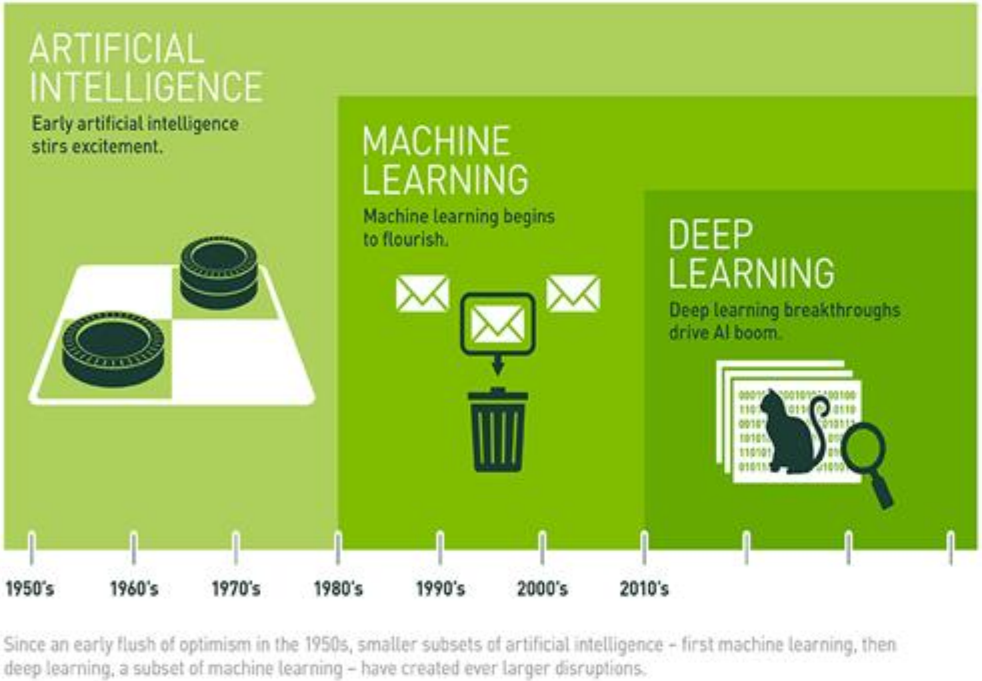
Deep Learning

Lecture 1
Neural networks

Giovanni Chierchia

Lecture 1 – Table of content

- **Introduction**
- **Review**
  - Supervised learning
  - Linear regression
  - Logistic regression
- **Neural networks**
  - Fully-connected layers
  - Loss function
  - Code Example



Introduction

Syllabus

Organization

What you should already know

- **Calculus & linear algebra**
  - Functions, gradients, matrices, vectors, …
- **Programming**
  - *Python & NumPy library (or MATLAB)*
- **Basics of machine learning**
  - *Linear regression, logistic regression, overfitting*

Syllabus

- **Lectures (8h)**
  - **(2h)** *Neural networks*
  - **(2h)** *Training + Best practices*
  - **(2h)** *Convolutional networks*
  - **(2h)** *Representation learning*
- **Labs (≥ 16h)**
  - *Deep learning project*
  - *Groups of 2-3 students*
- **Evaluation**
  - **(1h)** *MCQ*
  - **(3h)** *Oral presentation*

Last day of the course

Logistics

- **Course material**
  - *https://esiee.blackboard.com*

- **Grading**
  - ***MCQ*** → *50% of final mark*
  - ***Project*** → *50% of final mark*

- **Textbooks**
  - *G. James, Witten, Hastie, Tibshirani.* ***An Introduction to Statistical Learning.*** *Springer, 2017.*
  - *J. Watt, R. Borhani, A. Katsaggelos.* ***Machine Learning Refined.*** *Cambridge Univ. Press, 2016.*
  - *F. Chollet.* ***Deep Learning with Python.*** *Manning, 2017.*

About the project

- **Step 1 → Group Creation**
  - Form a team of 3 students *(+/-1 admitted if well justified)*
  - Let us know the team members by the end of the first lab
- **Step 2 → Project Selection**
  - Define the goal of your project
    - ***Option 1 → Build a neural network from scratch*** *(Available Online)*
    - ***Option 2 → Choose your own subject*** *(classification of real images, object detection, motion tracking, visual odometry, 3D reconstruction, …)*
  - Let us know your choice by the end of the second lab
- **Step 3 → Oral Presentation**
  - Prepare and give a presentation on the last day of class

Supervised learning

Fundamental hypothesis

Inductive bias

Learning

Context

- What is **machine learning** ?
  - *The ability of computers to learn without being explicitly programmed*

- There are several types of learning
  - ***Supervised*** → *Teach the computer how to do something*
  - ***Unsupervised*** → *Let the computer learn how to do something*
  - ***Reinforcement*** → *Allow the computer automate decision-making*

- Course objectives
  - *Study of neural networks for supervised learning*
  - *Special emphasis on computer vision*

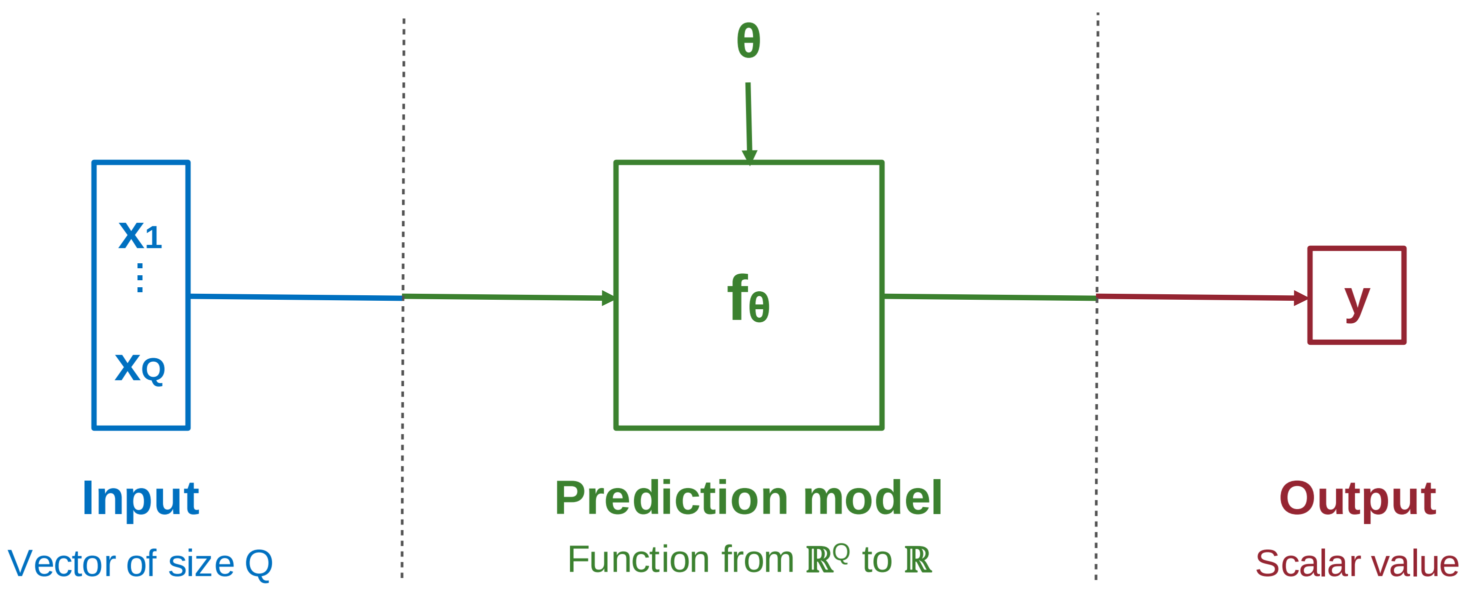
Training data

- **Fundamental hypothesis**
  - *Our goal is to predict an output from an input*
  - *We are given a dataset of input-output examples*
  - *We know there is a relationship between the input and the output*

	Input feature 1	*Input feature 2*	*Input feature 3*	*Input feature 4*	*Output*
	Size ($feet^2$)	**Number of bedrooms**	**Number of floors**	**Age of home (years)**	**Price ($1000)**
$(x^{(1)},y^{(1)})$ = *example 1*	$x_1^{(1)} = 2104$	$x_2^{(1)} = 5$	$x_3^{(1)} = 1$	$x_4^{(1)} = 45$	$y^{(1)} = 460$
$(x^{(2)},y^{(2)})$ = *example 2*	$x_1^{(2)} = 1416$	$x_2^{(2)} = 3$	$x_3^{(2)} = 2$	$x_4^{(2)} = 40$	$y^{(2)} = 232$
$(x^{(3)},y^{(3)})$ = *example 3*	$x_1^{(3)} = 1534$	$x_2^{(3)} = 3$	$x_3^{(3)} = 2$	$x_4^{(3)} = 30$	$y^{(3)} = 315$
$(x^{(4)},y^{(4)})$ = *example 4*	$x_1^{(4)} = 852$	$x_2^{(4)} = 2$	$x_3^{(4)} = 1$	$x_4^{(4)} = 36$	$y^{(4)} = 178$
	…	…	…	…	…

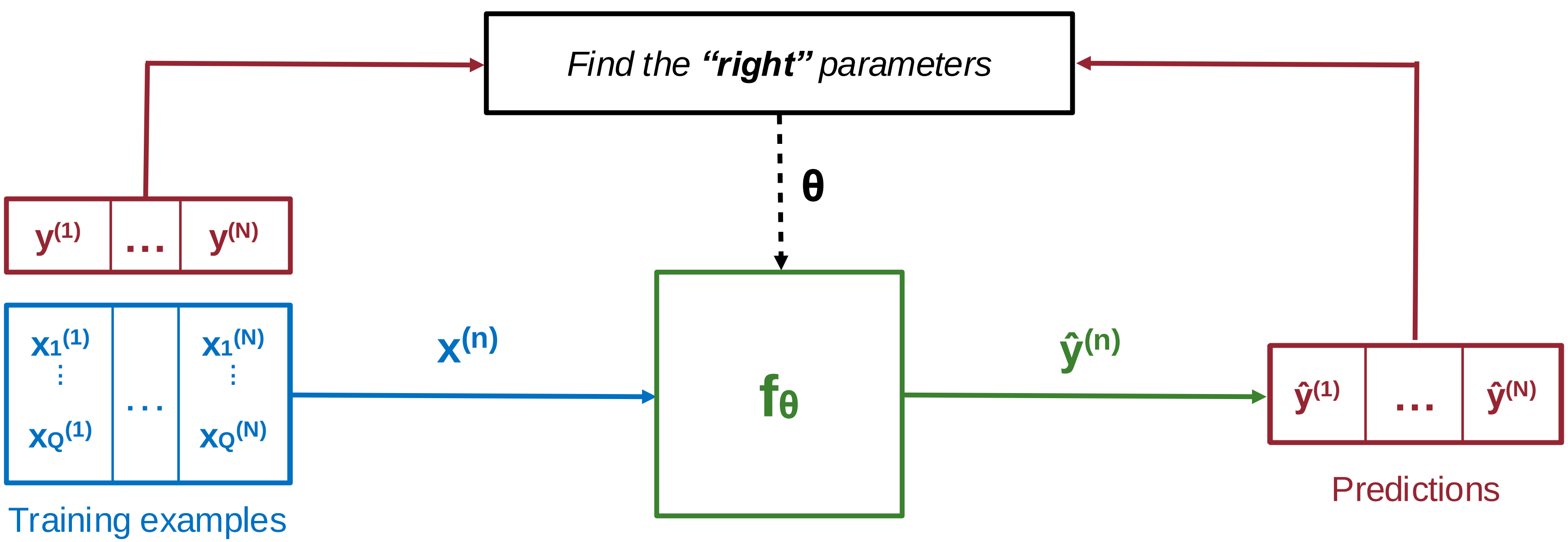
Prediction model

- **Generalization by inductive bias**
  - *We are interested in predicting the output for new unseen inputs*
  - *To do so, we use a parametric model $\boldsymbol{f_\theta}$ (where $\boldsymbol{\theta}$ is a vector of parameters)*



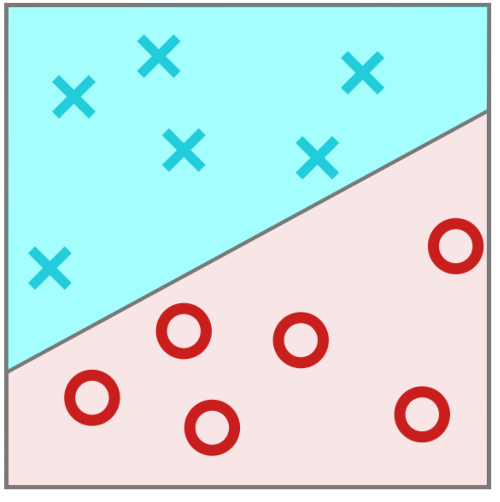
Training process

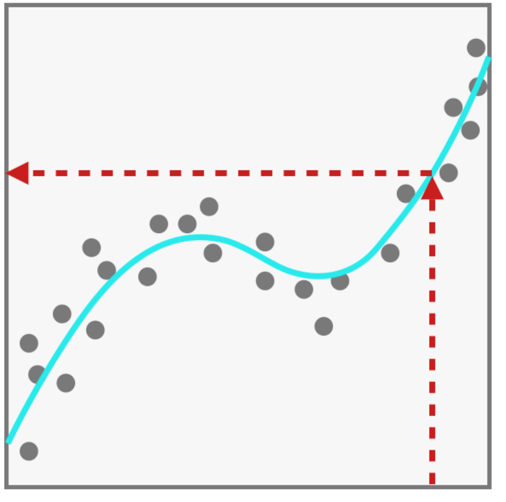
- **Learning**
  - *Our goal is to learn the prediction model* f_θ *from training data*
  - *This amounts to finding the "right values" for parameters* θ



Supervised learning

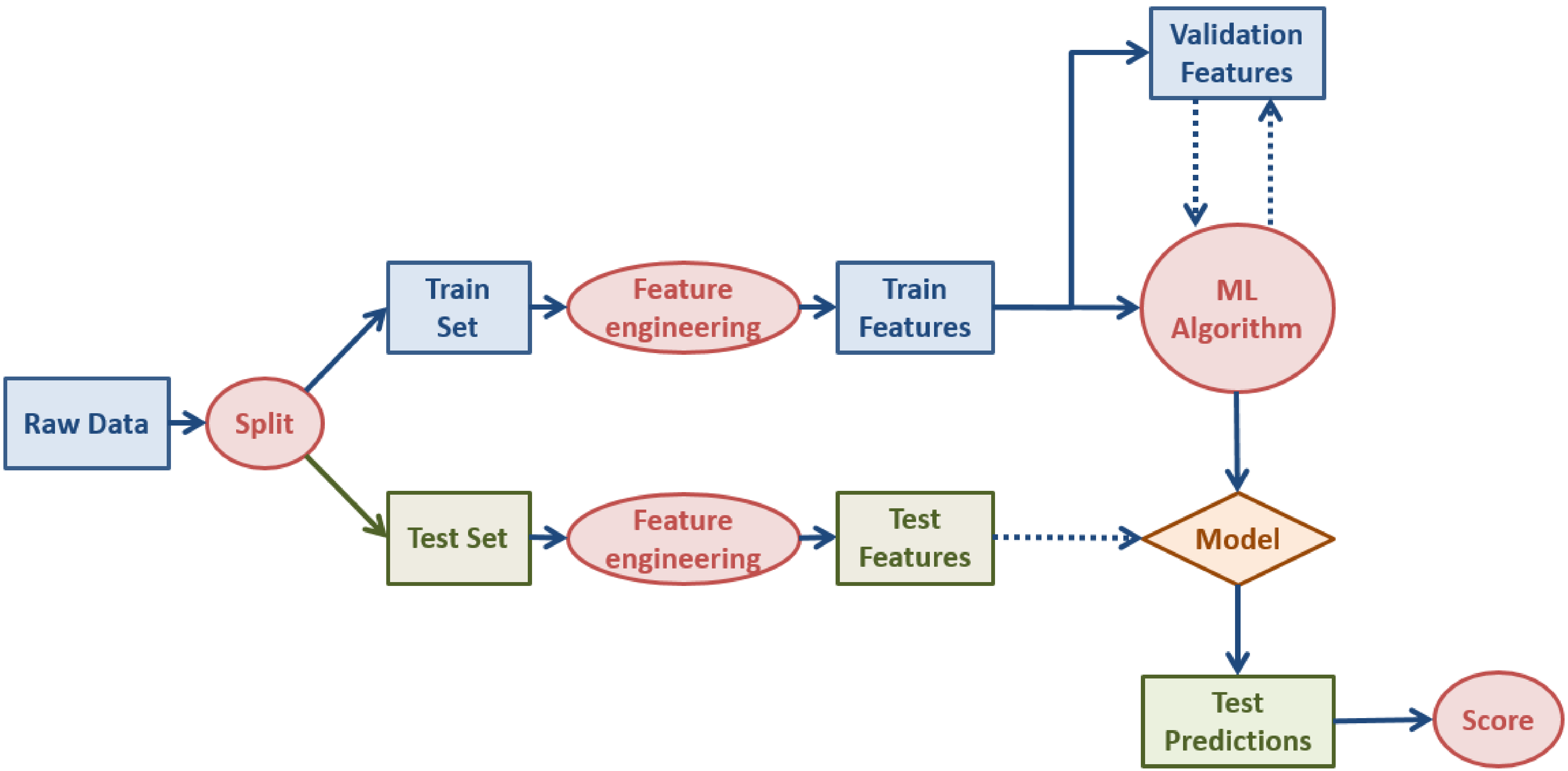
- **Two types of problems**
  - ***Regression*** ➔ *Learning how to predict a **continuous** output*
  - ***Classification*** ➔ *Learning how to predict a **discrete** output*





The big picture

- **Machine learning pipeline**



Linear regression

Training data

Prediction model

Cost function

Problem definition

- We are interested in understanding the relationship **f** between an input vector and a **continuous** output

$$f : \mathbb{R}^Q \mapsto \mathbb{R}$$

Input "x"
(vector of size Q)

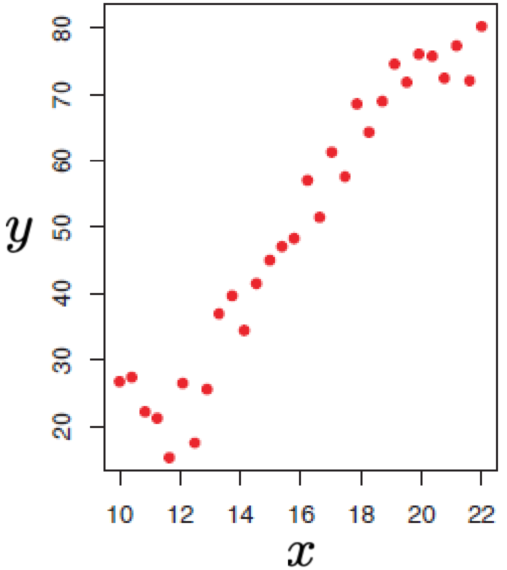
Output "y"
(continuous value)

Training data

- We are given a set of **input-output examples**

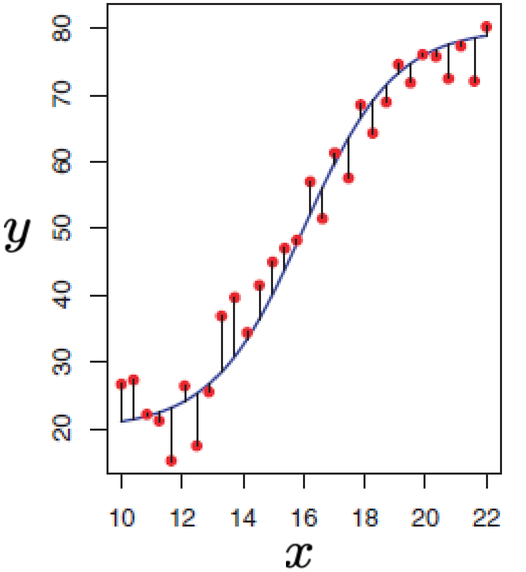
$$\left(\mathbf{x}^{(n)}, y^{(n)}\right) \in \mathbb{R}^Q \times \mathbb{R}$$

$$y^{(n)} = f\left(\mathbf{x}^{(n)}\right) + \varepsilon^{(n)}$$



$(Q = 1)$

***y*(*n*) is the approximated value of *f* at point *x*(*n*)**



Prediction model

- We represent **f** with a **parametric model**

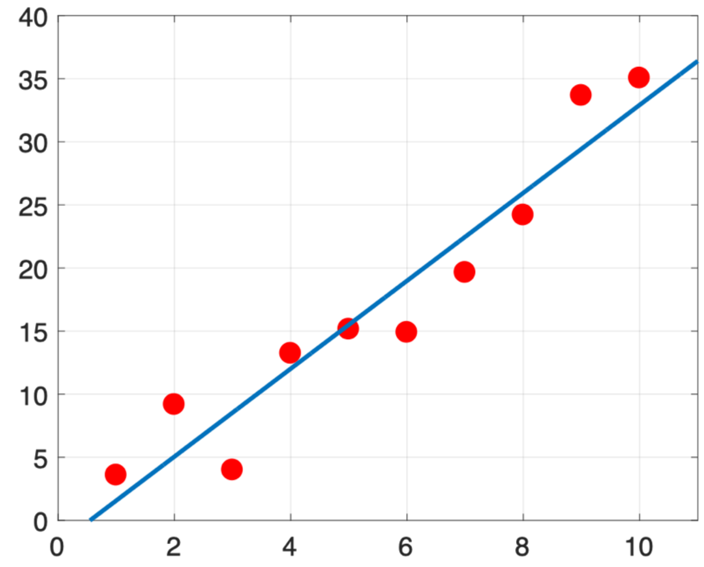
$$f(\mathrm{x}) \approx f_\theta(\mathrm{x})$$

where $\boldsymbol{\theta}$ *denotes a vector of parameters.*

- In particular, the model is **linear**

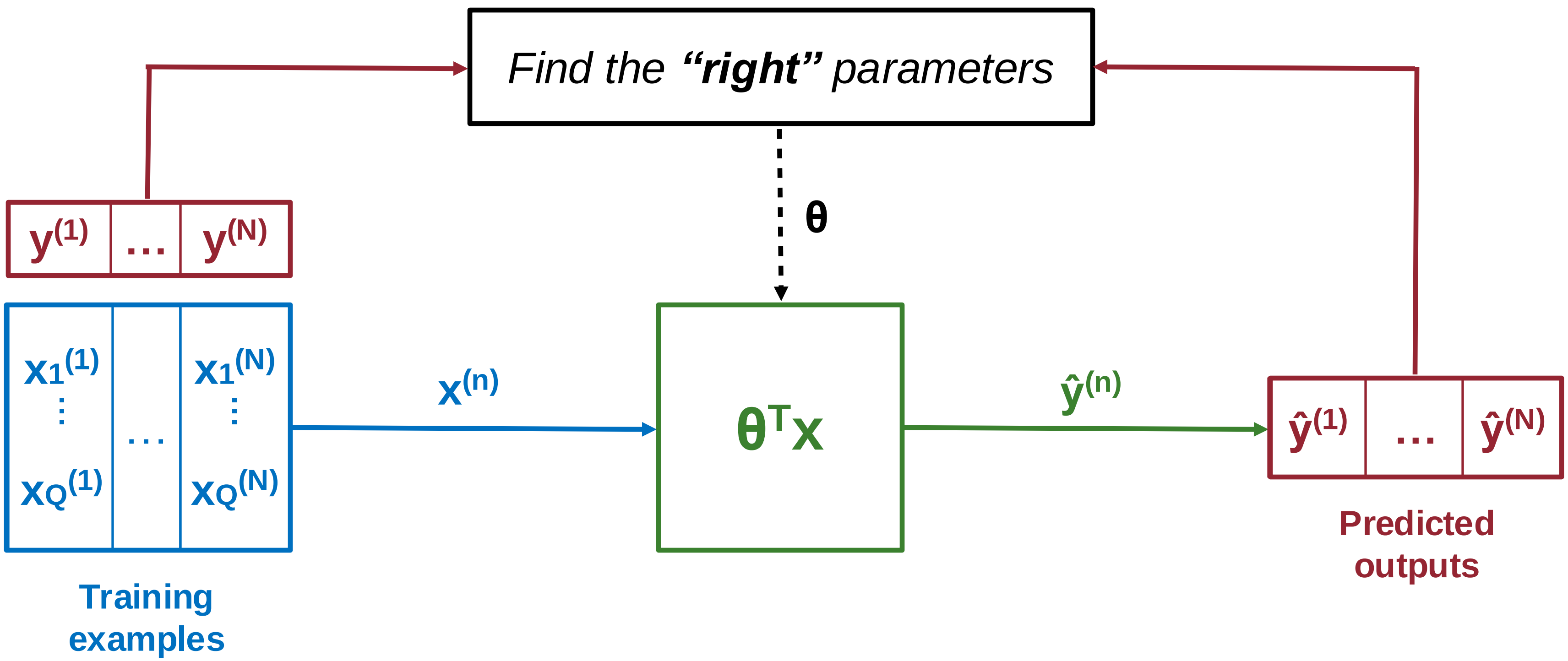
$$\begin{aligned} f_\theta(\mathrm{x}) &= \theta^\top \mathrm{x} \\ &= \theta_0 + \theta_1 x_1 + \cdots + \theta_Q x_Q \end{aligned}$$

with $\boldsymbol{\theta} = [\theta_0, \theta_1, \ldots, \theta_Q]^T$ *and* $\mathbf{x} = [1, x_1, \ldots, x_Q]^T$.



Cost function for regression (1/4)

- Our goal is to **learn** the prediction $\mathbf{f}_\theta$ from training data
  - *This amounts to finding the "right values" for parameters* $\boldsymbol{\theta}$

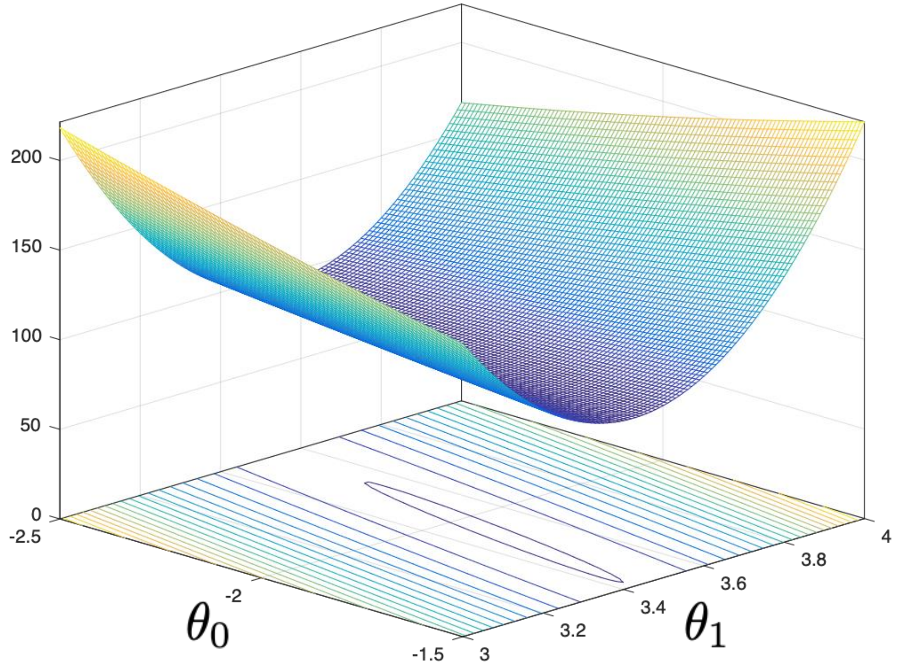


Cost function for regression (2/4)

- How to choose the "right values" for parameters **θ** ?
  - *We select **θ** such that the **model f_θ is fitted** to training data*

$$\widehat{\theta} = \arg\min_{\theta} \sum_{n=1}^{N} C\Big(f_\theta(\mathrm{x}^{(n)}), y^{(n)}\Big)$$

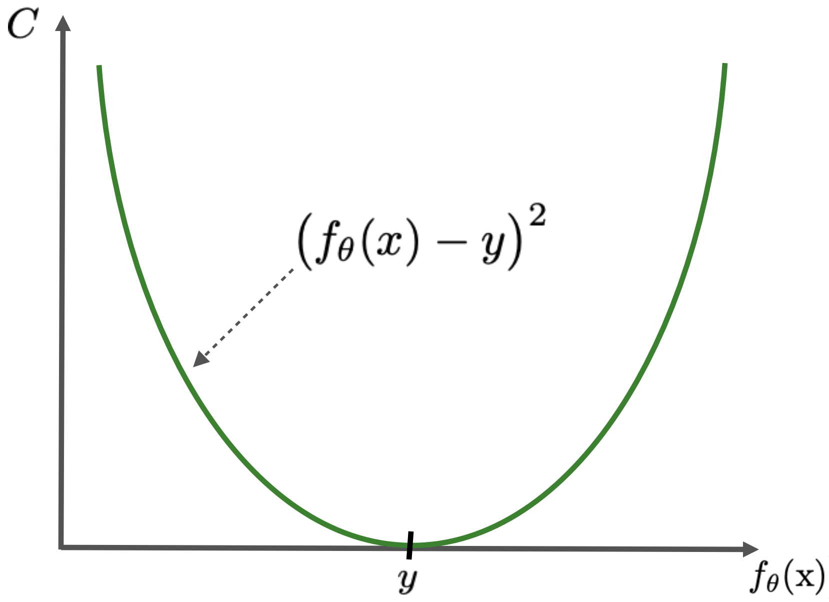
Cost function



Cost function for regression (3/4)

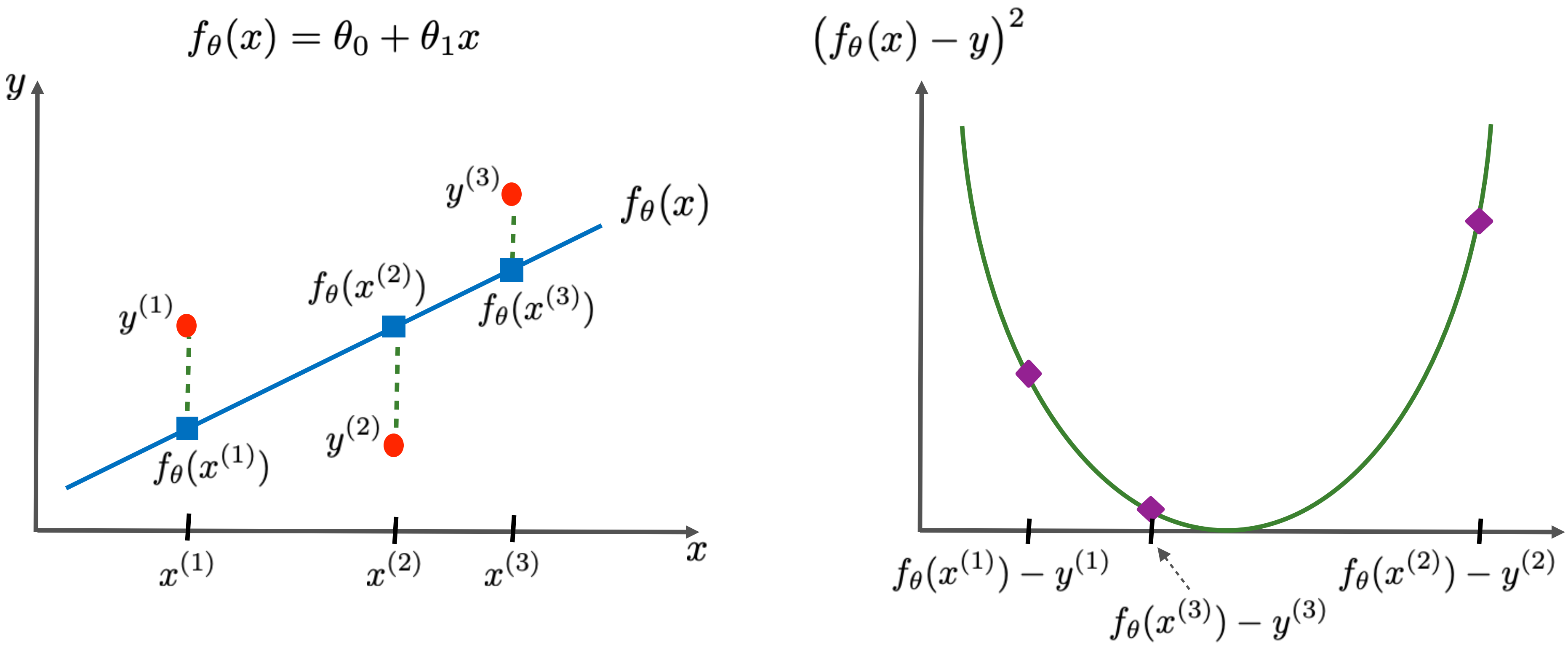
- How to measure the **fitting of f_θ** to the training data?
  - *for each example **(x, y)**, the prediction **f_θ(x)** must be close to **y***
  - *their distance is measured with the **squared cost function***

$$C\Big(f_\theta(\mathrm{x}), y\Big) = \big(f_\theta(\mathrm{x}) - y\big)^2$$



Cost function for regression (4/4)

- EXAMPLE. Linear regression with one feature (Q=1)



What we have seen so far…

- Key ingredients of **linear regression**
  - *Training data* → *Vector inputs — Continuous outputs*

$$\left(\mathrm{x}^{(n)}, y^{(n)}\right) \in \mathbb{R}^Q \times \mathbb{R} \qquad n = 1, \dots, N$$

  - *Prediction* → *Linear model*

$$f_\theta(\mathrm{x}) = \theta^\top \mathrm{x}$$

  - *Learning* → *Squared error function*

$$J(\theta) = \sum_{n=1}^{N} \left(f_\theta(\mathrm{x}^{(n)}) - y^{(n)} \right)^2$$

Logistic regression

Training data

Prediction model

Cost function

Decision boundary

Binary classification (1/2)

- Let's focus on classification with **two classes**
  - *The response variable **y** is a binary value*

$$y \in \{0, 1\}$$

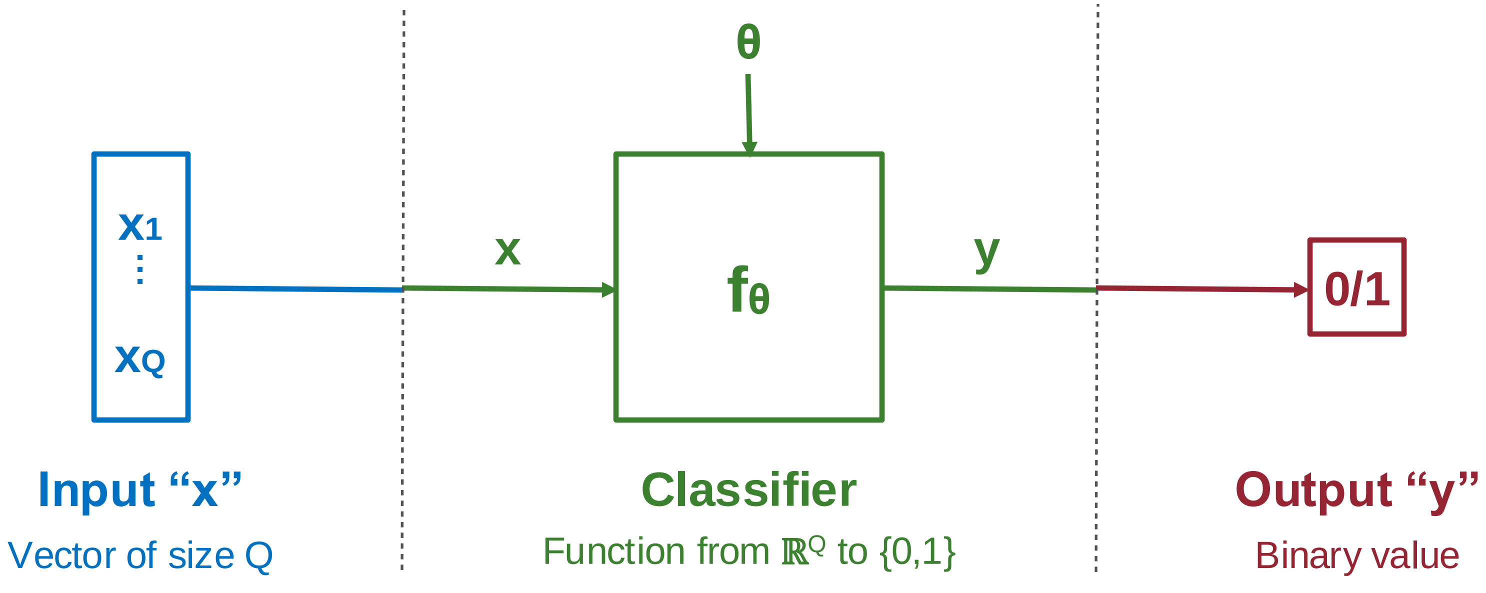
positive class (1)

negative class (0)

- Examples
  - *email → spam / not spam ?*
  - *online transaction → fraudulent (yes / no) ?*
  - *tumor → malignant / benign ?*

Binary classification (2/2)

- Our goal is to **predict** the class **y** from an observation **x**
  - *To do so, we use a parametric model* $\boldsymbol{f_\theta}$ *…*
  - *… where* $\boldsymbol{\theta = [\theta_0, \theta_1, \ldots, \theta_Q]^T}$ *is a vector of parameters to be estimated.*



Logistic model (1/3)

- How to **predict** a binary response variable ?
  - *Actually, we don't directly predict a binary outcome*
  - *Instead, we predict the **probability** that **y = 1** given **x***

$$f_\theta(\mathrm{x}) \approx \mathsf{P}(y = 1 \,|\, \mathrm{x})$$

  - *To do so, we use a **bounded** linear model*

$$f_\theta(\mathrm{x}) = g\big(\theta_0 + \theta_1 x_1 + \cdots + \theta_Q x_Q\big)$$

  - *where **g** is the **logistic function***

$$g(z) = \frac{1}{1 + e^{-z}}$$

Logistic model (2/3)

- The **logistic function** maps a real value between **0** and **1**
  - *Hence, it can be regarded as a probability.*

$$g(z) = \frac{1}{1+e^{-z}}$$

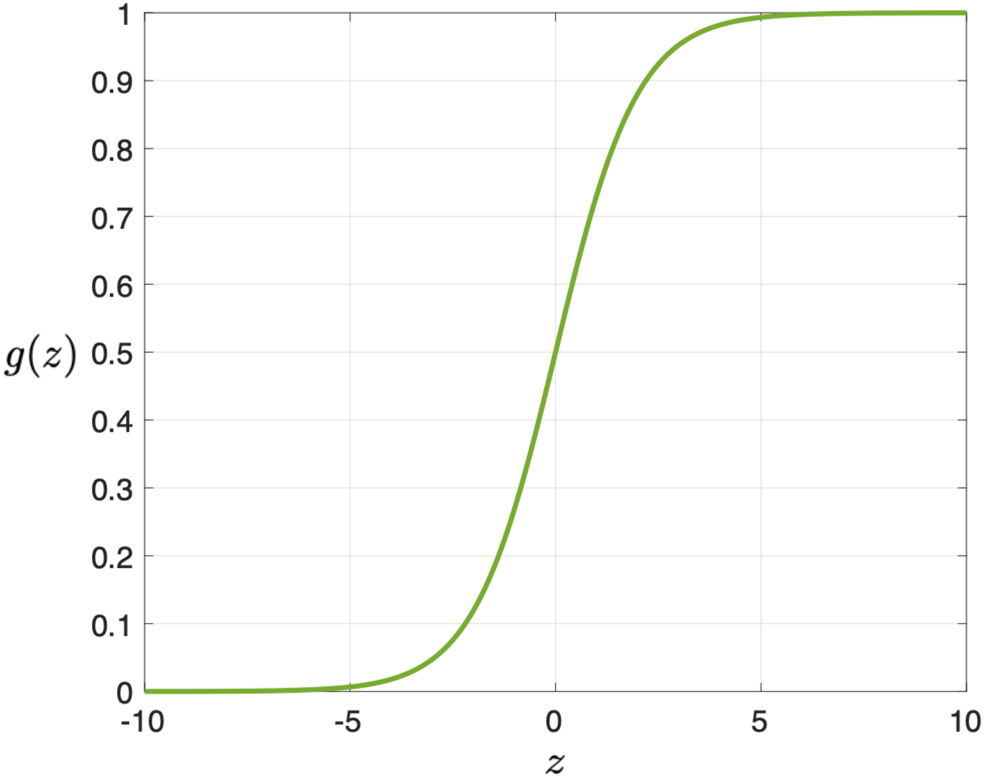
Properties

$$g(z) = \frac{e^z}{1+e^z}$$

$$g(-z) = 1 - g(z)$$

$$g'(z) = g(z)\big(1-g(z)\big)$$

$$g^{-1}(t) = \log\left(\frac{t}{1-t}\right)$$



Logistic model (3/3)

- Logistic model will be compactly written as

$$f_\theta(\mathrm{x}) = \frac{1}{1+\exp(-\theta^\top \mathrm{x})}$$

  - *NOTE 1:* ***x*** *and* ***θ*** *are column vectors of size Q+1 (with* $\boldsymbol{x_0 = 1}$*)*

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_Q \end{bmatrix} \qquad \mathrm{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_Q \end{bmatrix}$$

  - *NOTE 2:* *the linear combination of* ***x*** *and* ***θ*** *is a scalar product*

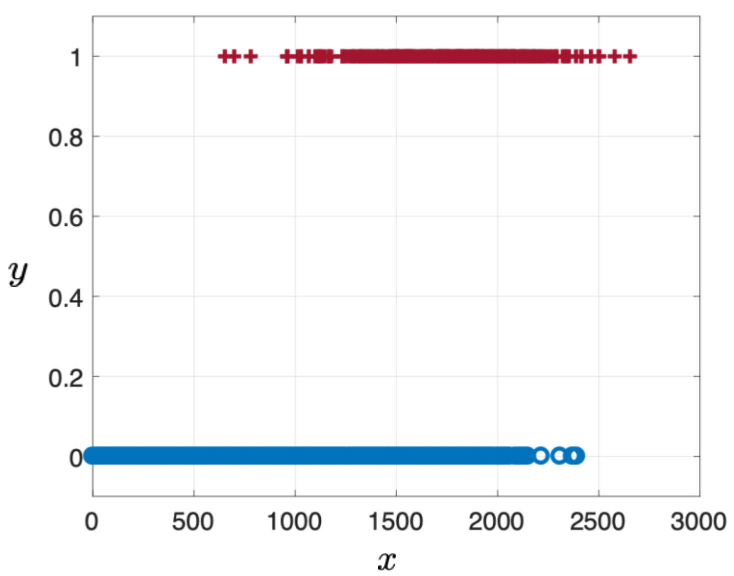
$$\theta^\top \mathrm{x} = [\theta_0\ \theta_1\ \dots\ \theta_Q] \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_Q \end{bmatrix} = \theta_0 + \theta_1 x_1 + \cdots + \theta_Q x_Q$$

Training data

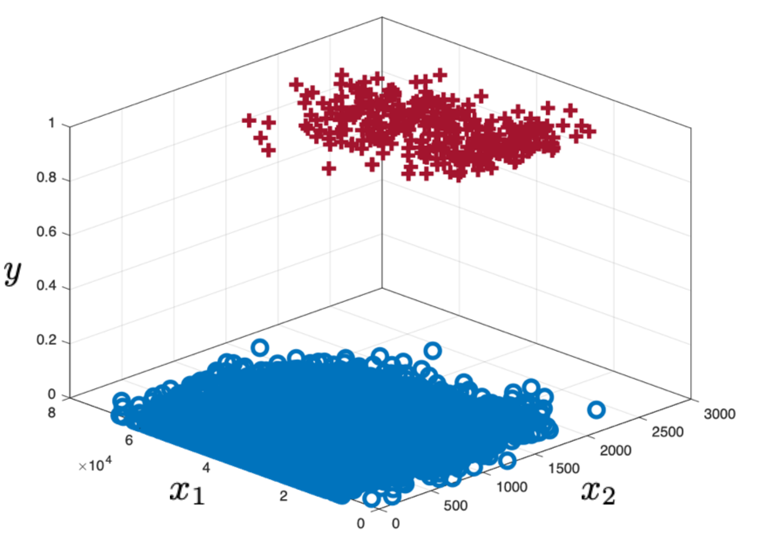
- We are given a set of **input-output examples**

$$(\mathrm{x}^{(n)}, y^{(n)}) \in \mathbb{R}^Q \times \{0,1\} \qquad n = 1, \ldots, N$$

Binary classification (Q=1)

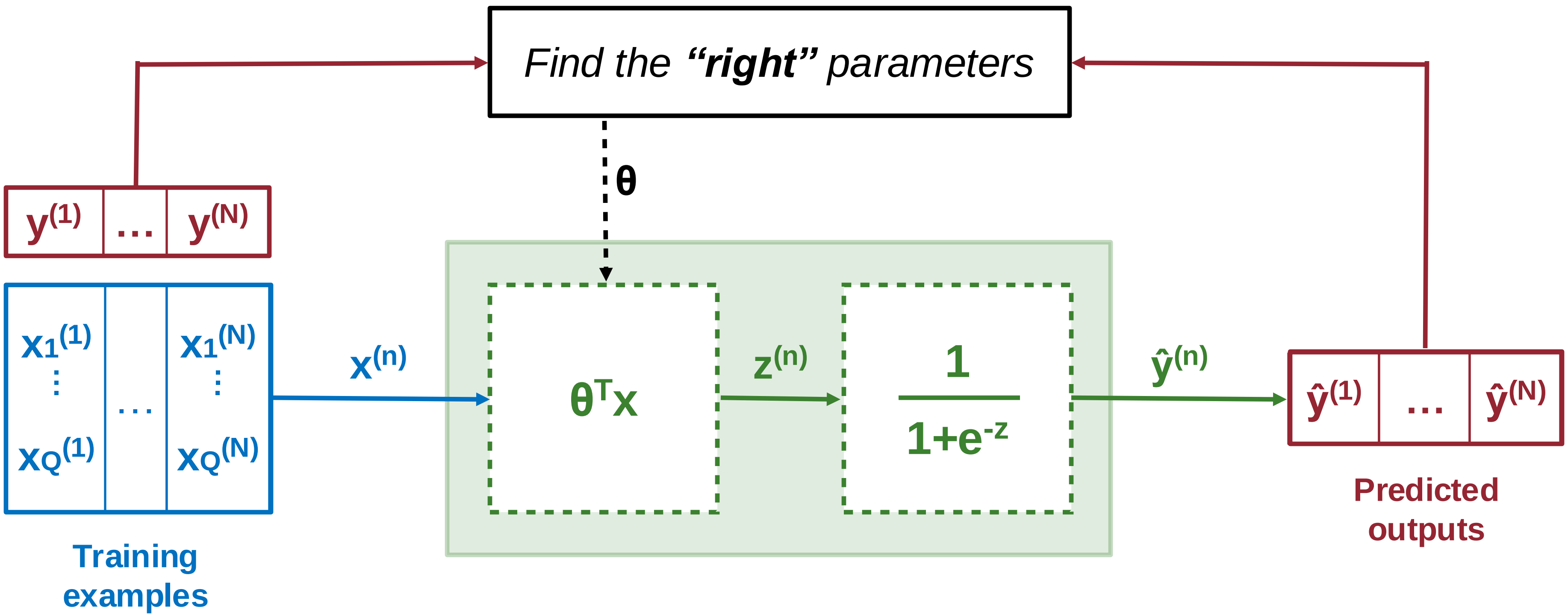


Binary classification (Q=2)



Learning (1/2)

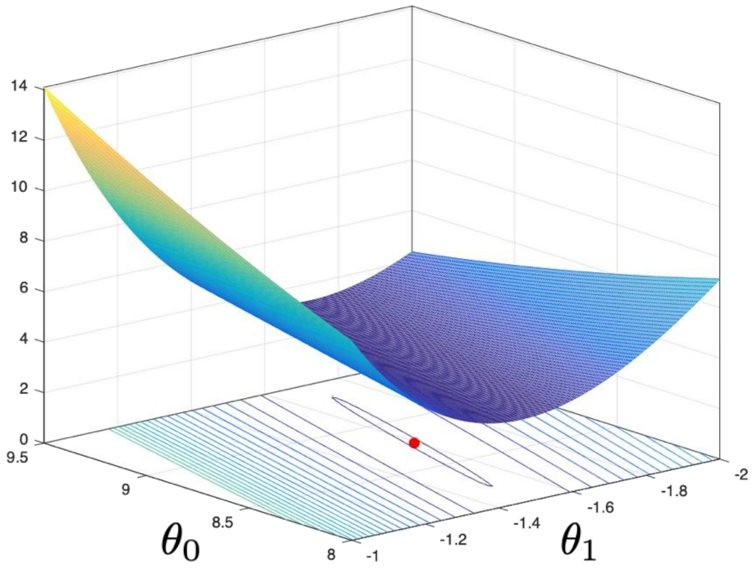
- Our goal is to **learn P(y=1|x)** from training data
  - *This amounts to finding the "right values" of* $\boldsymbol{\theta}$ *in the logistic model*

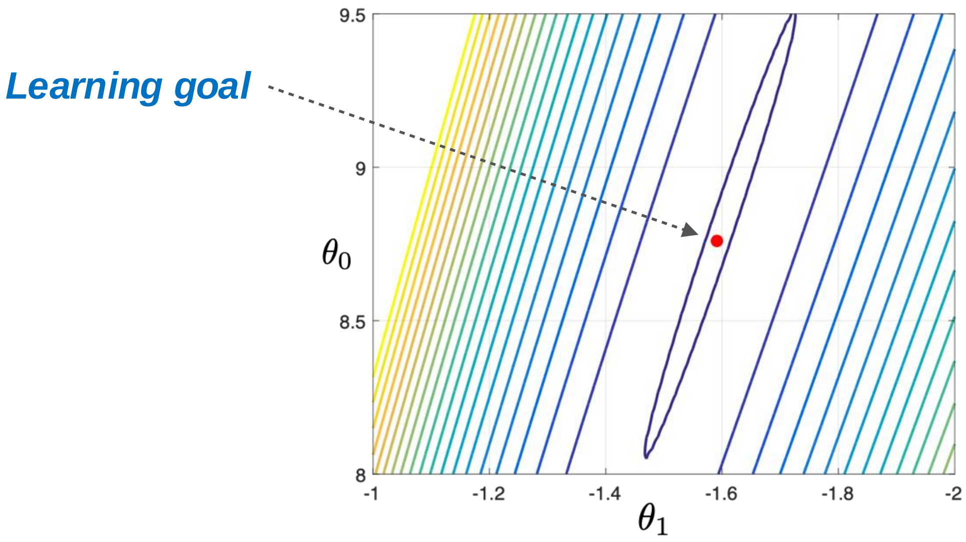


Learning (2/2)

- How to choose the "right values" for parameters **θ** ?
  - *We select **θ** such that the **model f_θ is fitted** to training data*

$$\widehat{\theta} = \arg\min_{\theta} \sum_{n=1}^{N} C\Big(f_\theta(\mathrm{x}^{(n)}), y^{(n)}\Big)$$

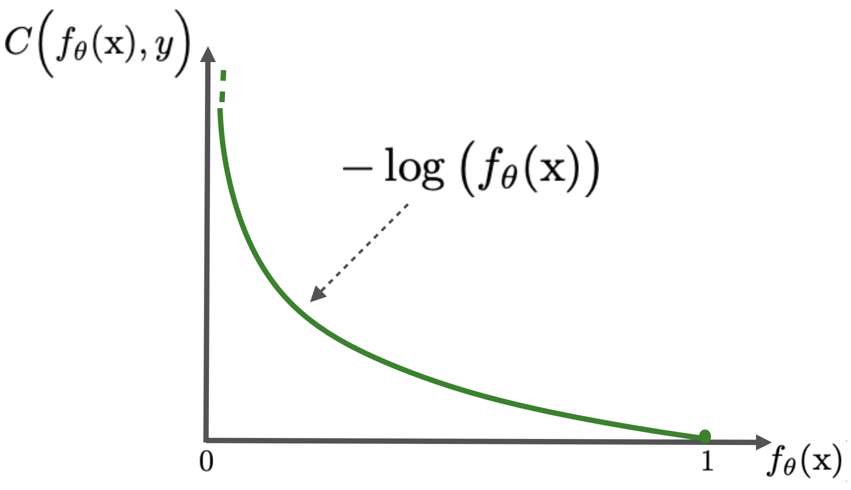




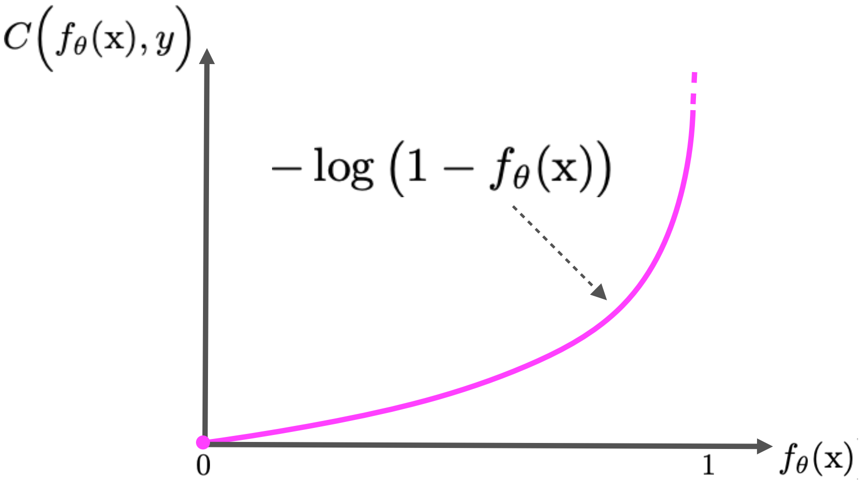
Cost function (1/2)

- How to measure the **fitting of f_θ** to the training data?
  - *for each example **(x, y)**, the prediction **f_θ(x)** must be close to **y***
  - *since **0 < f_θ(x) < 1**, the distance between **f_θ(x)** and **y** can be measured as*

distance to y = 1



distance to y = 0



Cost function (2/2)

- Data fitting is quantified by the **logarithm cost function**

$$C\Big(f_\theta(\mathrm{x}), y\Big) = \begin{cases} -\log\big(f_\theta(\mathrm{x})\big) & \text{if} \;\; y = 1 \\ -\log\big(1 - f_\theta(\mathrm{x})\big) & \text{if} \;\; y = 0 \end{cases}$$

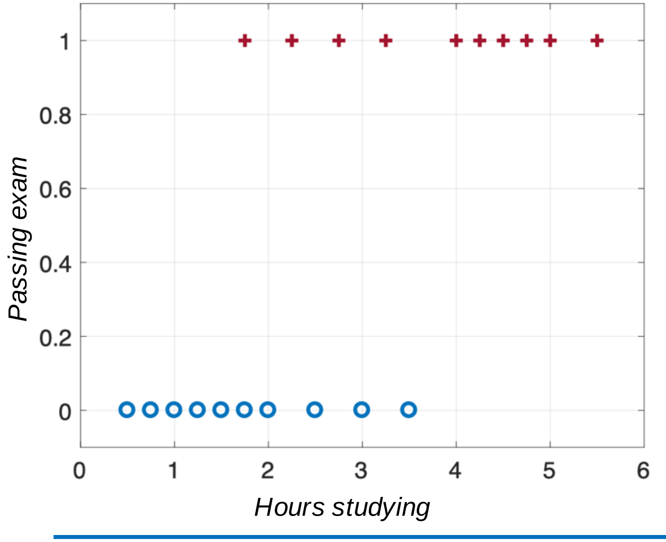
- which is exactly the anti-logarithm of **Bernoulli distribution**
  - *RECALL:* ***$f_\theta(x)$*** *is the probability that* ***y = 1***

$$\ell(y; \theta, \mathrm{x}) = \Big(f_\theta(\mathrm{x})\Big)^y \Big(1 - f_\theta(\mathrm{x})\Big)^{1-y}$$

Example (1/2)

- Suppose we wish to answer the following question
  - *A group of 20 students studied between 0 and 6 hours for an exam.*
  - *How does the number of hours spent studying affect the probability that the student will pass the exam?*

Training data



Prediction

Logistic model

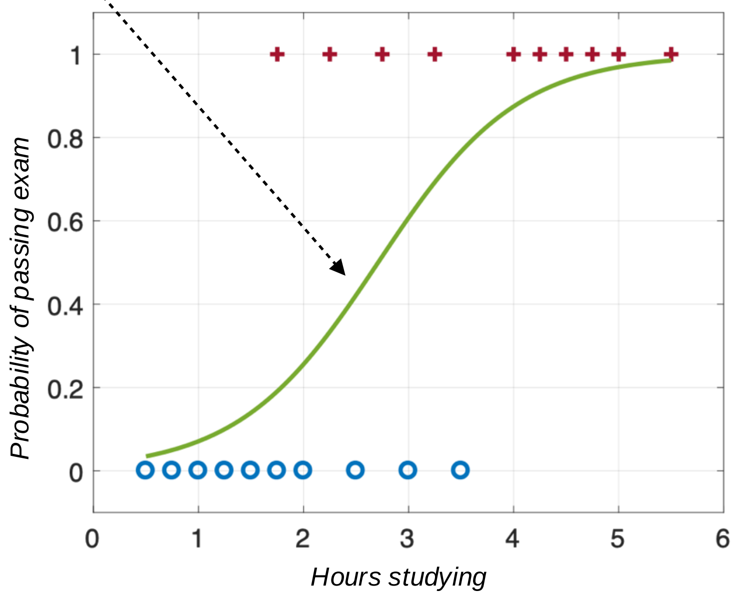
$$f_\theta(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$

Example (2/2)

- Learning yields the following parameters
  - *We will see later how to do this*

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} -4.07 \\ 1.50 \end{bmatrix} \quad \dashrightarrow \quad \text{Prob. of passing exam} = \frac{1}{1 + \exp(4.07 - 1.50 \times \text{Hours studying})}$$

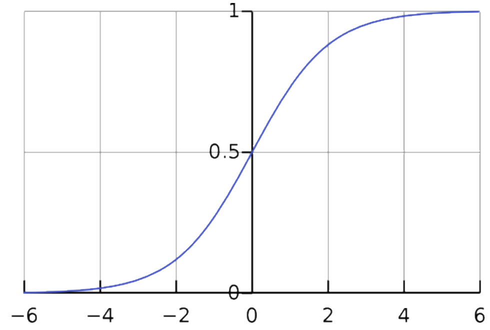
1	0.07
2	0.26
3	0.61
4	0.87
5	0.97
6	Compute it yourself !



Decision boundary (1/2)

- It is possible to show that

$$g(z) = \frac{1}{1+e^{-z}}$$



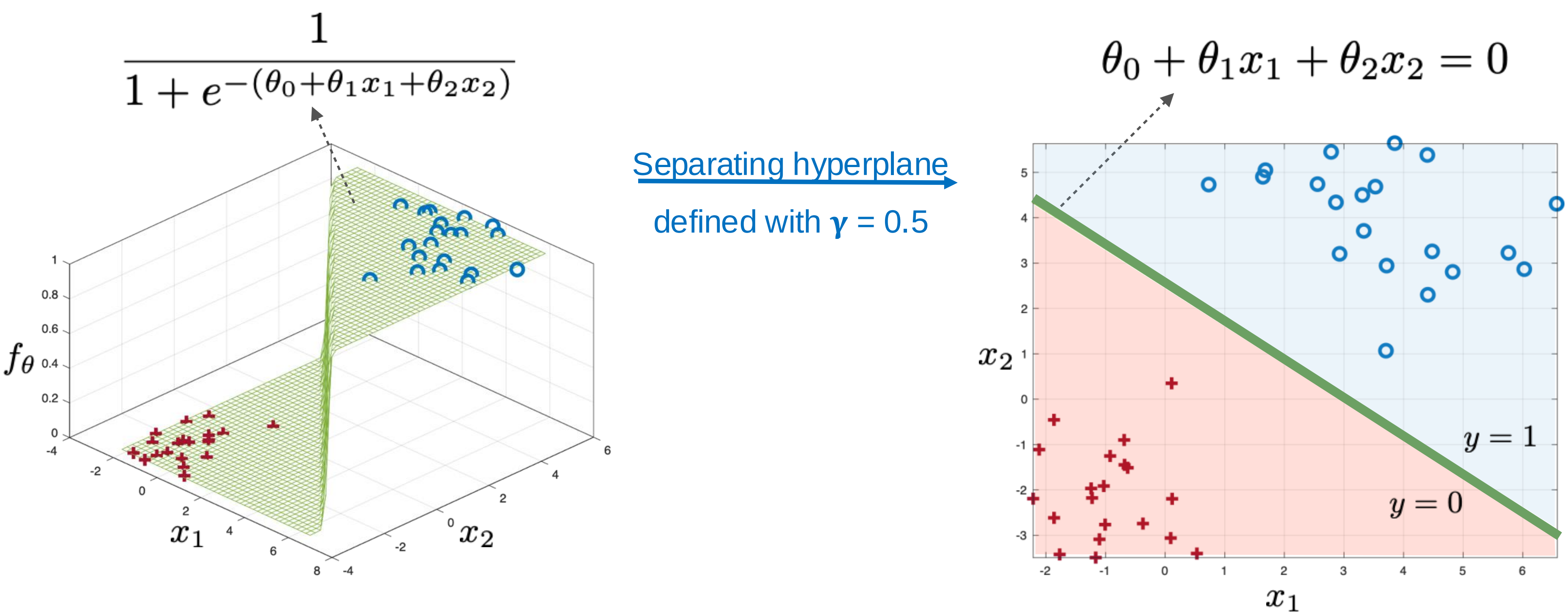
$$f_\theta(\mathrm{x}) = g(\theta^\top \mathrm{x}) \geq 0.5 \qquad \Leftrightarrow \qquad \theta^\top \mathrm{x} \geq 0$$
$$f_\theta(\mathrm{x}) = g(\theta^\top \mathrm{x}) < 0.5 \qquad \Leftrightarrow \qquad \theta^\top \mathrm{x} < 0$$

- Hence, thresholding by ***γ* = 0.5** is equivalent to

$$y_{\text{pred}} = \begin{cases} 1 & \text{if } \ \theta^\top \mathrm{x} \geq 0 \\ 0 & \text{if } \ \theta^\top \mathrm{x} < 0 \end{cases}$$

Decision boundary (2/2)

- Logistic regression is a **linear classifier**
  - *The feature space is split in two regions by a line*



Summary

- Key ingredients of **logistic regression**
  - *Training data → Vector inputs — Binary outputs*

$$\left(\mathrm{x}^{(n)}, y^{(n)}\right) \in \mathbb{R}^Q \times \{0,1\} \qquad n = 1, \dots, N$$

  - *Prediction → Logistic model*

$$f_\theta(\mathrm{x}) = \frac{1}{1 + e^{-\theta^\top \mathrm{x}}}$$

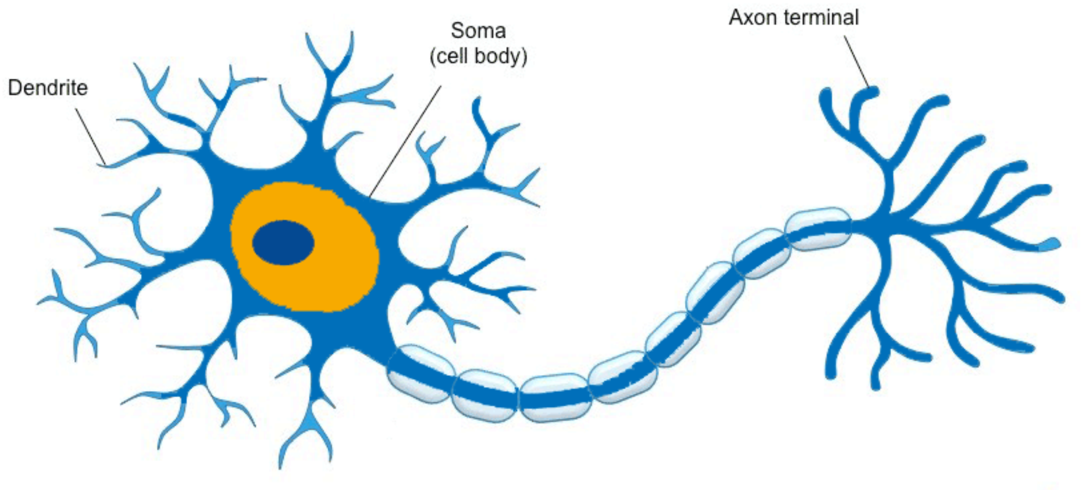
  - *Learning → Logarithmic cost function*

$$J(\theta) = \sum_{n=1}^{N} -y^{(n)} \log\left(f_\theta(\mathrm{x}^{(n)})\right) - (1 - y^{(n)}) \log\left(1 - f_\theta(\mathrm{x}^{(n)})\right)$$

Artificial neuron

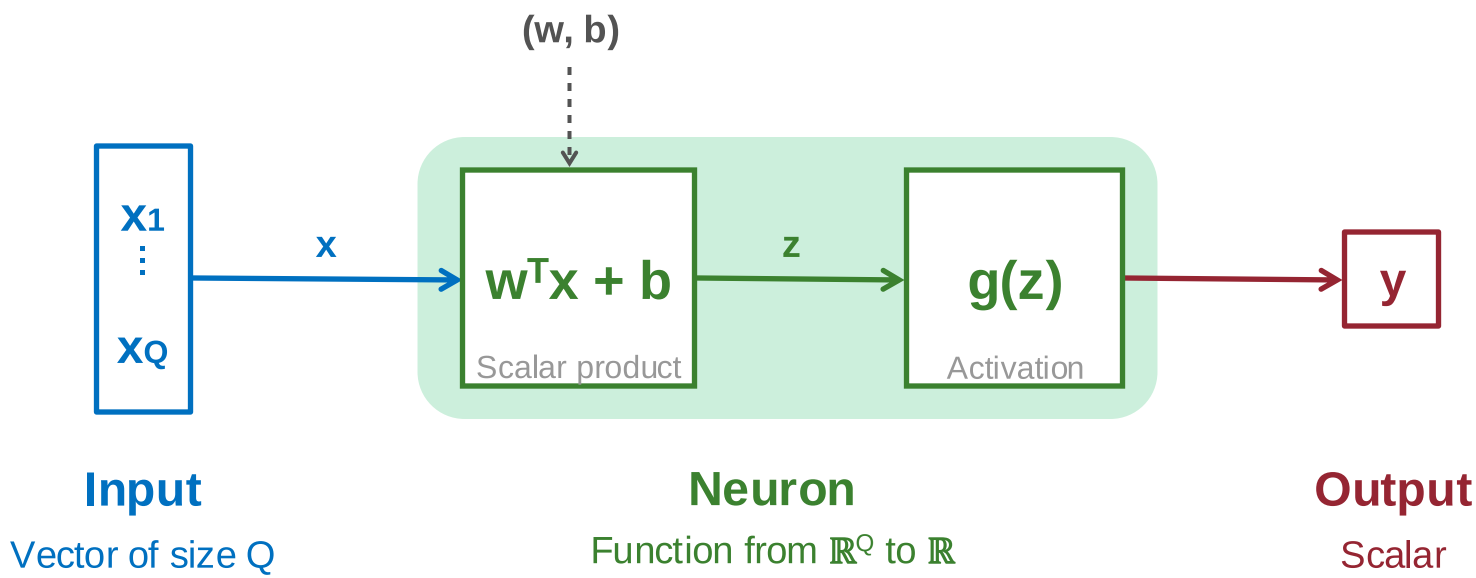
Artificial neuron (1/2)

- The **neuron** is the building block of neural networks
  - It is a single unit that inputs, processes and outputs information
  - It is part of a network formed by many interconnected neurons
  - The idea is **loosely** inspired from the nerve cells in the human brain



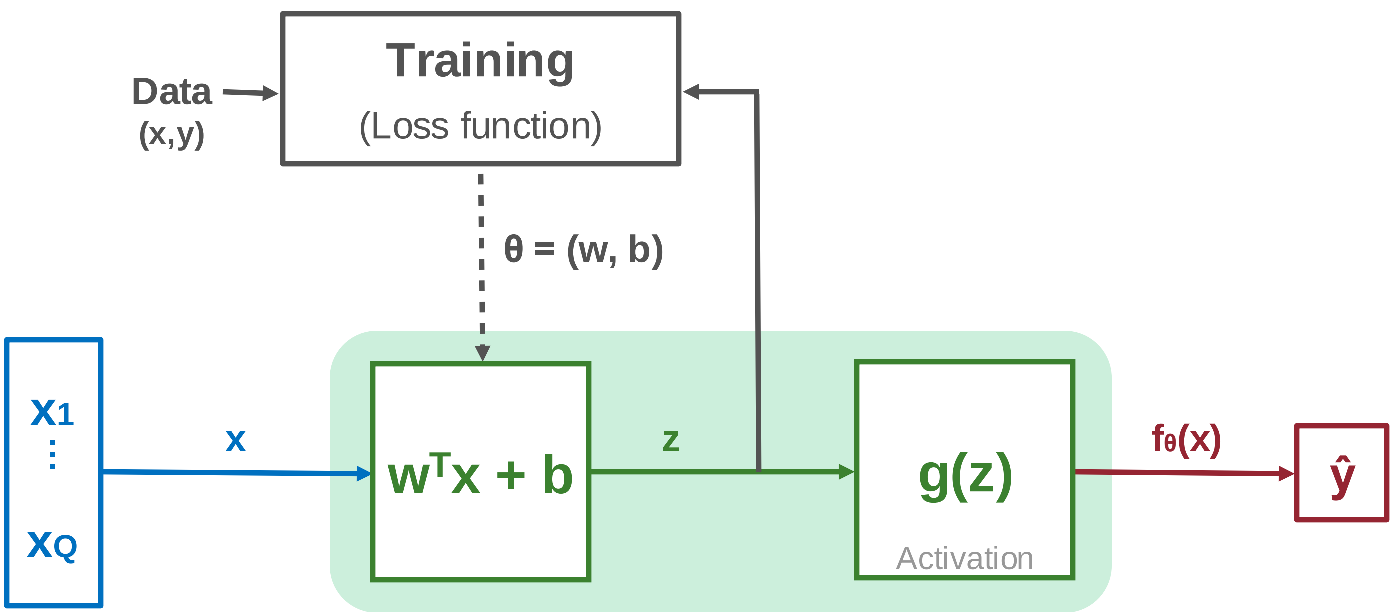
Artificial neuron (2/2)

- The **neuron** is a composition of two “simple” operations
  - **Scalar product ➜** Parameterized by **w** (vector) and **b** (scalar)
  - **Activation ➜** Function (usually nonlinear) without parameters



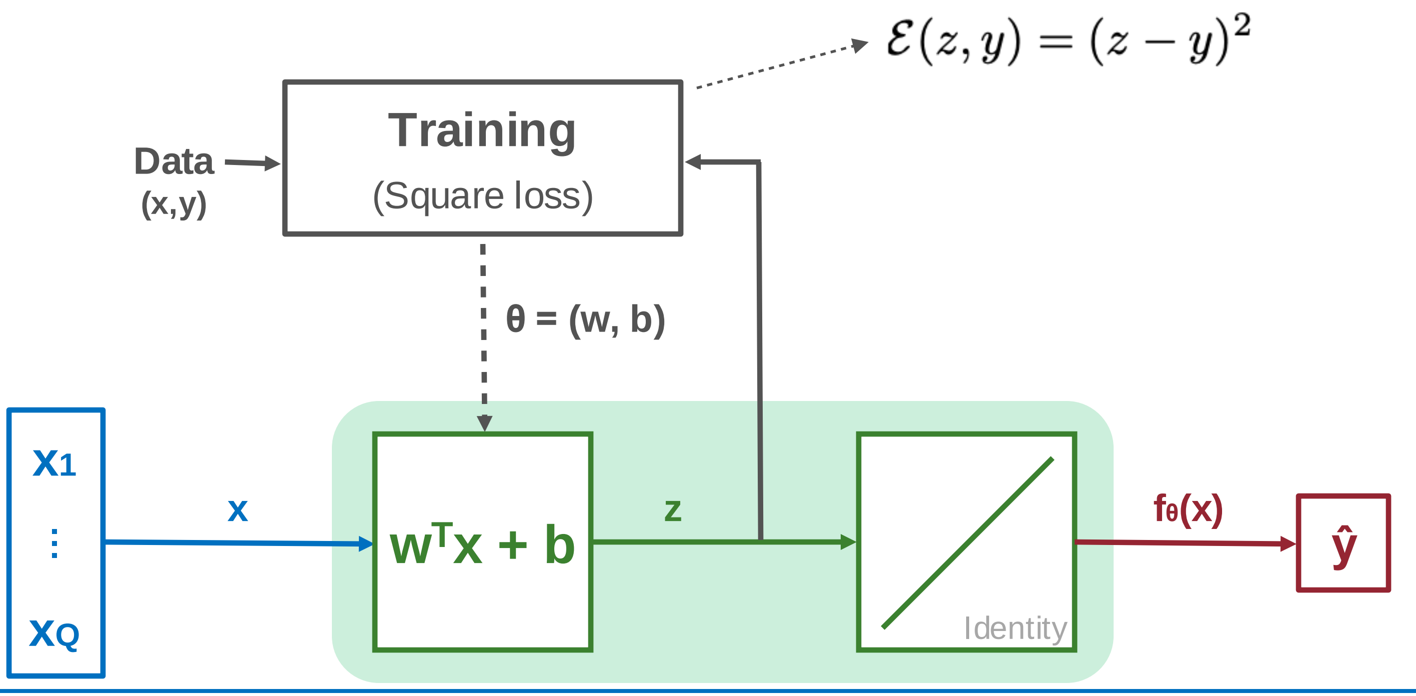
Linear models (1/4)

- **Linear models are neurons!**
  - **Activation** is adapted to the task (regression or classification)
  - **Training** uses a "loss function" adapted to the task



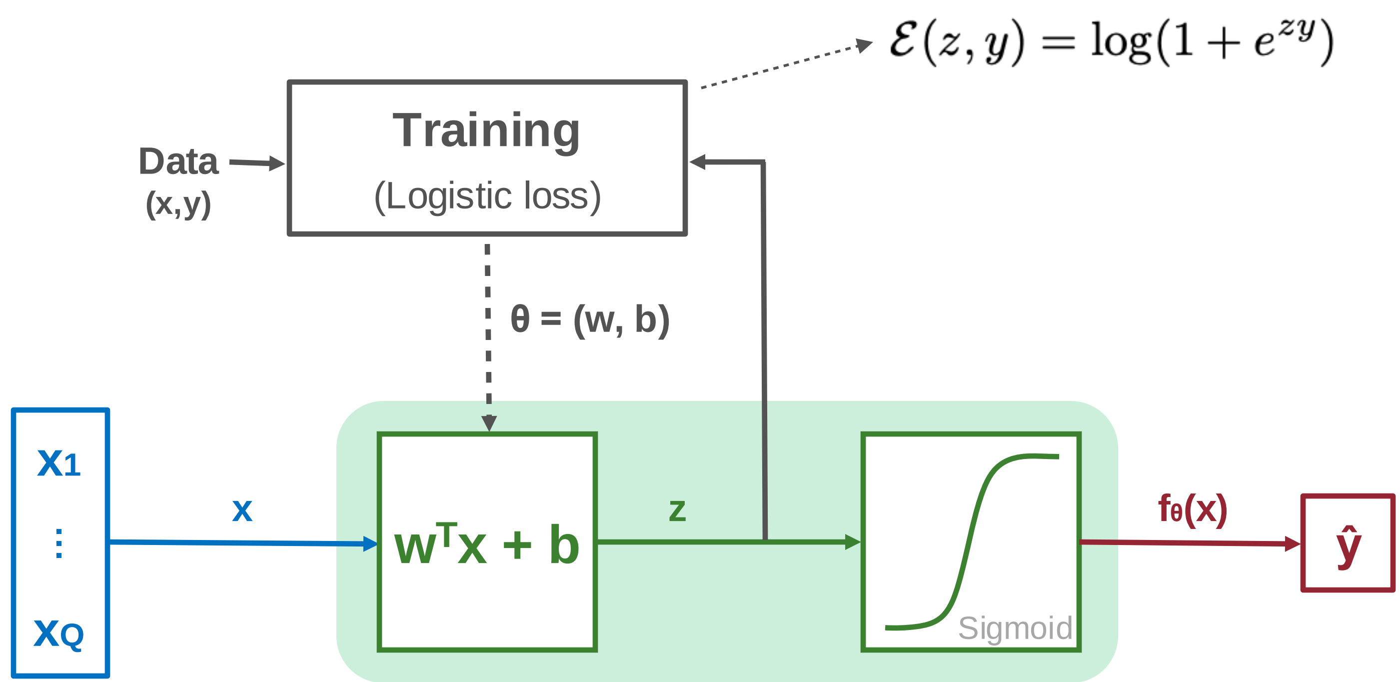
Linear models (2/4)

- **Linear regression**
  - **Activation →** None
  - **Training →** Square loss



Linear models (3/4)

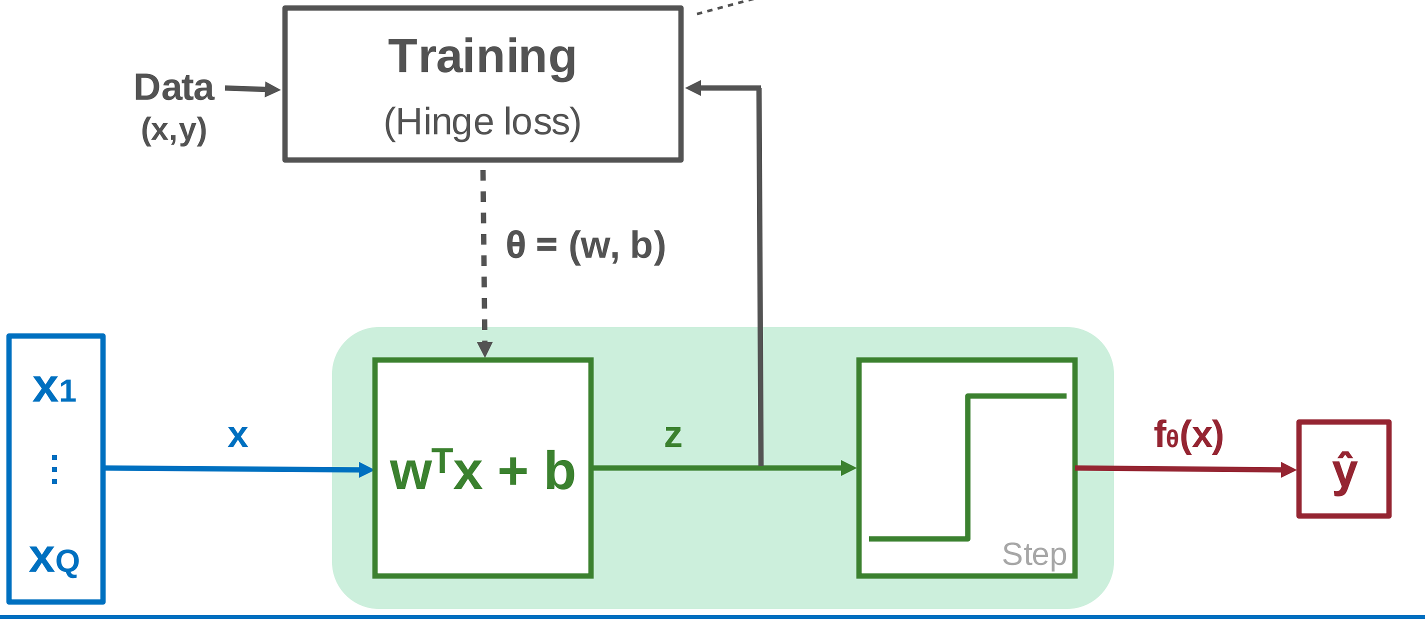
- **Logistic regression**
  - **Activation →** Sigmoid function
  - **Training →** Logistic loss



Linear models (4/4)

- **Support vector machine**
  - **Activation ➜** Step function
  - **Training ➜** Hinge loss

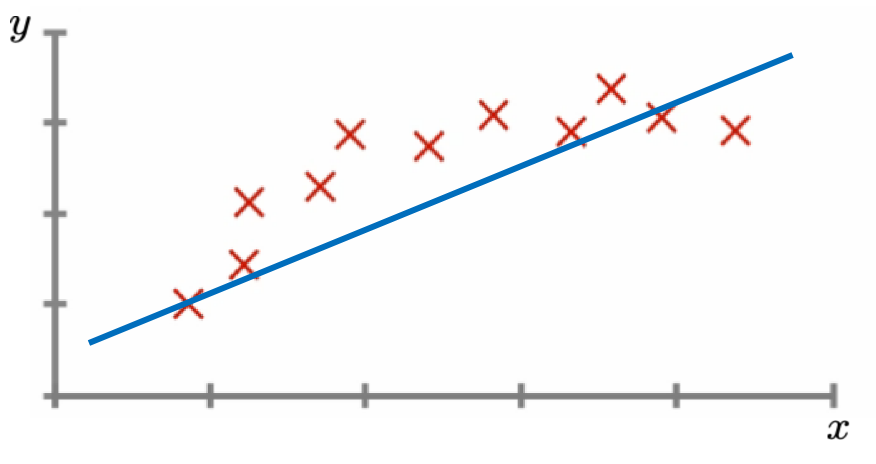
$$\mathcal{E}(z, y) = \max\{0, 1 + zy\}$$



Beyond linear models (1/3)

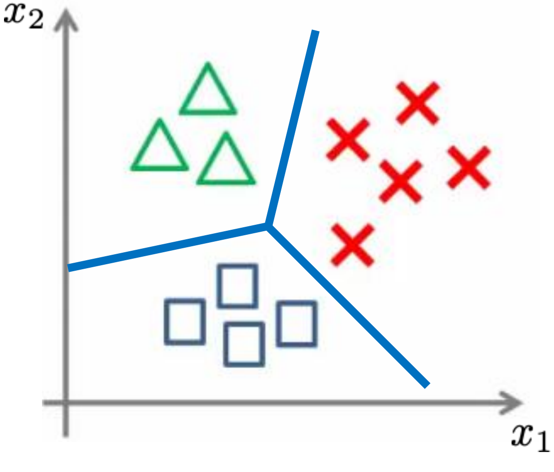
- Linear models have **low variance** and **high bias**
  - *Good choice when **Q >> N** (more input features than examples)*
  - *Prone to under-fitting when **Q << N** (large dataset)*

Regression



(Here, the input space is 1-dimensional)

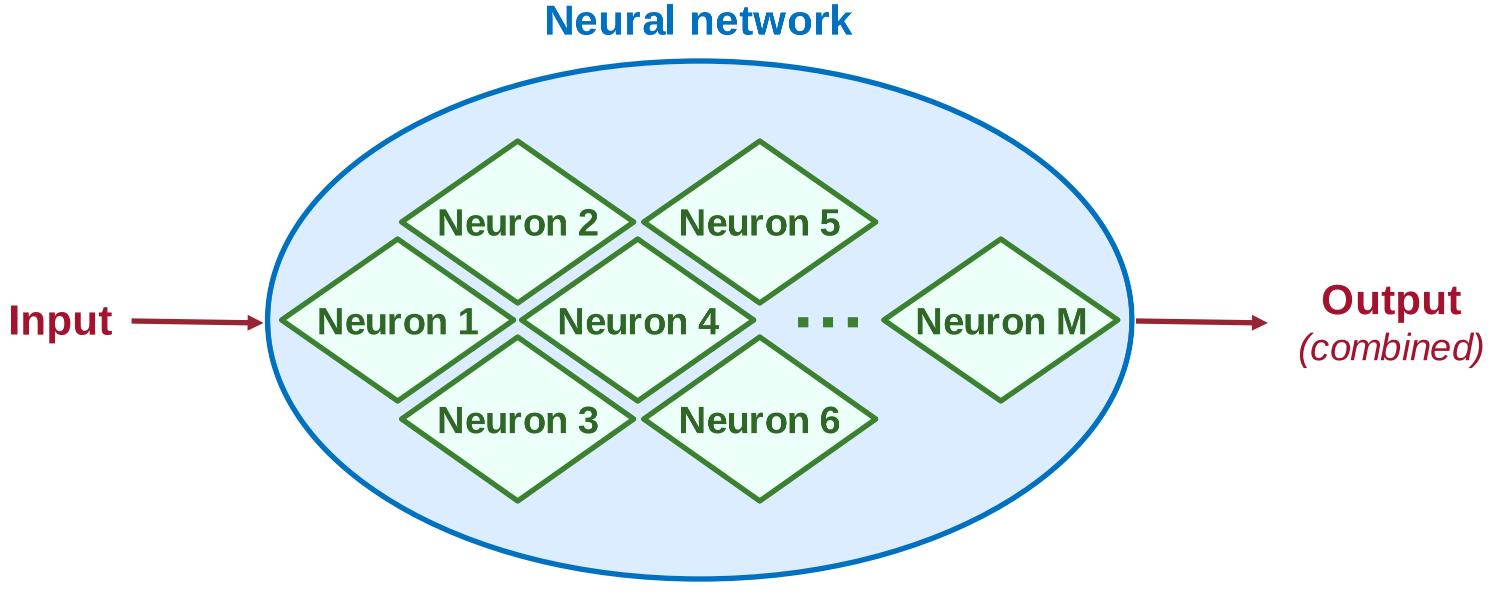
Classification



(Here, the input space is 2-dimensional)

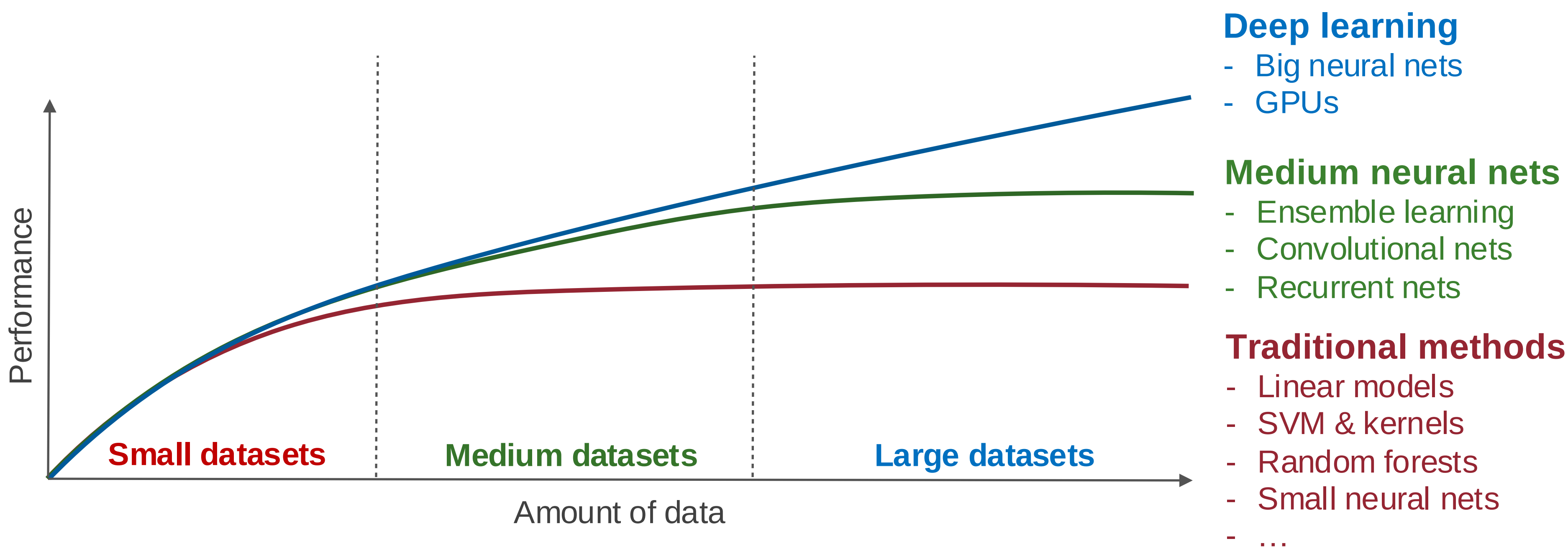
Beyond linear models (2/3)

- **Idea →** Improve accuracy through **ensemble learning**
  - Build a network composed of many neurons
  - Reduce the bias by increasing the variance



Beyond linear models (3/3)

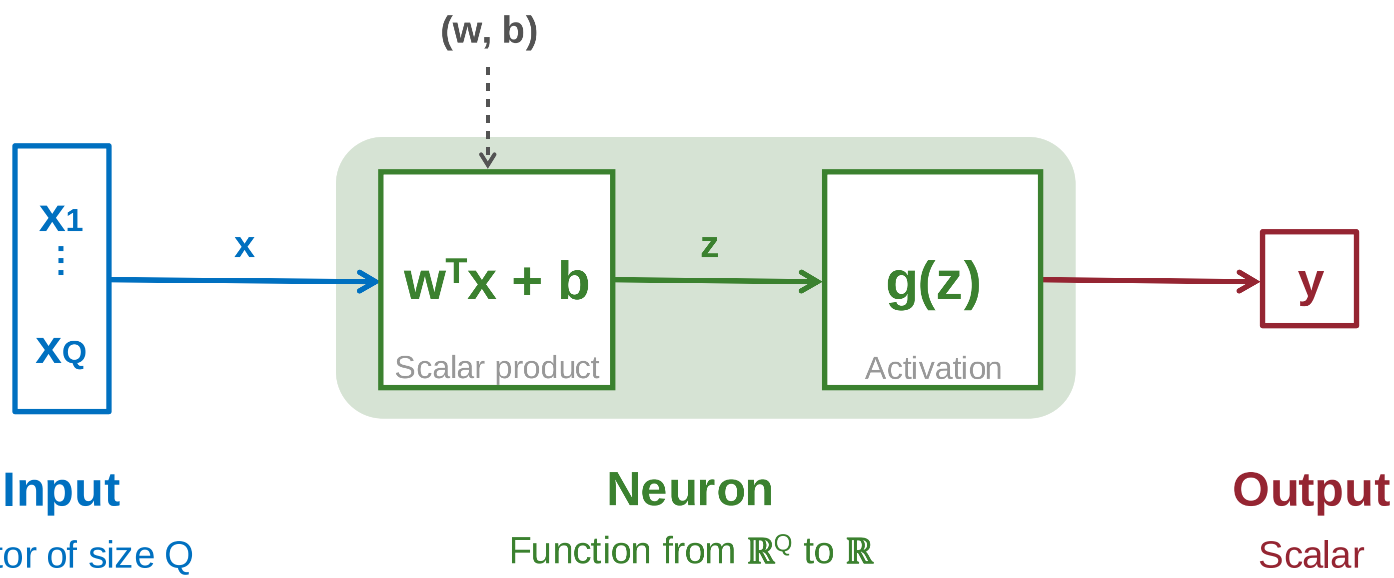
- Neural networks can outperform "traditional" techniques
  - ***Requirement 1*** *→ A large amount of homogeneous non-tabular data*
  - ***Requirement 2*** *→ Time and resources for intensive computing*



Fully-connected layer

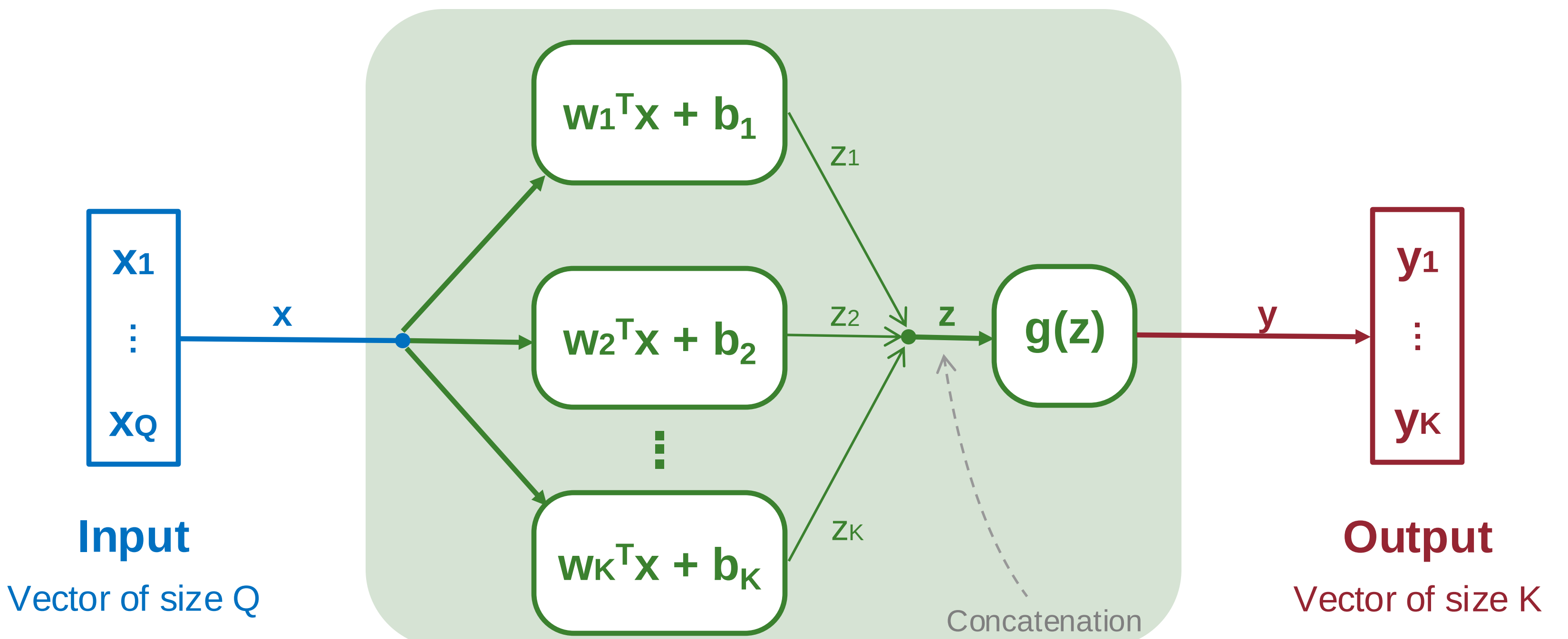
Scalar output

- A neuron takes in a **vector** and returns a **scalar value**
  - How to output a vector ?



Vector output

- **Fully-connected layer ➜** Stack of multiple neurons
  - **Input ➜** A vector supplied to all neurons
  - **Output ➜** A vector holding the values produced by all neurons



Matrix representation (1/2)

- Mathematical formulation of a layer with **K** neurons
  - **Weights →** Matrix that stacks the vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K$
  - **Biases →** Vector that holds the scalars $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_K$
  - **Activation →** Vector function of vector variable (from $\mathbb{R}^K$ to $\mathbb{R}^K$)

$$W = \begin{bmatrix} - \mathbf{w}_1^\top - \\ \vdots \\ - \mathbf{w}_K^\top - \end{bmatrix} \in \mathbb{R}^{K \times Q} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_K \end{bmatrix} \in \mathbb{R}^K \qquad \mathbf{g} \colon \mathbb{R}^K \to \mathbb{R}^K$$

Weight matrix — Bias vector — Activation function

Matrix representation (2/2)

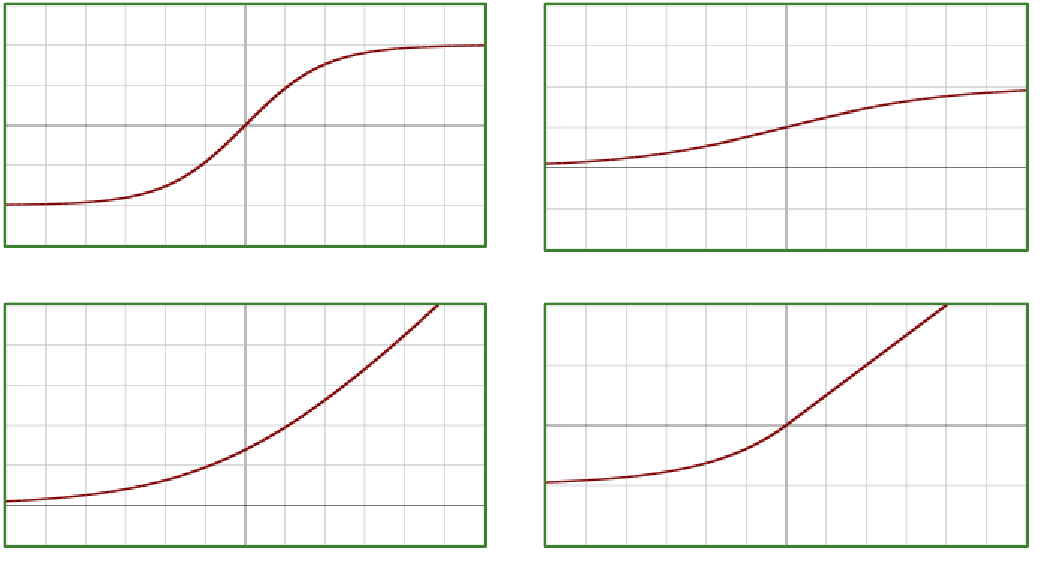
- Operations of a fully-connected layer
  - Matrix product
  - Bias addition
  - Activation

$$\mathbf{y} = \mathbf{g}(W\mathbf{x} + \mathbf{b}) \implies \mathbf{g}\left(\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{\mathbf{x}} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{\mathbf{b}} \right) = \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{\mathbf{y}}$$

Activation (1/4)

- Two types of activation
  - **Non-separable →** An operation applied to the vector as a whole
  - **Separable →** An operation applied separately to each element

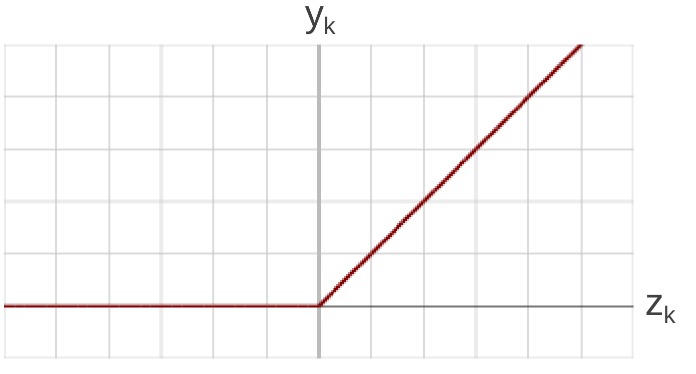
- **Examples**
  - Sigmoid
  - Logistic
  - Softmax
  - ReLU
  - Leaky ReLU
  - …



Activation (2/4)

- **Rectified Linear Unit** (ReLU)
  - Negative values are set to zero
  - Positive values are preserved
  - It is a **separable** activation

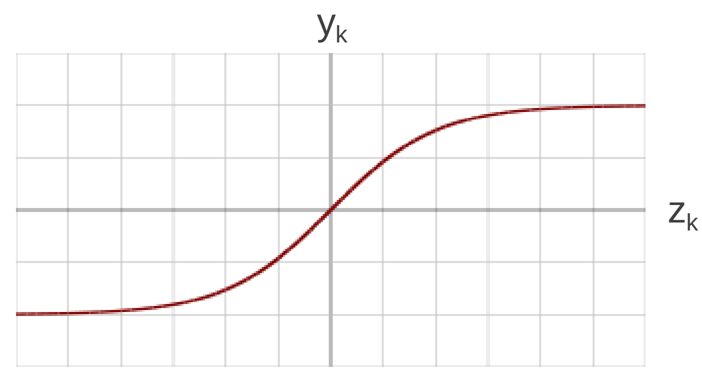
$$\mathbf{g}_{\text{relu}}(\mathbf{z}) = \begin{bmatrix} \max\{0, z_1\} \\ \max\{0, z_2\} \\ \vdots \\ \max\{0, z_K\} \end{bmatrix}$$



Activation (3/4)

- **Sigmoid**
  - Real values are mapped between zero and one
  - S-shaped curve
  - It is a **separable** activation

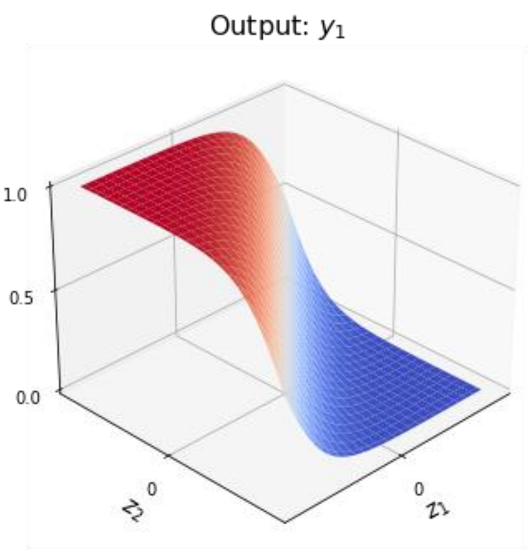
$$\mathbf{g}(\mathbf{z}) = \begin{bmatrix} \frac{1}{1+e^{-z_1}} \\ \frac{1}{1+e^{-z_2}} \\ \vdots \\ \frac{1}{1+e^{-z_K}} \end{bmatrix}$$

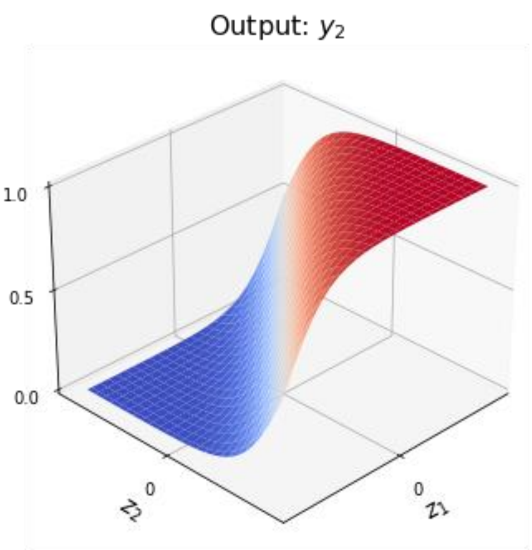


Activation (4/4)

- **Softmax**
  - A vector is transformed to have positive elements that sum to one
  - Generalization of the sigmoid to multiple dimensions
  - Smooth approximation of the “argmax” operation
  - It is a **non-separable** activation

$$\mathbf{g}(\mathbf{z}) = \begin{bmatrix} \frac{e^{z_1}}{e^{z_1}+\cdots+e^{z_K}} \\ \frac{e^{z_2}}{e^{z_1}+\cdots+e^{z_K}} \\ \vdots \\ \frac{e^{z_K}}{e^{z_1}+\cdots+e^{z_K}} \end{bmatrix}$$





Quiz

- Suppose you have a layer with **Q inputs** and **K outputs**. How many parameters (weights & biases) does it have?

  1) ***Q***

  2) ***Q + K***

  3) ***Q K***

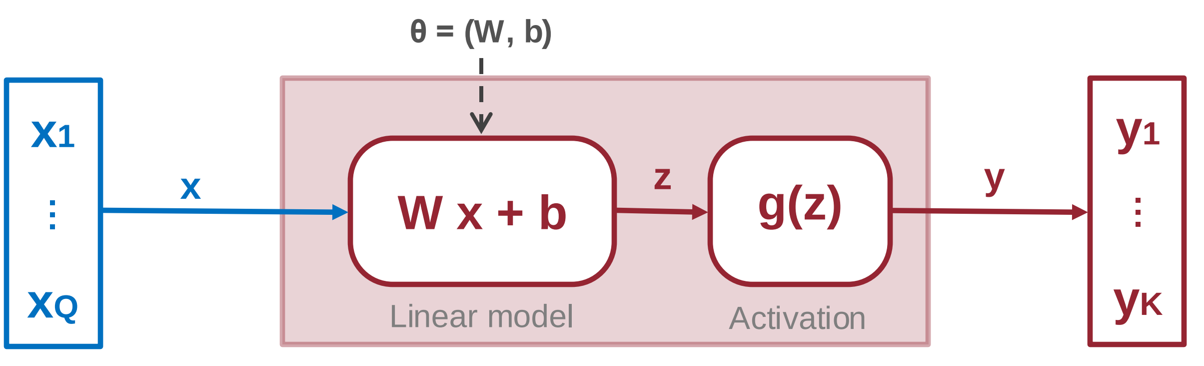
  4) ***Q K + K***

Neural networks

One-layer network (1/2)

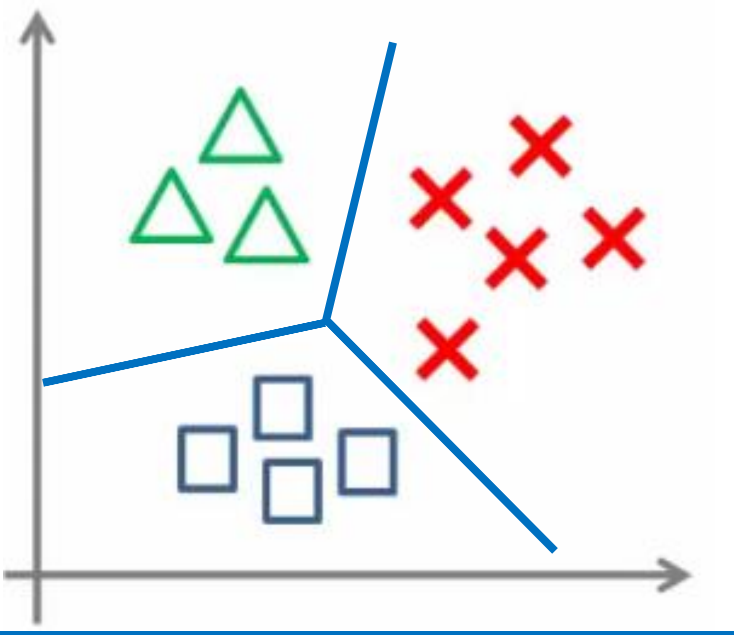
- **One-layer network**
  - A **single** fully-connected layer with **K ≥ 1** neurons

$$f_\theta(\mathbf{x}) = \mathbf{g}(W\mathbf{x} + \mathbf{b})$$



One-layer network (2/2)

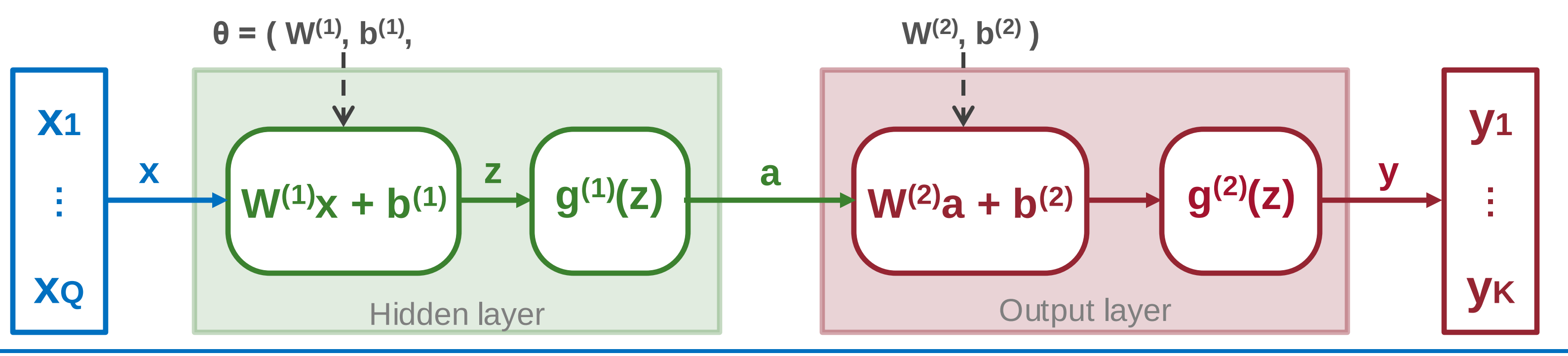
- This is a **linear model** with **K outputs**
  - **Regression ➜** Prediction of K values
  - **Classification ➜** Prediction of K classes



Two-layer network (1/3)

- **Two-layer network**
  - **Hidden layer →** A fully-connected layer with $\mathbf{M^{(1)}}$ neurons
  - **Output layer →** A fully-connected layer with $\mathbf{M^{(2)}} = \mathbf{K}$ neurons

$$\mathbf{a} = \mathbf{g}^{(1)}(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$f_\theta(\mathbf{x}) = \mathbf{g}^{(2)}(W^{(2)}\mathbf{a} + \mathbf{b}^{(2)})$$



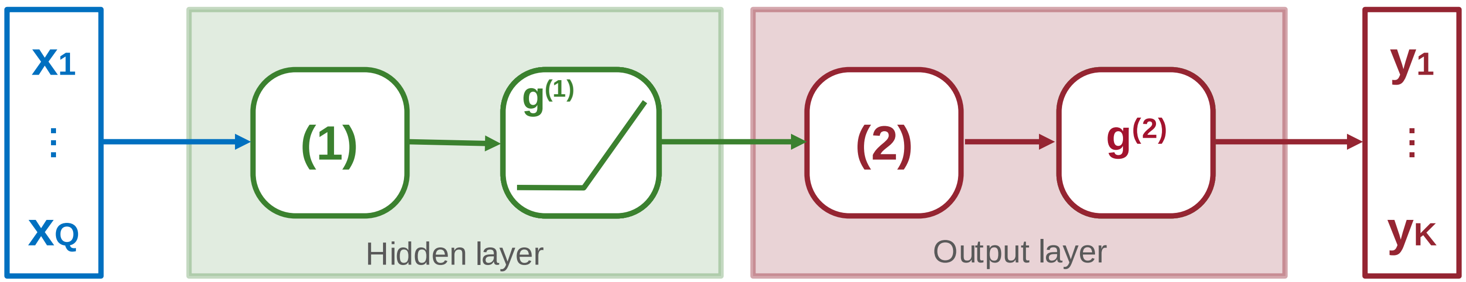
Two-layer network (2/3)

- This is a **nonlinear model** with **K outputs**
  - **Regression ➜** Prediction of K values
  - **Classification ➜** Prediction of K classes

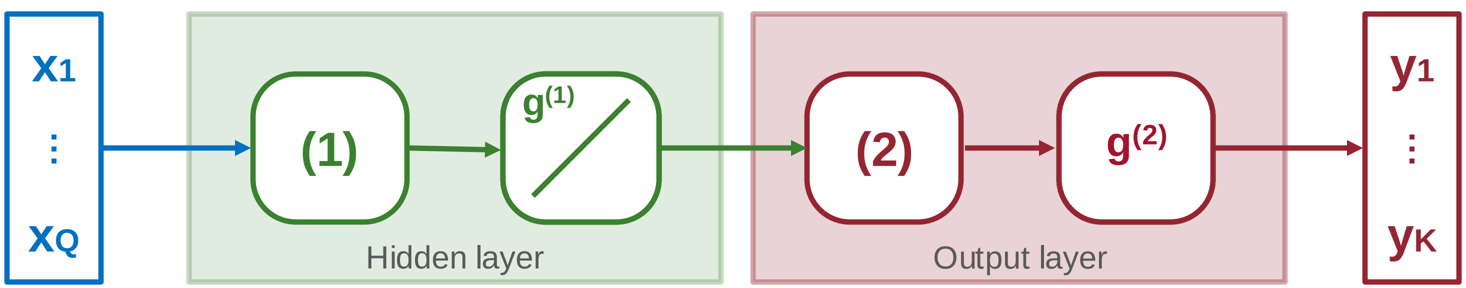
Two-layer network (3/3)

- The **hidden layer** must have a **nonlinear activation**
  - Otherwise the network behaves like a linear model

This is a **nonlinear** model



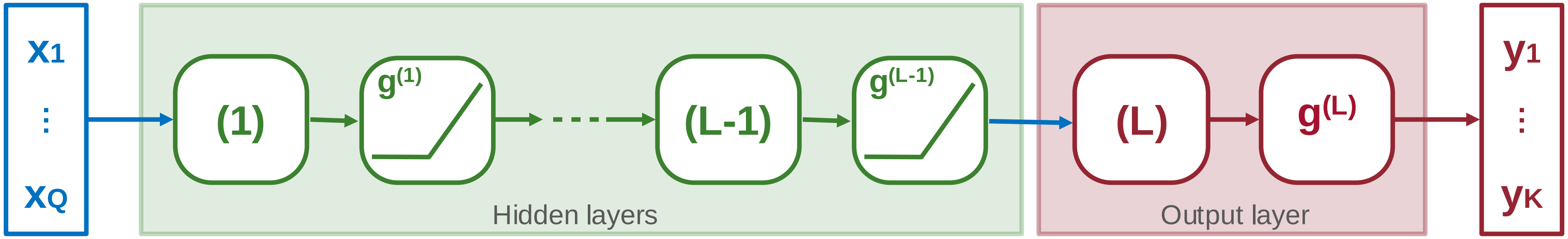
This is a **linear** model



Multilayer network

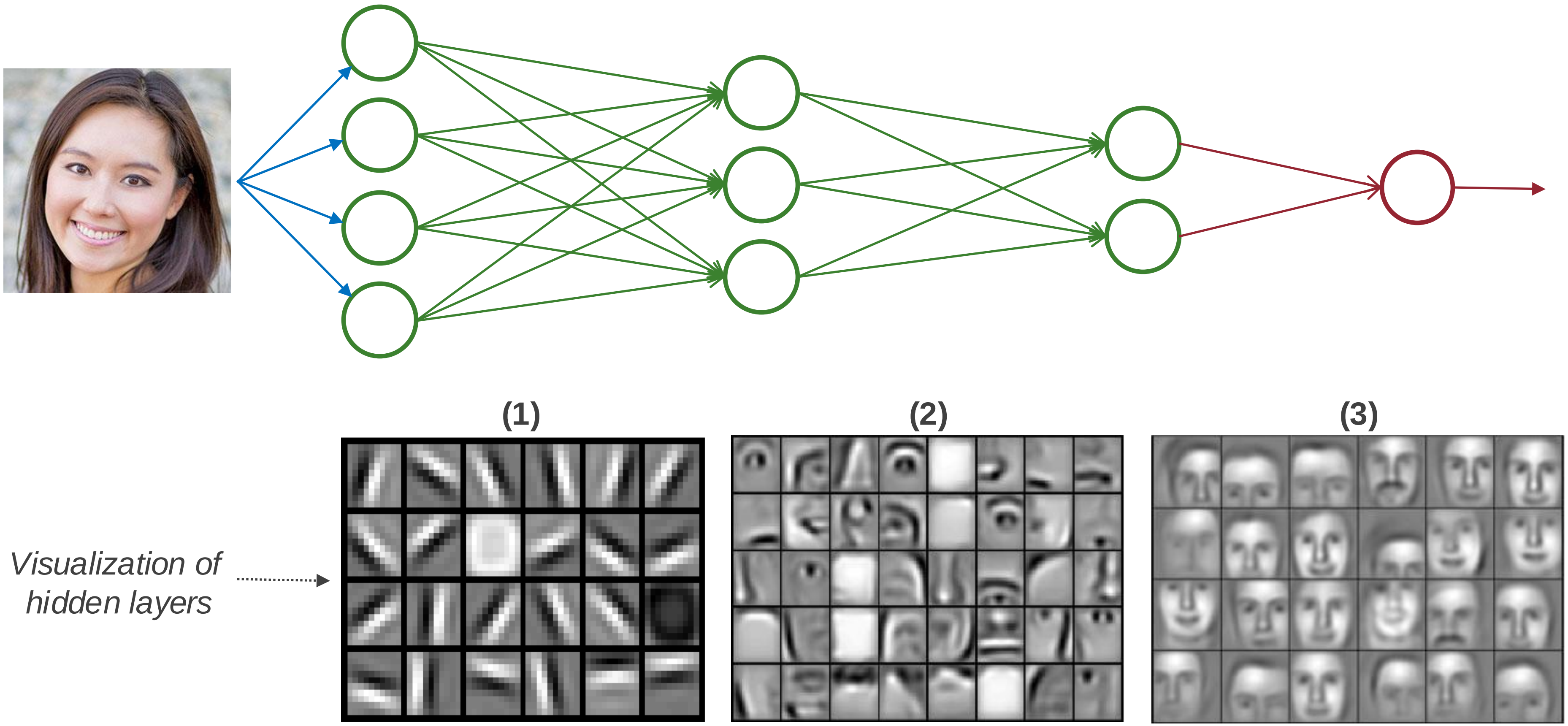
- **Multilayer network**
  - **Feed-forward →** Multiple layers arranged in series (no loops)

$$
\begin{aligned}
\mathbf{a}^{(1)} &= \mathbf{g}^{(1)}\big(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\big) \\
\mathbf{a}^{(2)} &= \mathbf{g}^{(2)}\big(W^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)}\big) \\
&\ \vdots \\
f_\theta(\mathbf{x}) &= \mathbf{g}^{(L)}\big(W^{(L)}\mathbf{a}^{(L-1)} + \mathbf{b}^{(L)}\big)
\end{aligned}
$$



Why neural networks? (1/2)

- Neural networks can learn a **hierarchical representation**



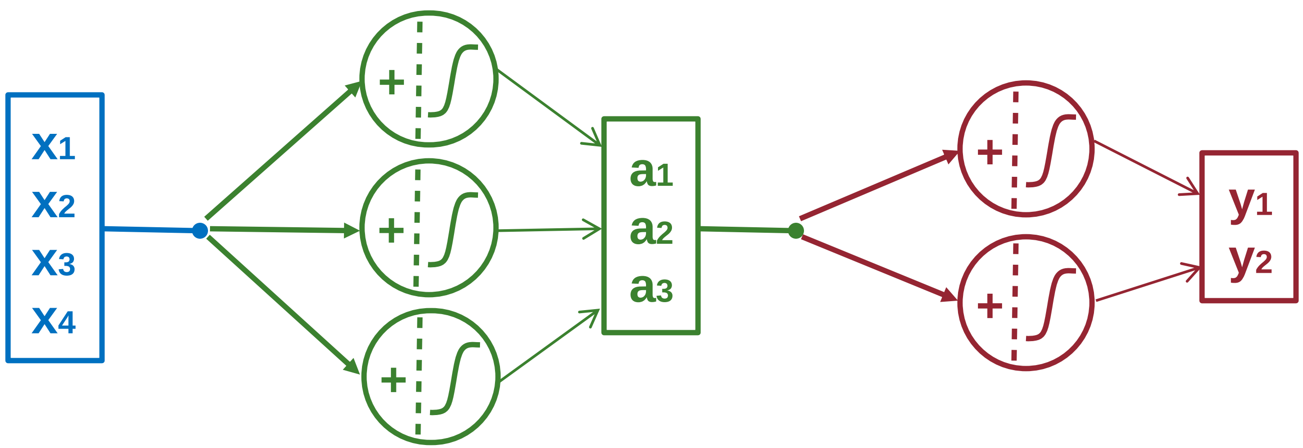
Why neural networks? (2/2)

- Neural networks can learn a **hierarchical representation**
  - ***First layer*** ➔ *Localization of edges in the input images*
  - ***Second layer*** ➔ *Grouping of edges into shapes (e.g., eyes, noses, …)*
  - ***Third layer*** ➔ *Formation of full objects (e.g., faces)*
  - ***Fourth layer*** ➔ *Object classification (e.g., face detection)*

Visualization of hidden unis

Quiz

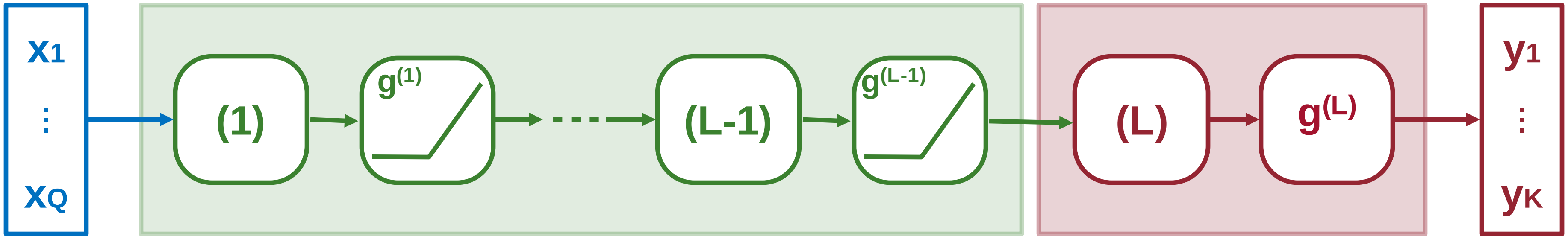
- Consider a network composed of a hidden layer with 3 neurons and an output layer with 2 neurons (see below).
  1) *What is the size of the weight matrix $W^{(1)}$ and the bias vector $b^{(1)}$?*
  2) *What is the size of the weight matrix $W^{(2)}$ and the bias vector $b^{(2)}$?*



Neural networks for regression/classification

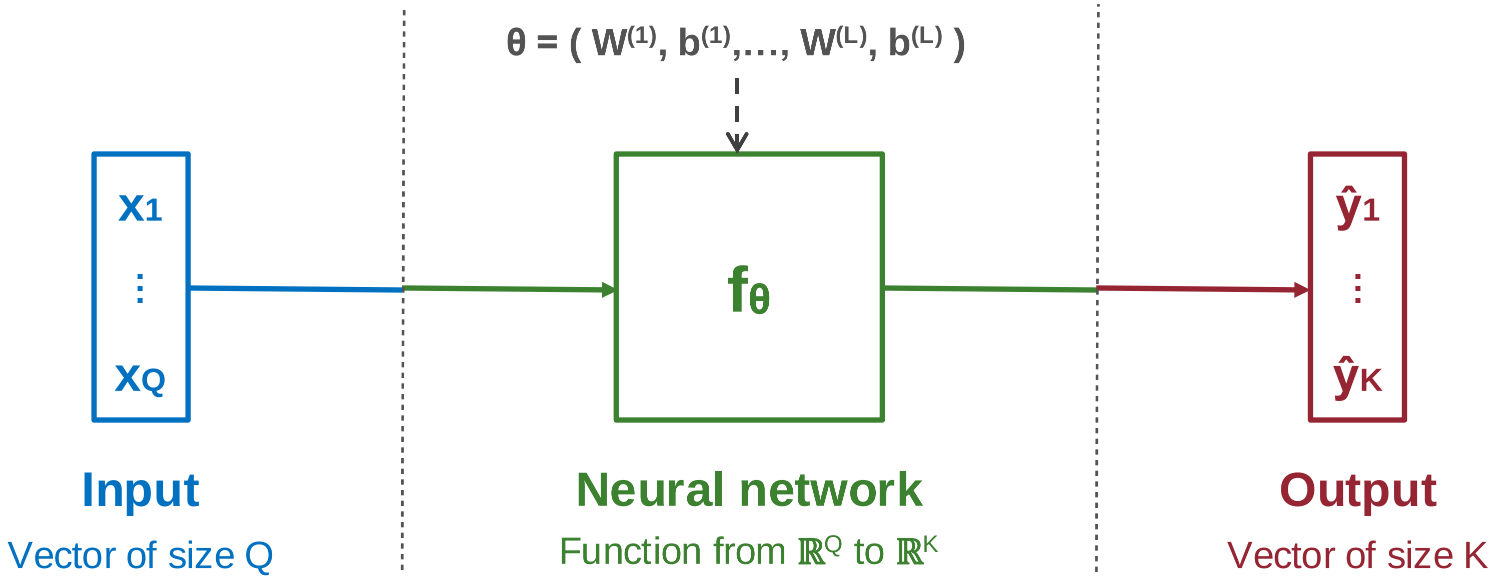
What we have seen so far…

- Neural network
  - **Hidden layers ➜** Activations must be nonlinear
  - **Output layer ➜** How to set it up?



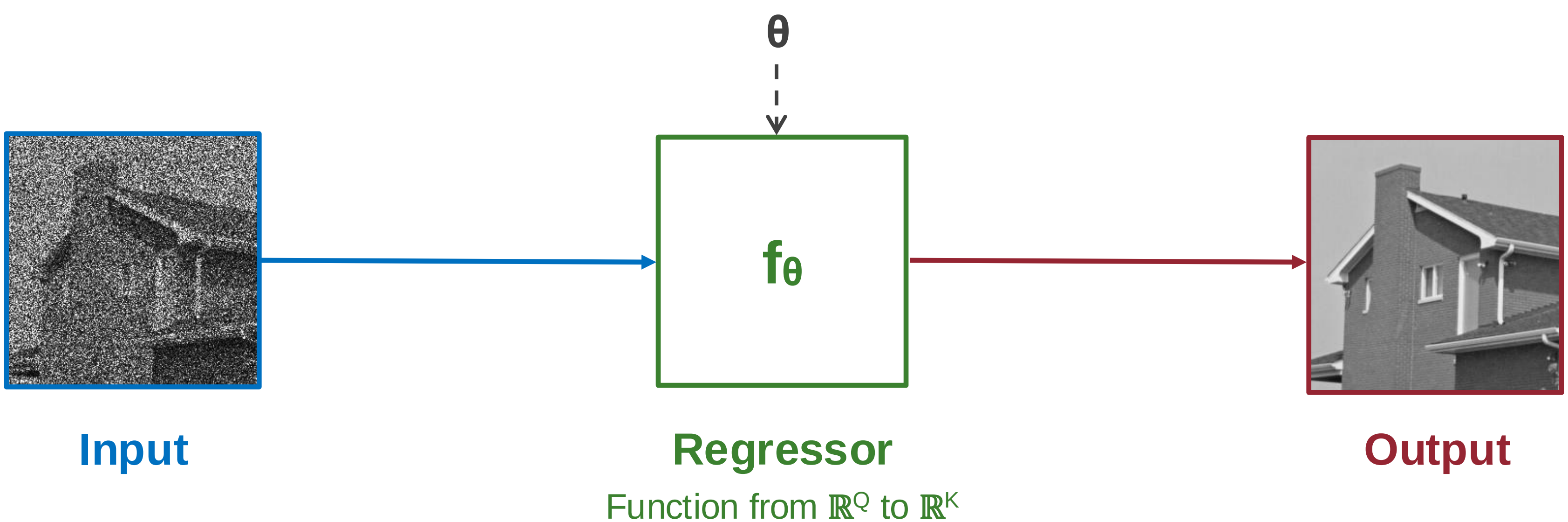
Input-Output of a neural network

- Neural network
  - **Input size ➜** Equal to the **"width" of neurons** in the first layer
  - **Output size ➜** Equal to the **number of neurons** in the output layer



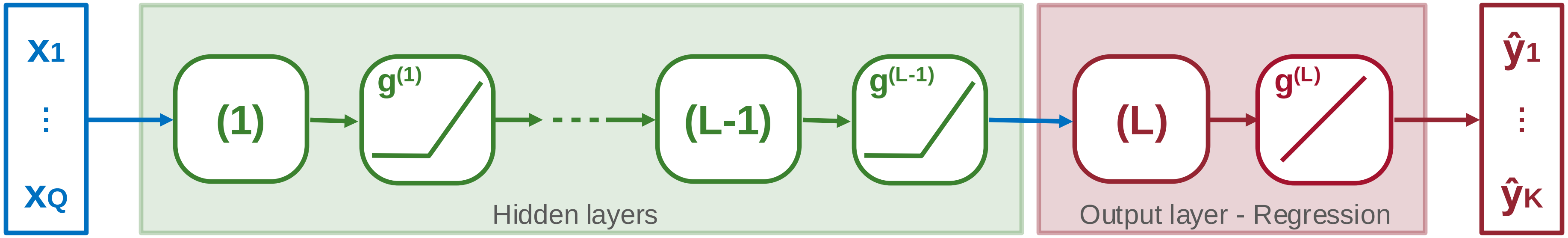
Multiple regression (1/3)

- **Multiple regression**
  - Regression with **K ≥ 1** target variables
  - **Example →** Image restoration



Multiple regression (2/3)

- **Output layer**
  - **Number of neurons →** Equal to the number of target variables
  - **Activation →** Identity



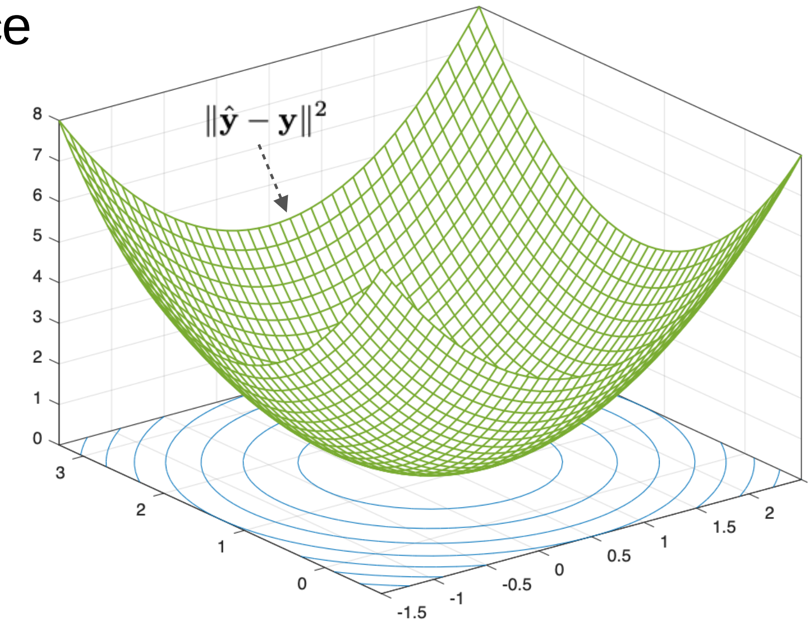
Multiple regression (3/3)

- **Training data ➜** Vector input – Vector output

$$\mathcal{S}_{\text{train}} = \left\{ (\mathbf{x}^{(n)}, \mathbf{y}^{(n)}) \in \mathbb{R}^Q \times \mathbb{R}^K \mid n = 1, \dots, N \right\}$$

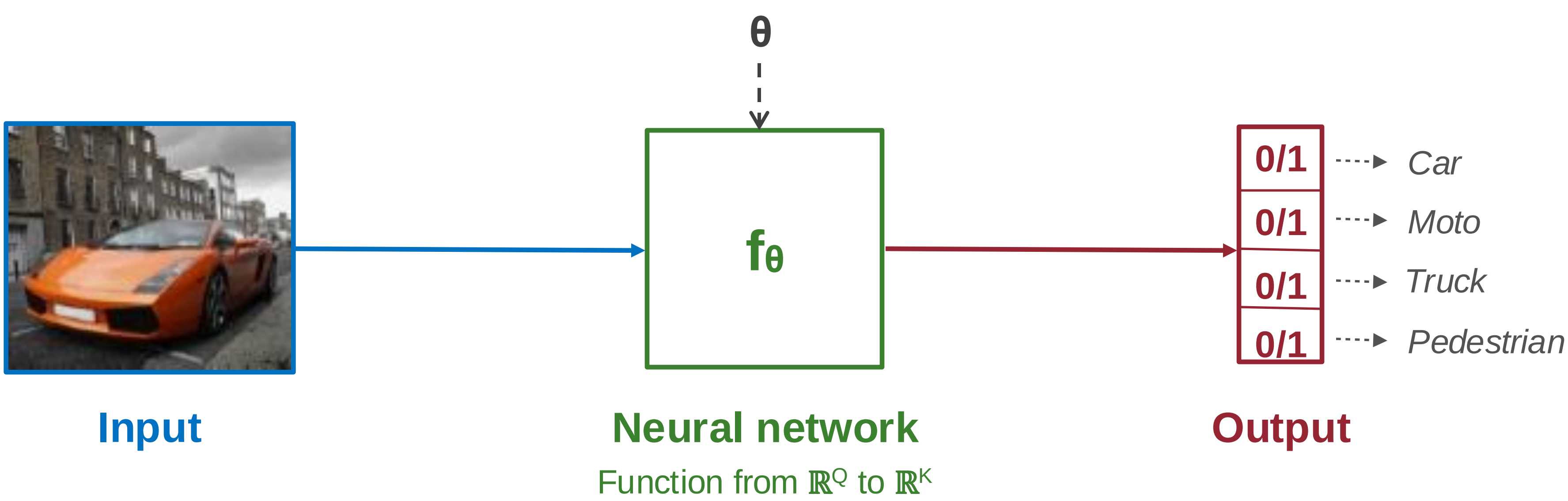
- **Loss function ➜** Euclidean distance

$$\mathcal{E}(\hat{\mathbf{y}}, \mathbf{y}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2$$



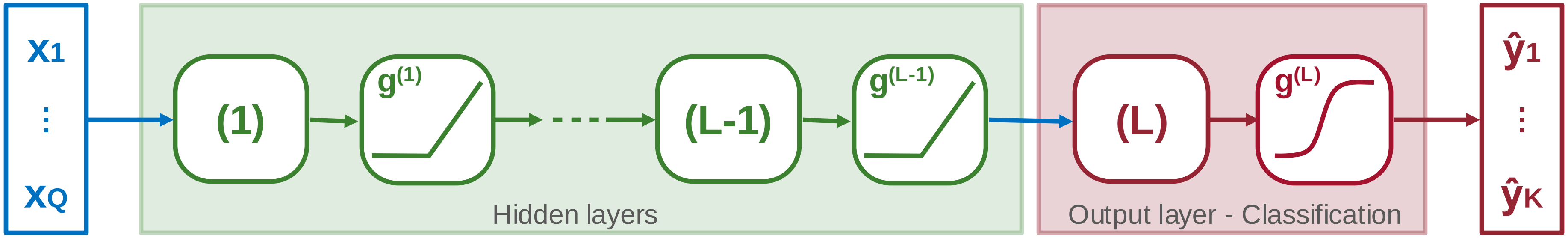
Multiclass classification (1/4)

- **Multiclass classification**
  - Classification with **K ≥ 2** classes
  - **Example →** Image classification



Multiclass classification (2/4)

- **Output layer**
  - **Number of neurons →** Equal to the number of classes
  - **Activation →** Softmax



Multiclass classification (3/4)

- **Training data** → Vector input — Binary vector output

$$\mathcal{S}_{\text{train}} = \left\{ (\mathbf{x}^{(n)}, \mathbf{y}^{(n)}) \in \mathbb{R}^Q \times \{0,1\}^K \mid n = 1, \ldots, N \right\}$$

  - Output vectors must be **one-hot encoded**

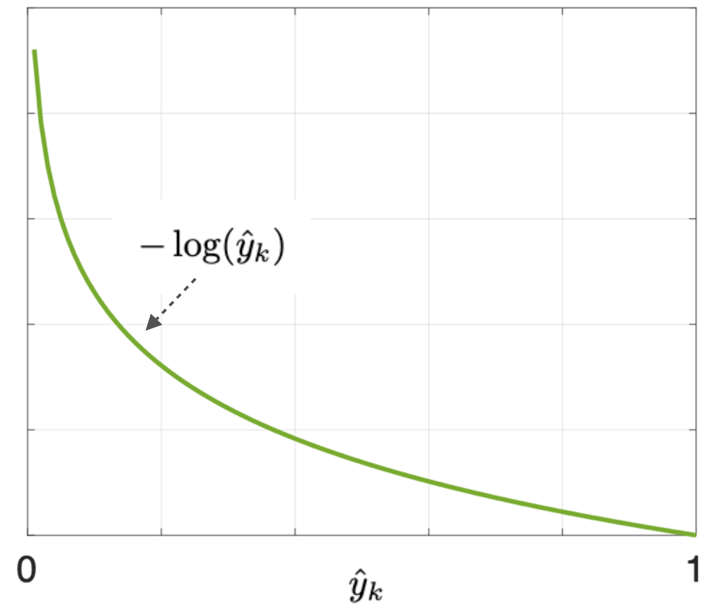
$$\mathrm{y}_{\text{class 1}} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \mathrm{y}_{\text{class 2}} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \qquad \cdots \qquad \mathrm{y}_{\text{class K}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

Multiclass classification (4/4)

- **Loss function ➜** Cross entropy

$$\mathcal{E}(\hat{\mathbf{y}}, \mathbf{y}) = \begin{cases} -\log\left(\hat{y}_1\right) & \text{if } y_1 = 1 \\ -\log\left(\hat{y}_2\right) & \text{if } y_2 = 1 \\ \quad\vdots & \\ -\log\left(\hat{y}_K\right) & \text{if } y_K = 1 \end{cases}$$

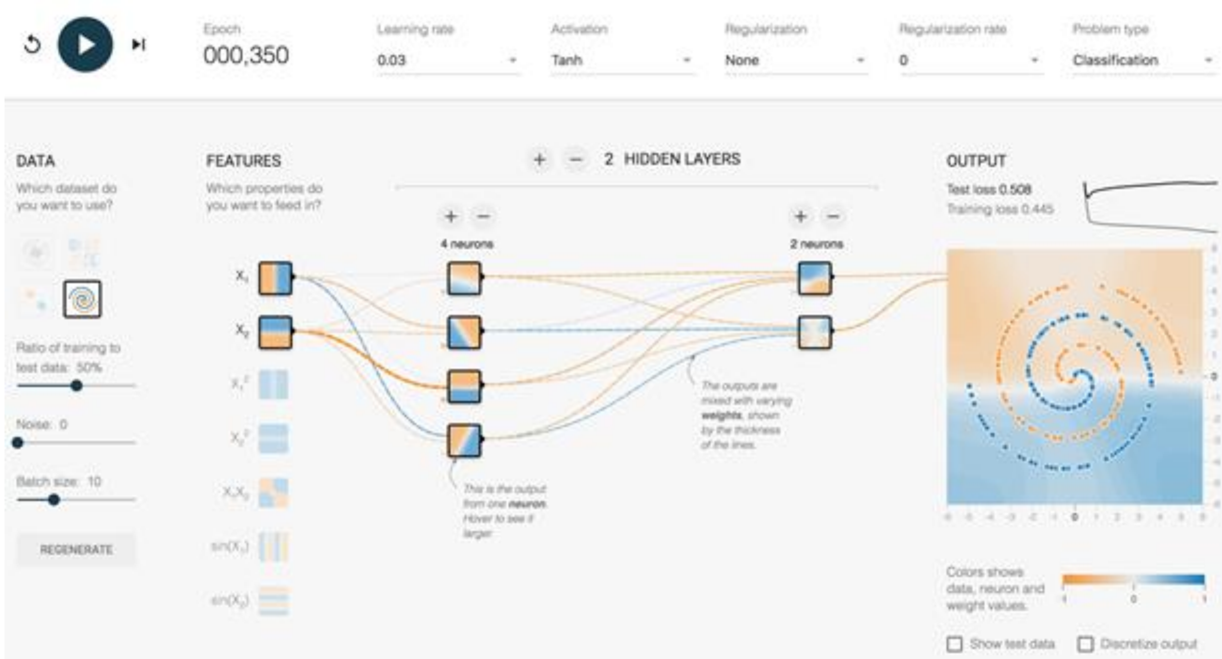
Only one is selected ← - - - *One-hot encoding*



Code examples

Live coding

- **Neural network playground →** http://playground.tensorflow.org
  - Tinker with a neural network in your browser
  - Useful to grasp the concepts introduced in this lecture



Keras (1/5)

- **Keras** is a Python library for deep learning
  - **Step 0 ➜ Import the library**

```
import keras
```

  - **Step 1 ➜ Load the dataset**

```
from keras.datasets import mnist

(images, labels), (test_images, test_labels) = mnist.load_data()
```

Keras (2/5)

- MNIST dataset requires a classifier with **10 classes**

  - **Step 2 → Define a two-layer network**

```
network = keras.models.Sequential()

network.add(keras.layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(keras.layers.Dense(10, activation='softmax'))
```

  - **Step 3 → Set the loss function and the optimizer**

```
network.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Keras (3/5)

- Data must be preprocessed before learning
  - **Step 4 → Normalization of inputs**

```
train_inputs = images.reshape((60000, 28 * 28)) \
                     .astype('float32') / 255

test_inputs = test_images.reshape((10000, 28 * 28)) \
                         .astype('float32') / 255
```

  - **Step 5 → One-hot encoding of outputs**

```
train_targets = np.eye(10)[labels]

test_targets = np.eye(10)[test_labels]
```

Keras (4/5)

- Now, all is ready for training the network
  - **Step 6 → Learning on the training set**

```
history = network.fit(train_inputs, train_targets, epochs=10, batch_size=1000)
```

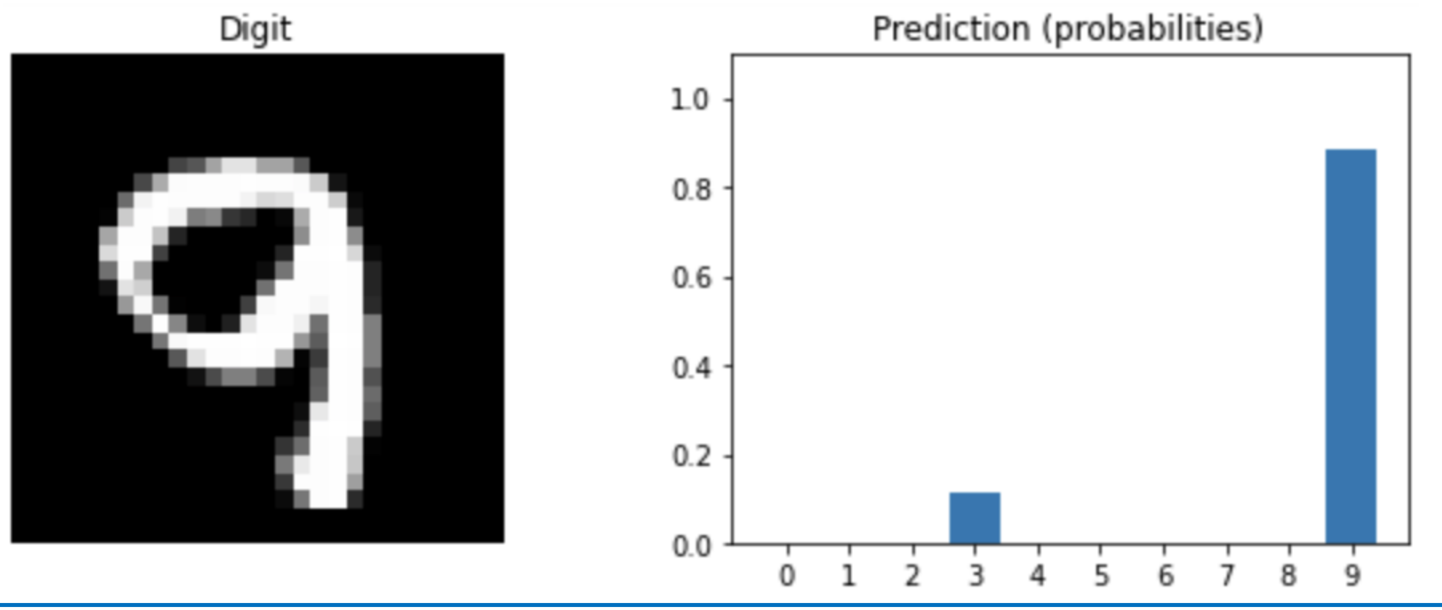
```
Epoch 1/10
60/60 [==============================] - 1s 10ms/step - loss: 0.0453 - accuracy: 0.9884
Epoch 2/10
60/60 [==============================] - 1s 10ms/step - loss: 0.0397 - accuracy: 0.9900
Epoch 3/10
60/60 [==============================] - 1s 10ms/step - loss: 0.0353 - accuracy: 0.9917
Epoch 4/10
60/60 [==============================] - 1s 11ms/step - loss: 0.0317 - accuracy: 0.9923
Epoch 5/10
60/60 [==============================] - 1s 11ms/step - loss: 0.0274 - accuracy: 0.9941
Epoch 6/10
60/60 [==============================] - 1s 14ms/step - loss: 0.0245 - accuracy: 0.9947
Epoch 7/10
60/60 [==============================] - 1s 10ms/step - loss: 0.0218 - accuracy: 0.9959
Epoch 8/10
60/60 [==============================] - 1s 10ms/step - loss: 0.0199 - accuracy: 0.9964
Epoch 9/10
60/60 [==============================] - 1s 10ms/step - loss: 0.0173 - accuracy: 0.9969
Epoch 10/10
60/60 [==============================] - 1s 10ms/step - loss: 0.0157 - accuracy: 0.9974
```

Keras (5/5)

- Finally, the network can be used to classify new data
  - **Step 7 → Evaluate the performance on the test set**

```
_, accuracy = network.evaluate(test_inputs, test_targets)
```

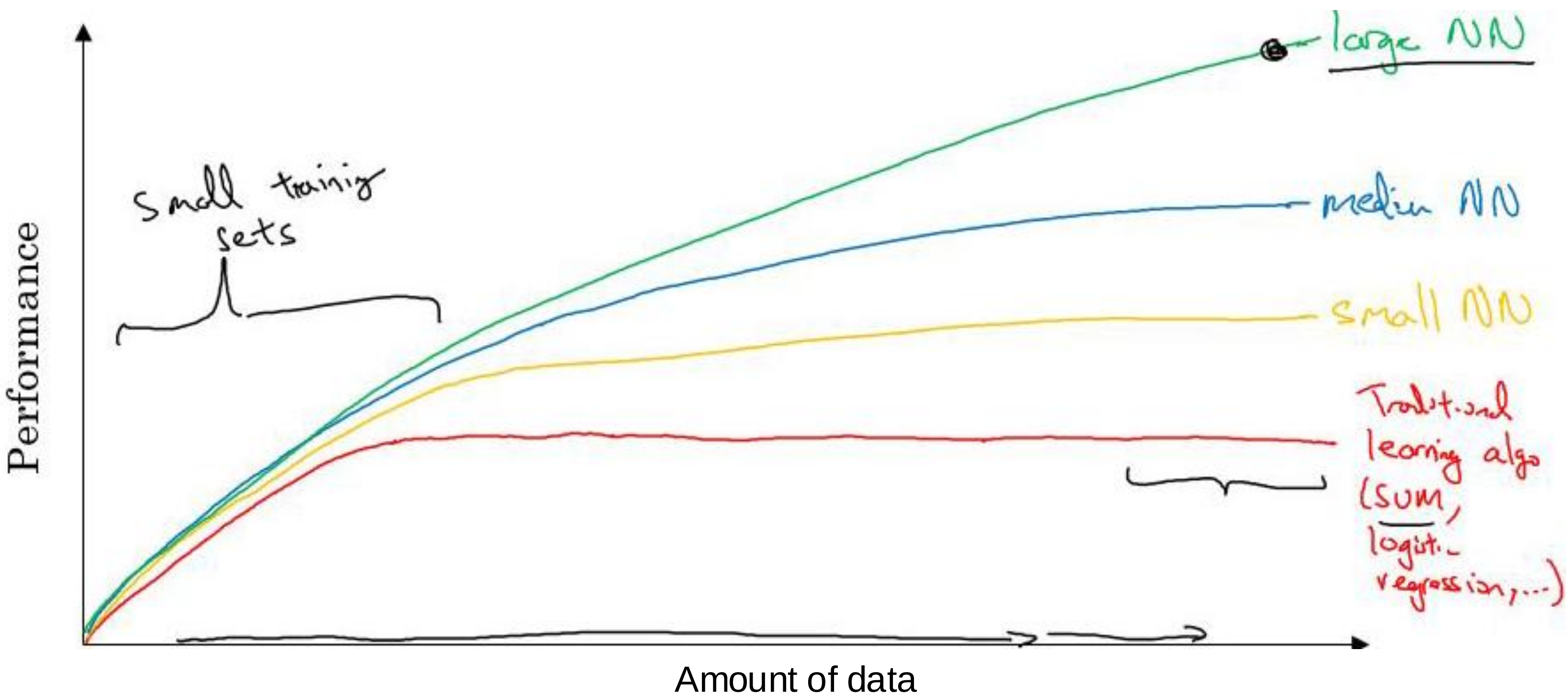
```
accuracy: 0.9804
```



Conclusion

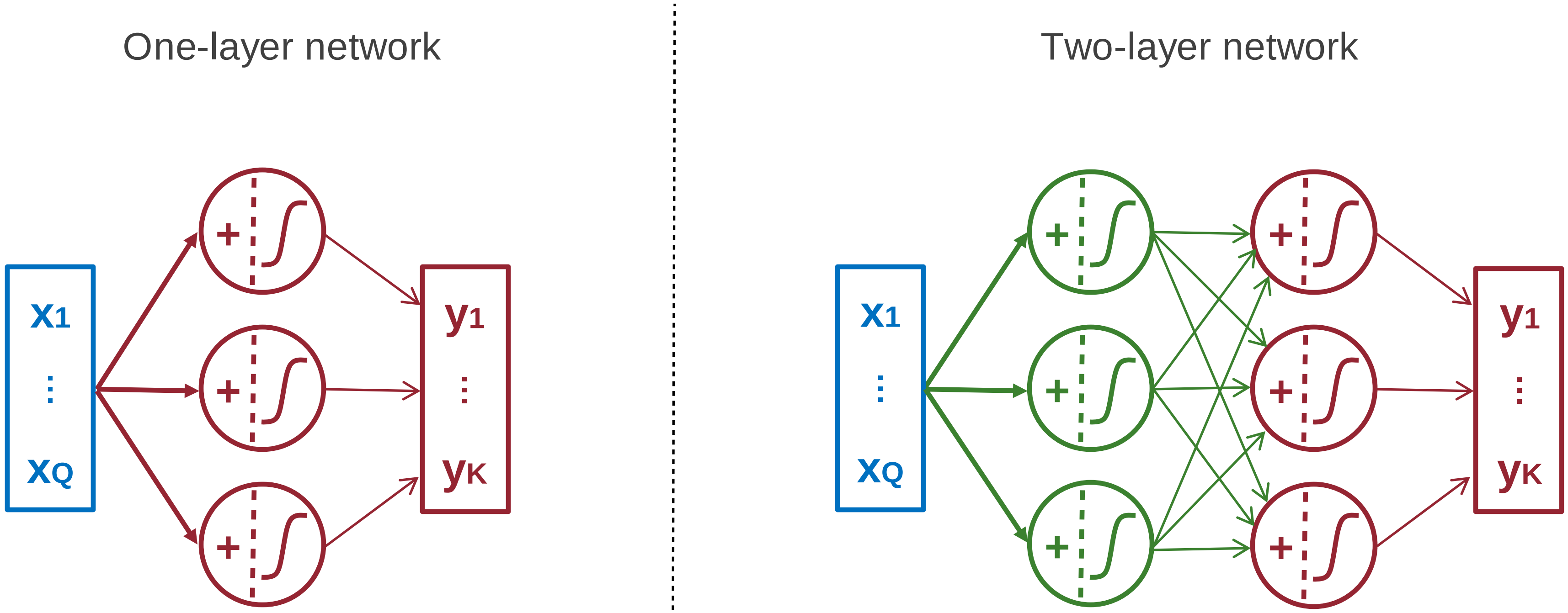
Quest for nonlinear models

- How to get better performance out of machine learning?
  - Use “more complex” nonlinear models
  - Use more data for training



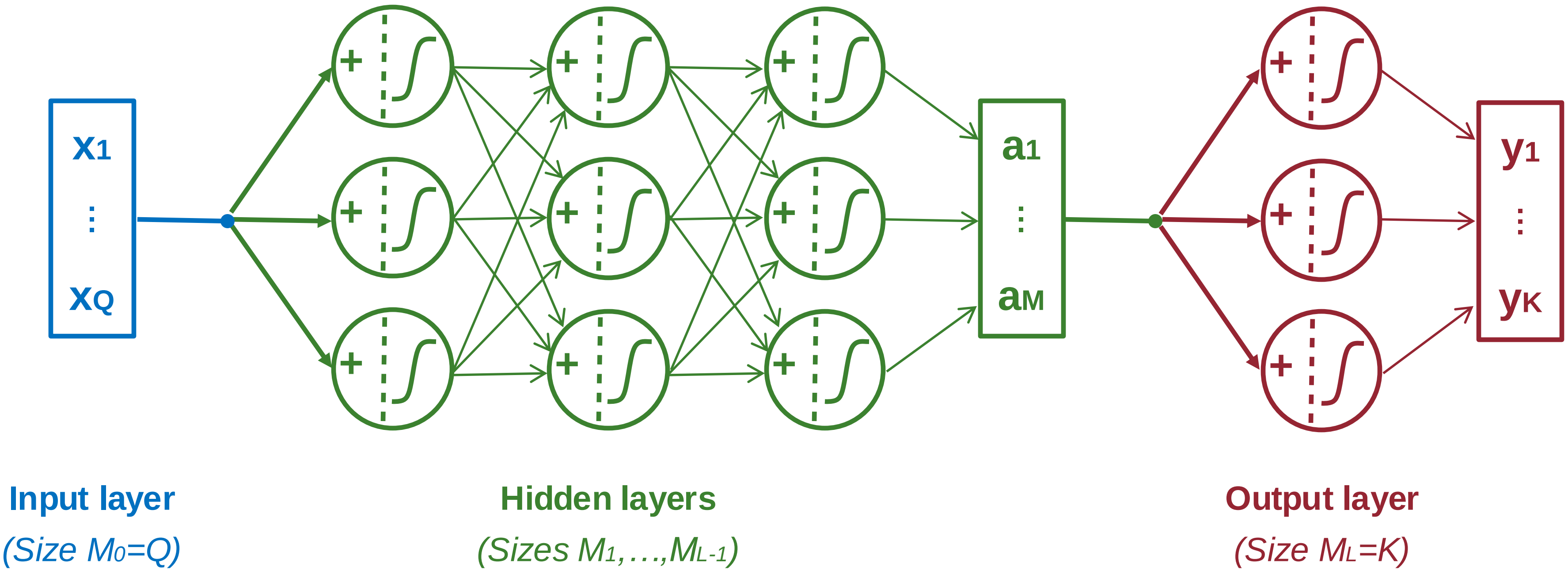
Multilayer networks (1/3)

- Neural networks consist of neurons organized in layers



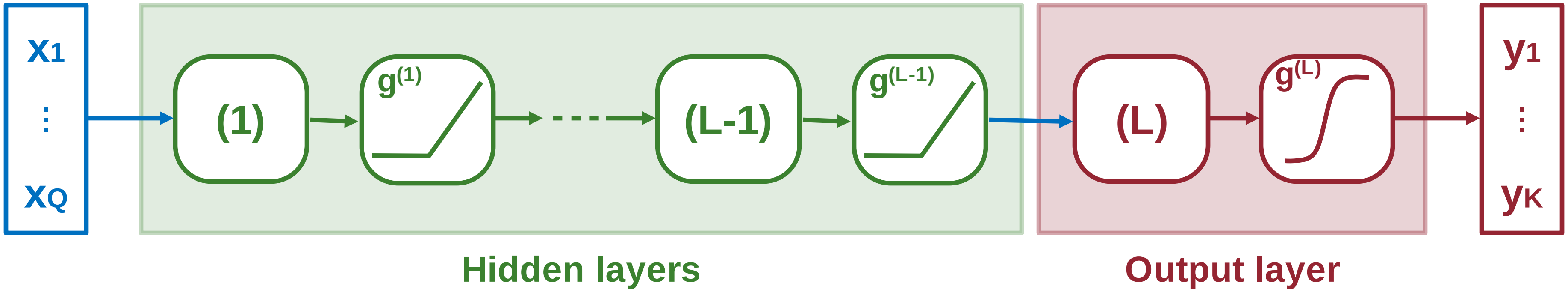
Multilayer networks (2/3)

- Adding more layers increases the learning capabilities



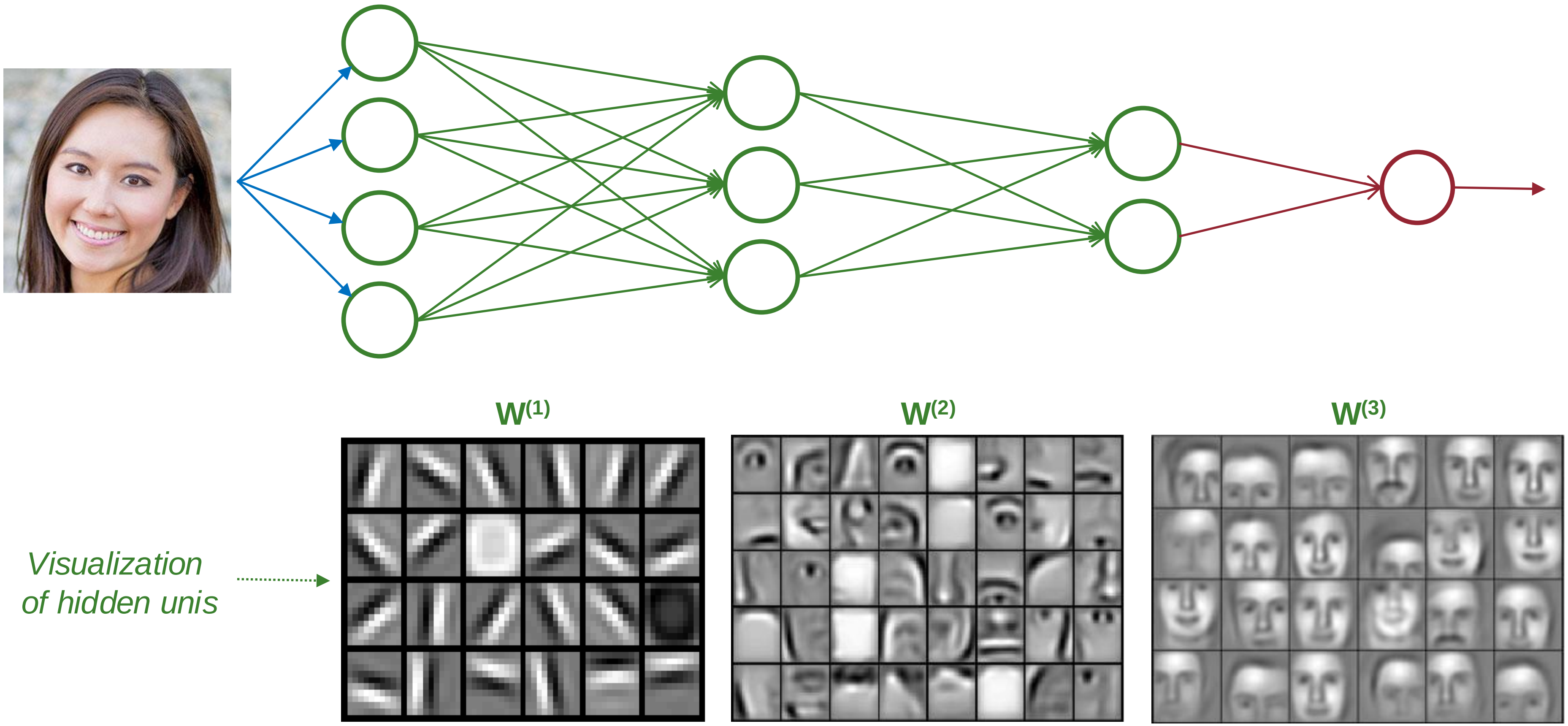
Multilayer networks (3/3)

- Architecture of neural networks
  - Hidden layers must have a nonlinear activation (e.g., ReLU)
  - The output layer must be adapted to the task
    - *Regression No activation*
    - *Classification Softmax*

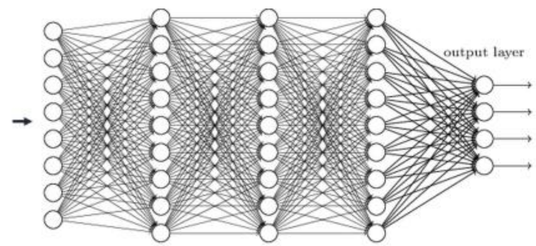


Hierarchical representation

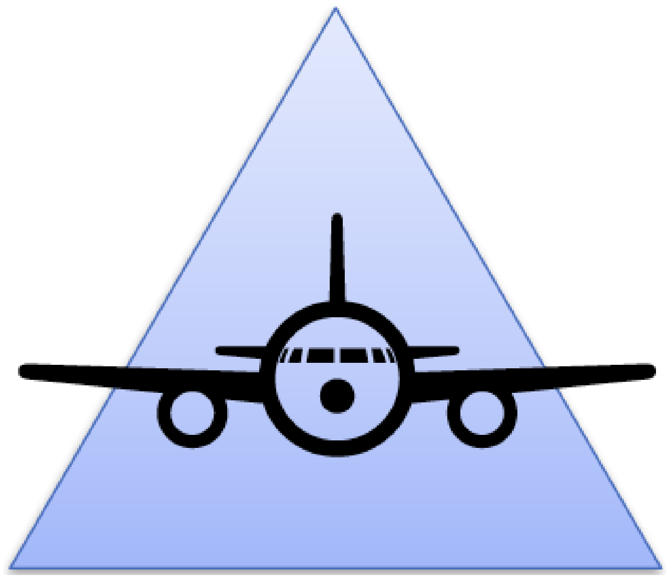
- Multilayer networks can learn a hierarchical representation

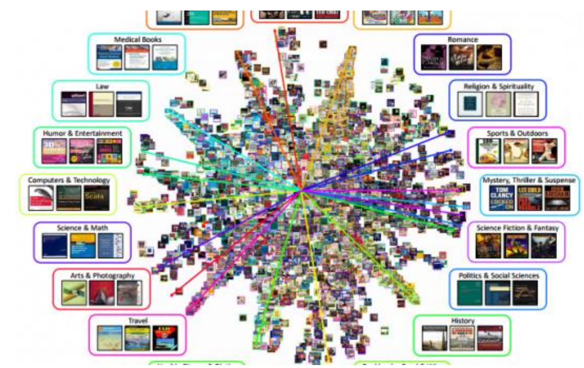


Why are neural networks successful?



High capacity models





Computing power

Lots of training data