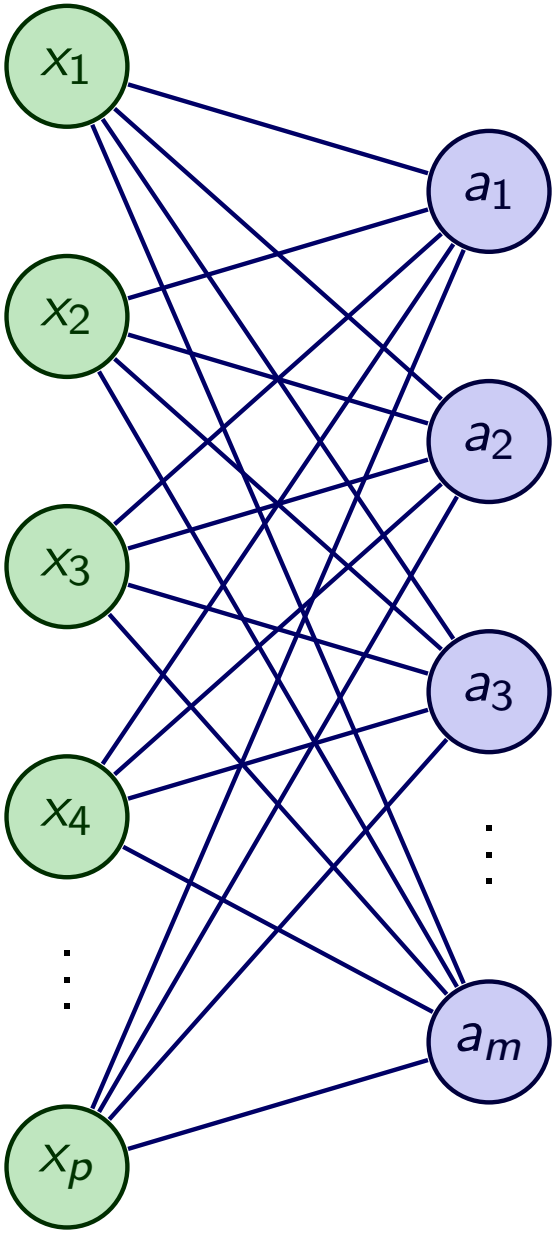


$$a_3 = \sigma\left(\underbrace{w_{3,1}x_1 + w_{3,2}x_2 + \ldots + w_{3,p}x_n + b_3}_{z_3}\right)$$
$$= \sigma(z_3)$$

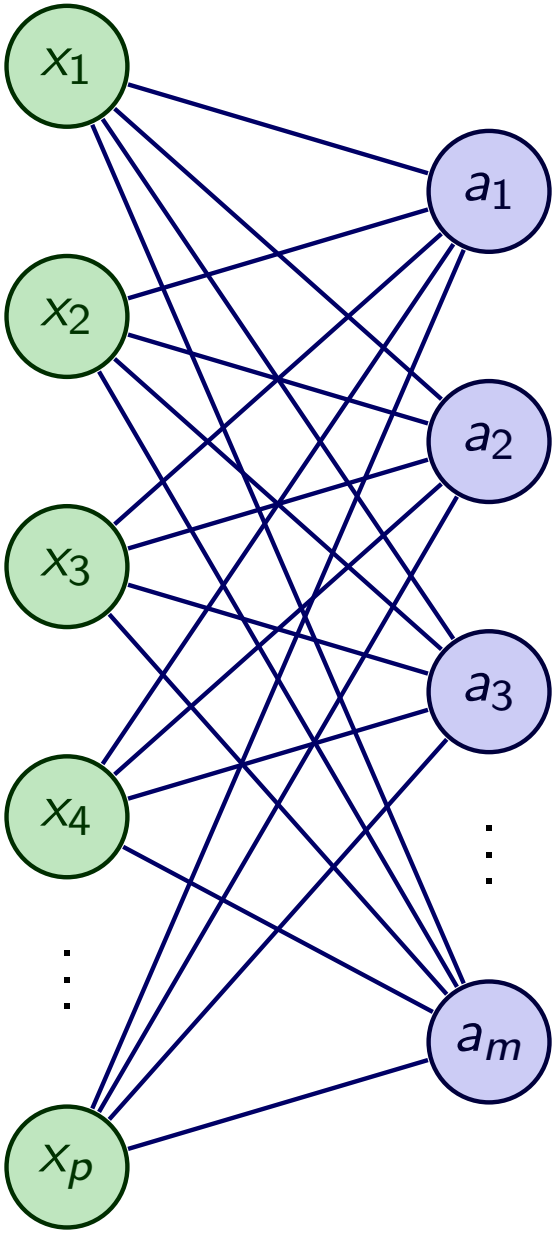


$$a_m = \sigma\left(\underbrace{w_{m,1}x_1 + w_{m,2}x_2 + \ldots + w_{m,p}x_n + b_m}_{z_m}\right)$$

$$= \sigma(z_m)$$



$$
\begin{cases}
a_1 = \sigma\left(w_{1,1}x_1 + w_{1,2}x_2 + \cdots + w_{1,p}x_p + b_1\right) \\
a_2 = \sigma\left(w_{2,1}x_1 + w_{2,2}x_2 + \cdots + w_{2,p}x_p + b_2\right) \\
a_3 = \sigma\left(w_{3,1}x_1 + w_{3,2}x_2 + \cdots + w_{3,p}x_p + b_3\right) \\
\quad \vdots = \qquad\qquad\qquad\qquad \vdots \\
a_m = \sigma\left(w_{m,1}x_1 + w_{m,2}x_2 + \cdots + w_{m,p}x_p + b_m\right)
\end{cases}
$$



$$
\begin{cases}
a_1 = \sigma(w_{1,1}x_1 + \cdots + w_{1,p}x_p + b_1) \\
a_2 = \sigma(w_{2,1}x_1 + \cdots + w_{2,p}x_p + b_2) \\
a_3 = \sigma(w_{3,1}x_1 + \cdots + w_{3,p}x_p + b_3) \\
\vdots = \qquad\qquad\qquad \vdots \\
a_m = \sigma(w_{m,1}x_1 + \cdots + w_{m,p}x_p + b_m)
\end{cases}
$$

$$
a = \sigma(\underbrace{Wx + b}_{z})
$$

Matrix notation

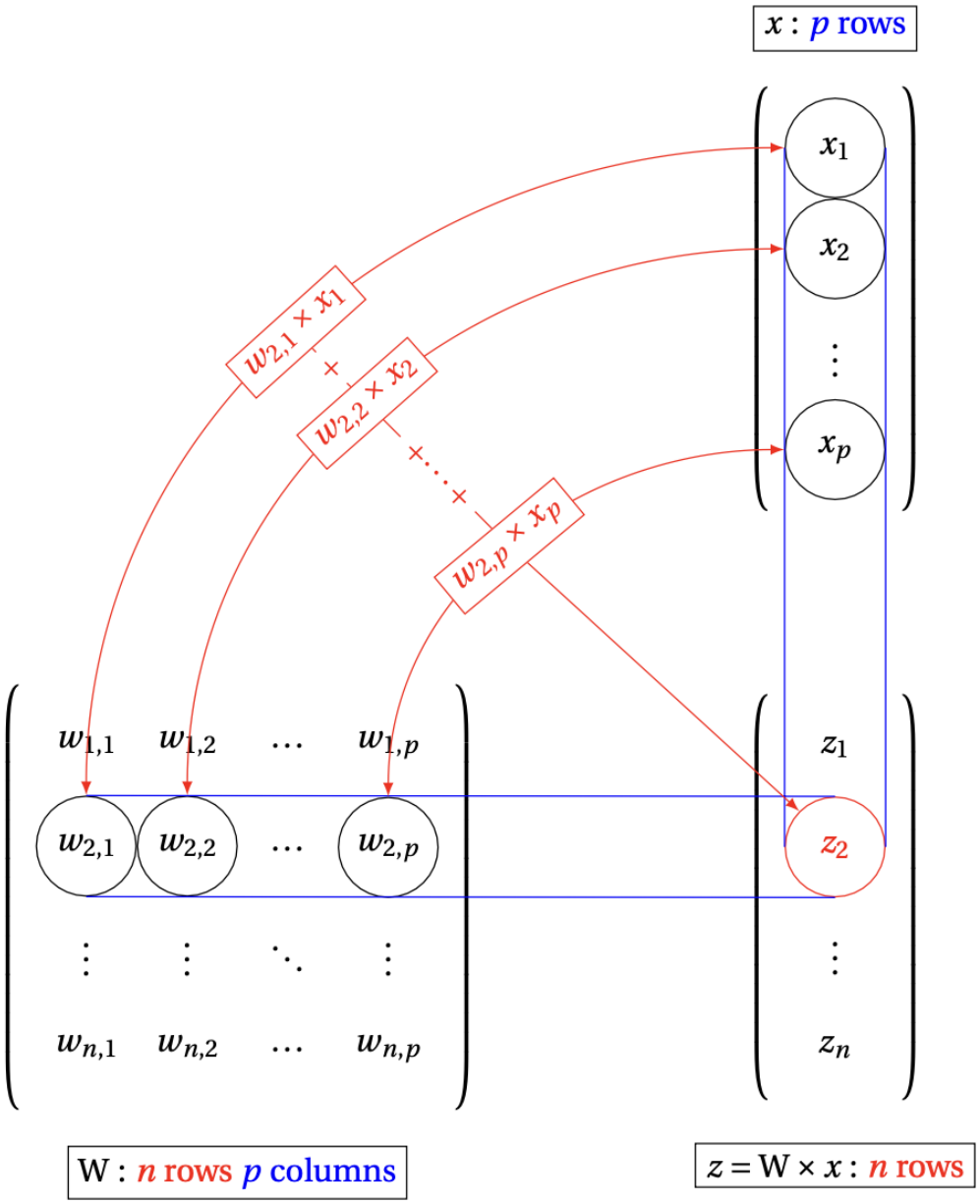
Vectorized form of neuron activations:

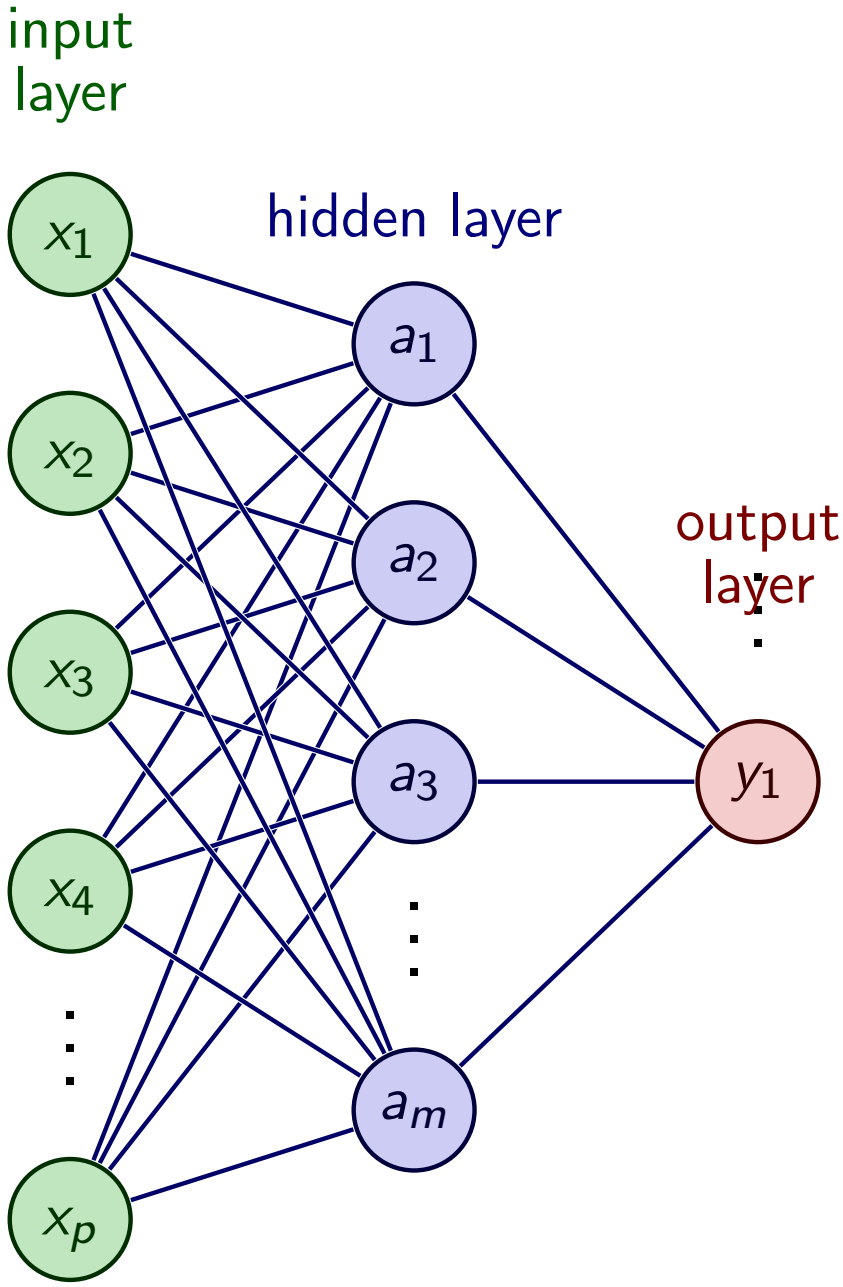
$$a = \sigma(Wx + b)$$

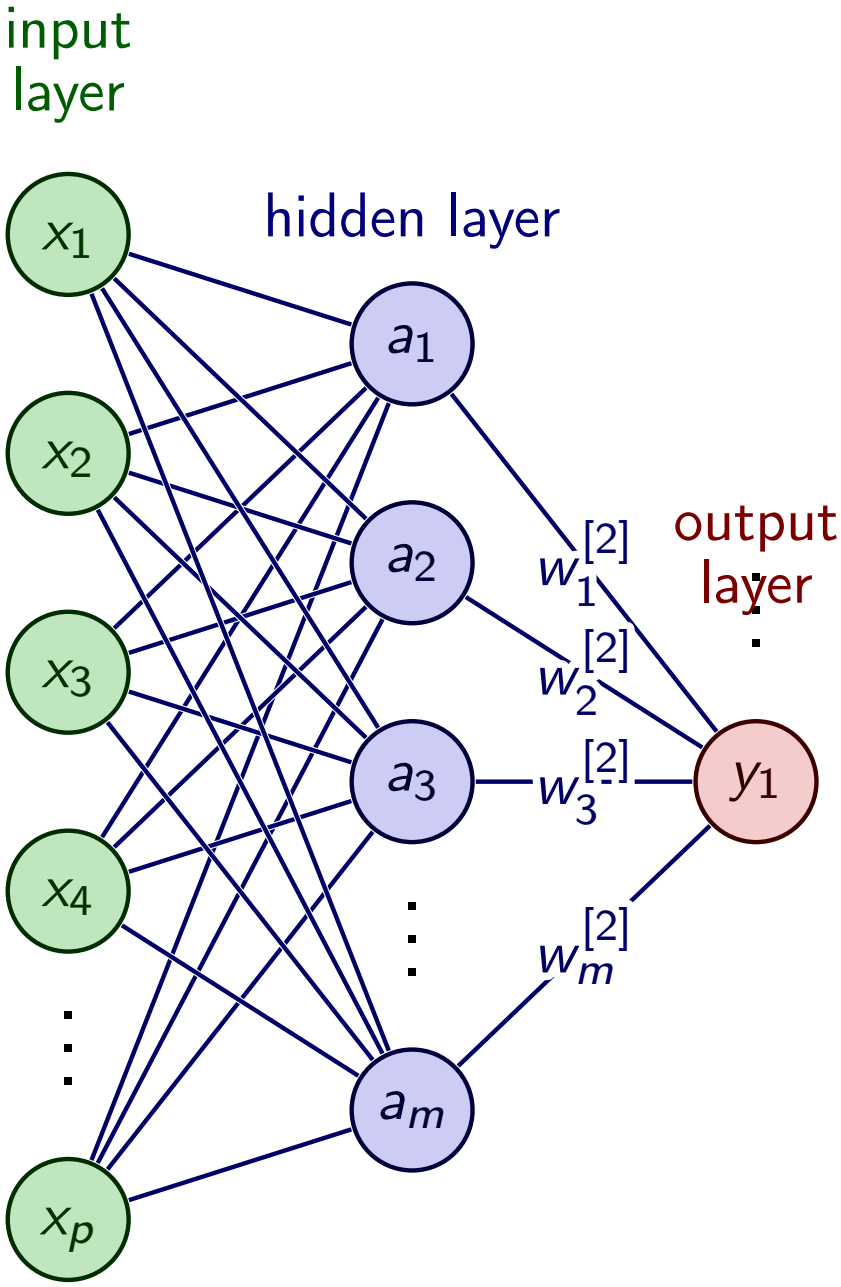
$$\overbrace{\begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix}}^{\text{a}} = \sigma\Bigg(\underbrace{\overbrace{\begin{pmatrix} w_{1,1} & \cdots & w_{1,p} \\ \vdots & \ddots & \vdots \\ w_{m,1} & \cdots & w_{m,p} \end{pmatrix}}^{W} \overbrace{\begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix}}^{\text{Input } x} + \overbrace{\begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}}^{b}}_{z}\Bigg)$$

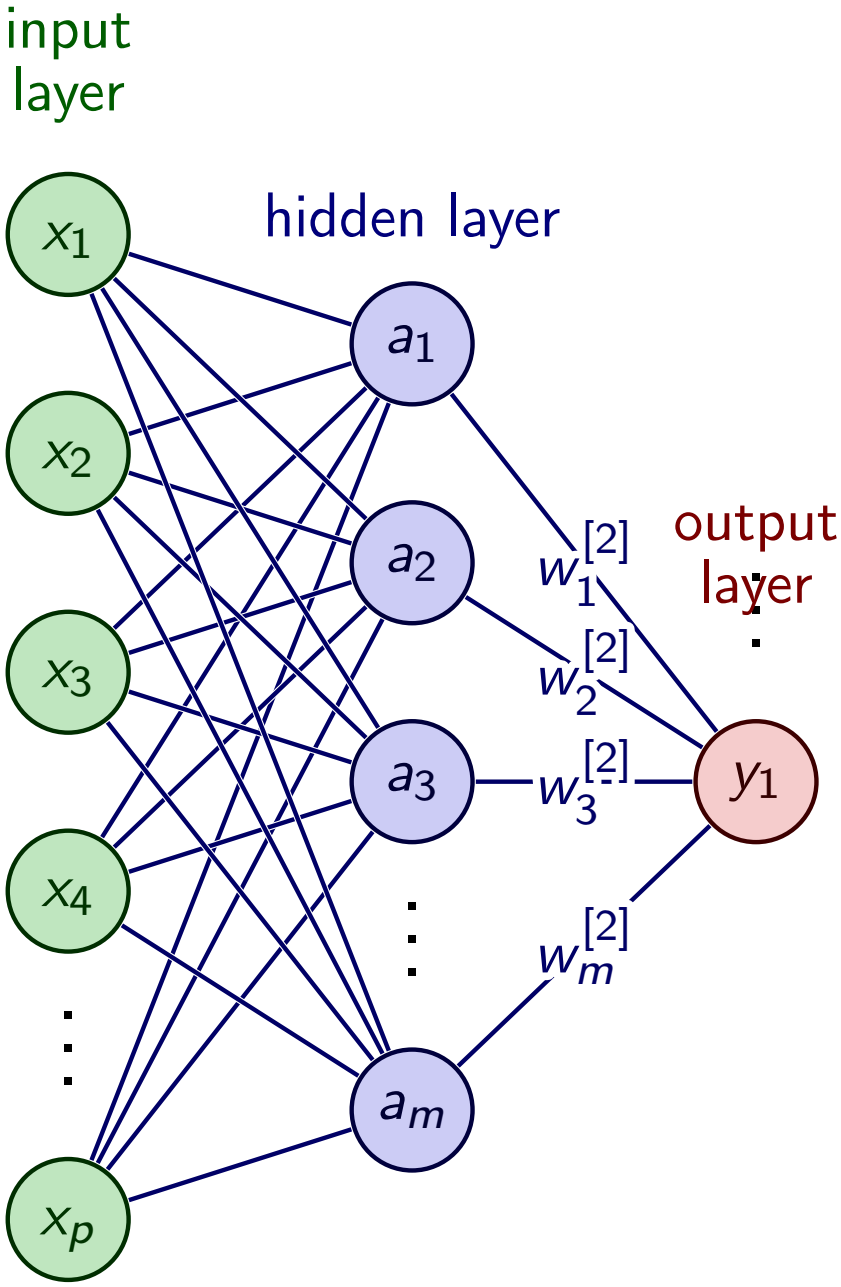
$$= \sigma \begin{pmatrix} z_1 \\ \vdots \\ z_p \end{pmatrix} = \begin{pmatrix} \sigma(z_1) \\ \vdots \\ \sigma(z_p) \end{pmatrix}$$

Matrix-vector multiplication: $W \times x = z$ with $b = 0$

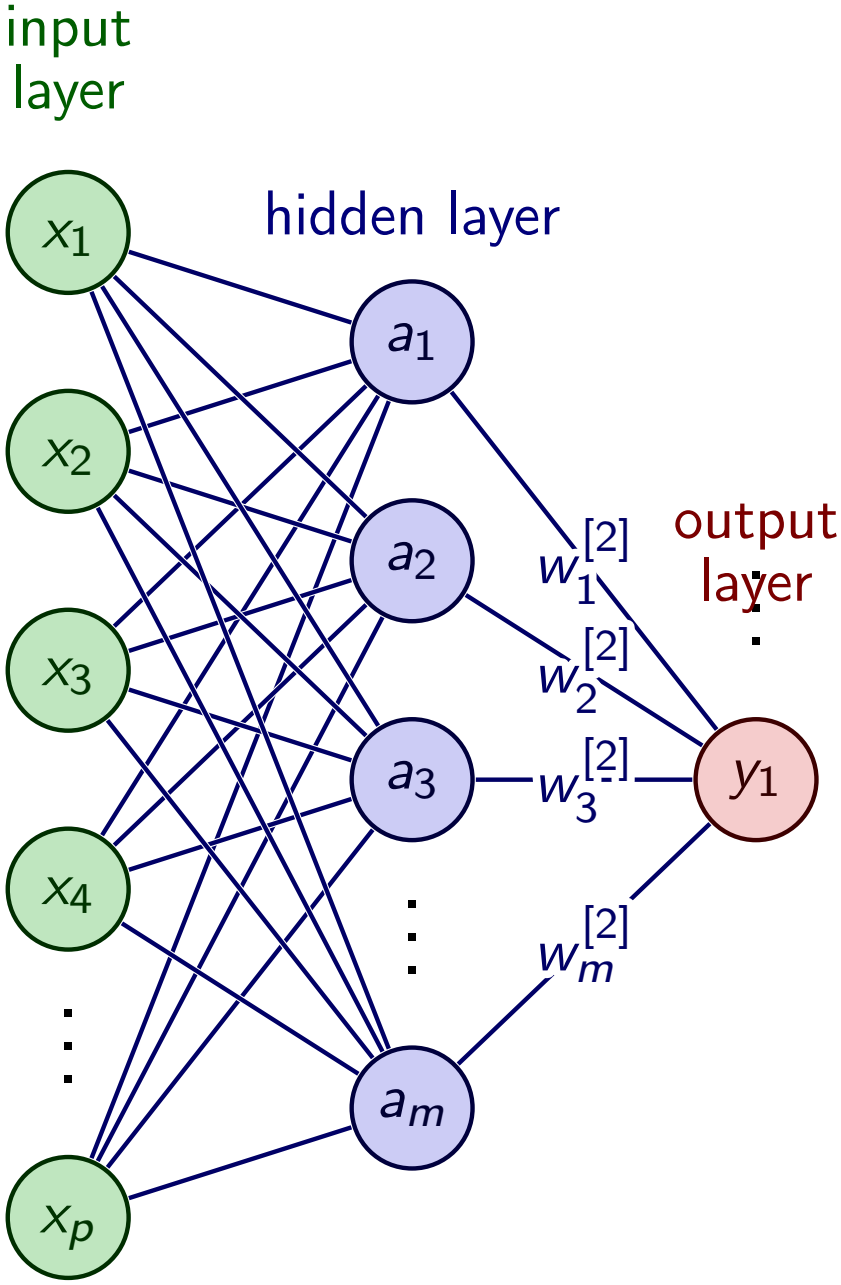



input
layer
hidden layer
output
layer
x_1
x_2
x_3
x_4
x_p
a_1
a_2
a_3
a_m
y_1


input
layer
hidden layer
output
layer
x_1
x_2
x_3
x_4
x_p
a_1
a_2
a_3
a_m
y_1
$w_1^{[2]}$
$w_2^{[2]}$
$w_3^{[2]}$
$w_m^{[2]}$



$$y = \sigma\left(w_1^{[2]} a_1 + w_2^{[2]} a_2 + \cdots + w_m^{[2]} a_m + b^{[2]}\right)$$
$$= \sigma\left(W^{[2]} a + b^{[2]}\right)$$

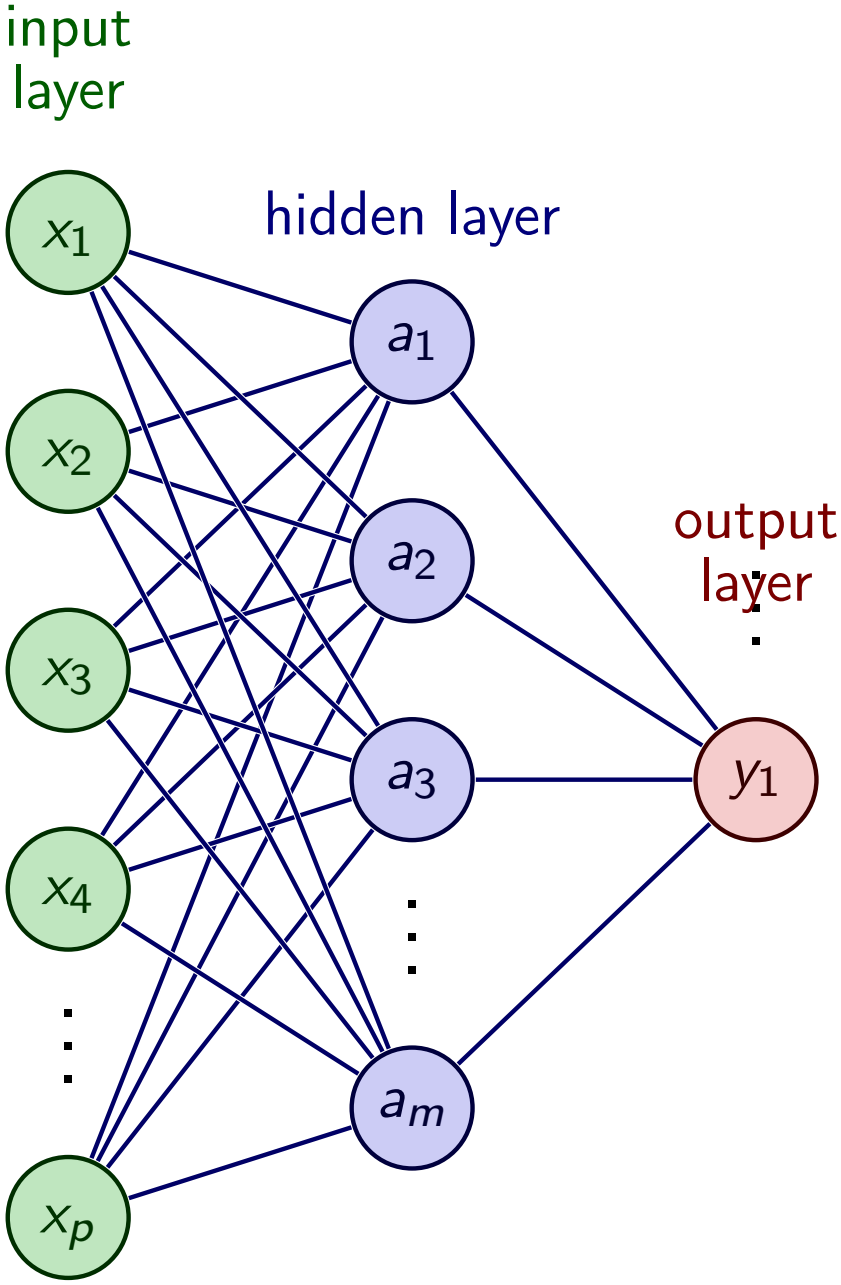


$$y = \sigma\left(\overbrace{w_1^{[2]}a_1 + w_2^{[2]}a_2 + \cdots + w_m^{[2]}a_m}^{W^{[2]}a} + b^{(2)}\right)$$

$$= \sigma\left(W^{[2]}a + b^{(2)}\right)$$

$$W^{[2]}a = \overbrace{\begin{pmatrix} w_1^{[2]} & w_2^{[2]} & \cdots & w_m^{[2]} \end{pmatrix}}^{W^{[2]}} \overbrace{\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix}}^{a}$$

$$= \left(w_1^{[2]}a_1 + w_2^{[2]}a_2 + \cdots + w_m^{[2]}a_m\right)$$

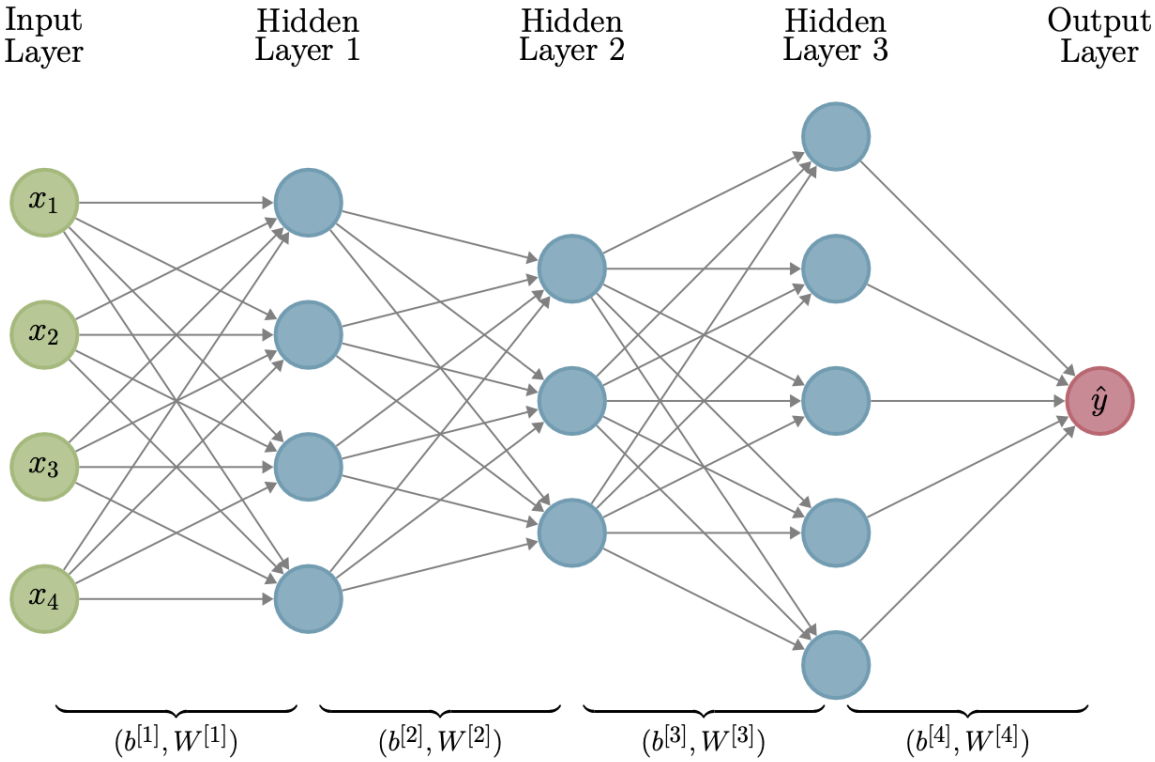


$$y = \sigma\left(W^{[2]} \underbrace{(\sigma(W^{[1]}x + b^{[1]})}_{a^{[1]}} + b^{[2]} \right)$$

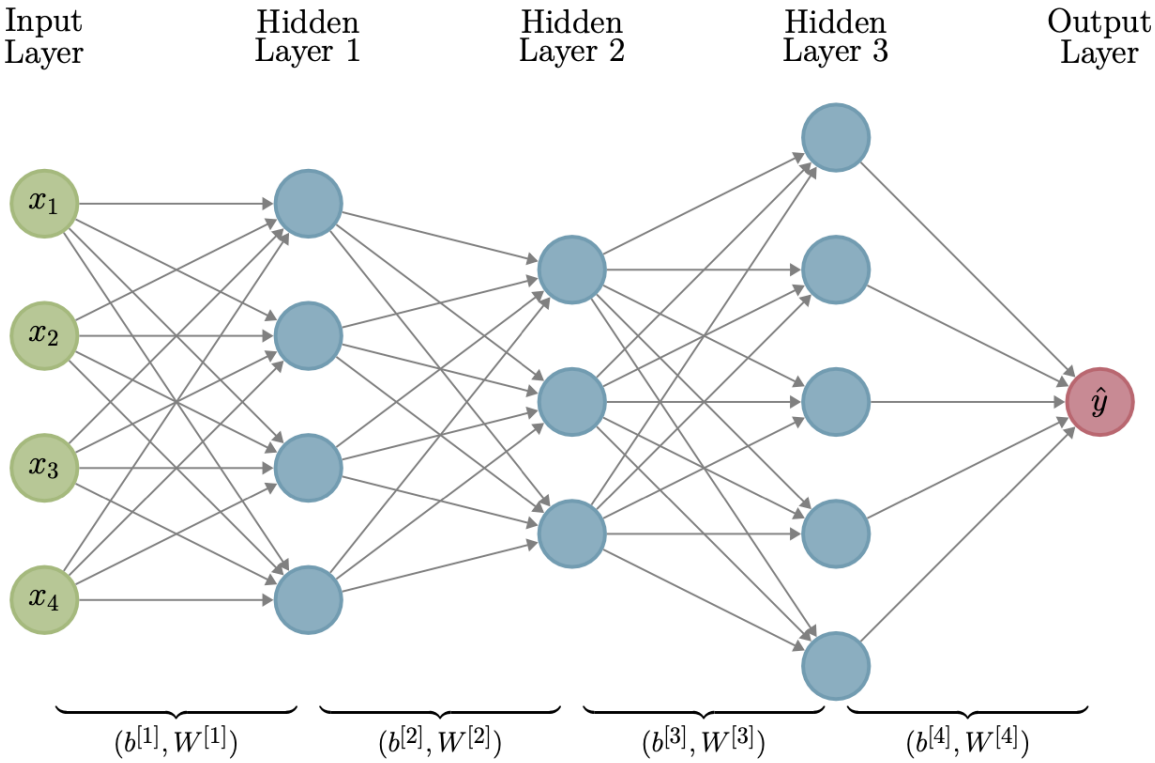
$$= \underbrace{f^{[2]}(f^{[1]}(x))}_{f_\theta(x)}$$

where $\theta = (W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]})$

Fully Connected Neural Network or feedforward networks

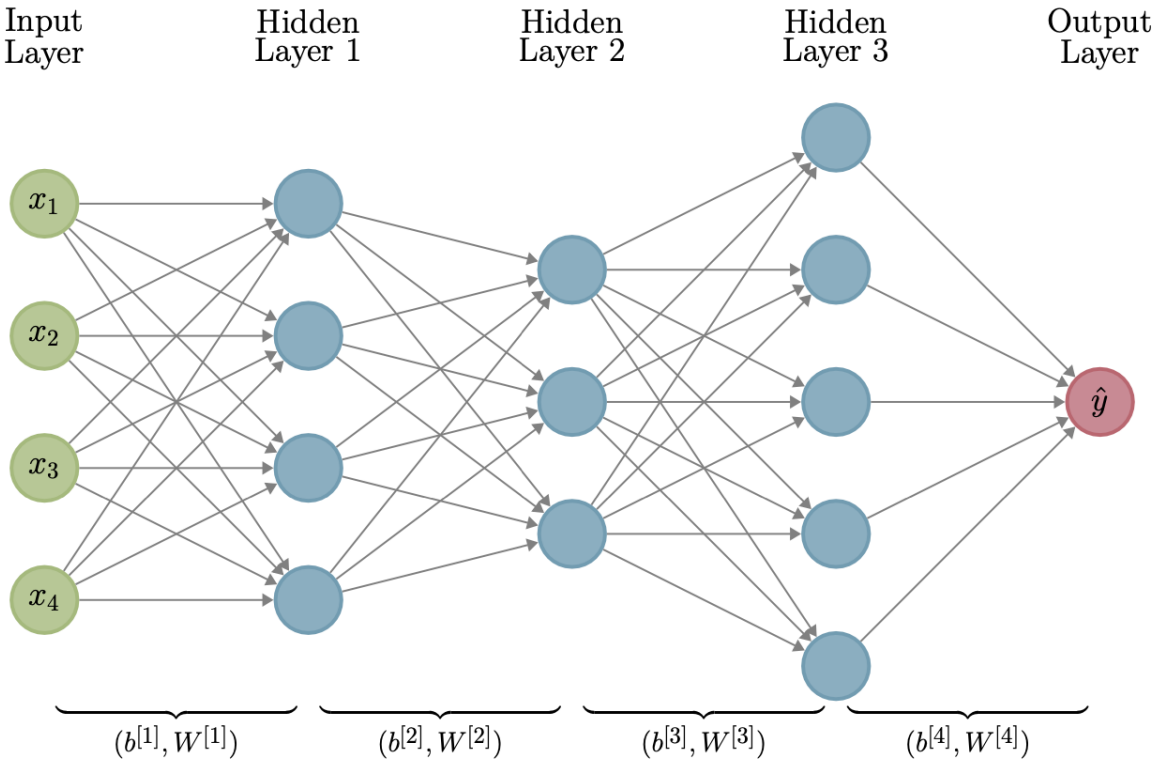


Fully Connected Neural Network or feedforward networks



Composition of function : $\hat{y} = f^{[L]}(f^{[L-1]}(f^{[L-2]}(\ldots(f^{[1]}(x))\ldots)))$

Fully Connected Neural Network or feedforward networks



Composition of function : $\hat{y} = f^{[L]}(f^{[L-1]}(f^{[L-2]}(\ldots(f^{[1]}(x))\ldots)))$

$$\hat{y} = \overbrace{\sigma^{[4]}(W^{[4]}\overbrace{\sigma^{[3]}(W^{[3]}\overbrace{\sigma^{[2]}(W^{[2]}\underbrace{\sigma^{[1]}(W^{[1]}x + b^{[1]})}_{f^{[1]}} + b^{[2]})}^{f^{[2]}} + b^{[3]})}^{f^{[3]}} + b^{[4]})}^{f^{[4]}}$$

Question 3

Does a fully connected neural network with one hidden layer allow interactions between input variables?

Why Single Hidden Layer is Useful?

Universal Approximation Theorem

Consider a continuous function $f^* : \mathcal{K} \to \mathbb{R}^q$ where $\mathcal{K} \subset \mathbb{R}^p$ is a compact set. Then for any any $\sigma(\cdot)$ non-polynomial activation function and any $\varepsilon > 0$ there exists an N_1 and parameters $W^{[1]} \in \mathbb{R}^{N_1 \times p}$, $b^{[1]} \in \mathbb{R}^{N_1}$, and $W^{[2]} \in \mathbb{R}^{q \times N_1}$, such that the function

$$f_\theta(x) = W^{[2]}\sigma(W^{[1]}x + b^{[1]}),$$

satisfies $||f_\theta(x) - f^*(x)|| < \varepsilon$ for all $x \in \mathcal{K}$.

Why Single Hidden Layer is Useful?

Universal Approximation Theorem

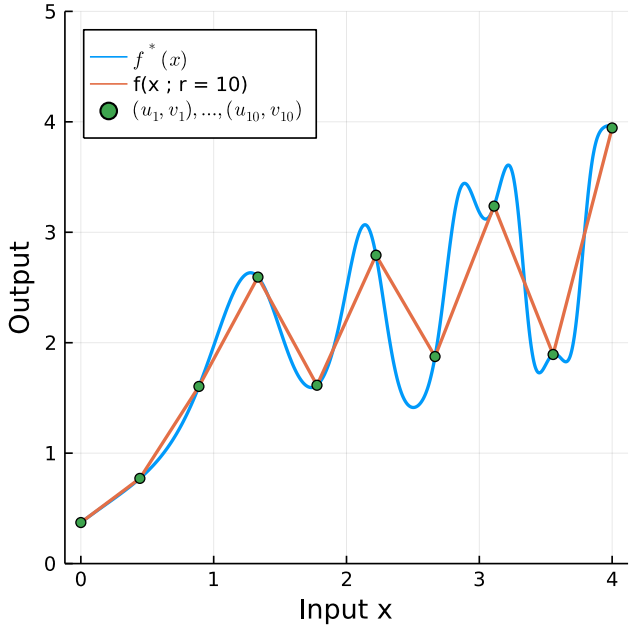
Consider a continuous function $f^* : \mathcal{K} \to \mathbb{R}^q$ where $\mathcal{K} \subset \mathbb{R}^p$ is a compact set. Then for any any $\sigma(\cdot)$ non-polynomial activation function and any $\varepsilon > 0$ there exists an N_1 and parameters $W^{[1]} \in \mathbb{R}^{N_1 \times p}$, $b^{[1]} \in \mathbb{R}^{N_1}$, and $W^{[2]} \in \mathbb{R}^{q \times N_1}$, such that the function

$$f_\theta(x) = W^{[2]}\sigma(W^{[1]}x + b^{[1]}),$$

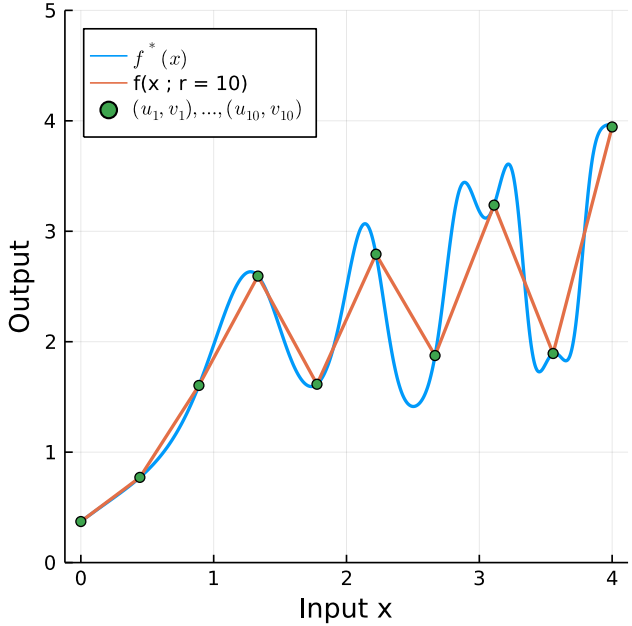
satisfies $||f_\theta(x) - f^*(x)|| < \varepsilon$ for all $x \in \mathcal{K}$.

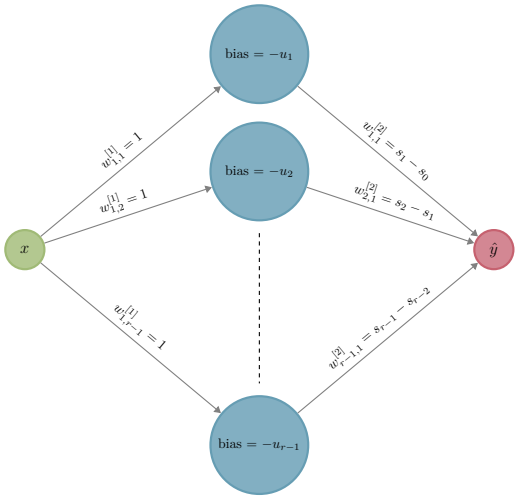
In simple words: *A neural network with just one hidden layer can, in principle, learn to approximate any smooth function, if it has enough hidden neurons.*

Simple Function Approximation

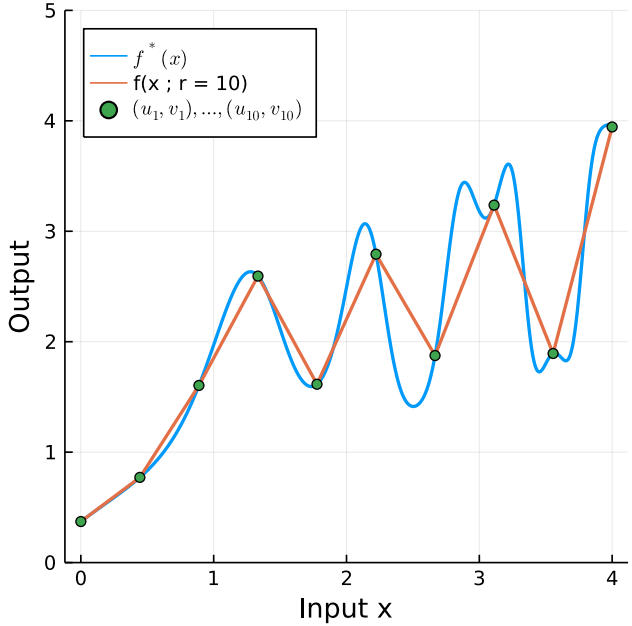


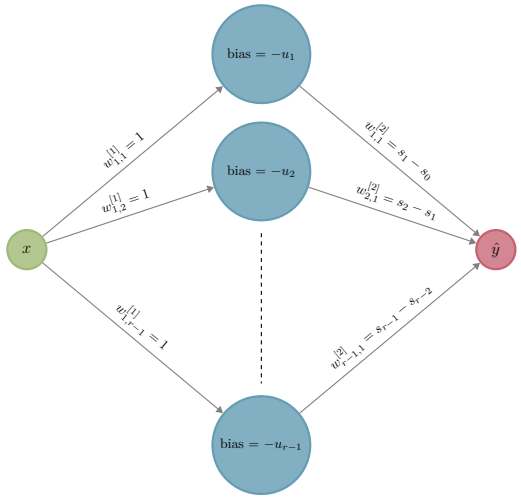
Simple Function Approximation

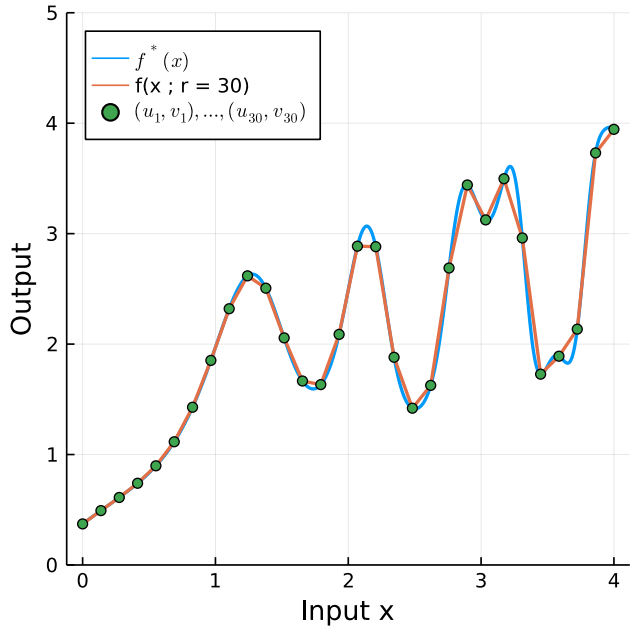




Simple Function Approximation

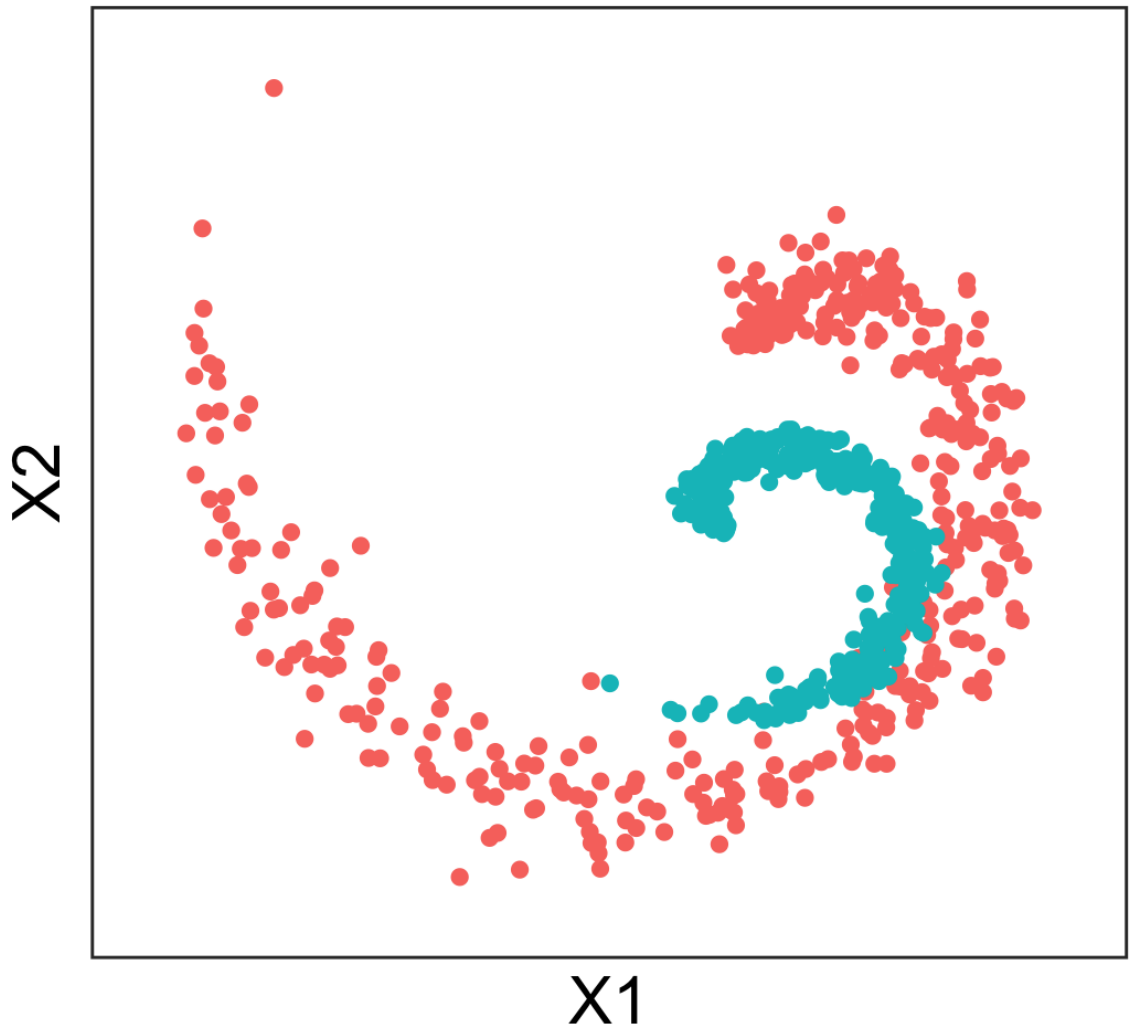






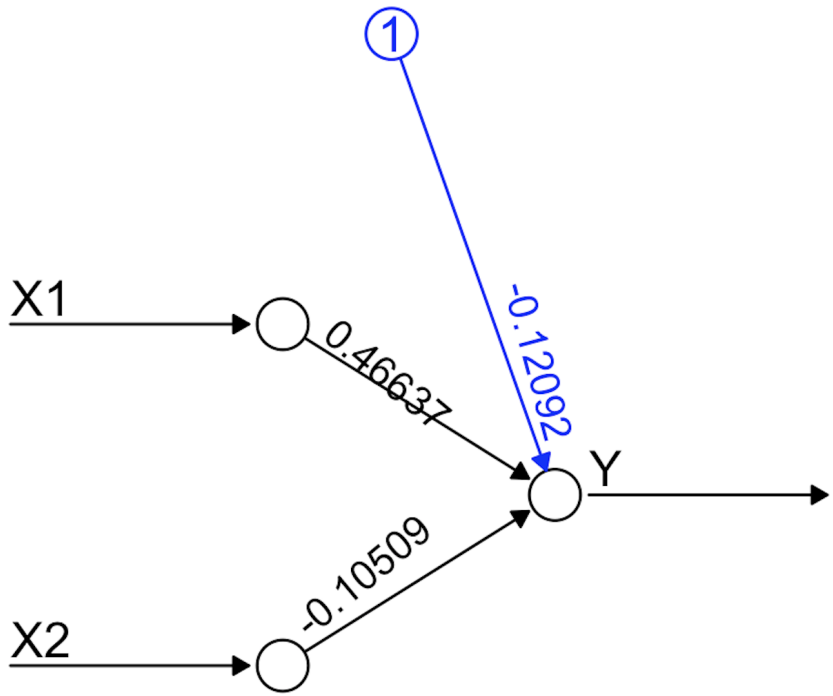
Expressivity:

Binary classification task with two features: x_1 and x_2



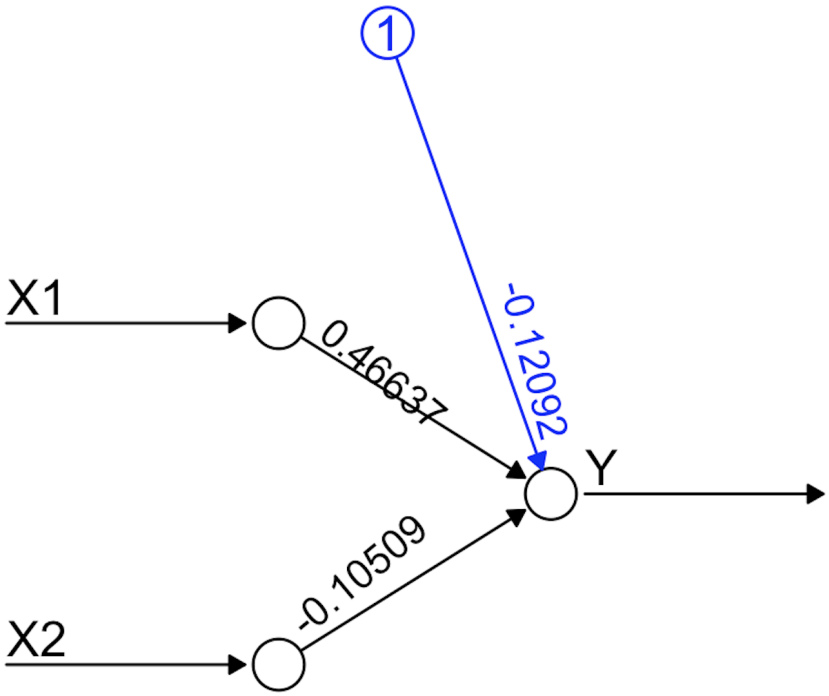
Sigmoid model: no hidden layer

Total parameter: 2 (weights) + 1 (biais) = 3



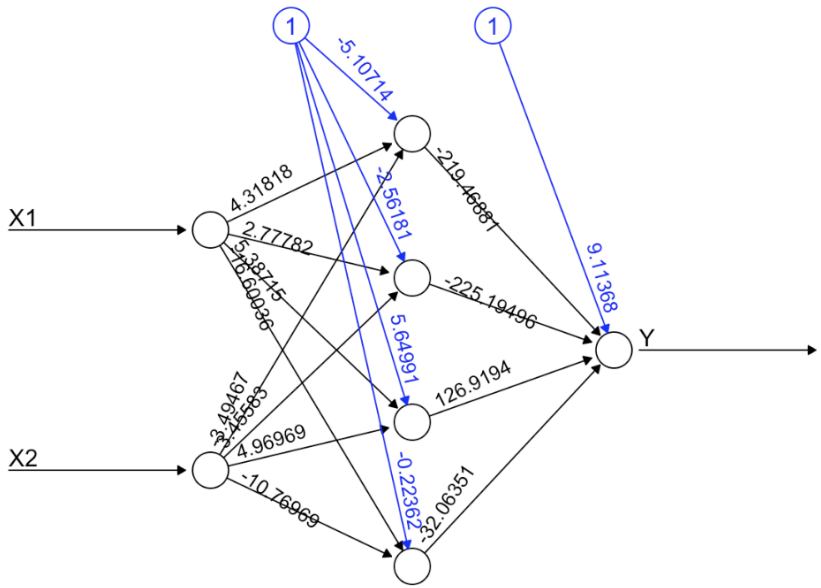
Sigmoid model: no hidden layer

Total parameter: 2 (weights) + 1 (biais) = 3



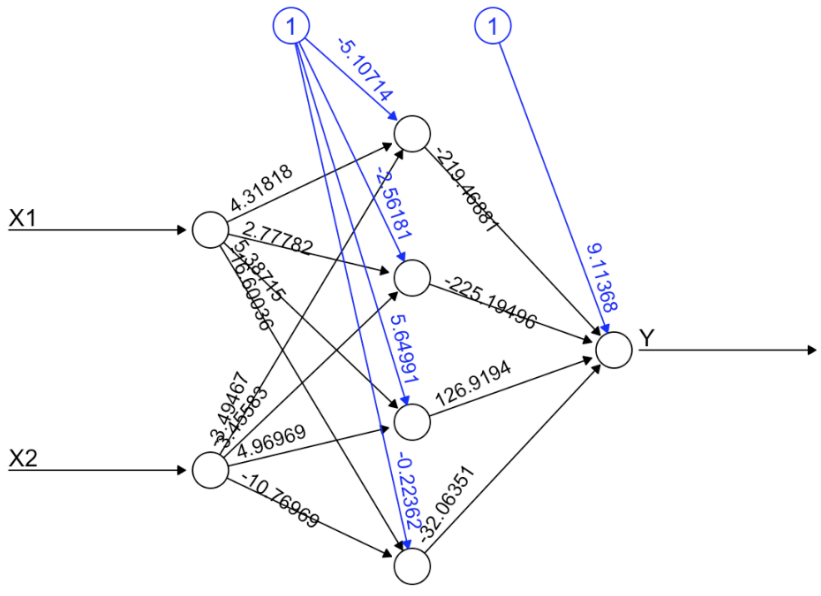
Expressivity: one hidden layer with 4 neurons

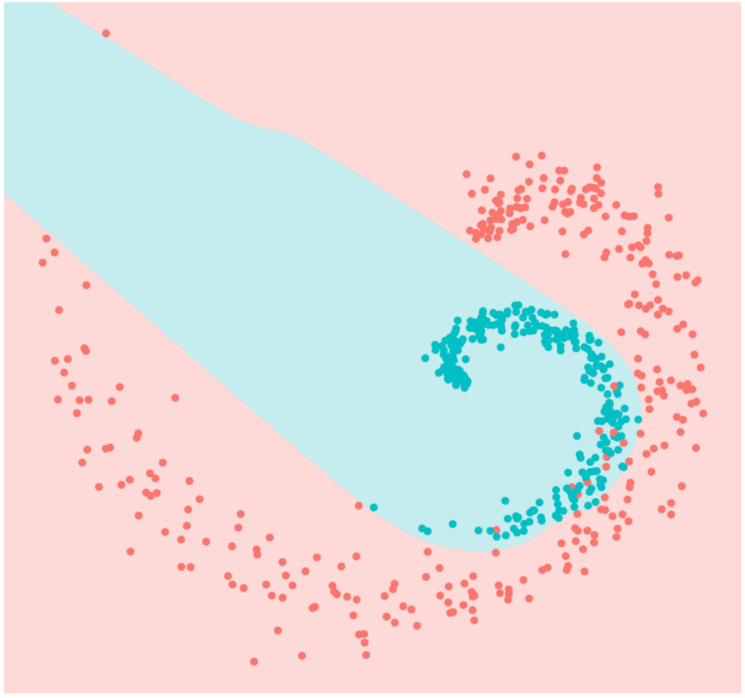
Total parameter: $\overbrace{4 \times 2 + 4}^{\text{hidden Layer 1}} + \overbrace{4 + 1}^{\text{output layer}} = 17$



Expressivity: one hidden layer with 4 neurons

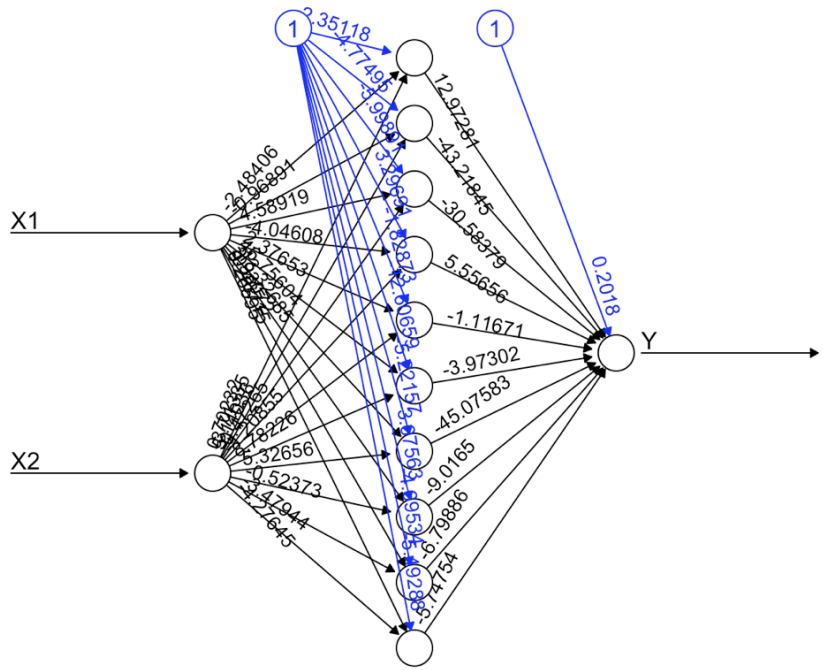
Total parameter: $\overbrace{4 \times 2 + 4}^{\text{hidden Layer 1}} + \overbrace{4 + 1}^{\text{output layer}} = 17$





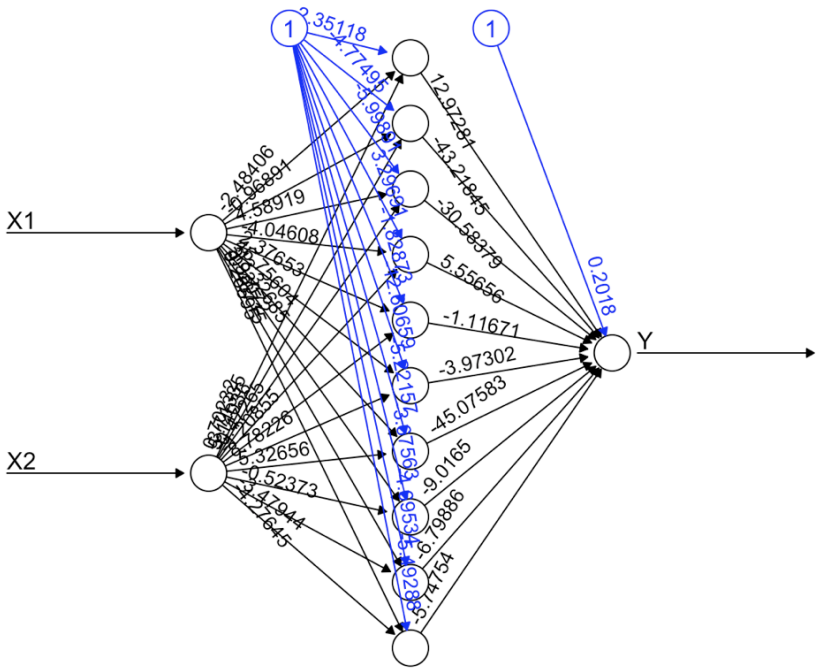
Expressivity: one hidden layer with 10 neurons

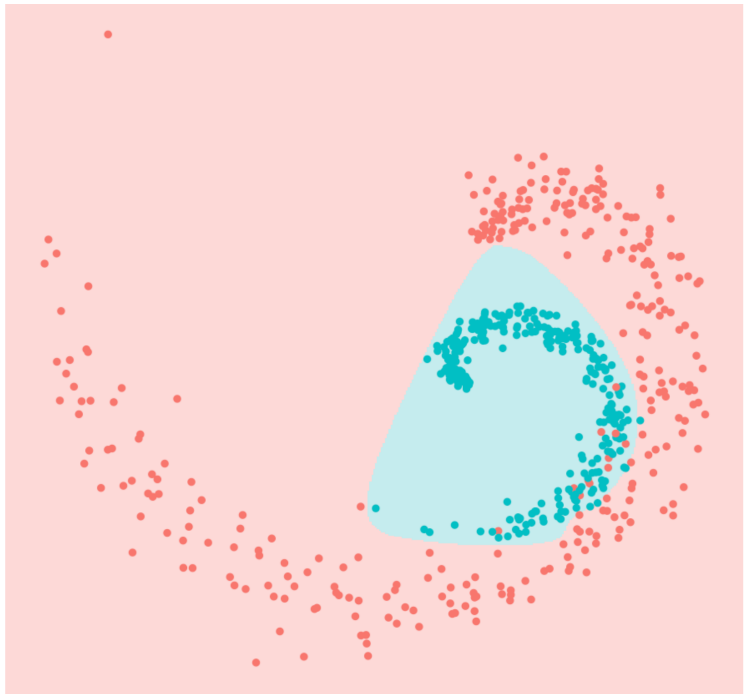
Total parameter: $\overbrace{10 \times 2 + 10}^{\text{hidden Layer 1}} + \overbrace{10 + 1}^{\text{output layer}} = 41$



Expressivity: one hidden layer with 10 neurons

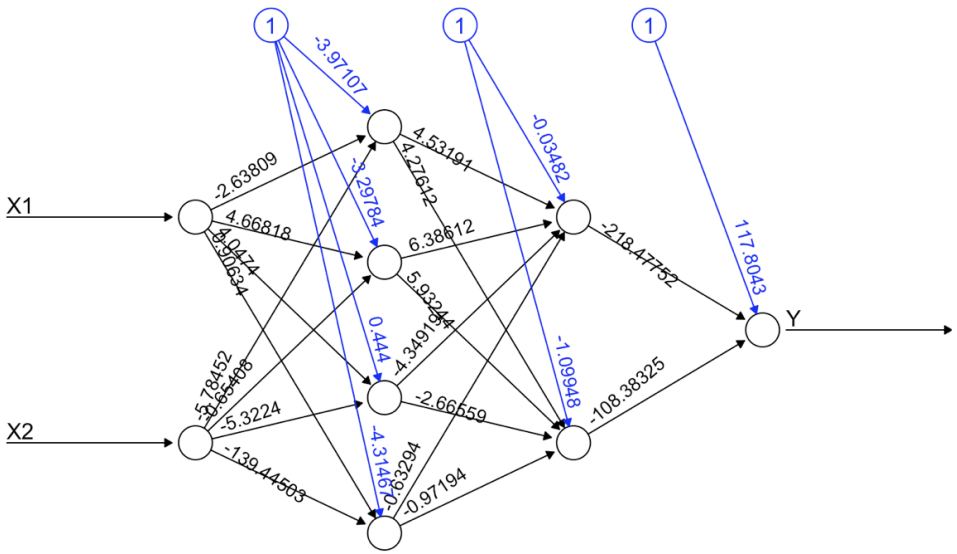
Total parameter: $\overbrace{10 \times 2 + 10}^{\text{hidden Layer 1}} + \overbrace{10 + 1}^{\text{output layer}} = 41$





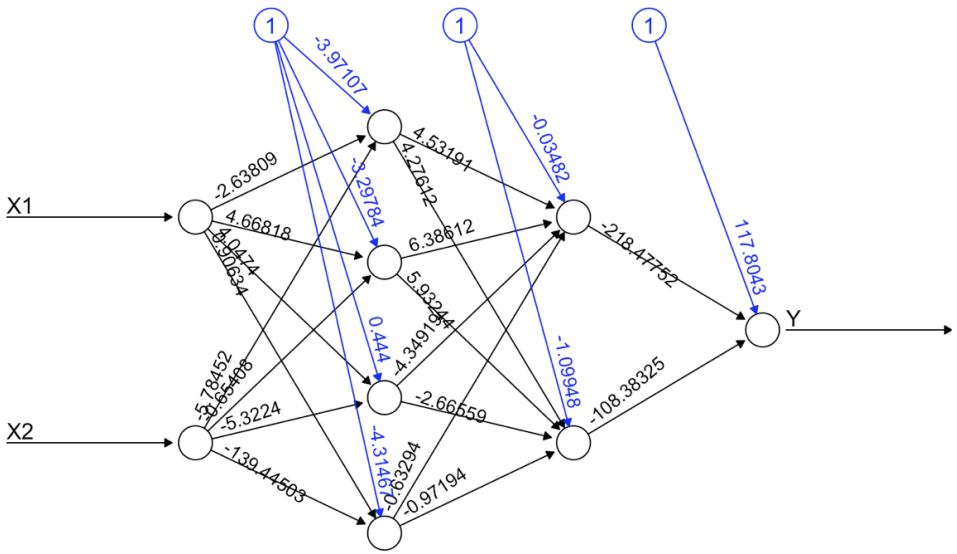
Why not going deep: 2 hidden layers

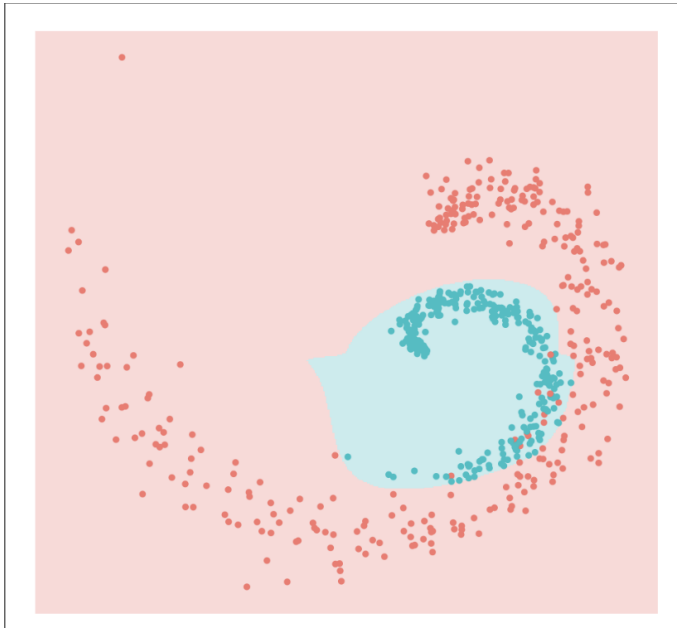
$$\text{Total parameter: } \overbrace{4\times 2+4}^{\text{hidden Layer 1}} + \overbrace{2\times 4+2}^{\text{hidden Layer 2}} + \overbrace{2+1}^{\text{output layer}} = 25$$



Why not going deep: 2 hidden layers

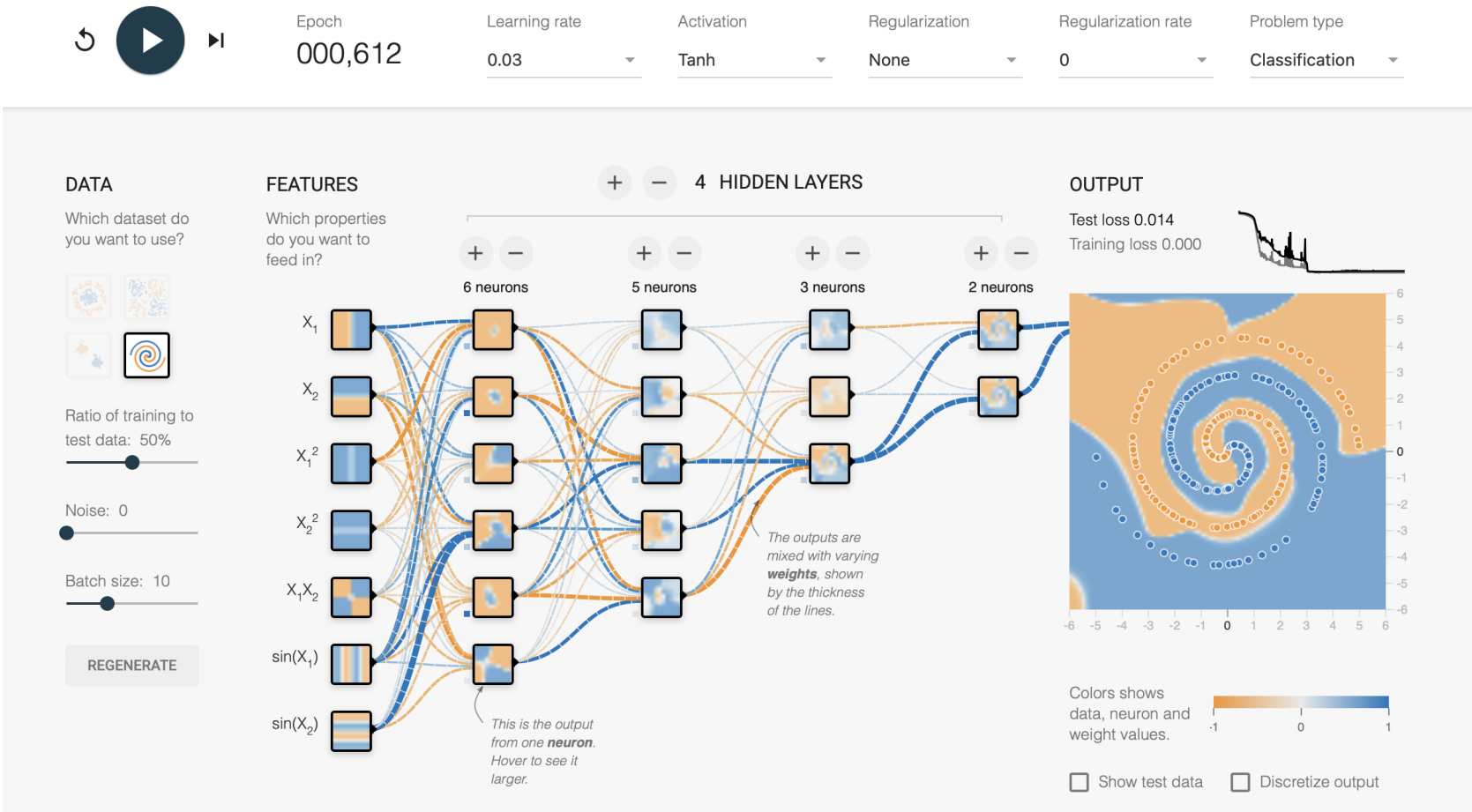
Total parameter: $\overbrace{4\times 2+4}^{\text{hidden Layer 1}} + \overbrace{2\times 4+2}^{\text{hidden Layer 2}} + \overbrace{2+1}^{\text{output layer}} = 25$





Short demo

https://playground.tensorflow.org



Why Go Deep?

- A single hidden layer can approximate any function
- But it may require a huge number of neurons

Why Go Deep?

- A single hidden layer can approximate any function
- But it may require a huge number of neurons
- Deep networks build functions **hierarchically**

$$\widehat{y} = f^{[L]}(f^{[L-1]}(f^{[L-2]}(\ldots(f^{[1]}(x))\ldots)))$$

- Each layer learns more abstract features

Shallow vs Deep Representations

Shallow network

- Many neurons
- Complex decision boundary learned at once

Deep network

- Fewer neurons per layer
- Decision boundary built step by step

Depth allows feature reuse and composition.

Expressivity of Deep Networks

Key Result

There exist functions that:

- require exponentially many neurons with one hidden layer
- can be represented with polynomial size deep networks

- Telgarsky (2016): *Benefits of depth in neural networks*
- Eldan & Shamir (2016): *The power of depth for feedforward networks*

Question 4

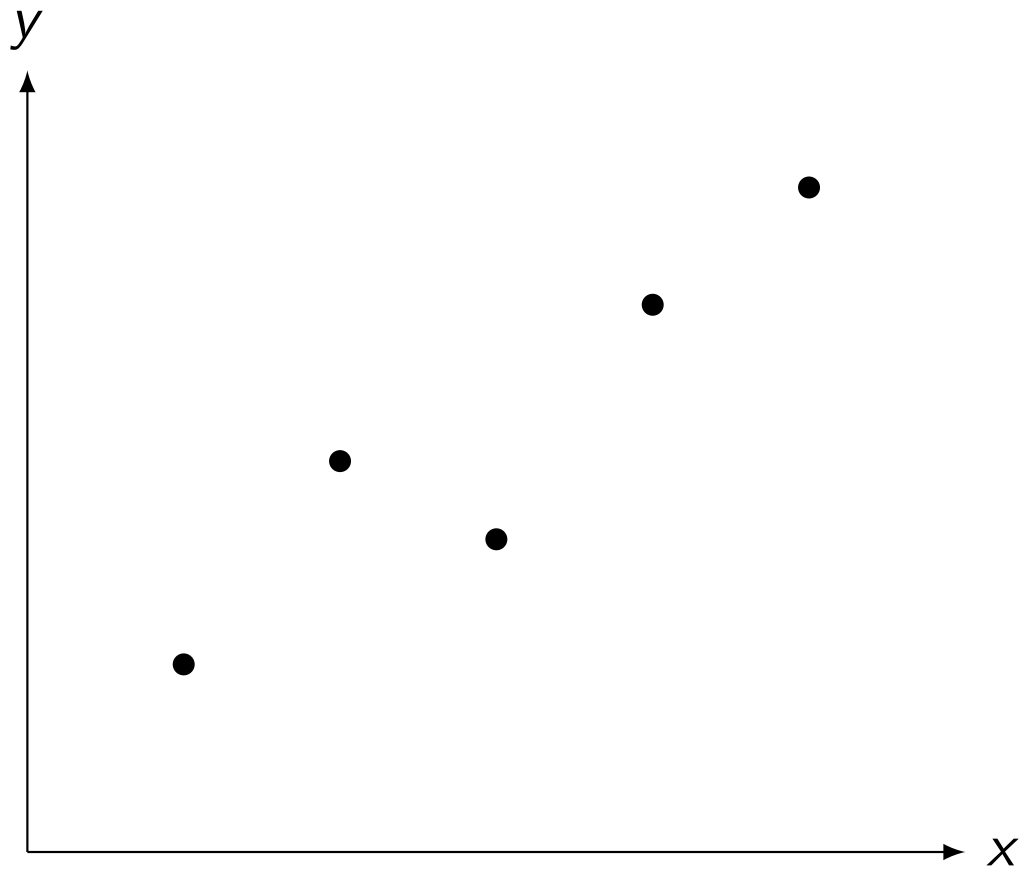
Does the fact that a neural network can approximate very complex functions guarantee that we can easily train it?

From Expressivity to Learning

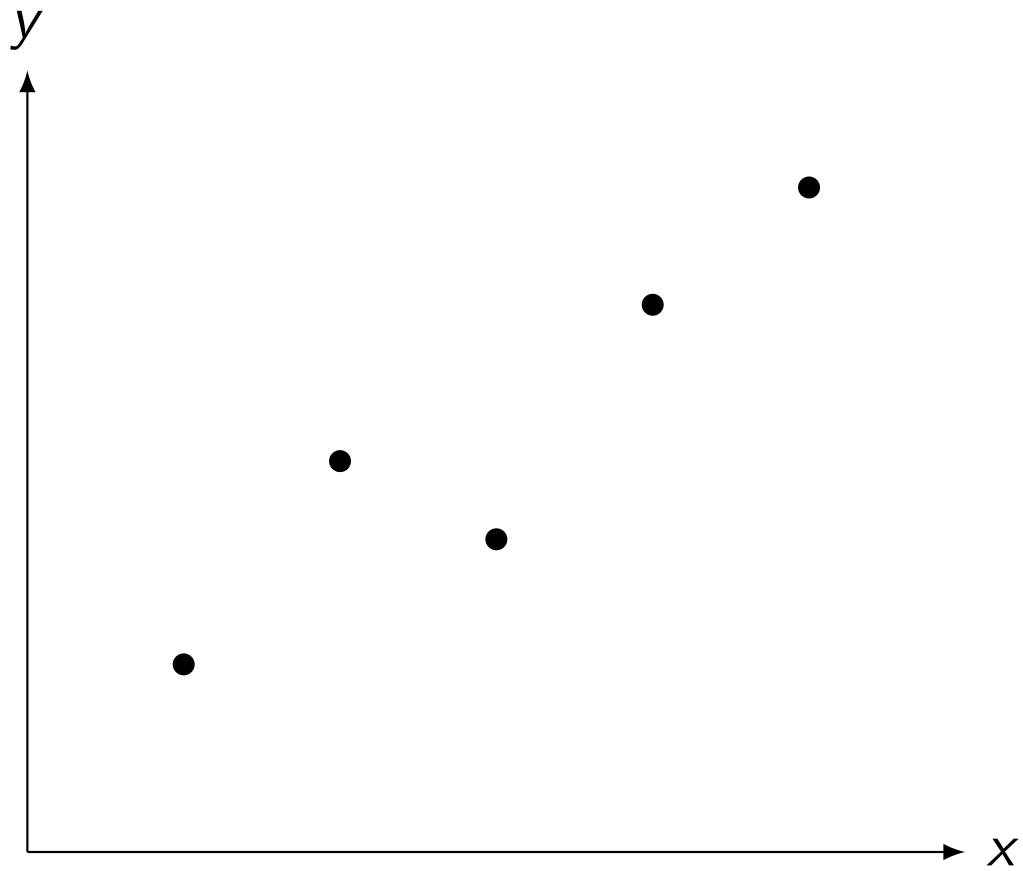
- Deep networks can represent very complex functions
- But representation alone is not enough
- We must **learn** the parameters from data

Learning = minimizing a loss (i.e. error) between predictions ($\widehat{y}$ and observed outcome y)

One-parameter model and no bias ($b = 0$)



One-parameter model and no bias ($b = 0$)

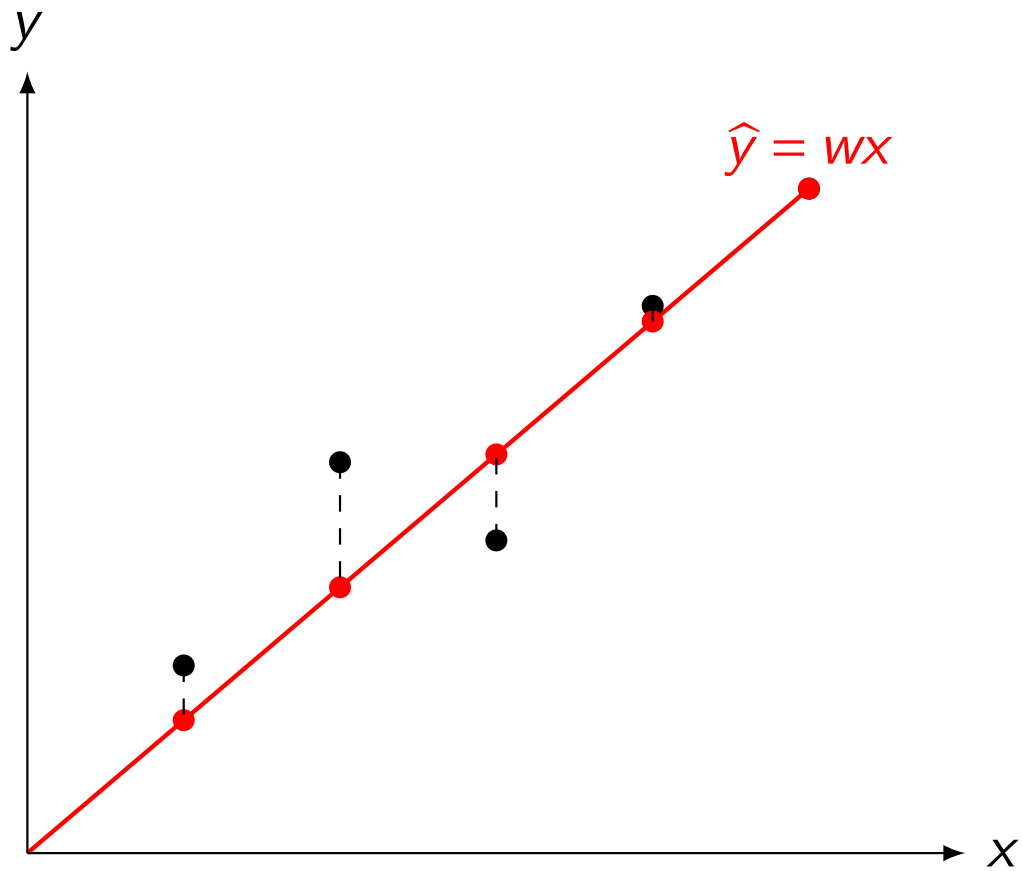


Simple linear model viewed as artificial neuron with a linear activation function ($\sigma(u) = u$).

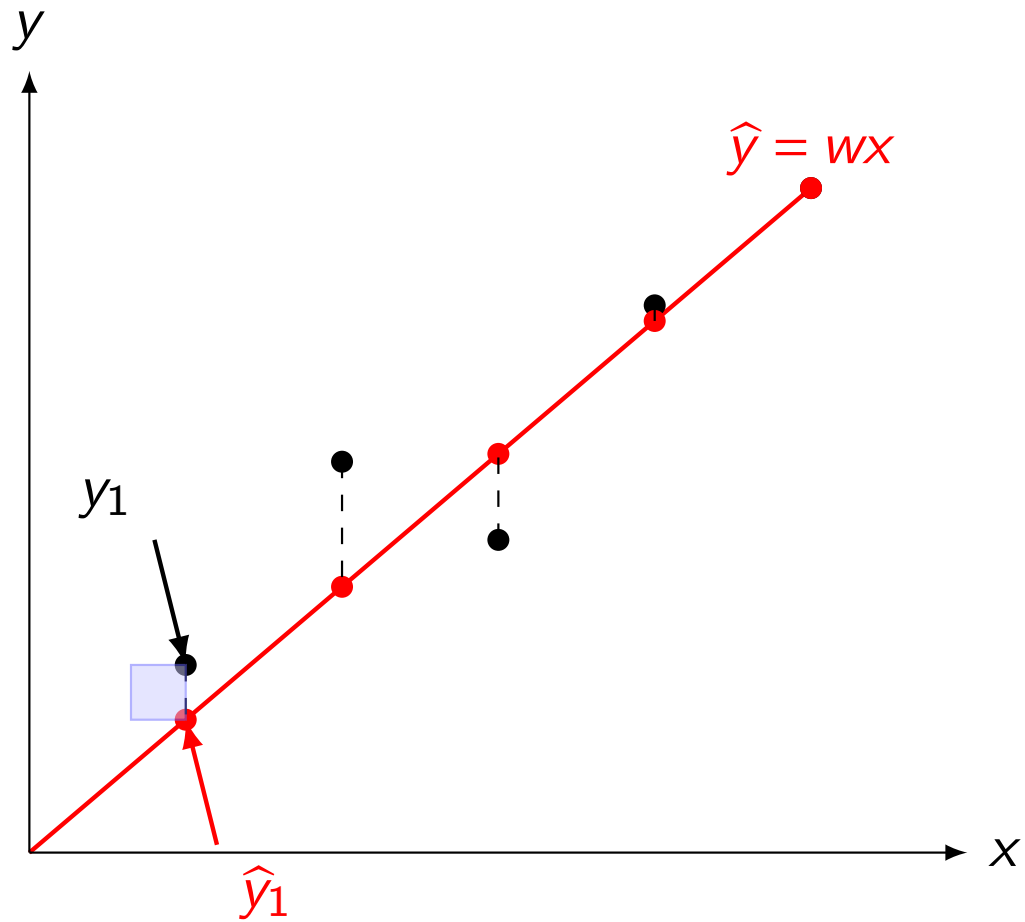
$$y = w \times x, \qquad \text{which best value for } w?$$

.

Find the best lines, i.e choose a suitable value for w

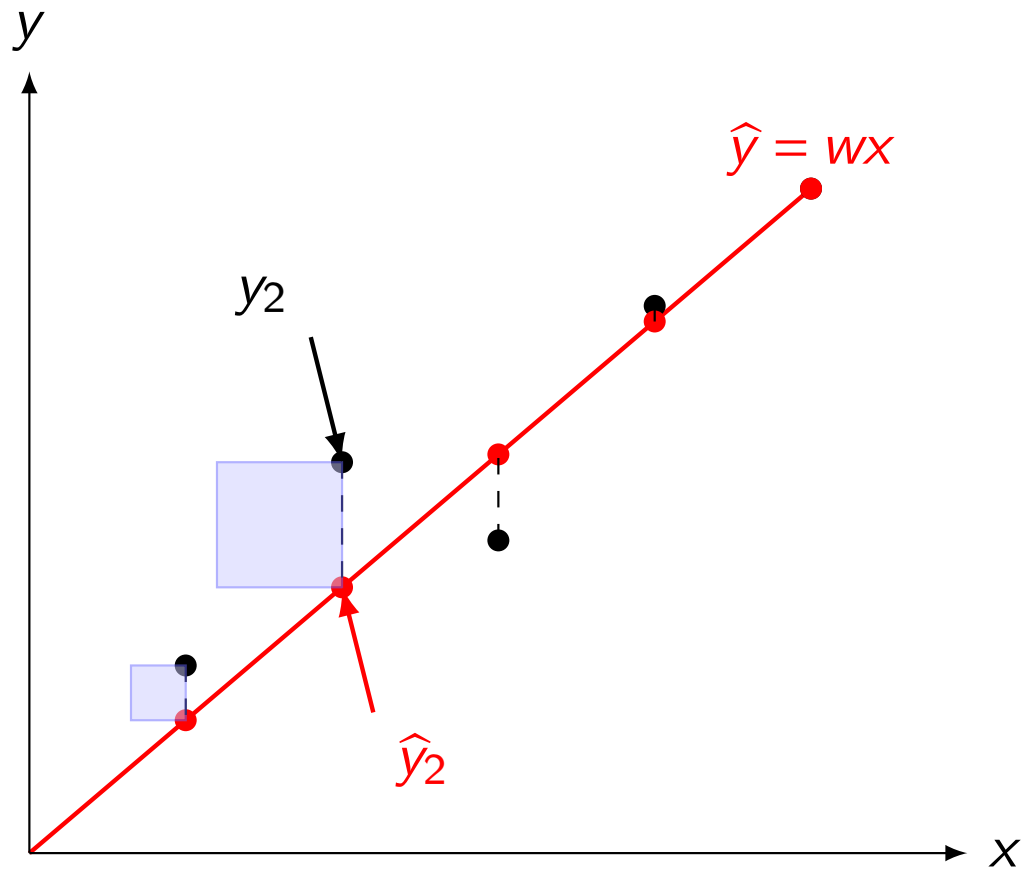


Find the best lines, i.e choose a suitable value for w



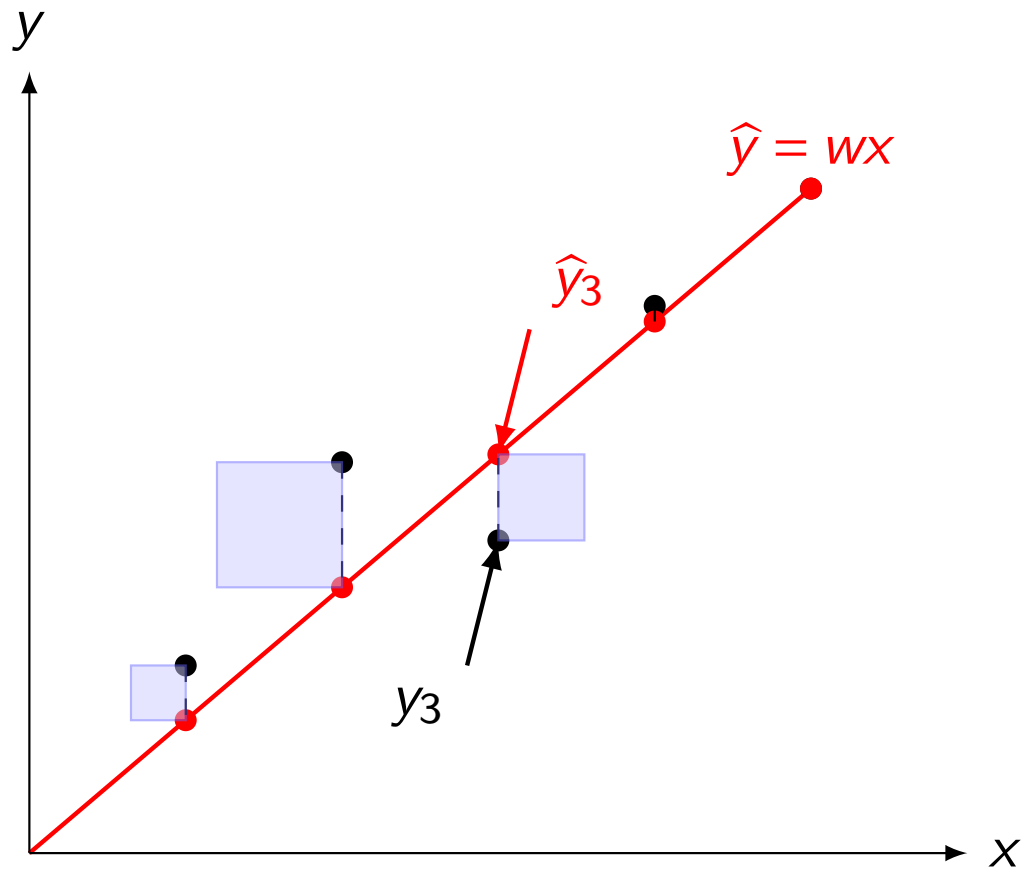
$$C_1(w) = (y_1 - \widehat{y}_1)^2$$

Find the best lines, i.e choose a suitable value for w



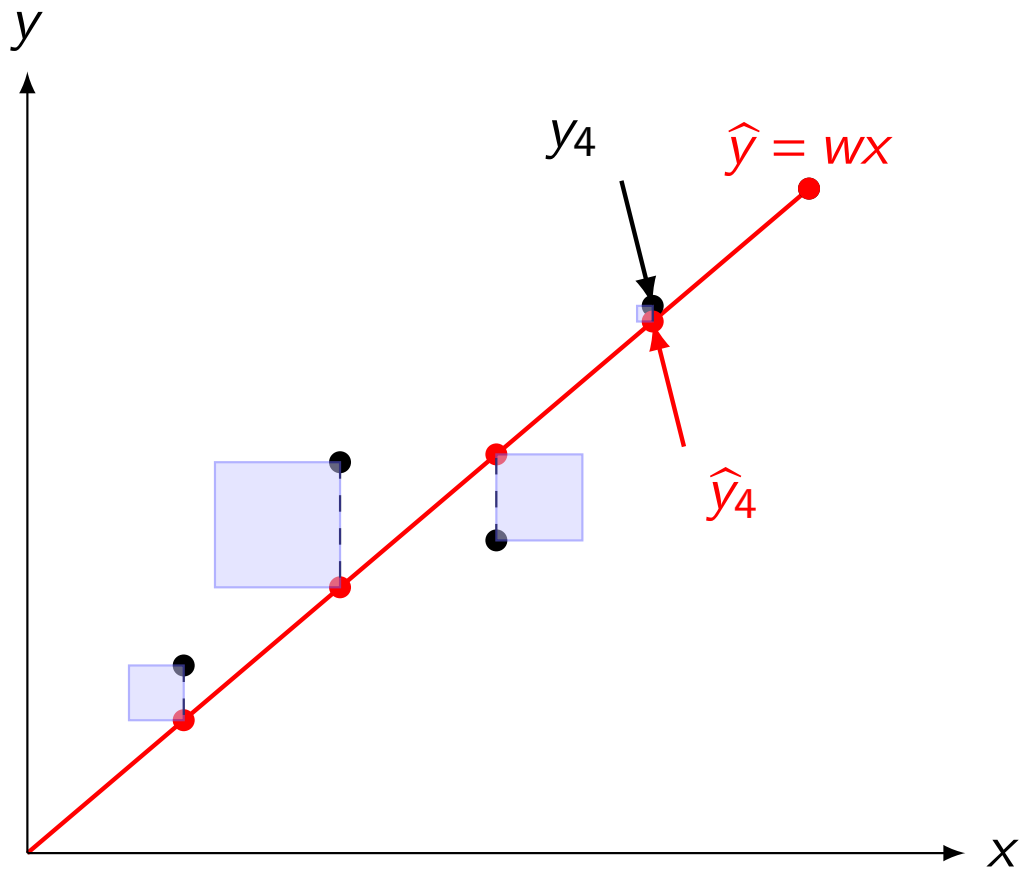
$$C_2(w) = (y_2 - \widehat{y}_2)^2$$

Find the best lines, i.e choose a suitable value for w



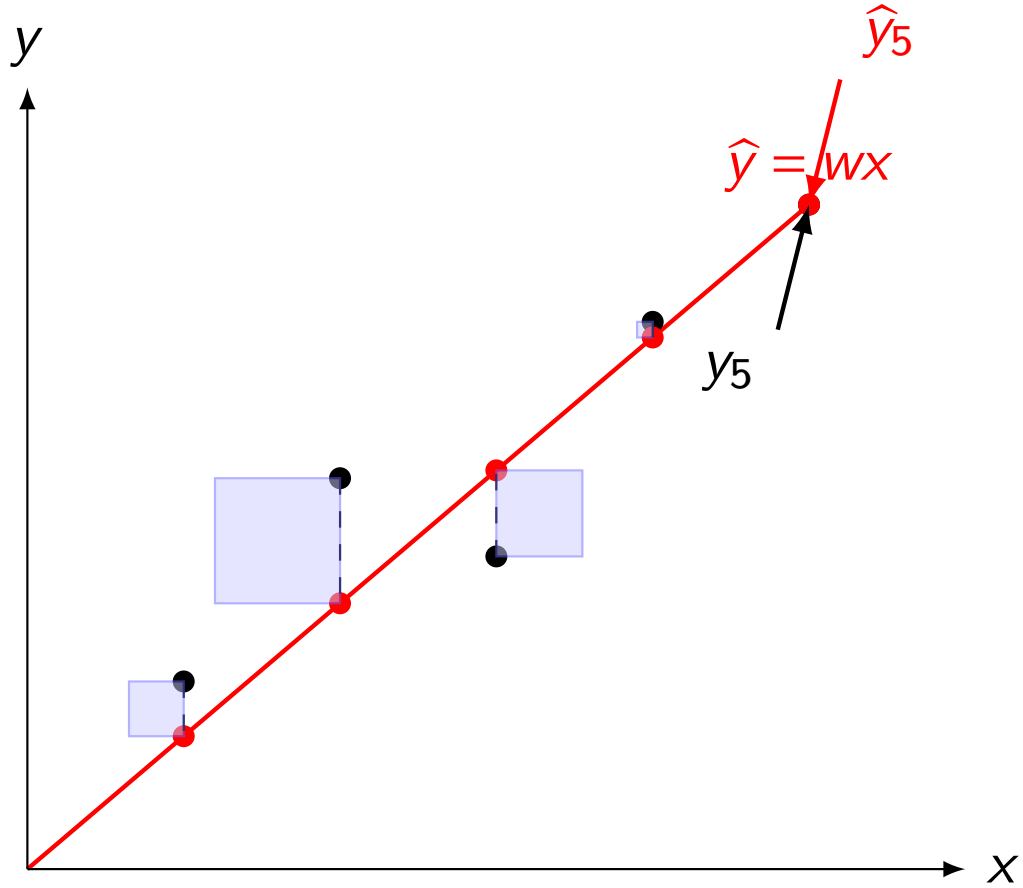
$$C_3(w) = (y_3 - \widehat{y}_3)^2$$

Find the best lines, i.e choose a suitable value for w



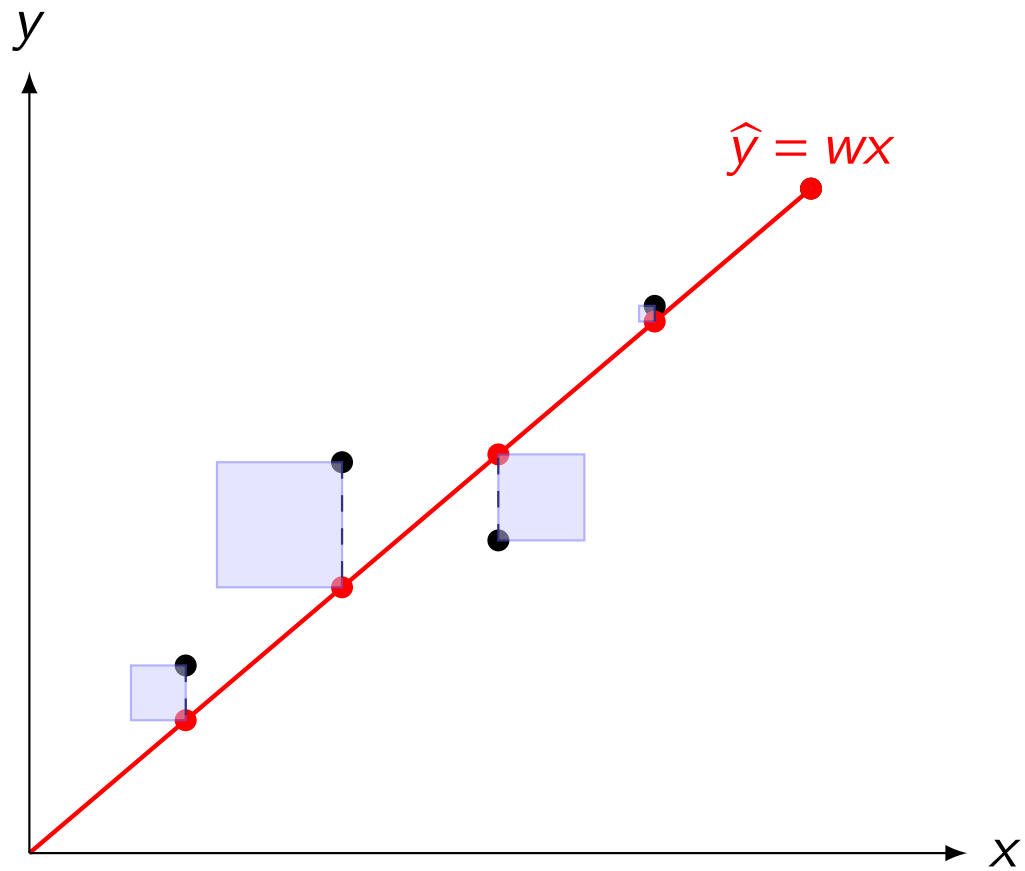
$$C_4(w) = (y_4 - \widehat{y}_4)^2$$

Find the best lines, i.e choose a suitable value for w



$$C_5(w) = (y_5 - \widehat{y}_5)^2$$

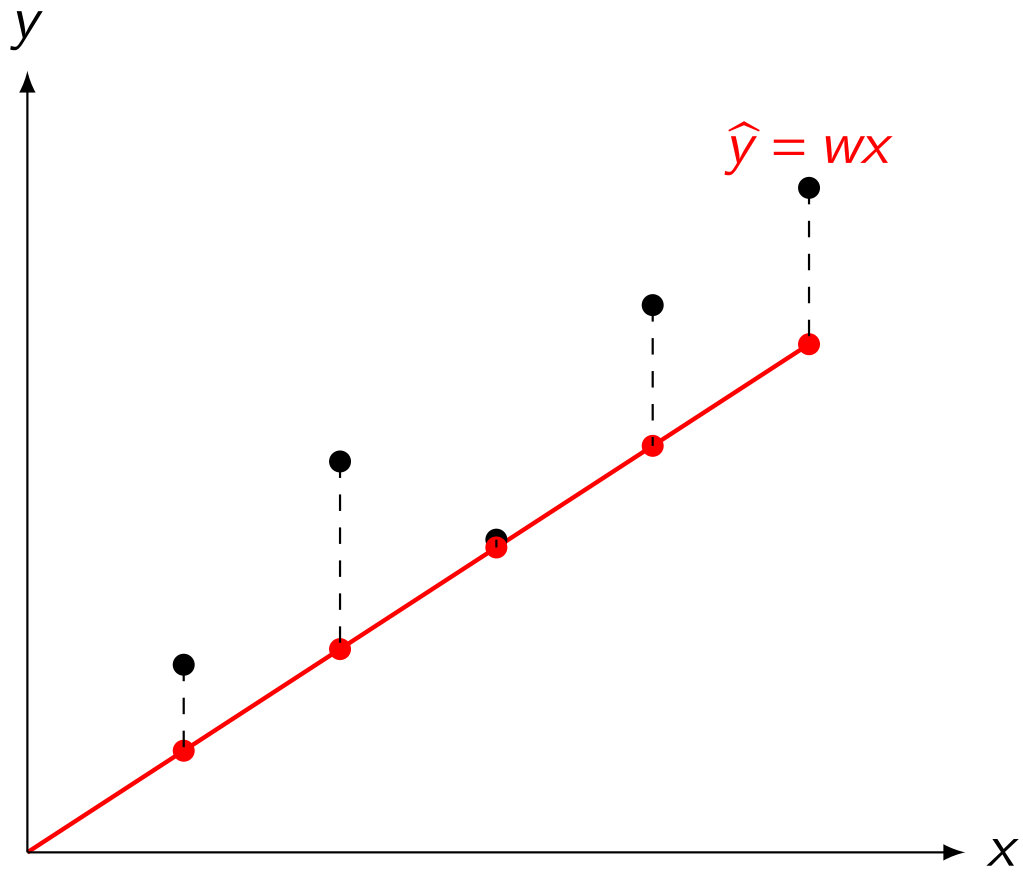
Find the best lines, i.e choose a suitable value for w



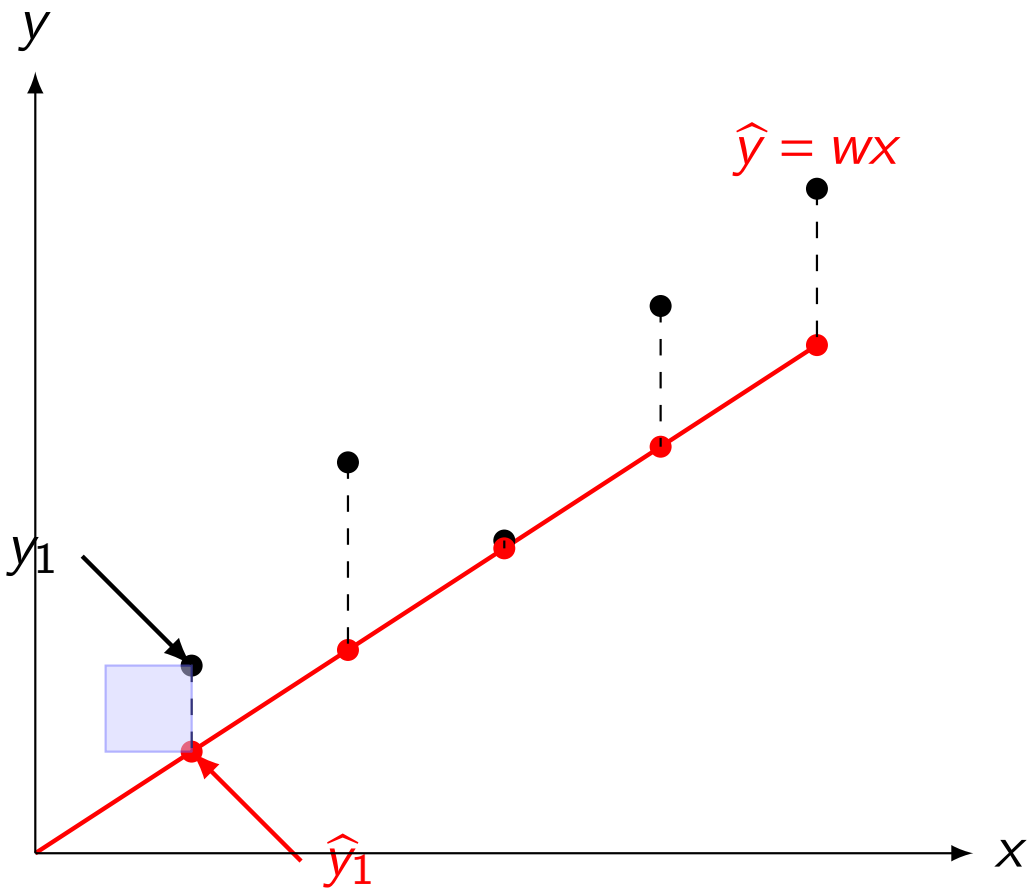
Here $w = 0.85$ and total loss is:

$$C(w) = \sum_{i=1}^{5} C_i(w) = 5.4$$

Another try with $w = 0.65$

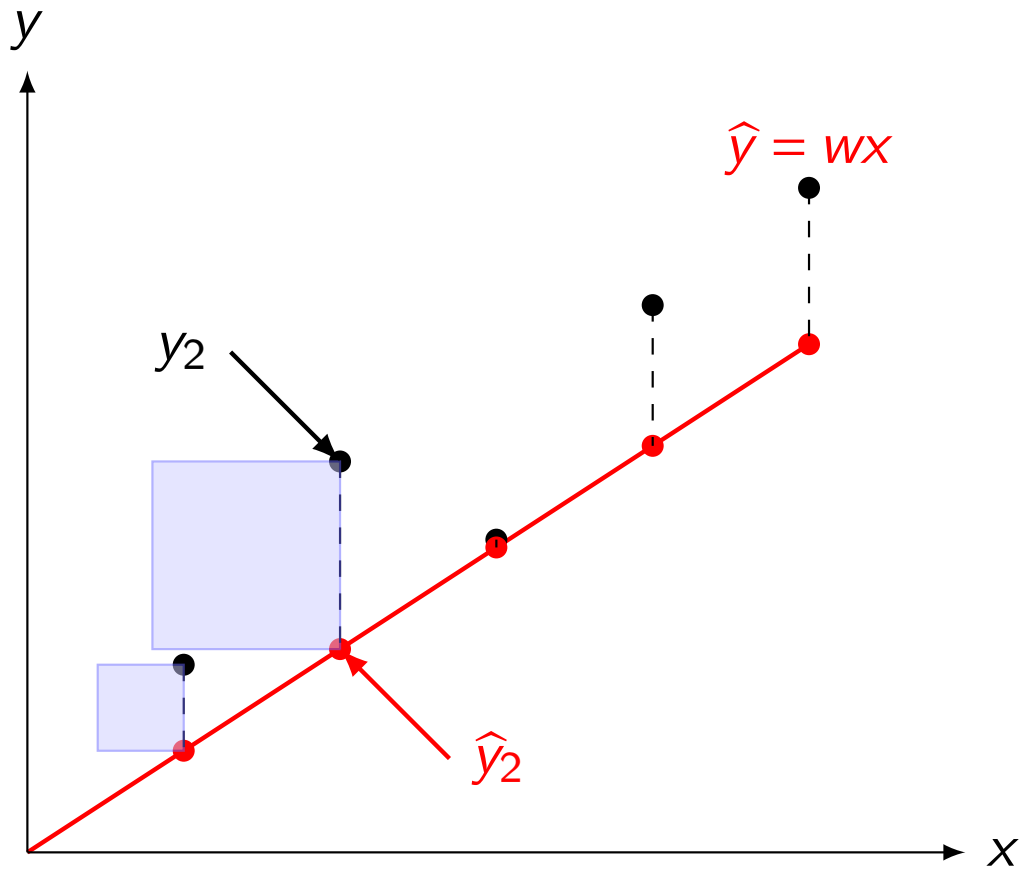


Another try with $w = 0.65$



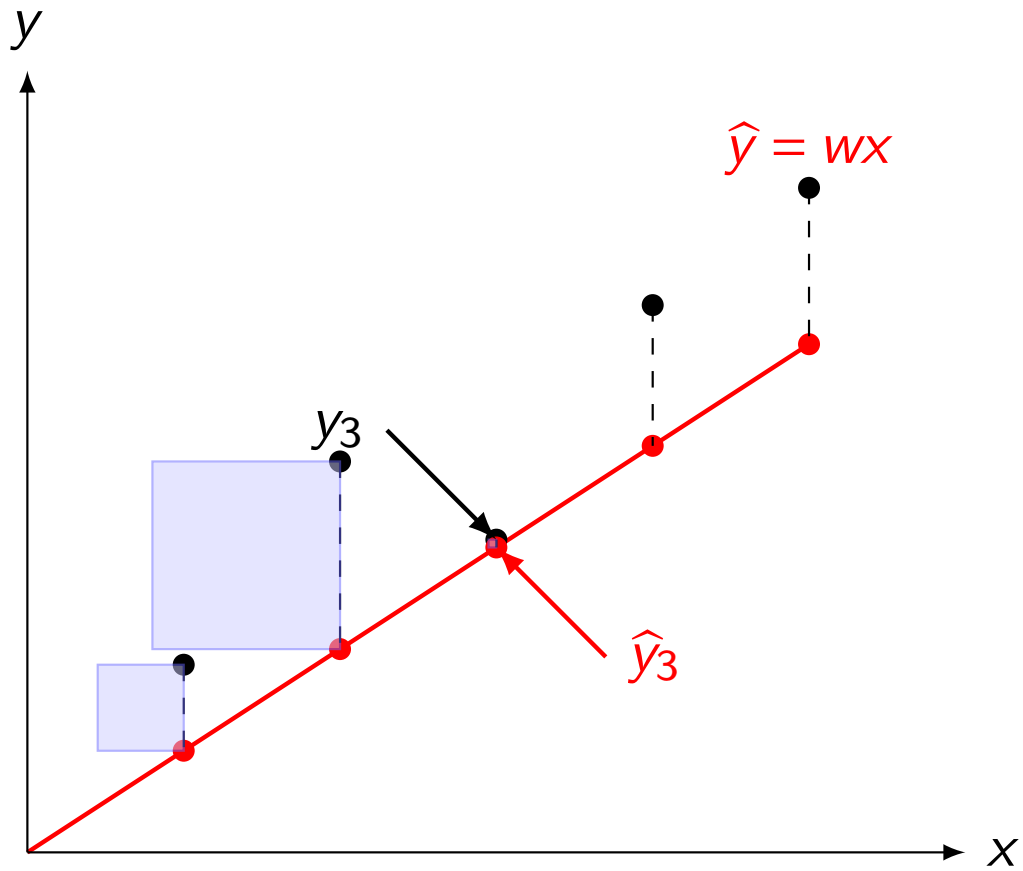
$$C_1(w) = (y_1 - \widehat{y}_1)^2$$

Another try with $w = 0.65$



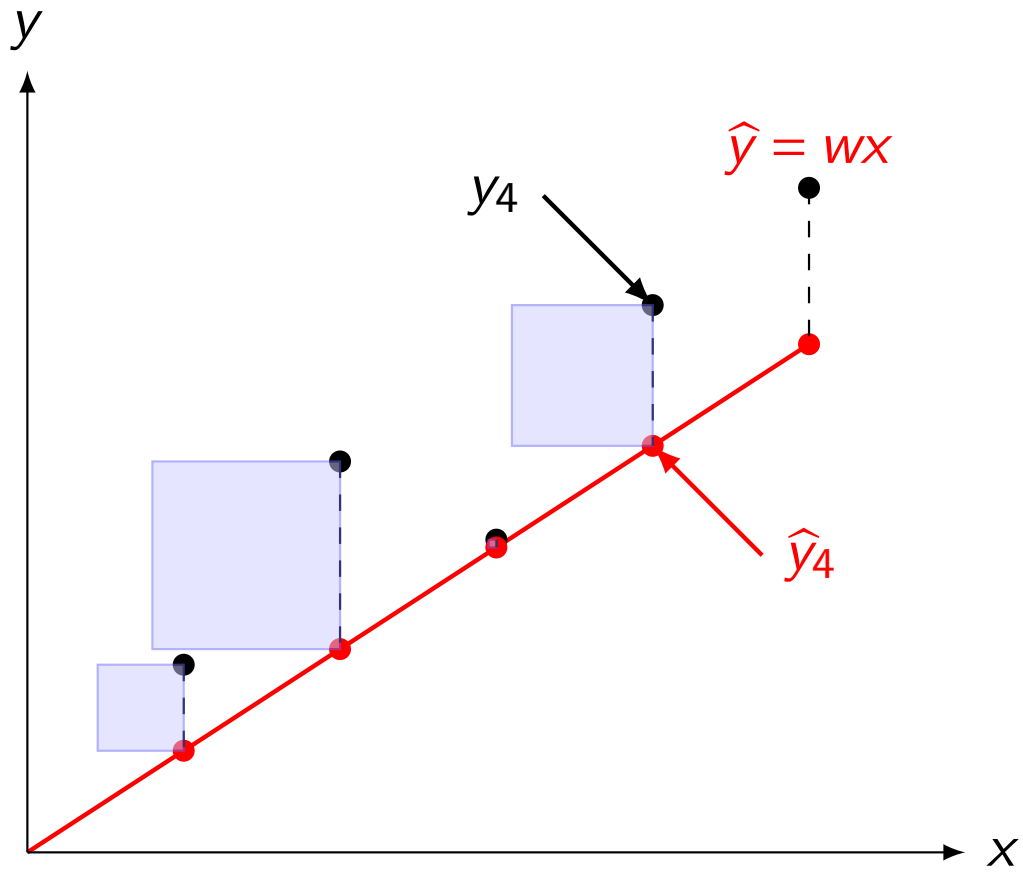
$$C_2(w) = (y_2 - \widehat{y}_2)^2$$

Another try with $w = 0.65$



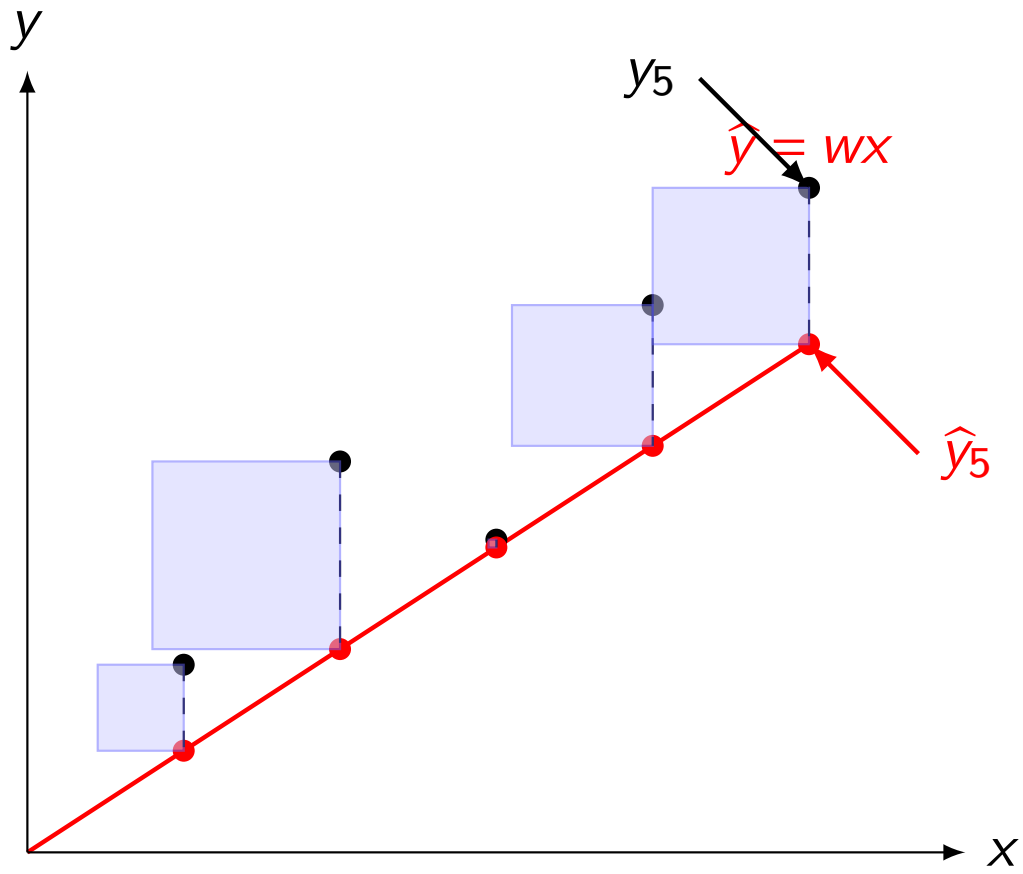
$$C_3(w) = (y_3 - \widehat{y}_3)^2$$

Another try with $w = 0.65$



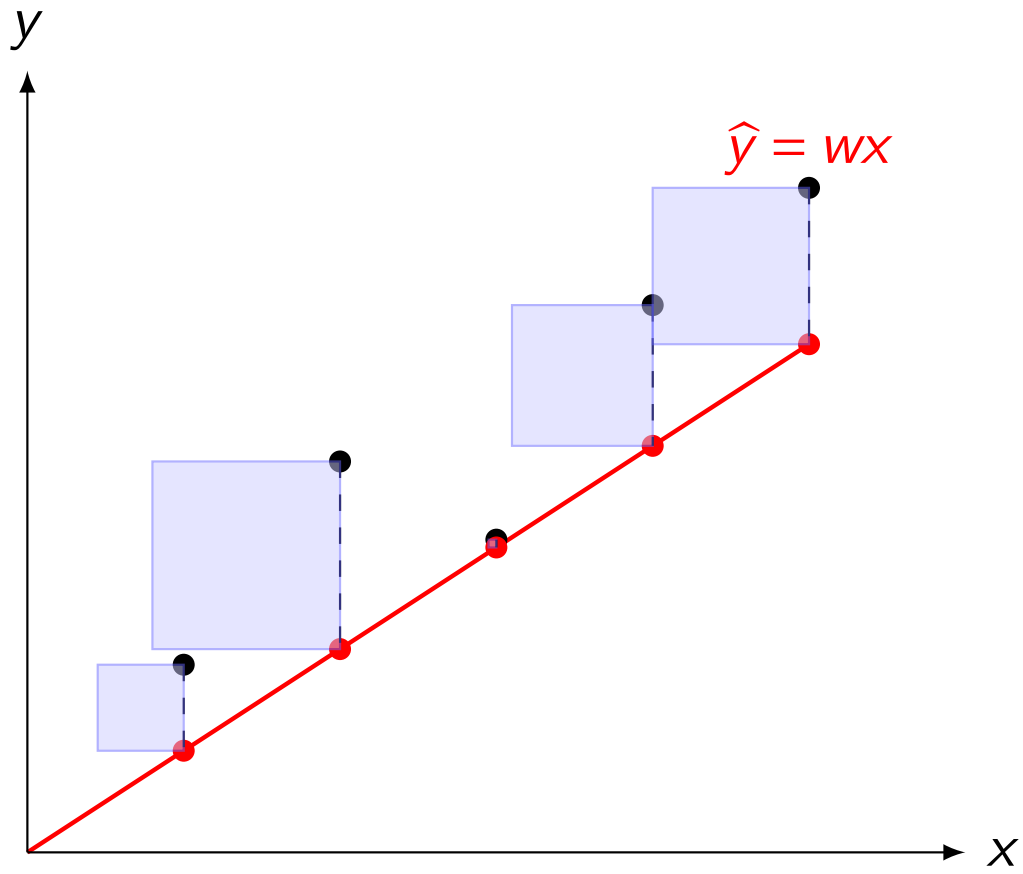
$$C_4(w) = (y_4 - \widehat{y}_4)^2$$

Another try with $w = 0.65$



$$C_5(w) = (y_5 - \widehat{y}_5)^2$$

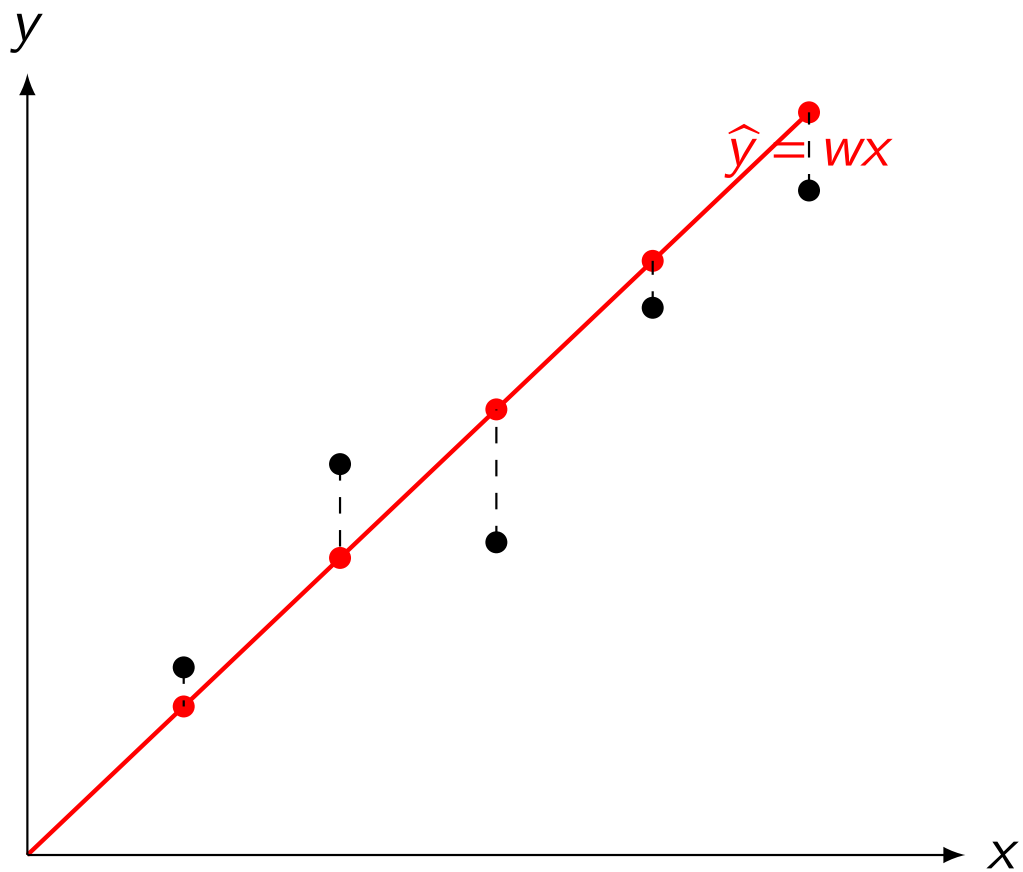
Another try with $w = 0.65$



$$C(w) = \sum_{i=1}^{5} C_i(w) = 12.3$$

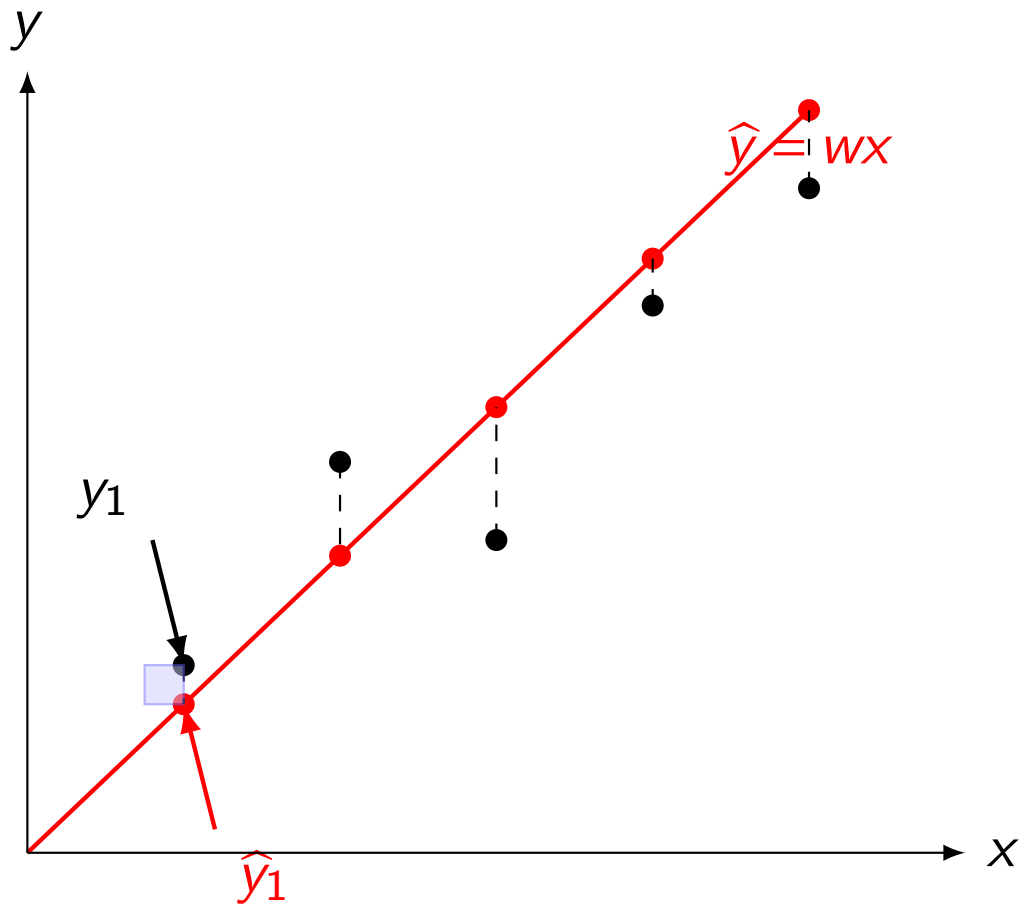
Another try with $w = 0.95$

Here a linear activation function $(\sigma(u) = u)$



Another try with $w = 0.95$

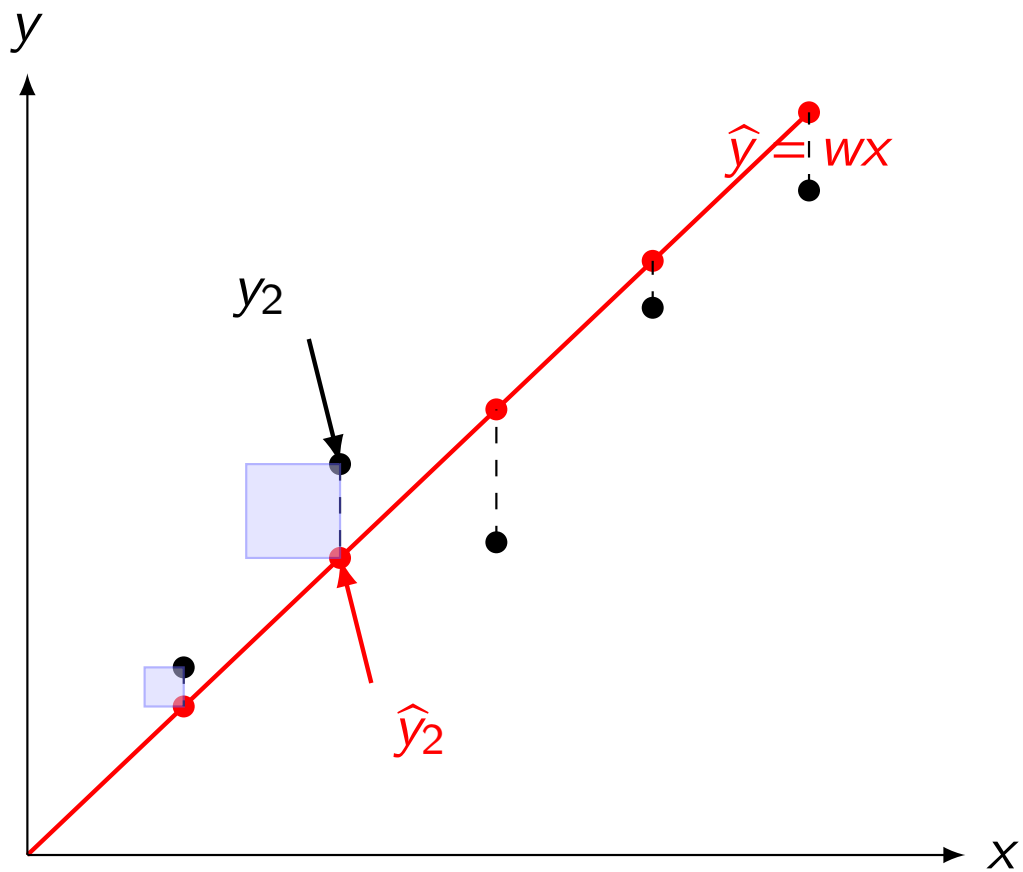
Here a linear activation function $(\sigma(u) = u)$



$$C_1(w) = (y_1 - \widehat{y}_1)^2$$

Another try with $w = 0.95$

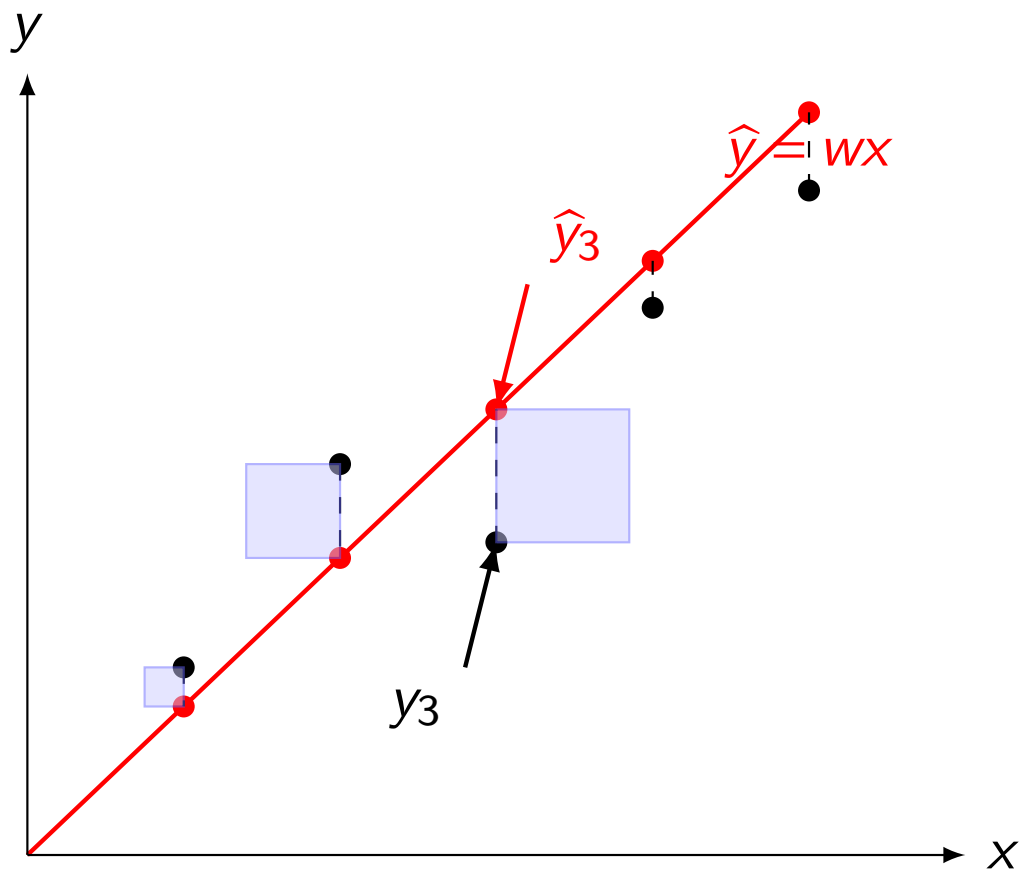
Here a linear activation function $(\sigma(u) = u)$



$$C_2(w) = (y_2 - \widehat{y}_2)^2$$

Another try with $w = 0.95$

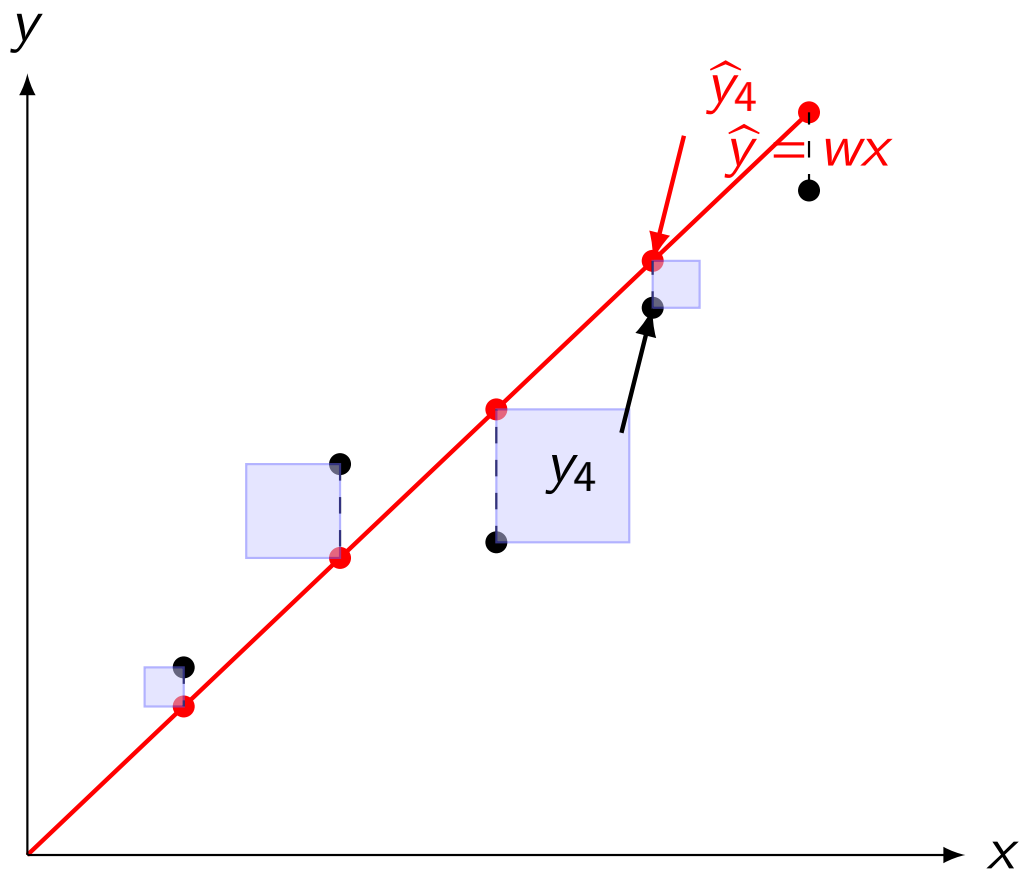
Here a linear activation function $(\sigma(u) = u)$



$$C_3(w) = (y_3 - \widehat{y}_3)^2$$

Another try with $w = 0.95$

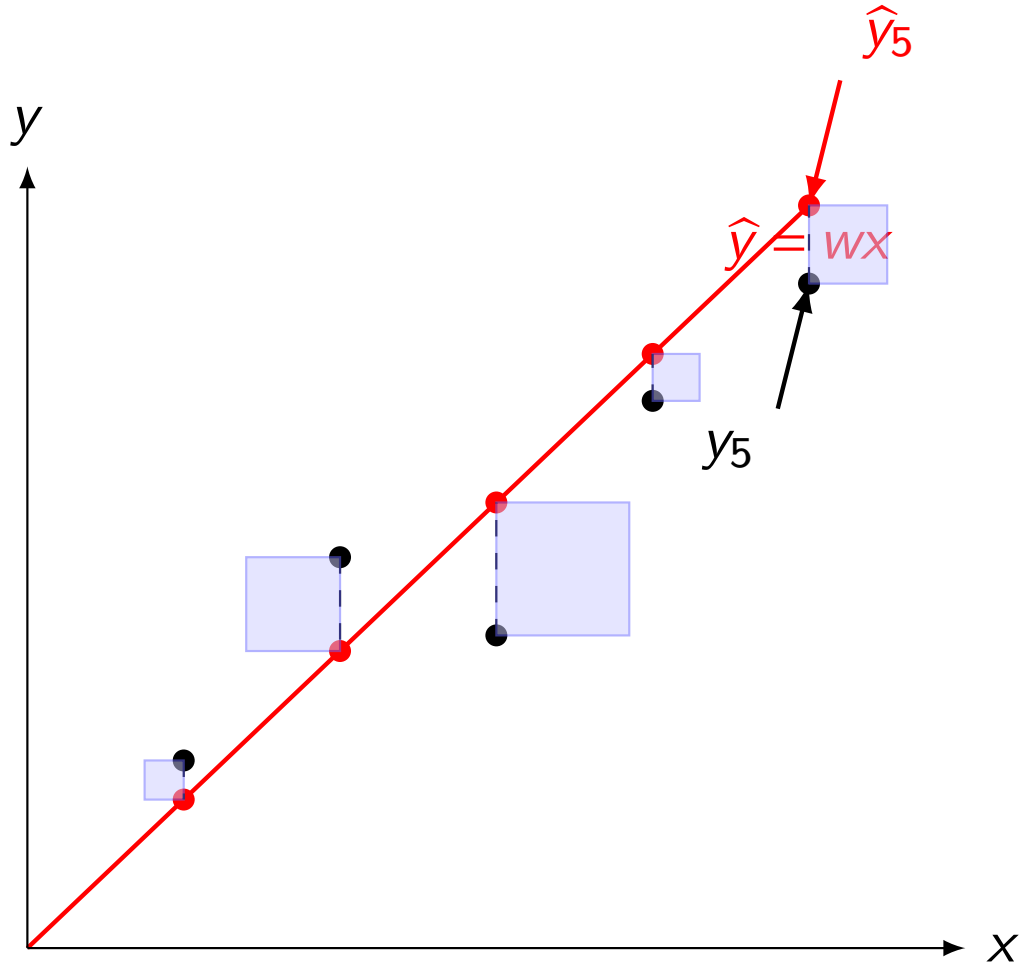
Here a linear activation function $(\sigma(u) = u)$



$$C_4(w) = (y_4 - \widehat{y}_4)^2$$

Another try with $w = 0.95$

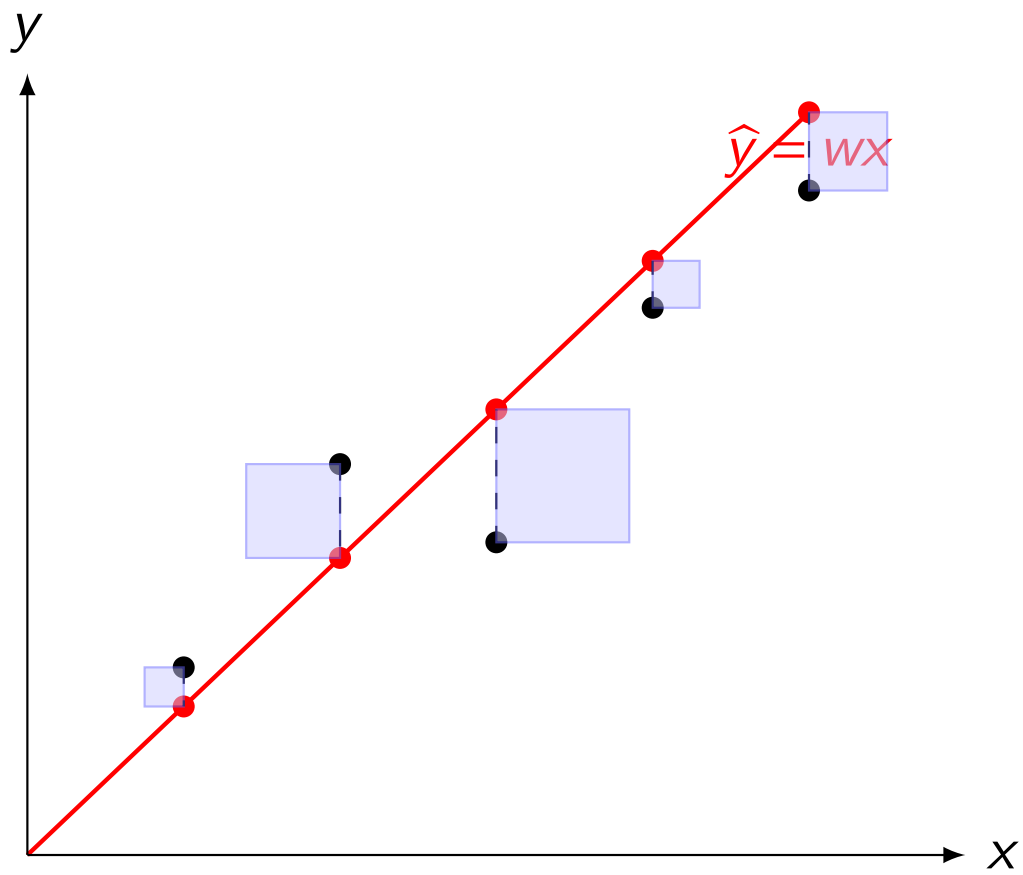
Here a linear activation function $(\sigma(u) = u)$



$$C_5(w) = (y_5 - \widehat{y}_5)^2$$

Another try with $w = 0.95$

Here a linear activation function $(\sigma(u) = u)$



$$C(w) = \sum_{i=1}^{5} C_i(w) = 4.9$$

We want to minize the loss function

- DL exploits iterative optimisation method to train (find) suitable parameter.
- Gradient Descent (GD) algorithm and variations are the key tools.
- Gradient means derivative (slope of the function) for one parameter model.
- An estimate of the slope of the function (rise divided by run) at a specific point θ:

$$\frac{\text{Rise}}{\text{Run}} = \frac{C(\theta+\Delta) - C(\theta)}{(\theta+\Delta) - \Delta}$$
$$\approx C'(\theta) \quad \text{when } \Delta \text{ positive small number}$$

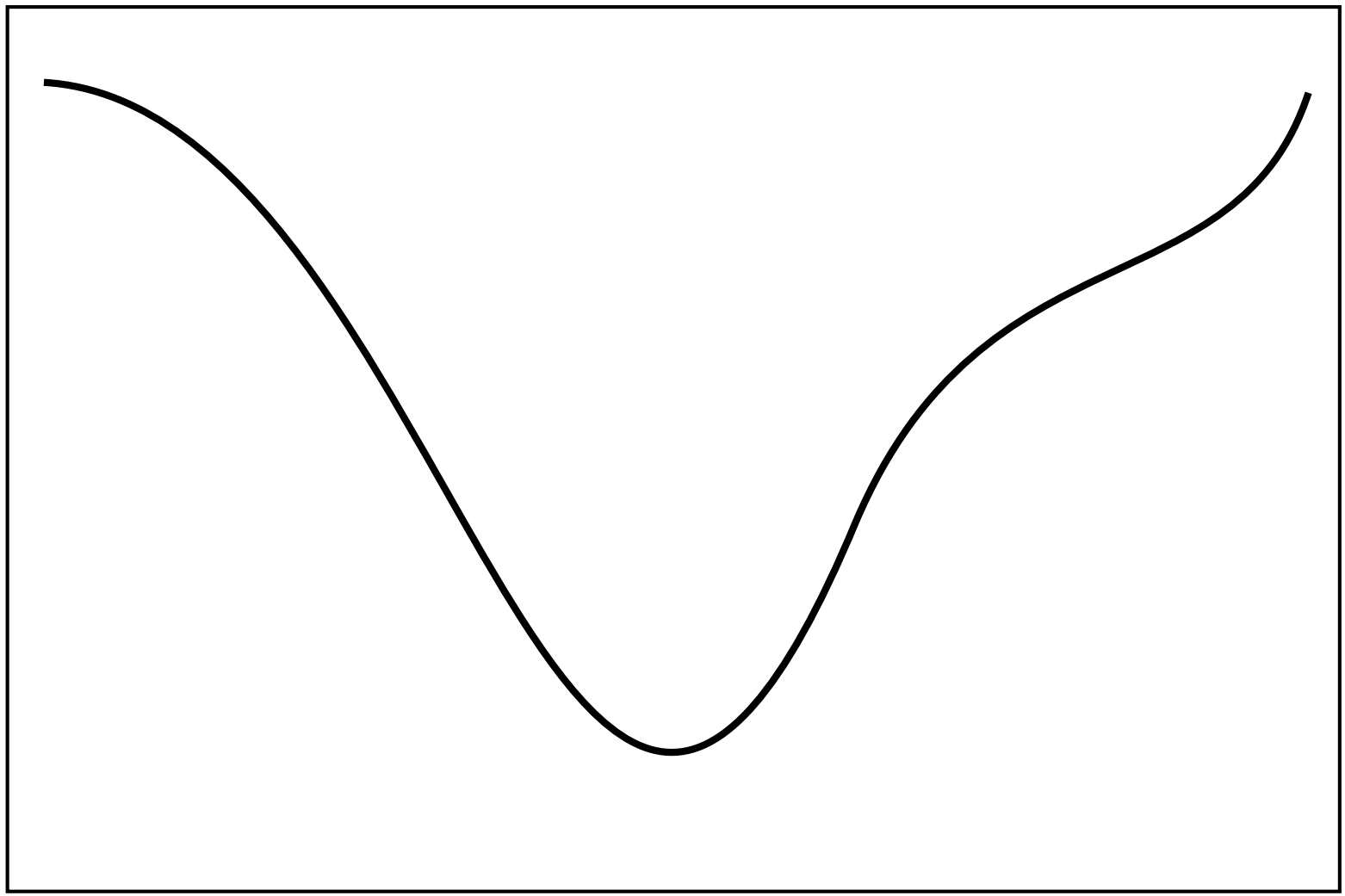
We want to minize the loss function

- DL exploits iterative optimisation method to train (find) suitable parameter.
- Gradient Descent (GD) algorithm and variations are the key tools.
- Gradient means derivative (slope of the function) for one parameter model.
- An estimate of the slope of the function (rise divided by run) at a specific point θ:

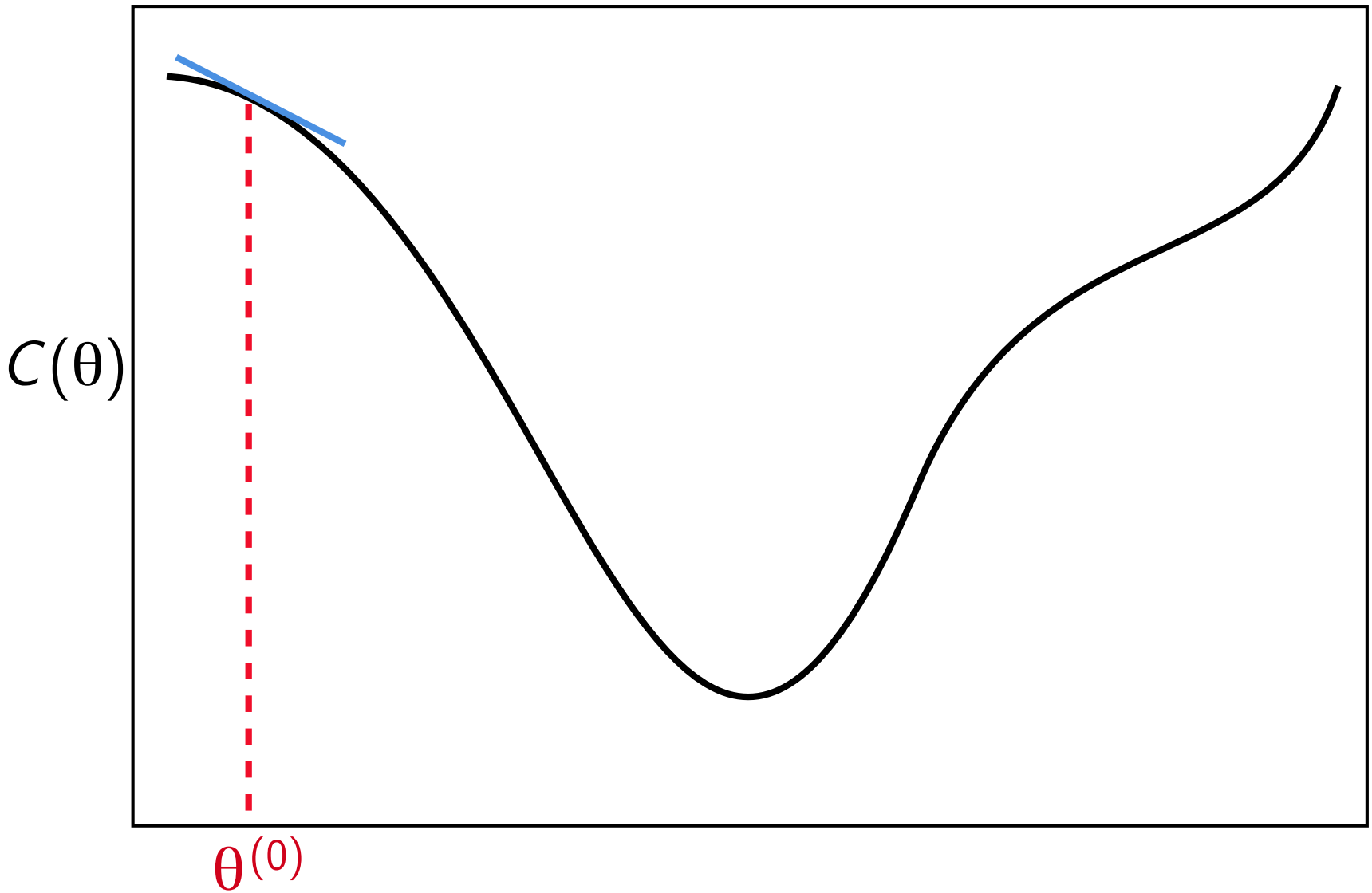
$$\frac{\text{Rise}}{\text{Run}} = \frac{C(\theta+\Delta) - C(\theta)}{(\theta+\Delta) - \Delta}$$
$$\approx C'(\theta) \quad \text{when } \Delta \text{ positive small number}$$

- Gradient is related to vector parameter: noted $\nabla C(\theta)$ $(\theta \in \mathbb{R}^p)$

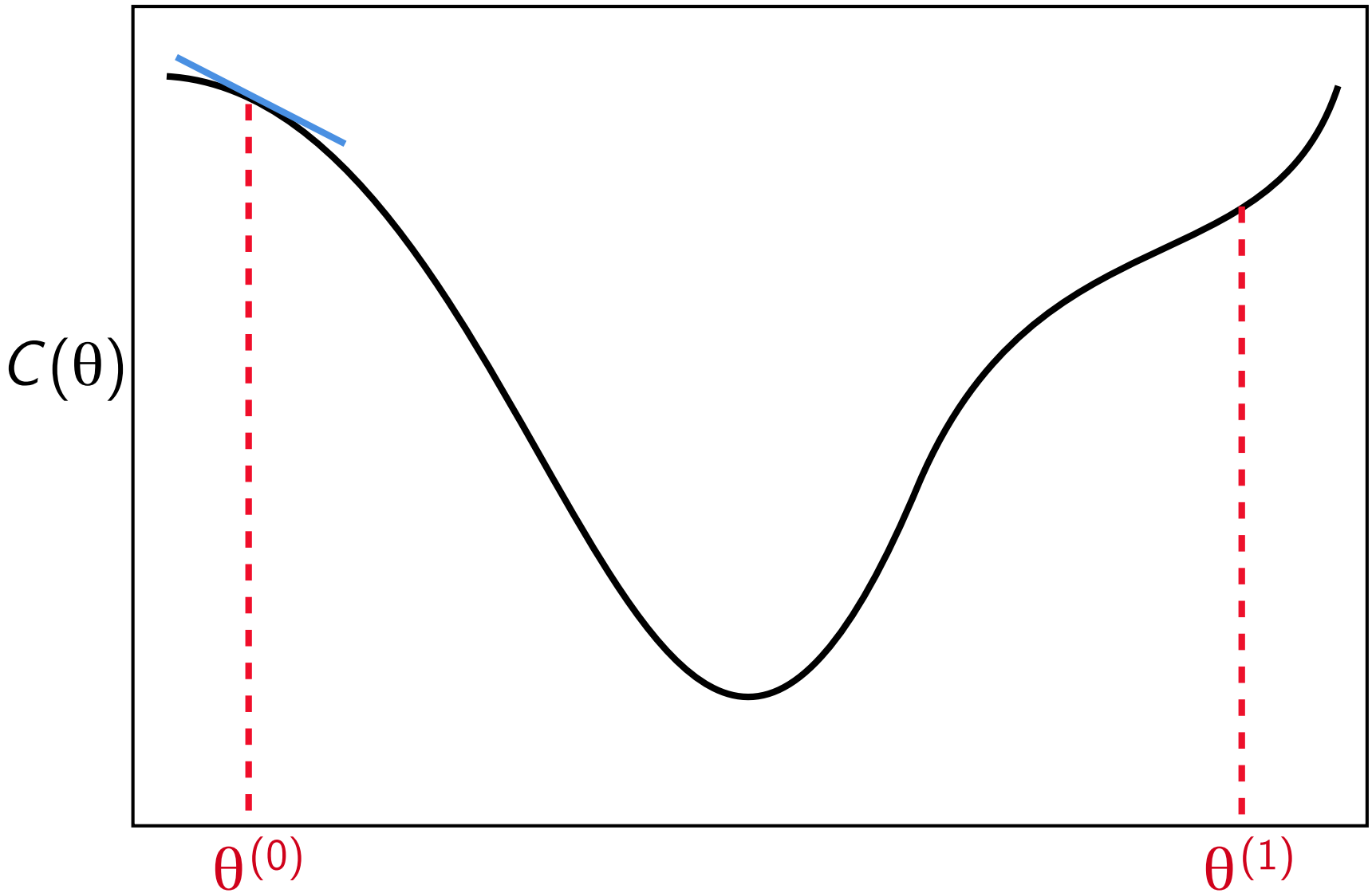
Gradient Descent in action: $\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla C(\theta^{(t)})$



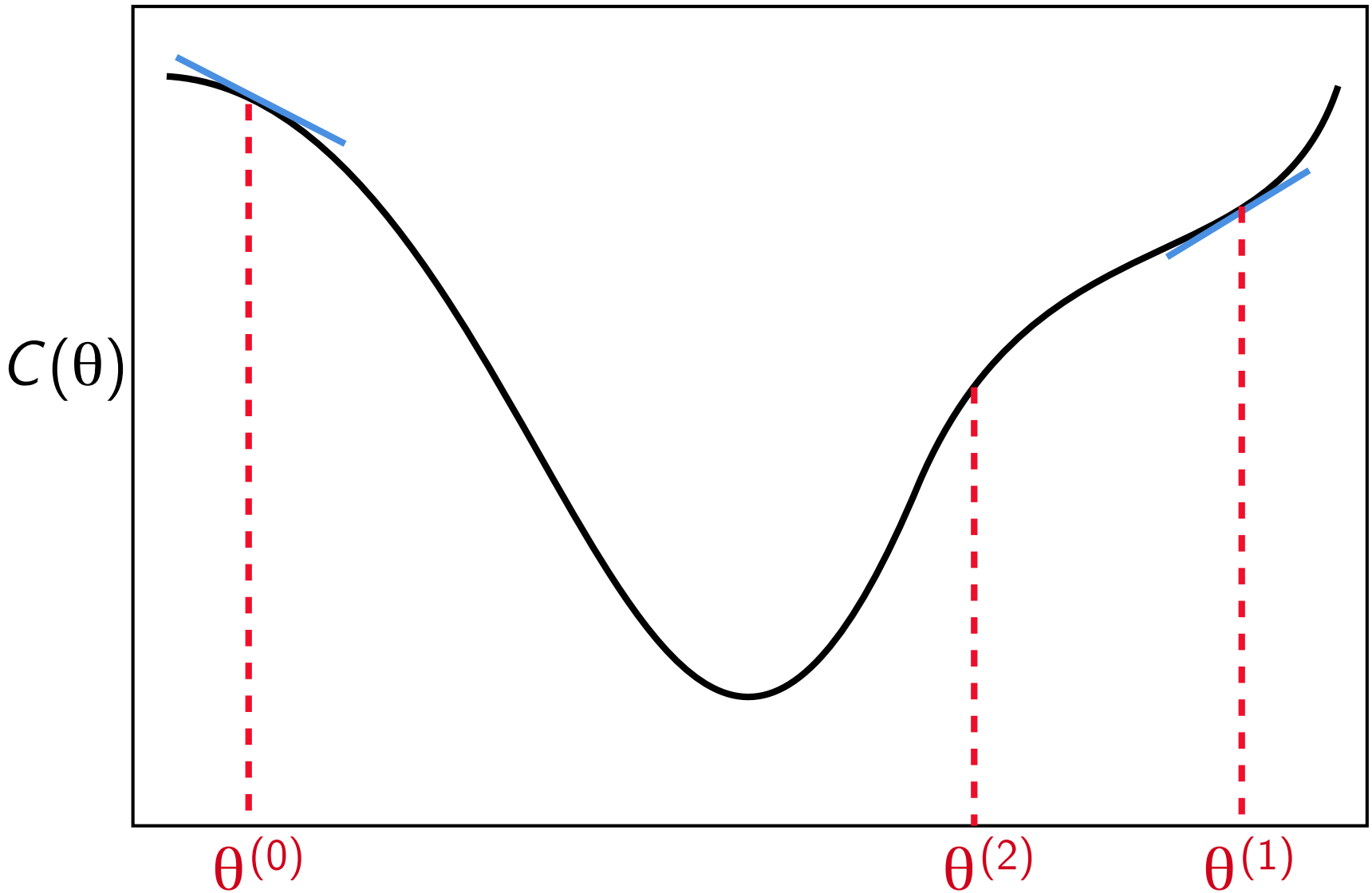
Gradient Descent: Initial guess $\theta^{(0)}$ and compute $\nabla C(\theta^{(0)})$



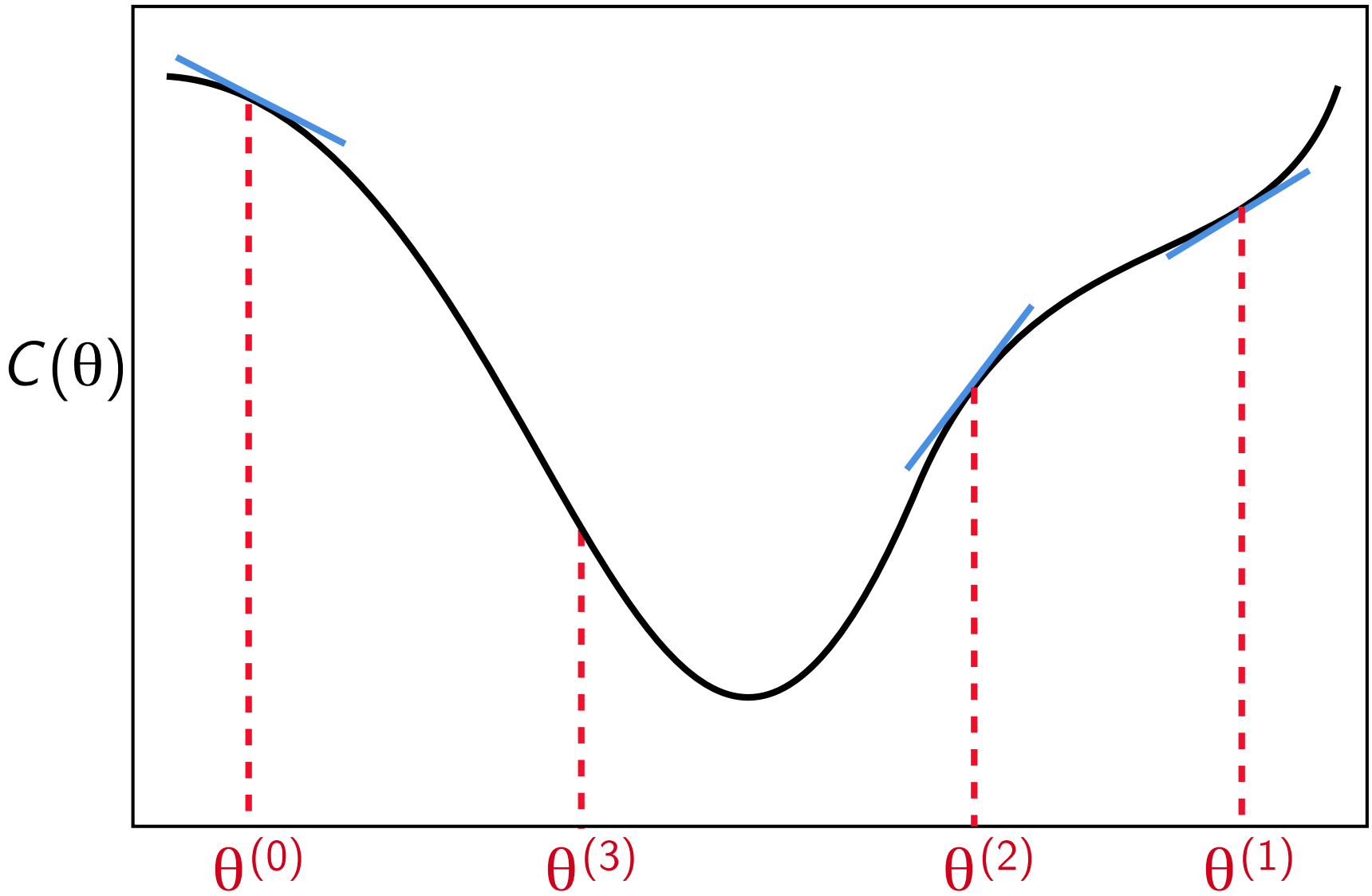
Gradient Descent moves: $\theta^{(1)} = \theta^{(0)} - \alpha \nabla C(\theta^{(0)})$



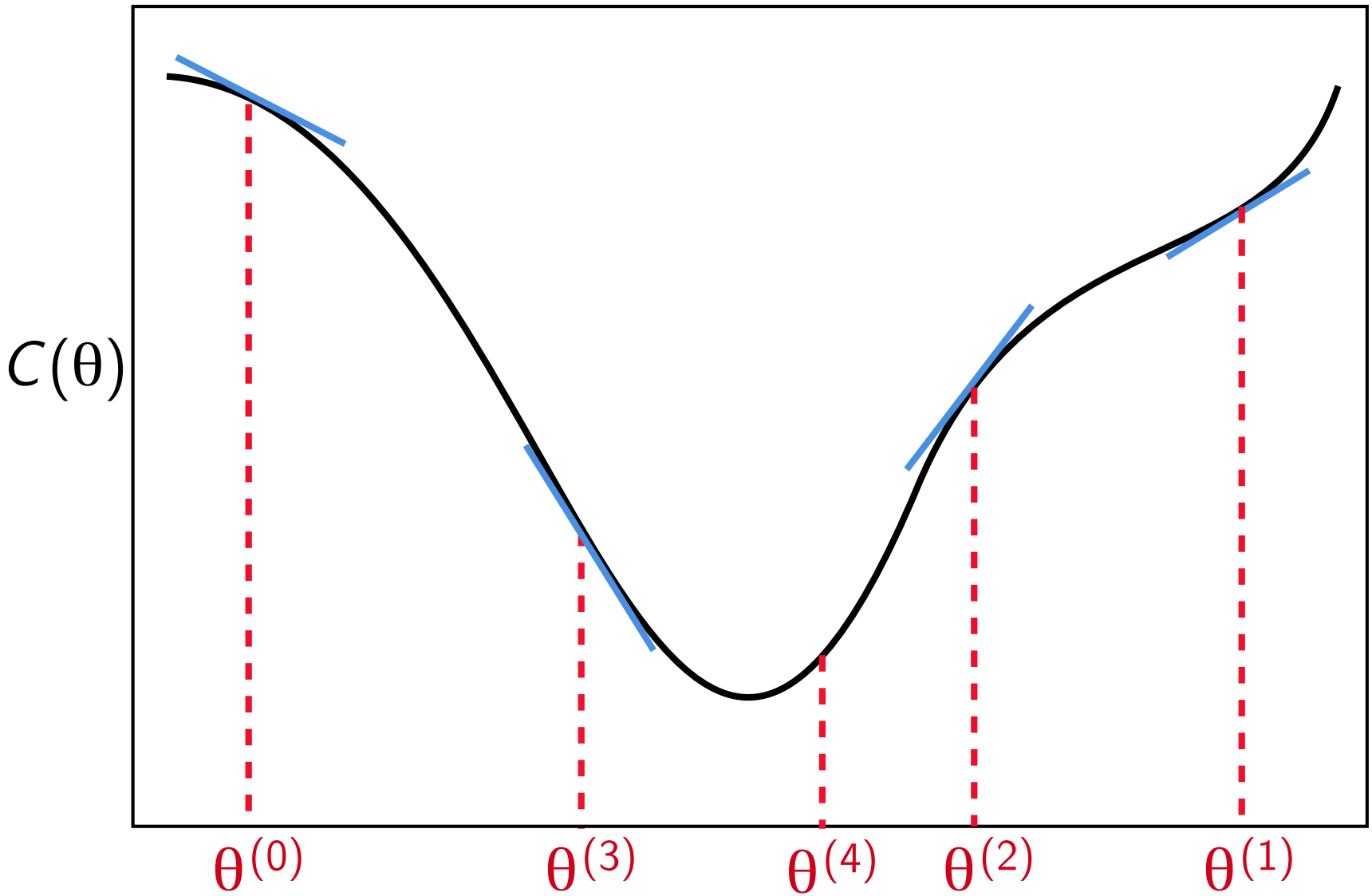
Gradient Descent moves: $\theta^{(2)} = \theta^{(1)} - \alpha \nabla C(\theta^{(1)})$



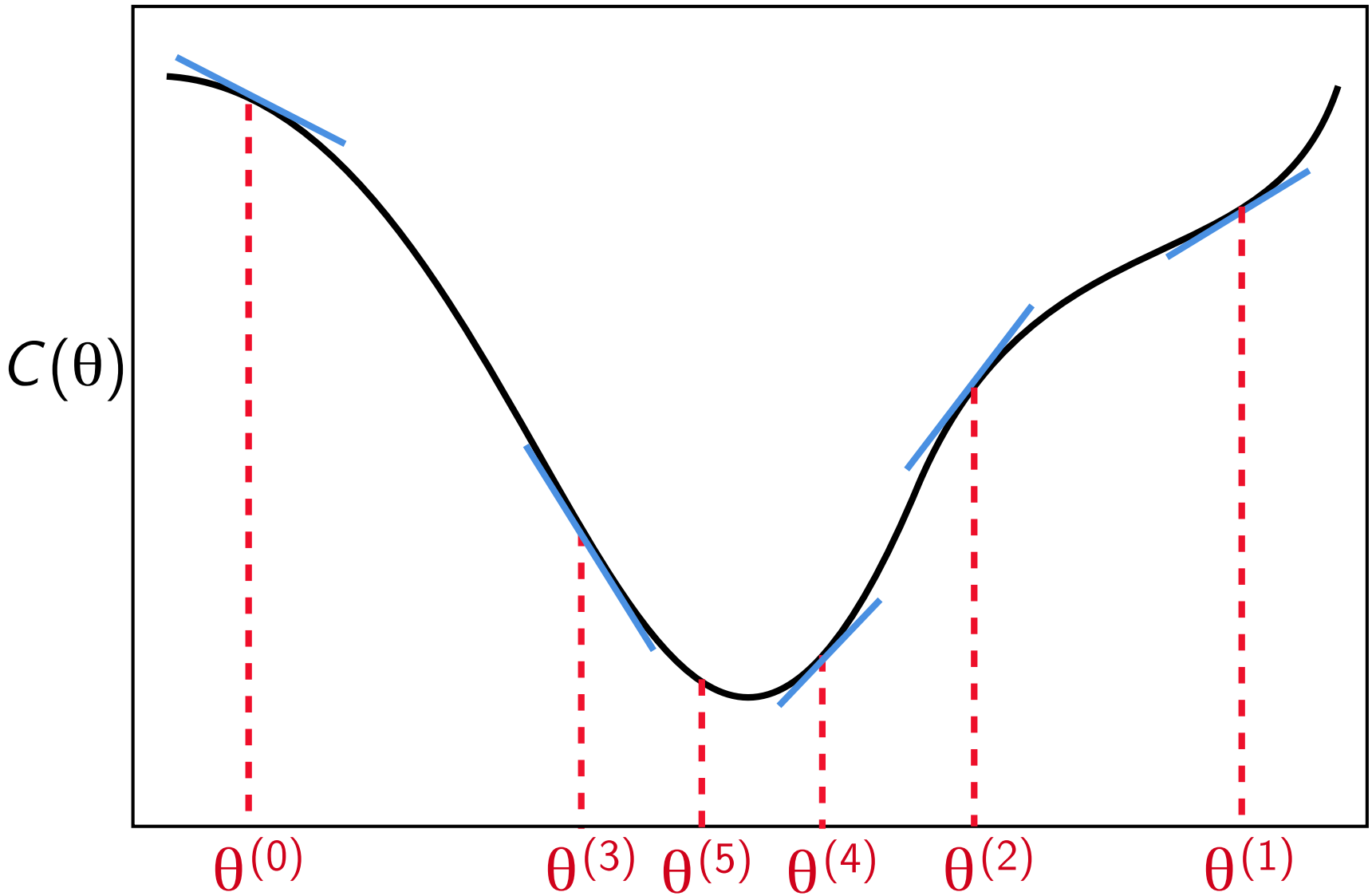
Gradient Descent moves: $\theta^{(3)} = \theta^{(2)} - \alpha \nabla C(\theta^{(2)})$



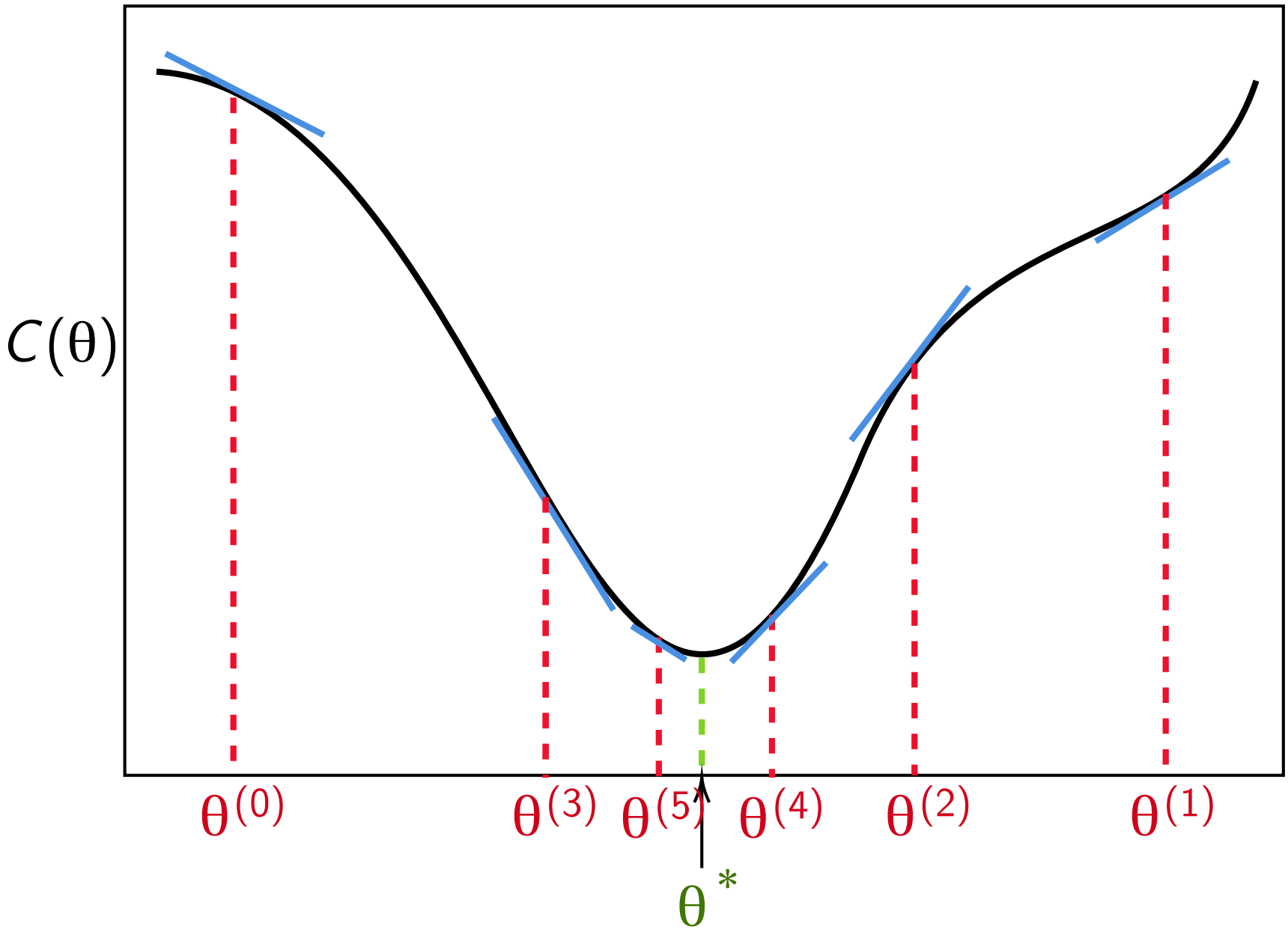
Gradient Descent moves: $\theta^{(4)} = \theta^{(3)} - \alpha \nabla C(\theta^{(3)})$



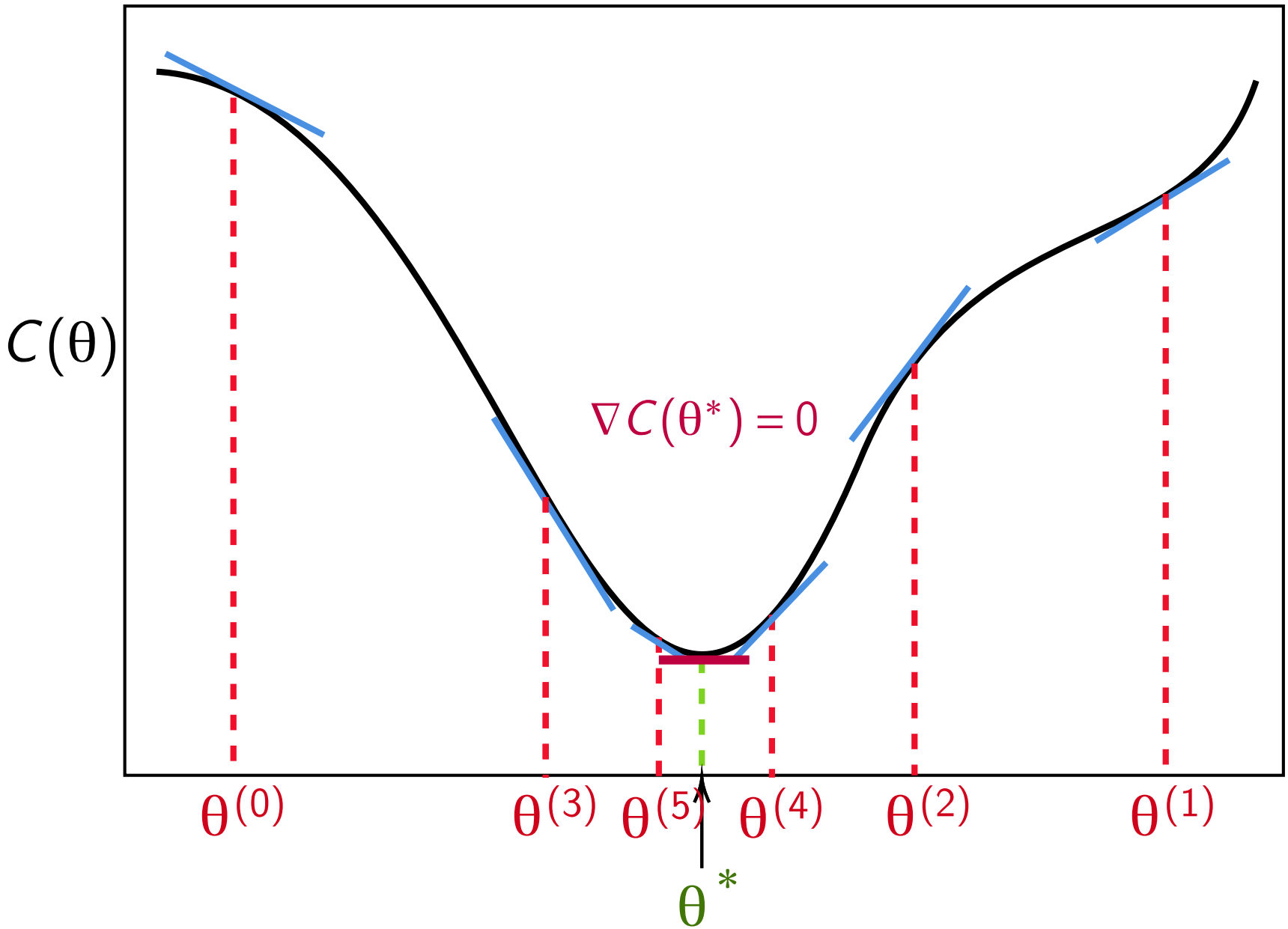
Gradient Descent moves: $\theta^{(5)} = \theta^{(4)} - \alpha \nabla C(\theta^{(4)})$



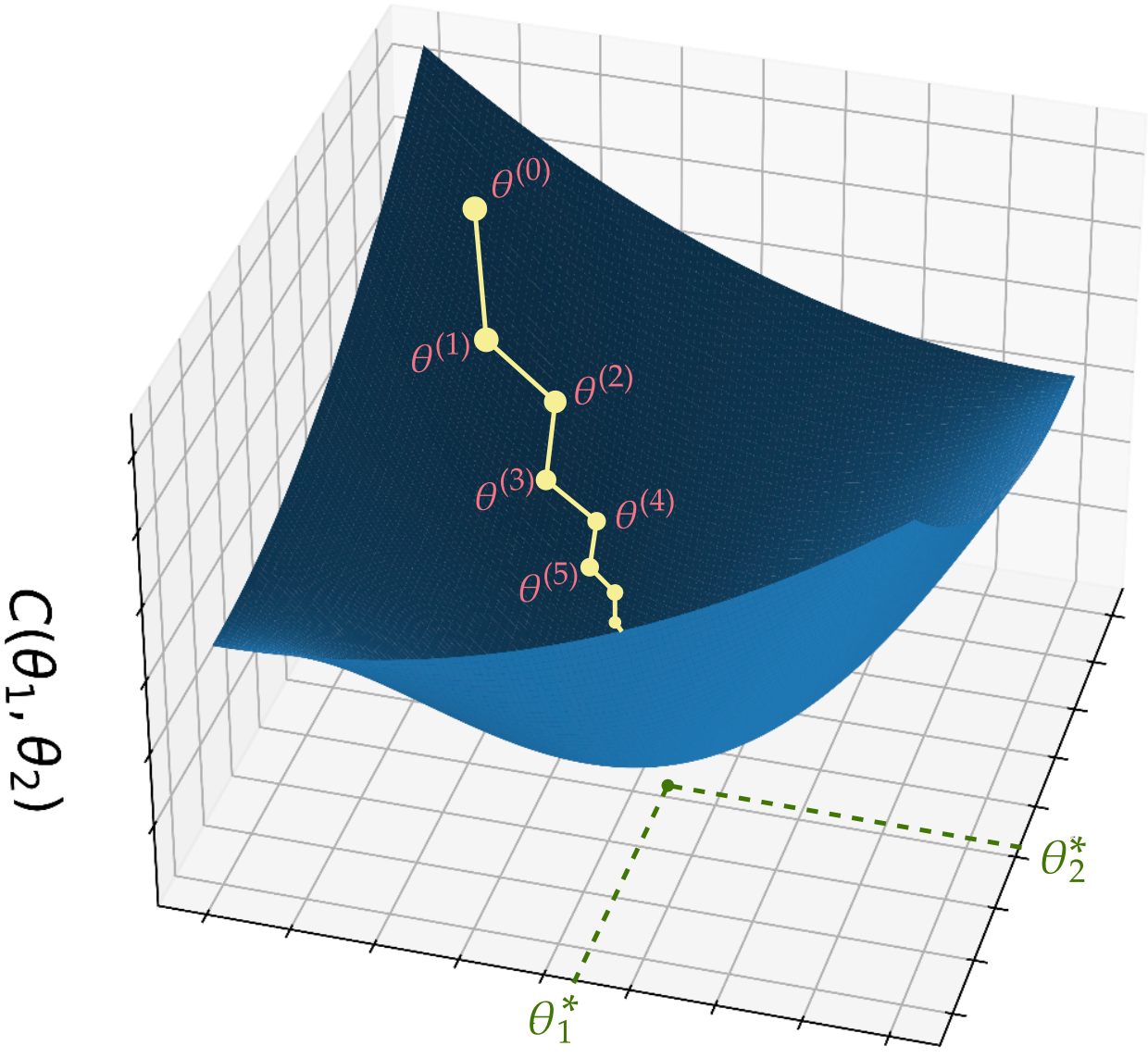
Gradient Descent moves: $\theta^* = \theta^{(5)} - \alpha \nabla C(\theta^{(5)})$



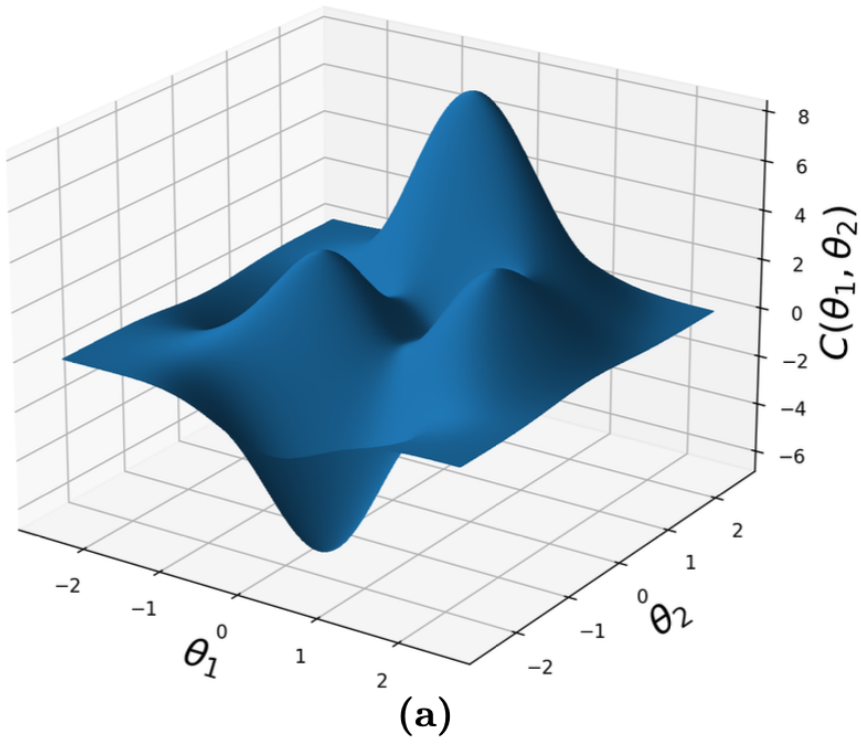
Gradient Descent terminates: $\nabla C(\theta^*) \approx 0$

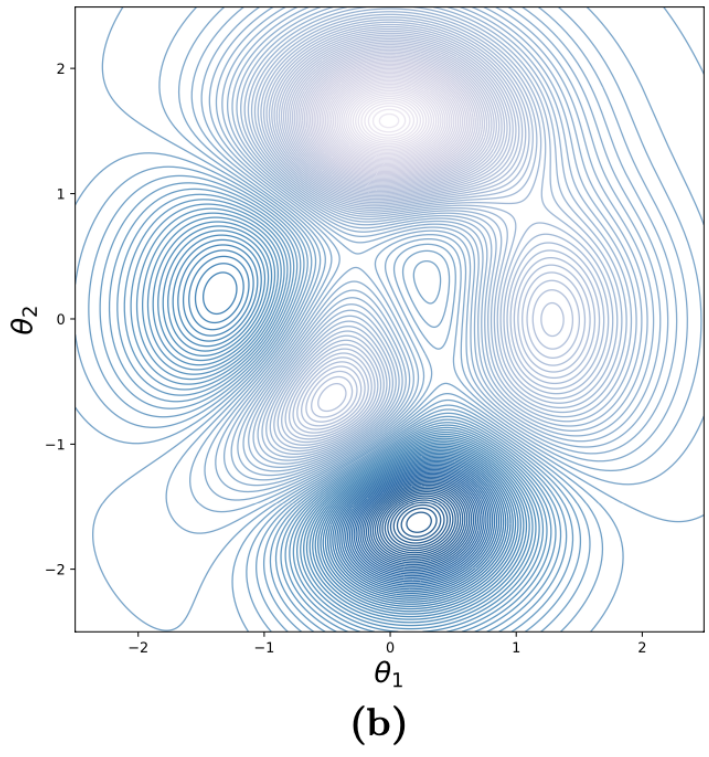


Loss landscape: two parameters



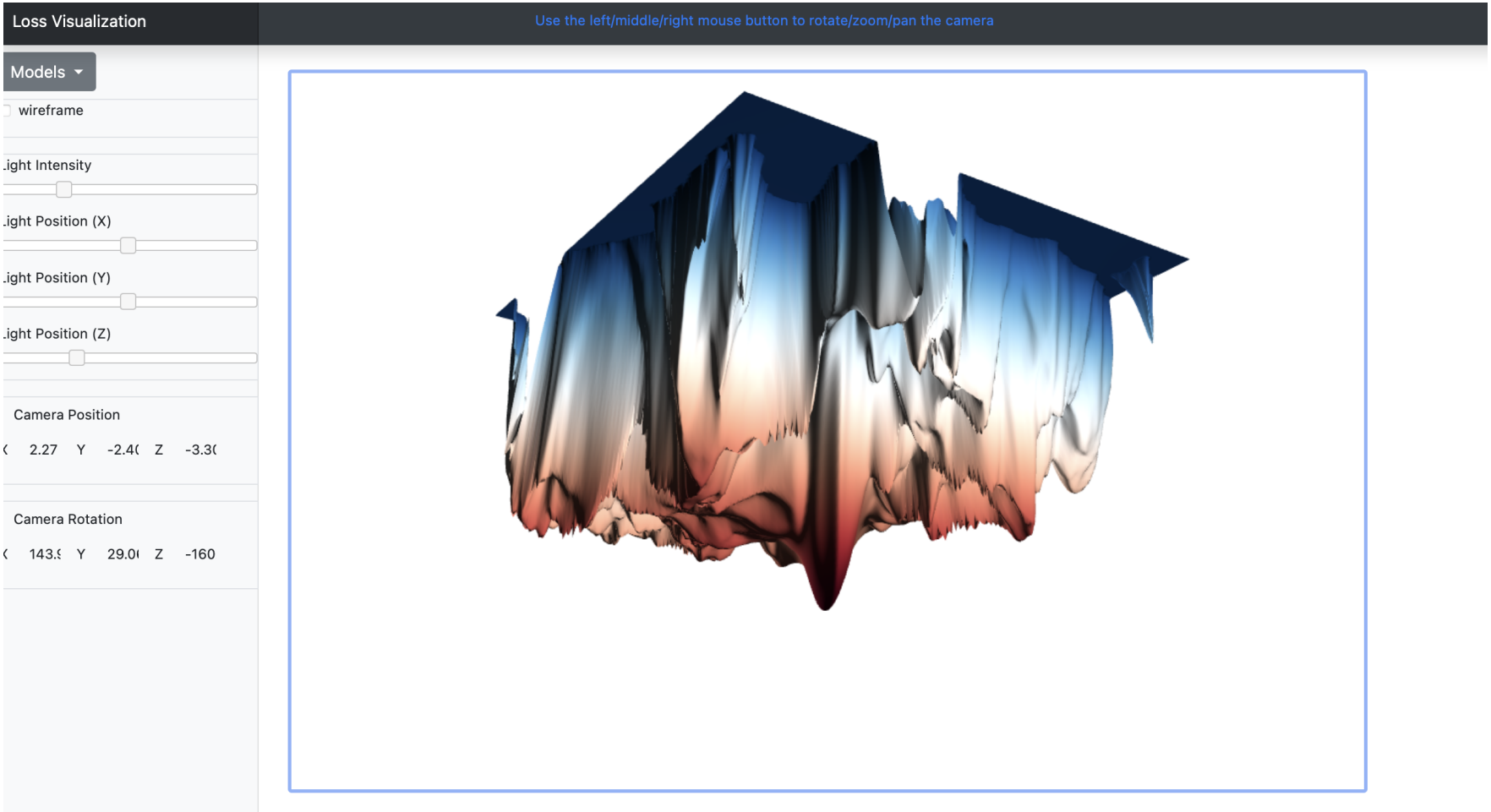
Loss landscape: 3D surface plot and topographical map





Loss landscape is quite complex

Interactive loss landscape visualization



Why Gradient Descent is expensive?

$$C(\theta; \text{Data}) = \frac{1}{n} \sum_{i=1}^{n} C_i(\theta), \qquad i.e, \qquad C_i(\theta) = (y_i - \widehat{y}_i)^2$$

Why Gradient Descent is expensive?

$$C(\theta; \text{Data}) = \frac{1}{n} \sum_{i=1}^{n} C_i(\theta), \qquad i.e, \qquad C_i(\theta) = (y_i - \hat{y}_i)^2$$

- To compute the gradient, GD must look at **all training data** at every step:

$$\nabla C(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla C_i(\theta)$$

- For modern datasets: millions of data points and millions of parameters.
- Each update becomes **slow** and **computationally expensive**.

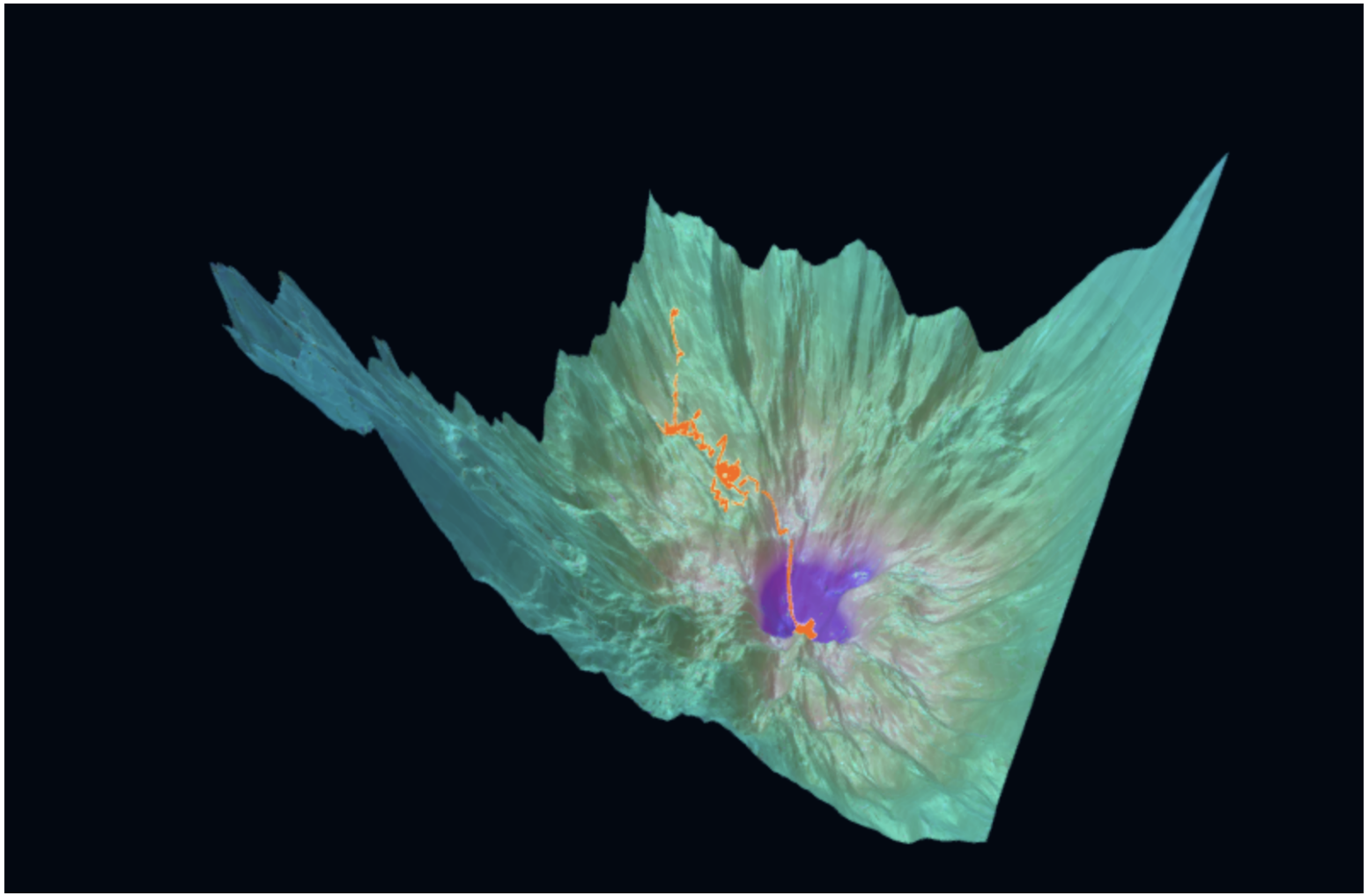
Key idea: *Deep learning replaces full Gradient Descent with faster approximations, such as Stochastic Gradient Descent (SGD) and mini-batch GD, which use only a small subset of the data at each step.*

Question 5

Does gradient descent guarantee that we find the global minimum of the loss function?

Visualisation: `https://losslandscape.com/`

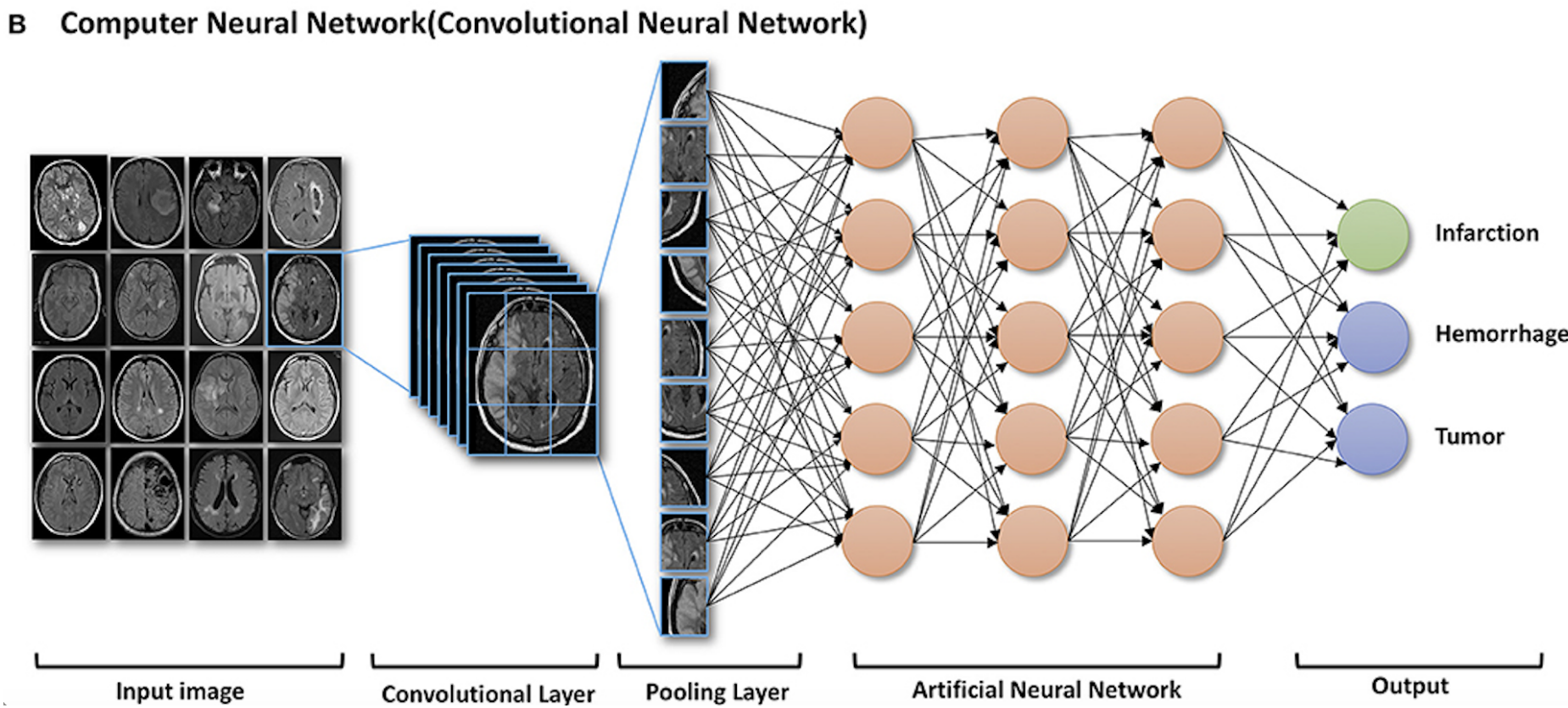
Explorer



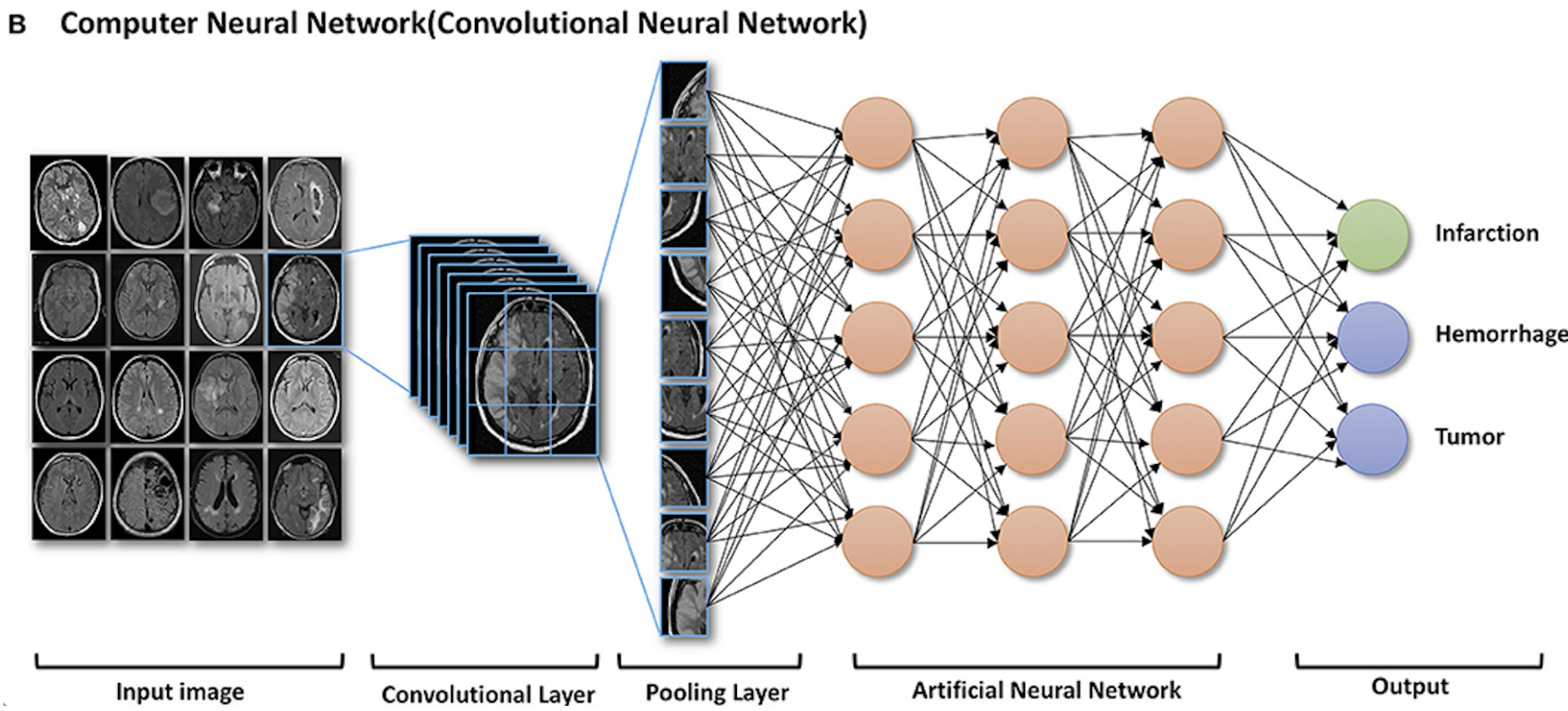
Visualisation: video

https://losslandscape.com/videos/

CNN: the math behind ?



CNN: the math behind ?



The key is the Convolution operation

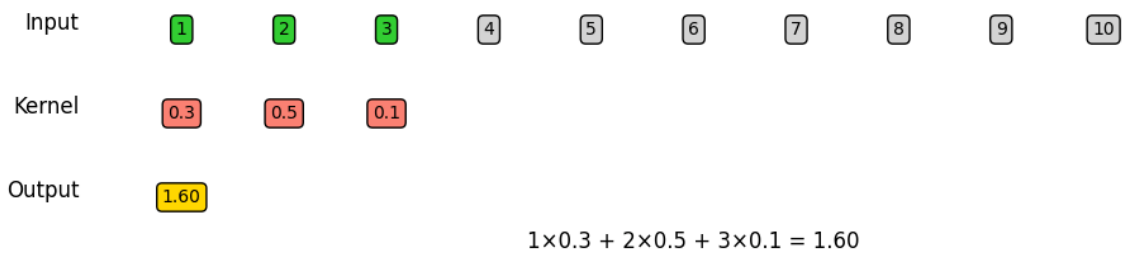
Convolution 1D

Given an input vector $x = [x_1, x_2, \ldots, x_{10}]$ and a kernel (filter) $w = [w_1, w_2, w_3]$, the output z of the 1D convolution operation is :

$$z = w \star x$$

where $\star$ denotes the convolution operation. Each element is computed as:

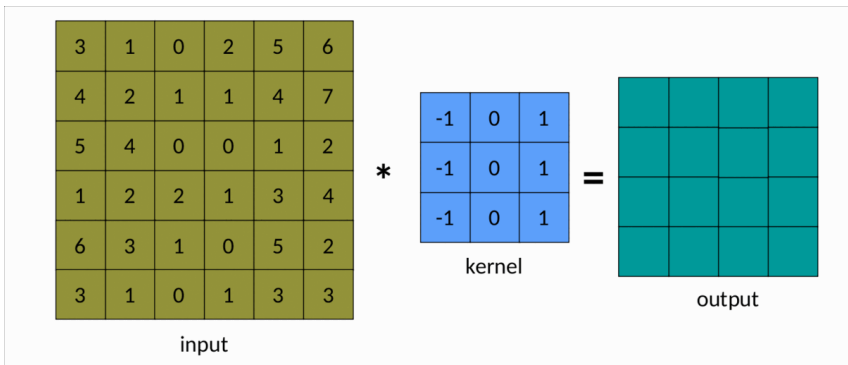
$$z_i = w_1 \cdot x_i + w_2 \cdot x_{i+1} + w_3 \cdot x_{i+2}$$



Convolution 2D: $z = W \star x$

A 3×3 Kernel W applied on an 6×6 input matrix

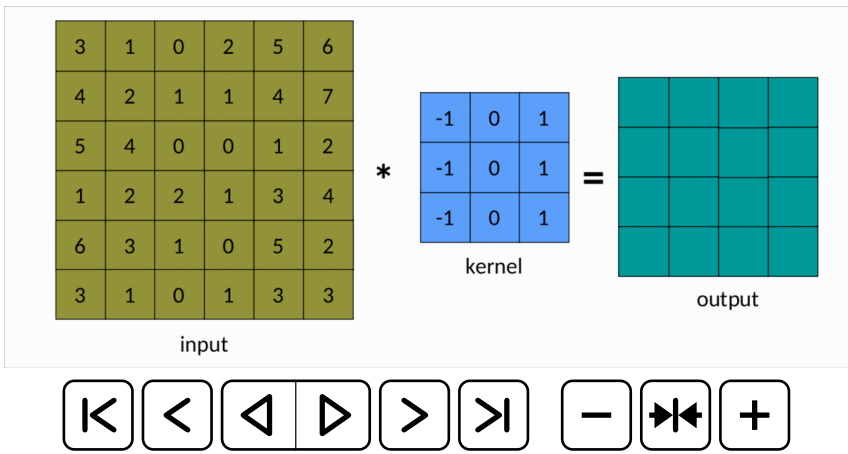
$$\underbrace{\begin{bmatrix} -11 & -4 & 9 & 12 \\ -7 & -6 & 5 & 11 \\ -9 & -8 & 6 & 7 \\ -7 & -4 & 8 & 7 \end{bmatrix}}_{z} = \underbrace{\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}}_{W} \star \underbrace{\begin{bmatrix} 3 & 1 & 0 & 2 & 5 & 6 \\ 4 & 2 & 1 & 1 & 4 & 7 \\ 5 & 4 & 0 & 0 & 1 & 2 \\ 1 & 2 & 2 & 1 & 3 & 4 \\ 6 & 3 & 1 & 0 & 5 & 2 \\ 3 & 1 & 0 & 1 & 3 & 3 \end{bmatrix}}_{x}$$



Convolution 2D: $z = W \star x$

A 3×3 Kernel W applied on an 6×6 input matrix

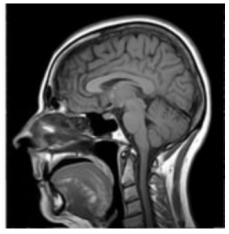
$$\underbrace{\begin{bmatrix} -11 & -4 & 9 & 12 \\ -7 & -6 & 5 & 11 \\ -9 & -8 & 6 & 7 \\ -7 & -4 & 8 & 7 \end{bmatrix}}_{z} = \underbrace{\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}}_{W} \star \underbrace{\begin{bmatrix} 3 & 1 & 0 & 2 & 5 & 6 \\ 4 & 2 & 1 & 1 & 4 & 7 \\ 5 & 4 & 0 & 0 & 1 & 2 \\ 1 & 2 & 2 & 1 & 3 & 4 \\ 6 & 3 & 1 & 0 & 5 & 2 \\ 3 & 1 & 0 & 1 & 3 & 3 \end{bmatrix}}_{x}$$



$$z_{i,j} = \sum_{m=0}^{2} \sum_{n=0}^{2} W_{m,n} \cdot x_{i+m-1,j+n-1}$$

Example of 2d convolution filter

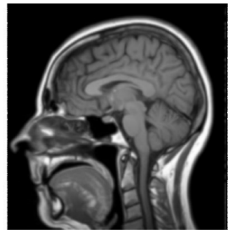
0	0	0
0	1	0
0	0	0



Identity Kernel: Just the starting image itself.

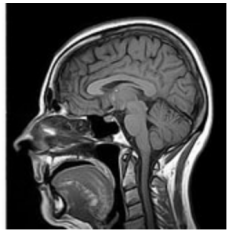
$$\frac{1}{9}$$

1	1	1
1	1	1
1	1	1



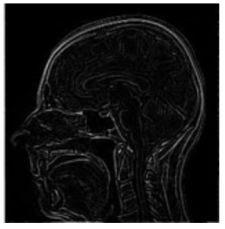
Box Blur Kernel: Each pixel is averaged equally with its 8 neighbors.

0	-1	0
-1	5	-1
0	-1	0



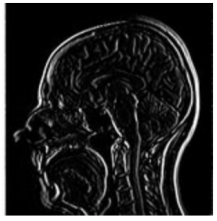
Sharpen Kernel:: Differences emphasized with its adjacent pixel values.

-1	-1	-1
-1	8	-1
-1	-1	-1



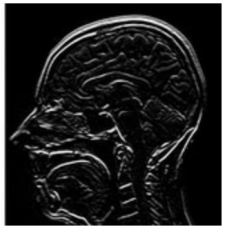
Outline ("Edge") Kernel: Highlights neighboring pixels with large differences of intensity.

1	0	-1
2	0	-2
1	0	-1



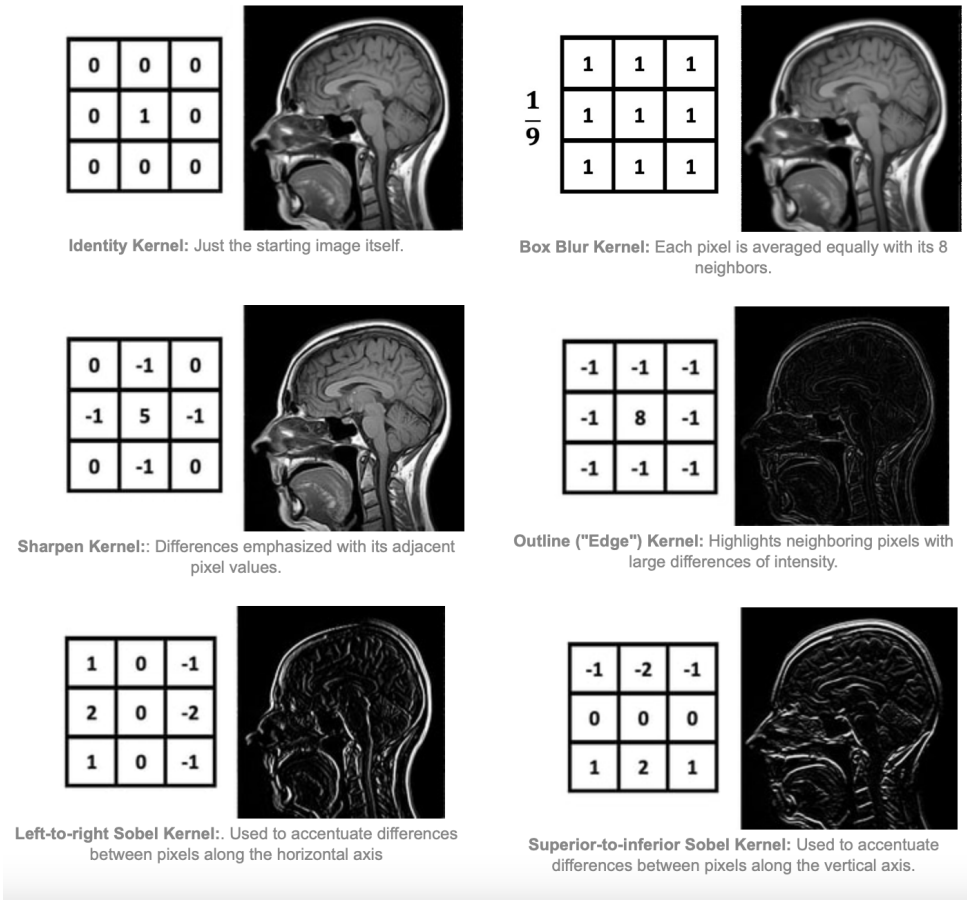
Left-to-right Sobel Kernel:. Used to accentuate differences between pixels along the horizontal axis

-1	-2	-1
0	0	0
1	2	1



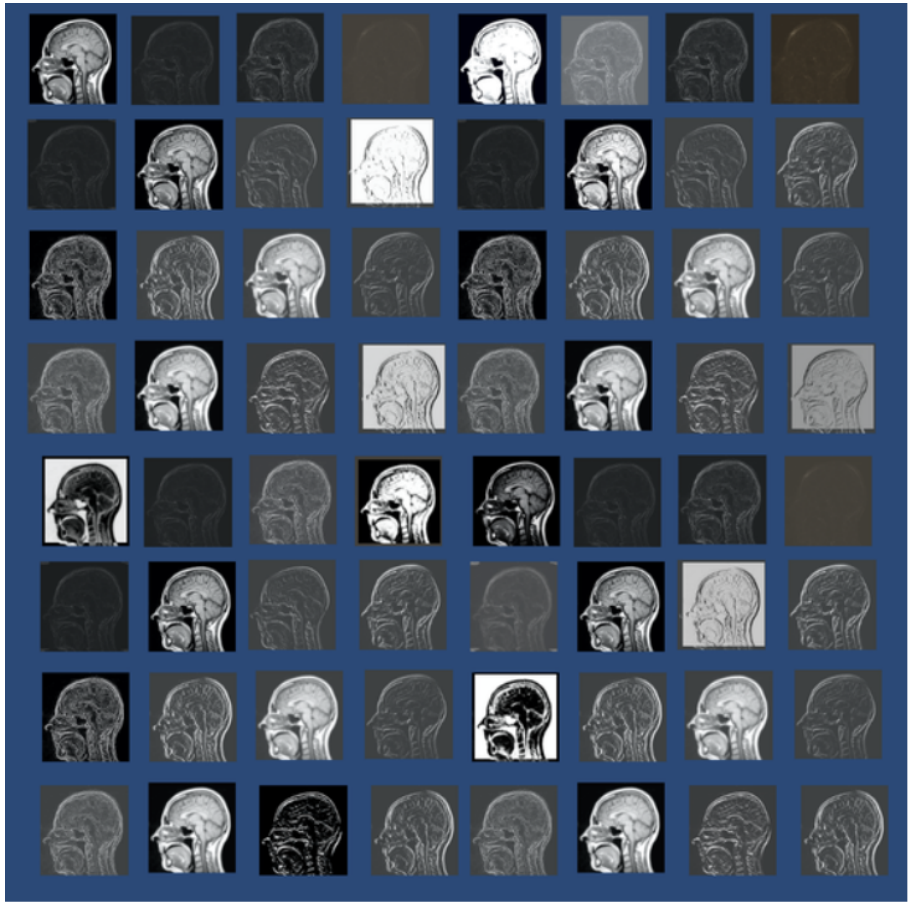
Superior-to-inferior Sobel Kernel: Used to accentuate differences between pixels along the vertical axis.

Example of 2d convolution filter



The key is the Convolution operation

CNN: Kernels are learned



Why Convolutional Kernels?

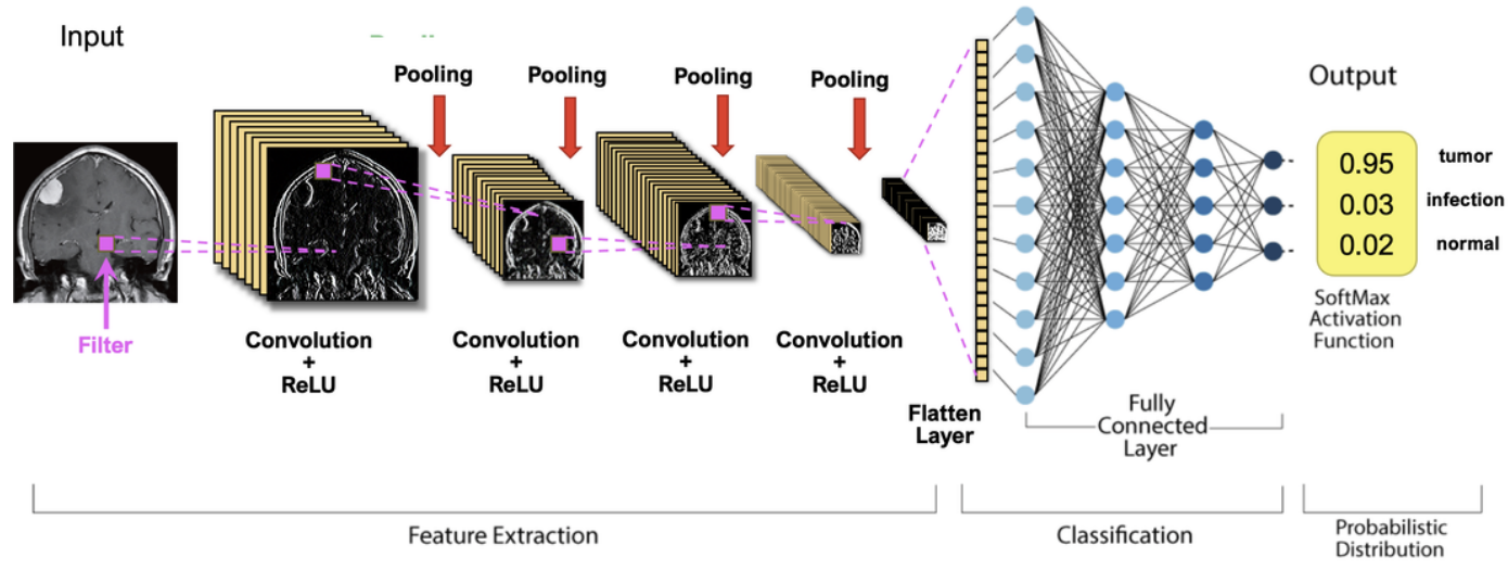
Key idea: Convolutional kernels allow neural networks to efficiently detect local patterns that can appear anywhere in the input.

- **Locality**
  - Nearby pixels (or signal values) are strongly related
  - Clinically relevant structures (edges, textures, lesions) are local
  - Convolution focuses on small neighborhoods
- **Translation invariance**
  - A relevant pattern matters regardless of its position
  - The same kernel is applied across the entire image
  - Enables detection of the same feature anywhere
- **Parameter efficiency**
  - Fully connected layers require many parameters
  - Convolution uses shared weights
  - Fewer parameters $\Rightarrow$ better generalization

Take-home message: *What matters locally matters everywhere.*

CNN as a Hierarchical Feature Extractor

A CNN is a sequence of convolutional layers that progressively extract more abstract features before making a prediction.

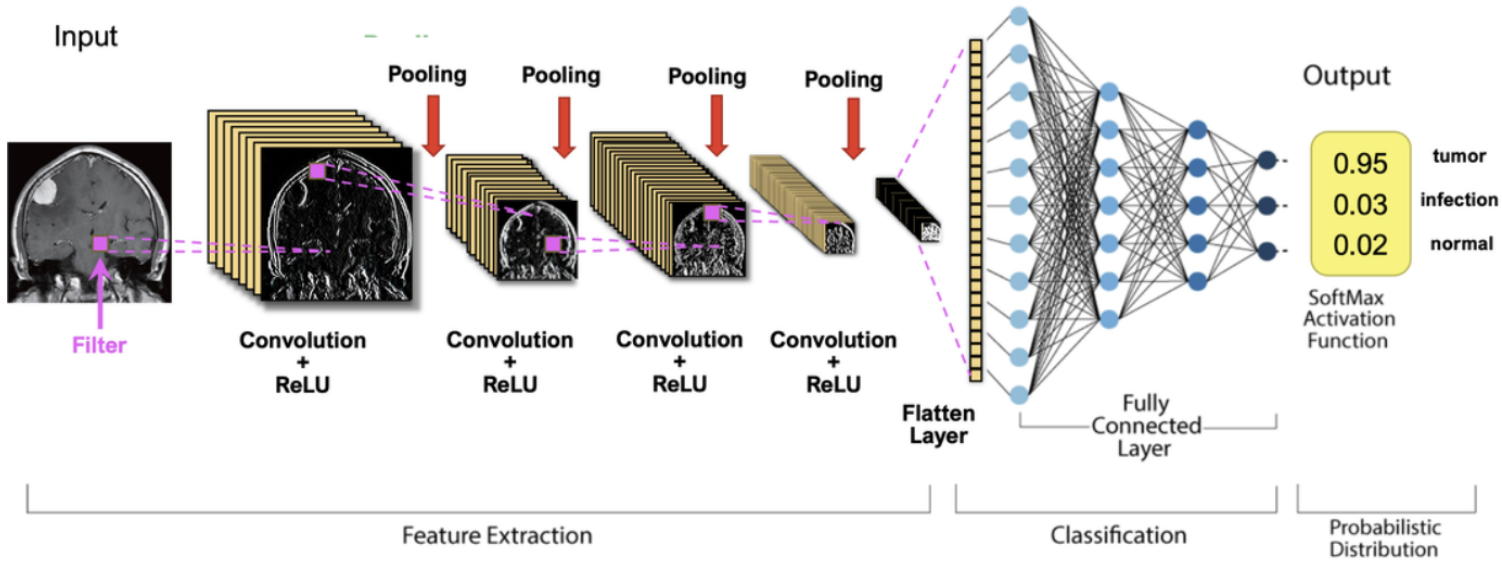


Input → **Conv + ReLU** → **Conv + ReLU** → **Conv + ReLU**

Image / Signal | Edges, textures | Shapes, regions | High-level patterns

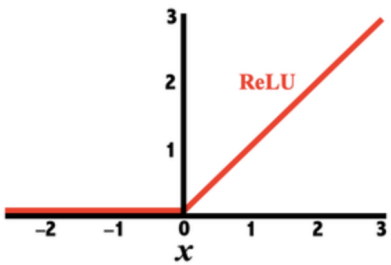
CNN as a Hierarchical Feature Extractor

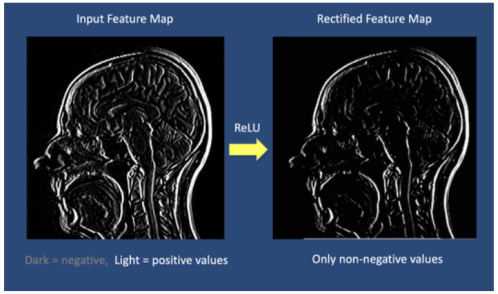
A CNN is a sequence of convolutional layers that progressively extract more abstract features before making a prediction.



Input → **Conv + ReLU** → **Conv + ReLU** → **Conv + ReLU**

Image / Signal | Edges, textures | Shapes, regions | High-level patterns

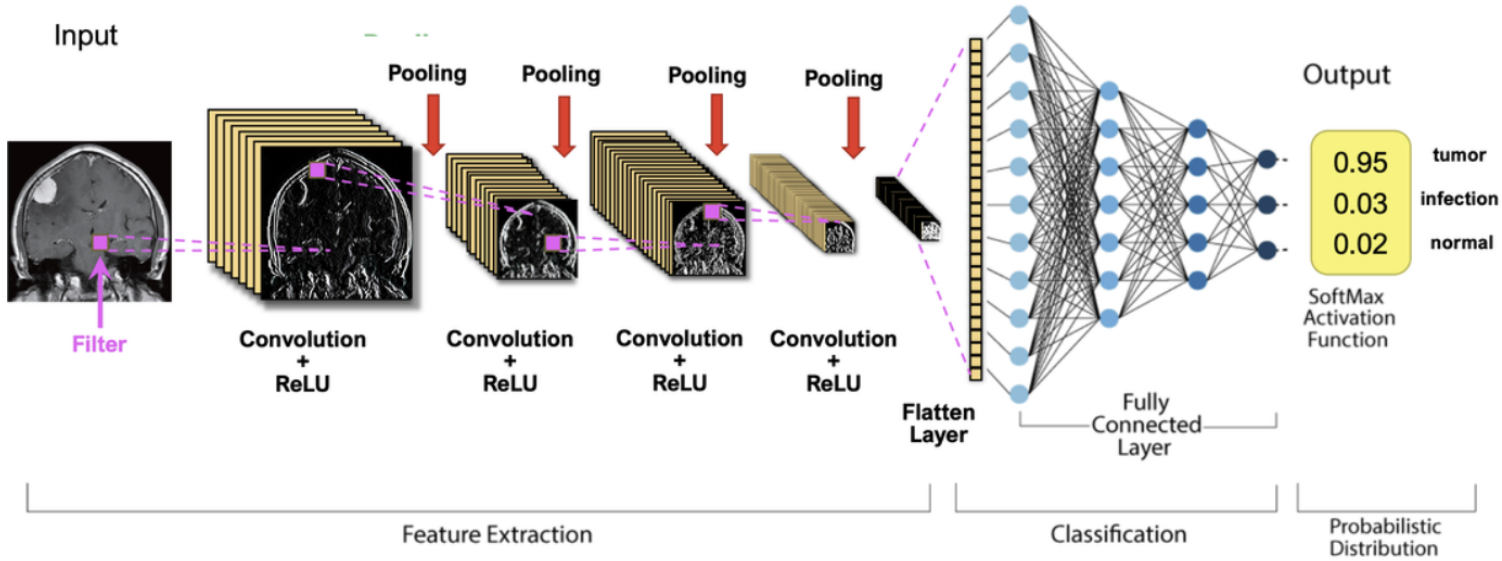




Appearance of one of the above 64 feature maps before and after application of ReLU activation function

CNN as a Hierarchical Feature Extractor

A CNN is a sequence of convolutional layers that progressively extract more abstract features before making a prediction.



Input →	**Conv + ReLU** →	**Conv + ReLU** →	**Conv + ReLU**
Image / Signal	Edges, textures	Shapes, regions	High-level patterns

- Convolution layers perform **feature extraction**
- Fully connected layers perform **decision making**

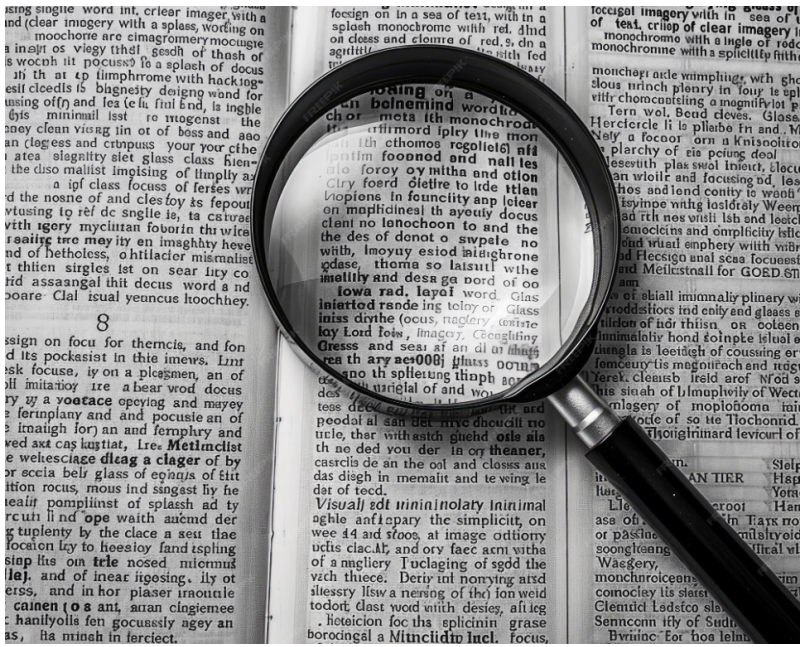
Analogy: from edges → structures → diagnosis

Question 6?

Do convolutional neural networks reduce the number of parameters by sharing the same filter across different parts of the image?

Do I still have your attention?

Now, let's see how machines focus on the important information, just like clinicians do.



"Attention is all you need" – Vaswani et al., 2017

Attention: Selecting Relevant Information

Human Attention

- When analysing imaging, clinicians focus on suspicious regions
- When reading clinical notes, doctors focus on key symptoms

Key Idea

Attention = focusing on relevant information while ignoring distractions

Attention: Selecting Relevant Information

Human Attention

- When analysing imaging, clinicians focus on suspicious regions
- When reading clinical notes, doctors focus on key symptoms

Key Idea

Attention = focusing on relevant information while ignoring distractions

Example: Clinical Note

> The patient reports a **headache** and took ibuprofen for arthritis.

Clinical Question:
What is the patient's main symptom?

Model Needs To Focus On

Headache

Attention: Selecting Relevant Information

Human Attention

- When analysing imaging, clinicians focus on suspicious regions
- When reading clinical notes, doctors focus on key symptoms

Key Idea

Attention = focusing on relevant information while ignoring distractions

Example: Clinical Note

> The patient reports a **headache** and took ibuprofen for arthritis.

Clinical Question:
What is the patient's main symptom?

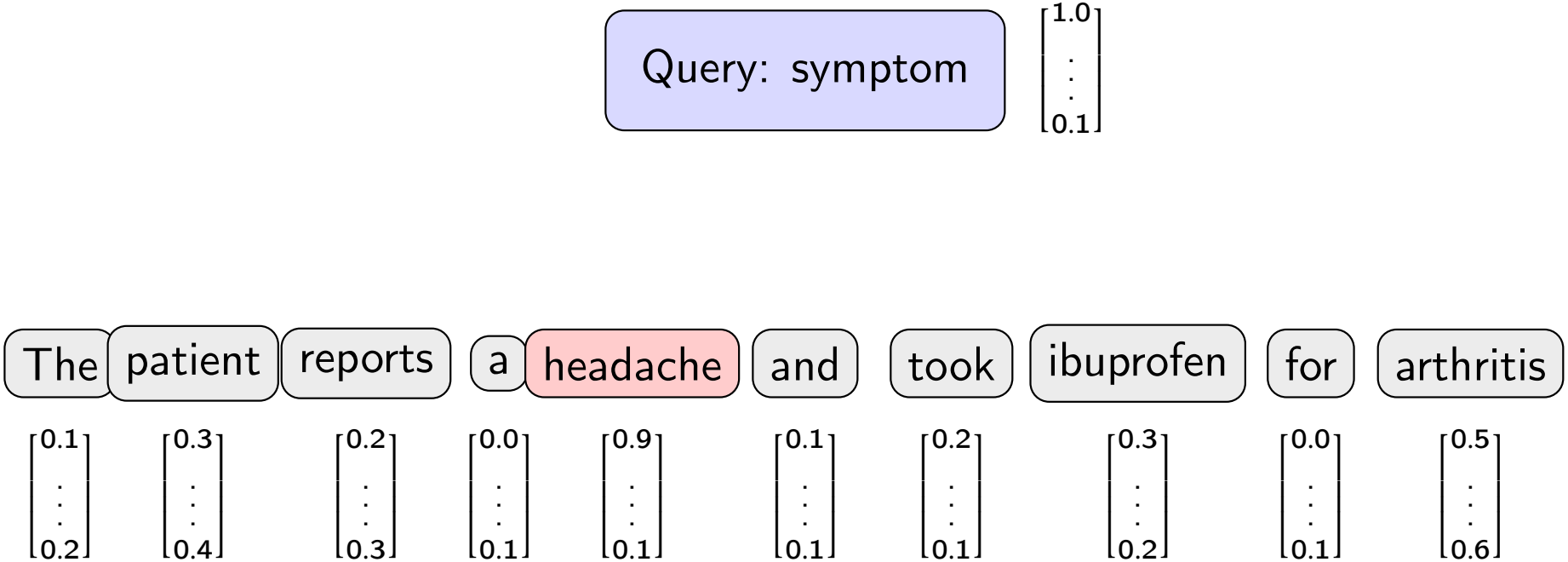
Model Needs To Focus On

Headache

Attention helps models focus on the right words, just like clinicians do.

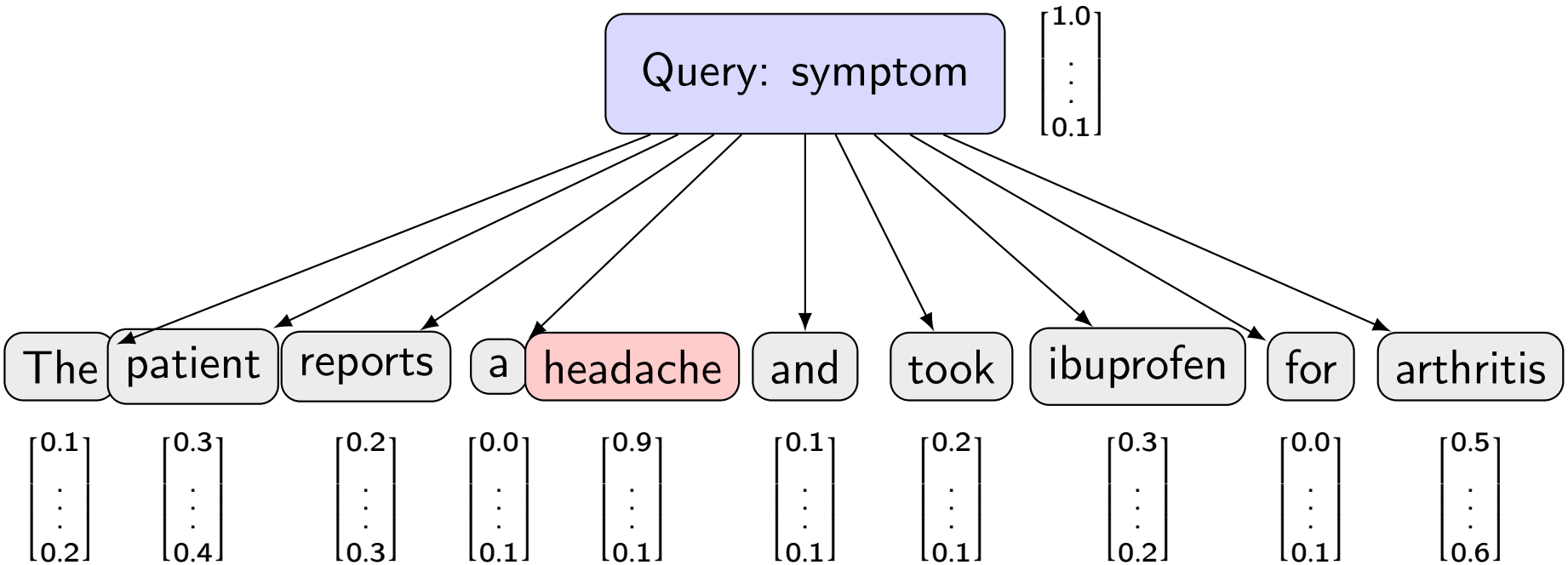
How Does a Model Use Attention?

Clinical Question: What is the patient's main symptom?



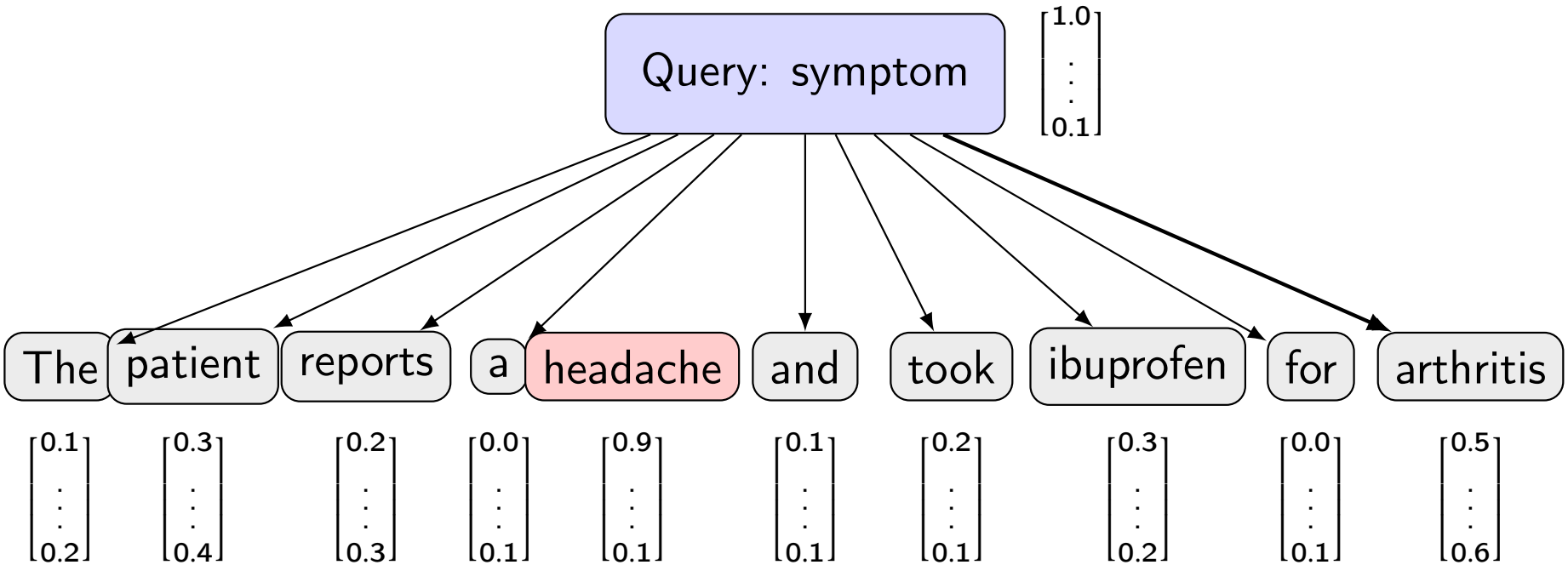
How Does a Model Use Attention?

Clinical Question: What is the patient's main symptom?



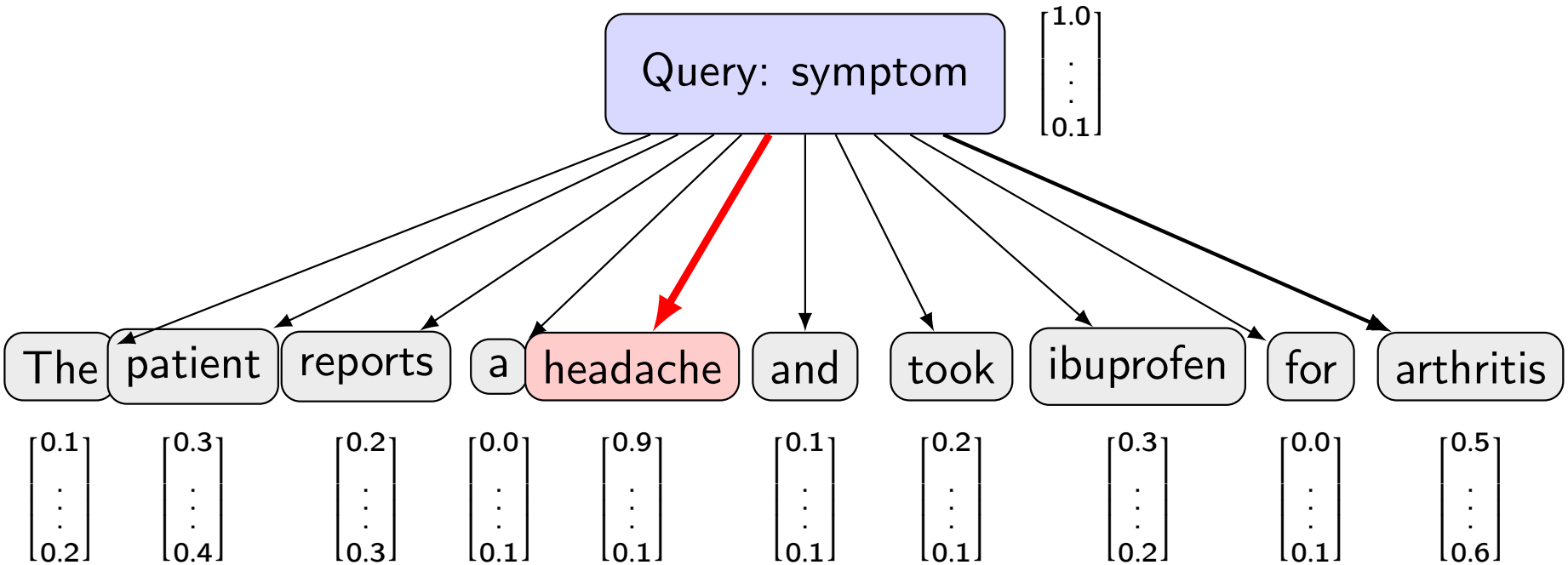
How Does a Model Use Attention?

Clinical Question: What is the patient's main symptom?



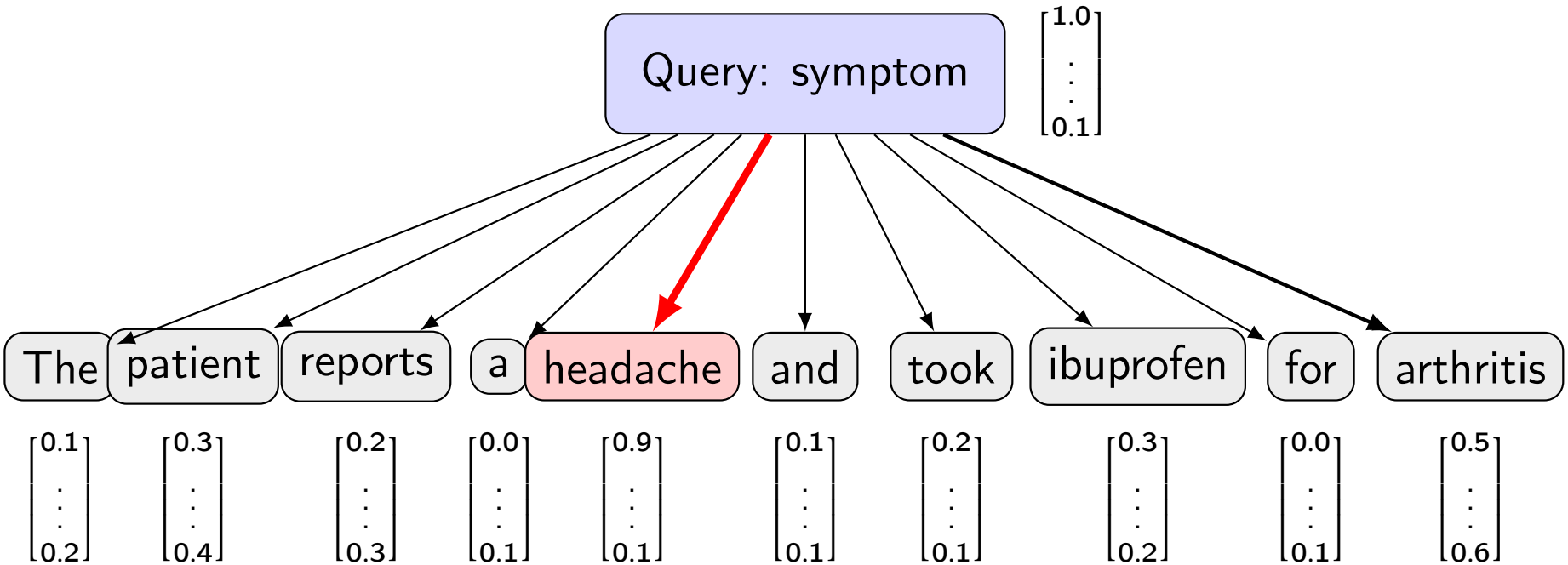
How Does a Model Use Attention?

Clinical Question: What is the patient's main symptom?



How Does a Model Use Attention?

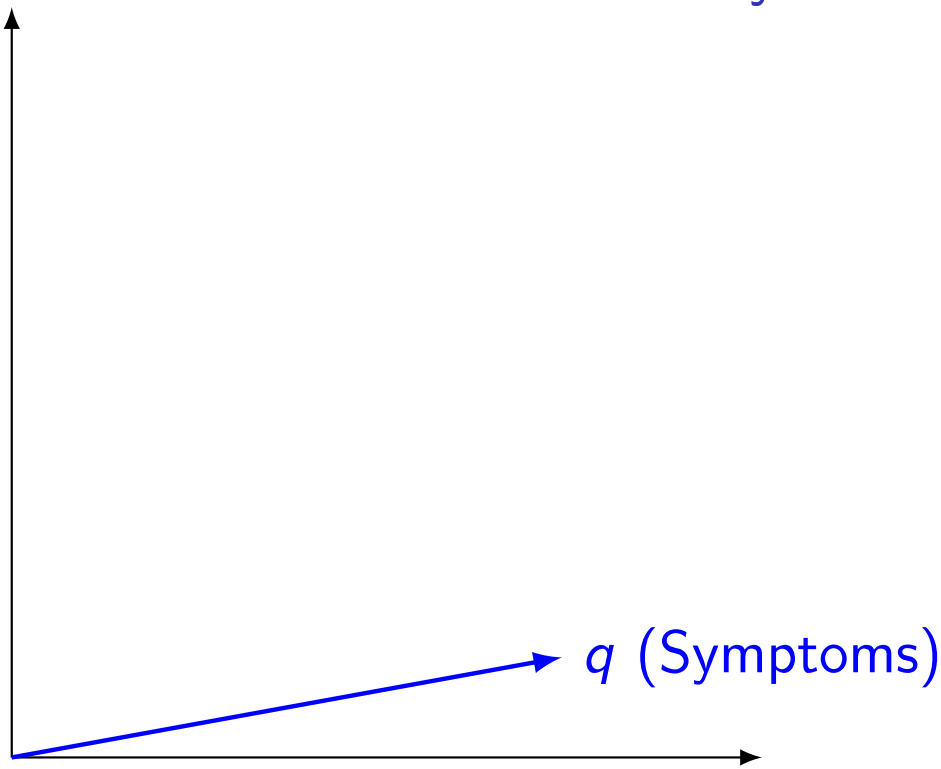
Clinical Question: What is the patient's main symptom?



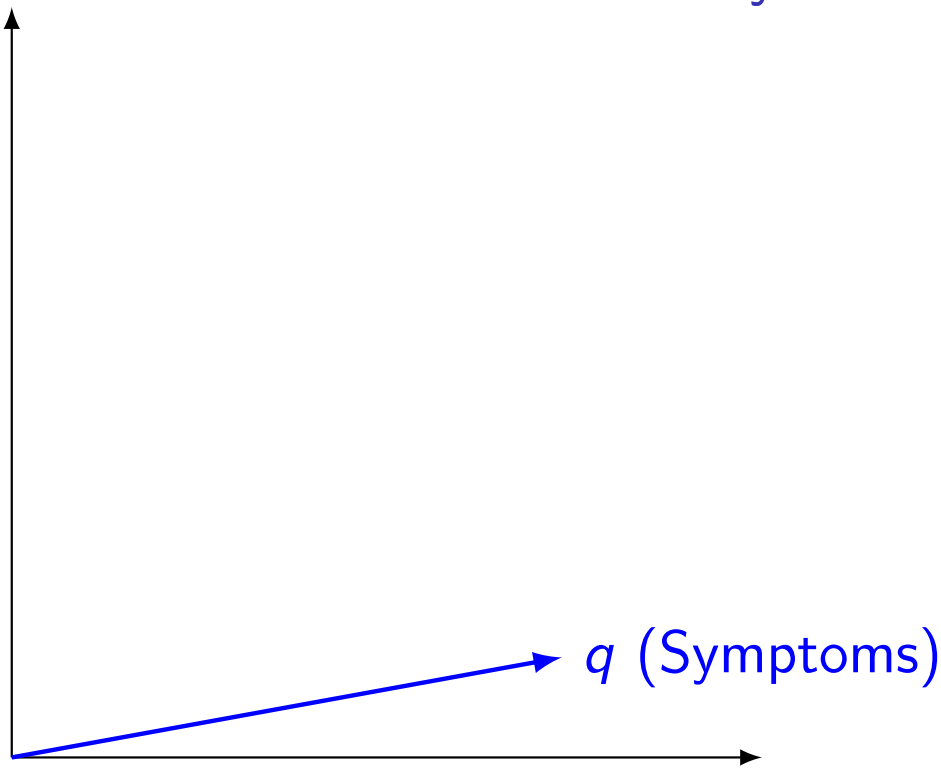
Key Idea

The model compares the question with each word and assigns importance using Dot Product.

Why Does Dot Product Measure Similarity?



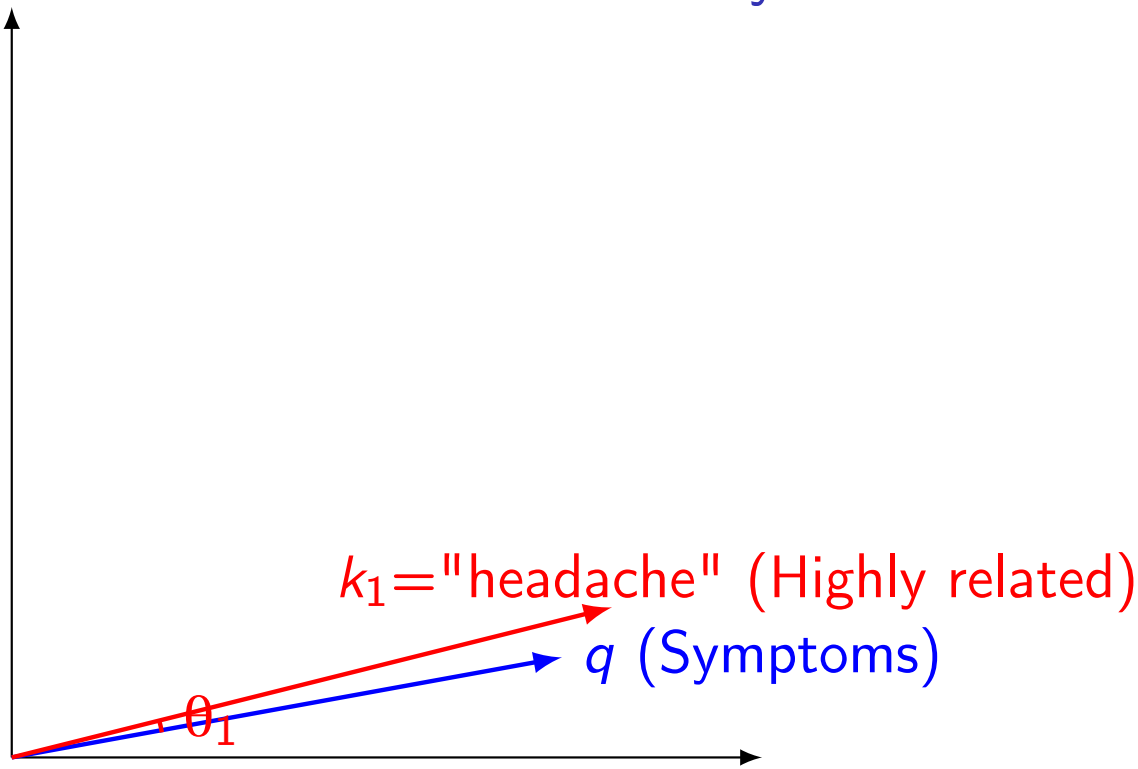
Why Does Dot Product Measure Similarity?



$$q \cdot k = \overbrace{||q||}^{\text{Norm of } q} \times \underbrace{||k||}_{\text{Norm of } k} \times \cos(\theta)$$

- Small angle $\rightarrow$ High similarity
- 90° angle $\rightarrow$ No similarity

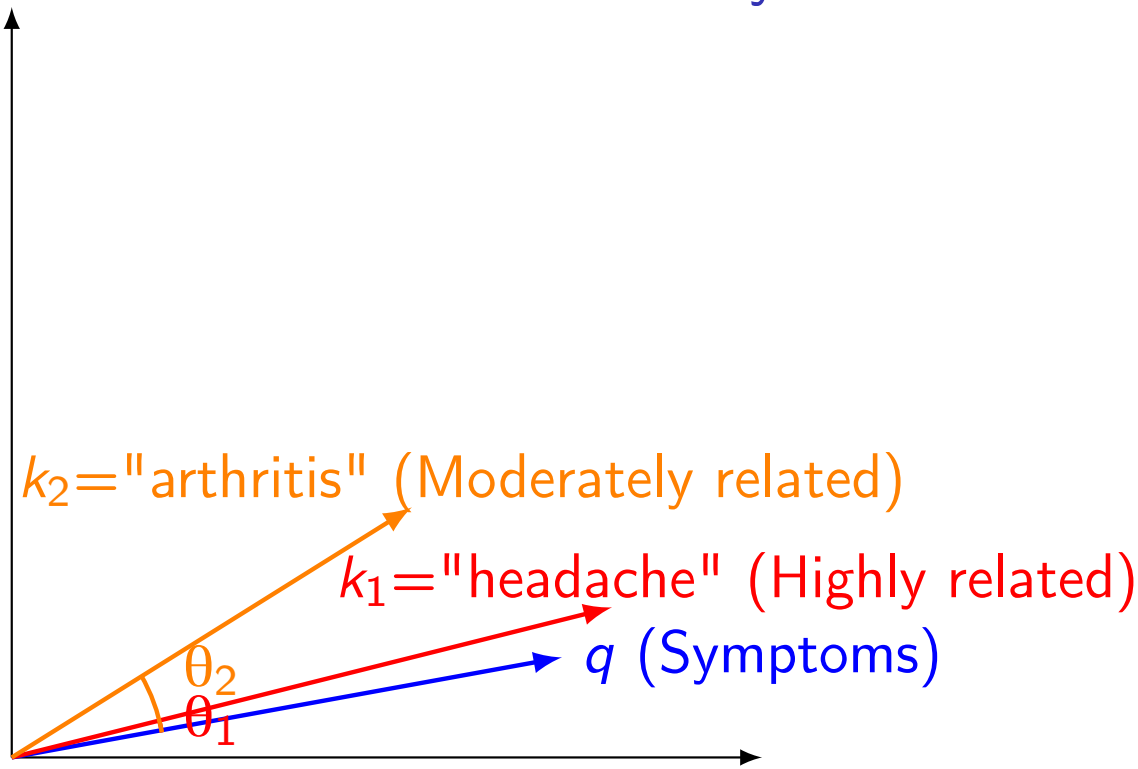
Why Does Dot Product Measure Similarity?



$$q \cdot k = \overbrace{||q||}^{\text{Norm of } q} \times \underbrace{||k||}_{\text{Norm of } k} \times \cos(\theta)$$

- Small angle $\rightarrow$ High similarity
- 90° angle $\rightarrow$ No similarity

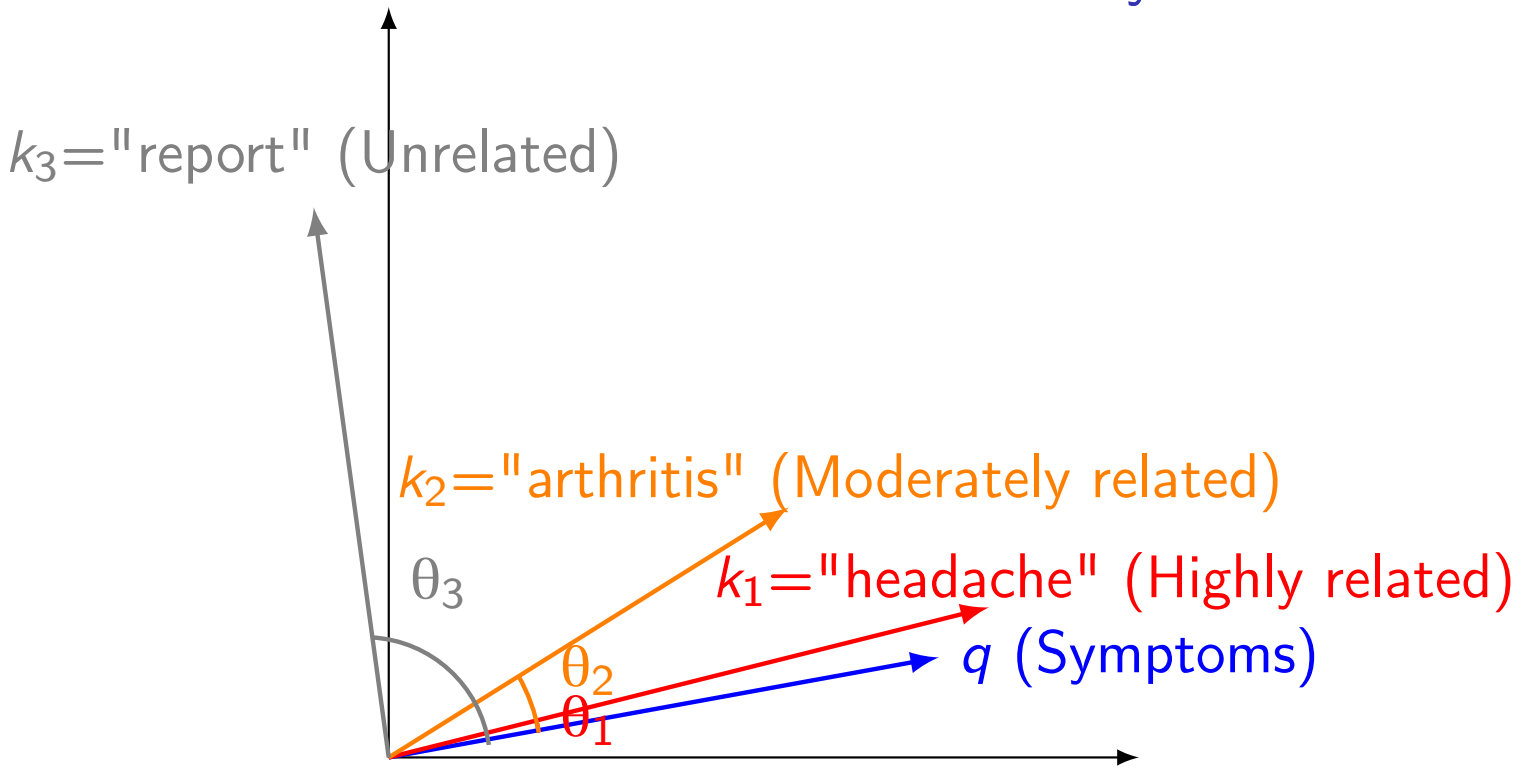
Why Does Dot Product Measure Similarity?



$$q \cdot k = \overbrace{||q||}^{\text{Norm of } q} \times \underbrace{||k||}_{\text{Norm of } k} \times \cos(\theta)$$

- Small angle $\rightarrow$ High similarity
- 90° angle $\rightarrow$ No similarity

Why Does Dot Product Measure Similarity?



$$q \cdot k = \overbrace{||q||}^{\text{Norm of } q} \times \underbrace{||k||}_{\text{Norm of } k} \times \cos(\theta)$$

- Small angle $\rightarrow$ High similarity
- 90° angle $\rightarrow$ No similarity

From Similarity to Answer: Attention Mechanism

Step 1: Compute Attention Weights

- Softmax converts these similarity scores into weights α_i between 0 and 1
- Words more relevant to the question get higher weights

Word (k_i)	Weight α_i
The	0.02
patient	0.05
reports	0.03
a	0.01
headache	**0.75**
and	0.02
took	0.03
ibuprofen	0.05
for	0.02
arthritis	0.02

From Similarity to Answer: Attention Mechanism

Step 1: Compute Attention Weights

- Softmax converts these similarity scores into weights α_i between 0 and 1
- Words more relevant to the question get higher weights

Word (k_i)	Weight α_i
The	0.02
patient	0.05
reports	0.03
a	0.01
headache	**0.75**
and	0.02
took	0.03
ibuprofen	0.05
for	0.02
arthritis	0.02

Step 2: Build Context Vector

- Multiply each word vector by its attention weight α_i
- Sum weighted vectors to get the context vector:

$$c = \sum_i \alpha_i v_i$$

Context Vector $c \rightarrow$ mostly influenced by headache

Step 3: Model Output

- Context vector c is fed into the model's prediction layer
- Produces the answer to the clinical question

Main Symptom = Headache

Question 7?

Do attention mechanisms allow a model to give different importance to various parts of the input??

Thank you for your attention!

Key takeaway: The math behind these AI models is nothing new, just familiar concepts applied in clever ways.

Thank you for your attention!

Key takeaway: The math behind these AI models is nothing new, just familiar concepts applied in clever ways.

Question 8: *After this lecture, do neural networks feel less like a black box?*

Any questions?

Thanks to my co-authors: **Sarat Moka** and **Yoni Nazarathy**

1. Liquet, B., Moka, S., & Nazarathy, Y. (2024). *Mathematical Engineering of Deep Learning, 1st Edition*. Available online: https://deeplearningmath.org

2. Liquet, B., Moka, S., Nazarathy, Y. (2024). Book chapter. *Navigating Mathematical Basics: A Primer for Deep Learning in Science* Available online: https://deeplearningmath.org

Question 9: *Could you provide your email for the practical? We'll send you important information there.*