Image Analysis and Computer Vision

Giacomo Boracchi

giacomo.boracchi@polimi.it

February 14th 2024

UEM, Maputo

https://boracchi.faculty.polimi.it

Image Classification

Giacomo Boracchi

giacomo.boracchi@polimi.it

February 14th 2024

UEM, Maputo

https://boracchi.faculty.polimi.it

Course Slides

Slides can be found on my website

https://boracchi.faculty.polimi.it/

and follow Tutorials and Talks

https://boracchi.faculty.polimi.it/seminars.html

Colab Folder

In this folder you will find, regularly updated notebooks

https://drive.google.com/drive/folders/10j99rb2kKo4KpLxca-uMe7uesy-8RZeD

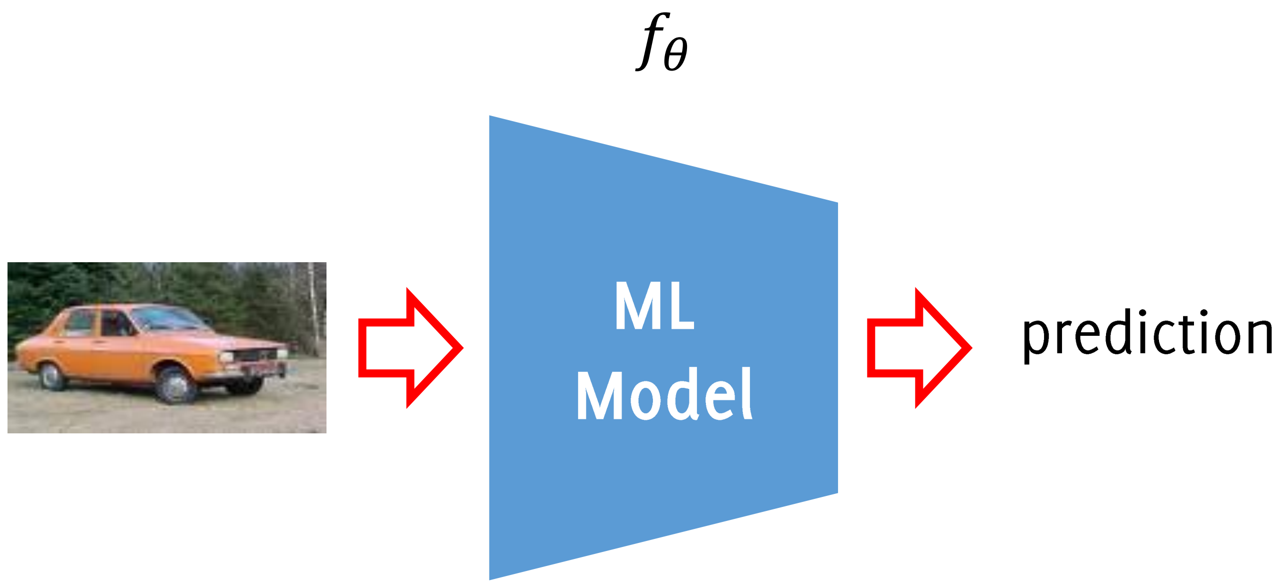
Notebooks require you to “fill in” some codes or to extend codes we illustrate during lectures to new data/new challenges

A Machine Learning Take on Image Understanding

Machine Learning Paradigms

Supervised Learning

- Classification
- Regression



$$x \Rightarrow f_\theta(x) = y$$

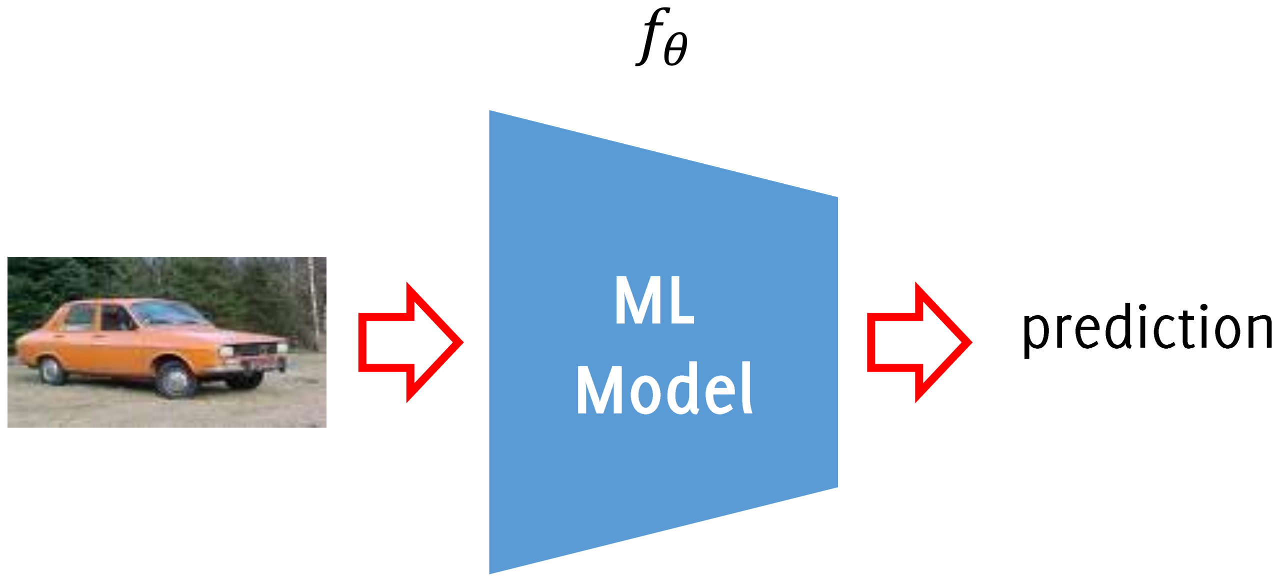
Machine Learning Paradigms

Supervised Learning

- Classification
- Regression

Unsupervised Learning

- Clustering
- Anomaly Detection
- ...



x ⇨ $f_\theta(x) = y$

Machine Learning Paradigms

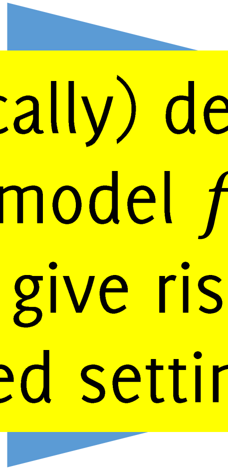
Supervised Learning

- Classification
- Regression

Unsupervised

- Clustering
- Anomaly Detection
- …

f_θ



prediction

Learning consists is (automatically) defining the parameters θ of the model f.
Different settings applies, which give rise to the supervised and unsupervised settings

x ⇨ $f_\theta(x) = y$

Supervised Learning

In **Supervised Learning** we are given a training in the form:

$$TR = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

where

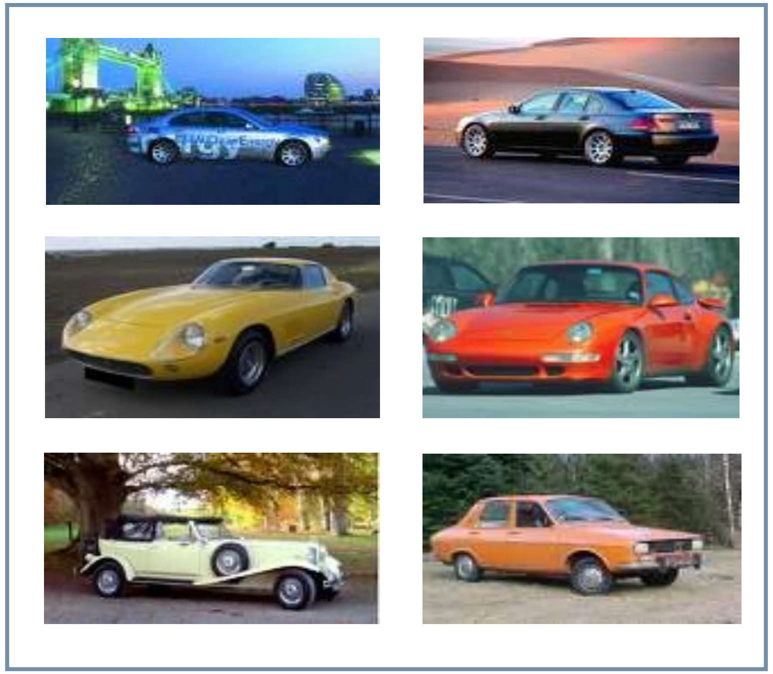
- $x_i \in \mathbb{R}^d$ is the input
- $y_i \in \Lambda$ is the target, the expected output of the model to x_i

The set Λ can be

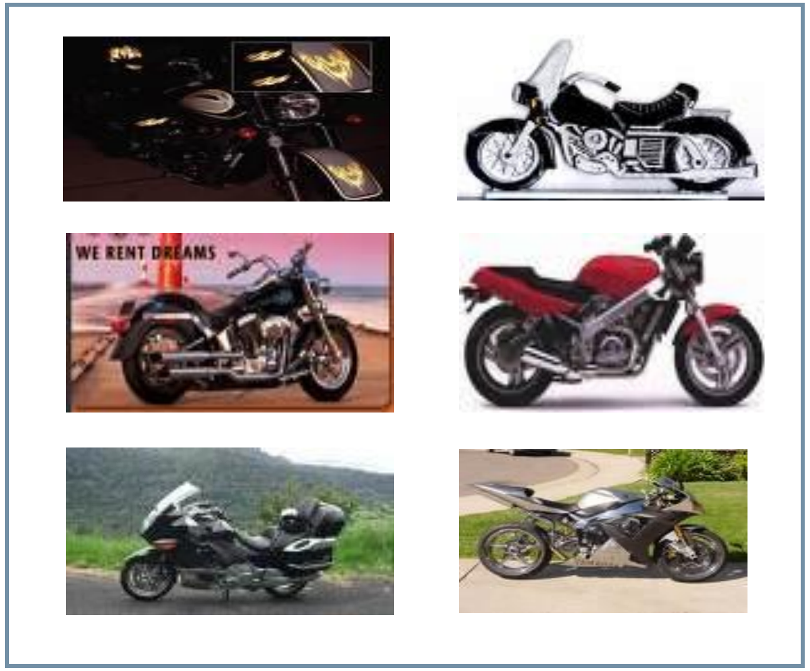
- A discrete set, as in classification $\Lambda = \{"\text{brown}", "\text{green}", "\text{blue}"\}$ (e.g., possible eye colors)
- An ordinal set (often continuous set, $\mathbb{R}$) in case of regression.

Λ can be also multivariate (e.g., regressing weight and height of an individual or estimating they eye colors and heirs color)

Training Set for (binary) Image Classification



Cars

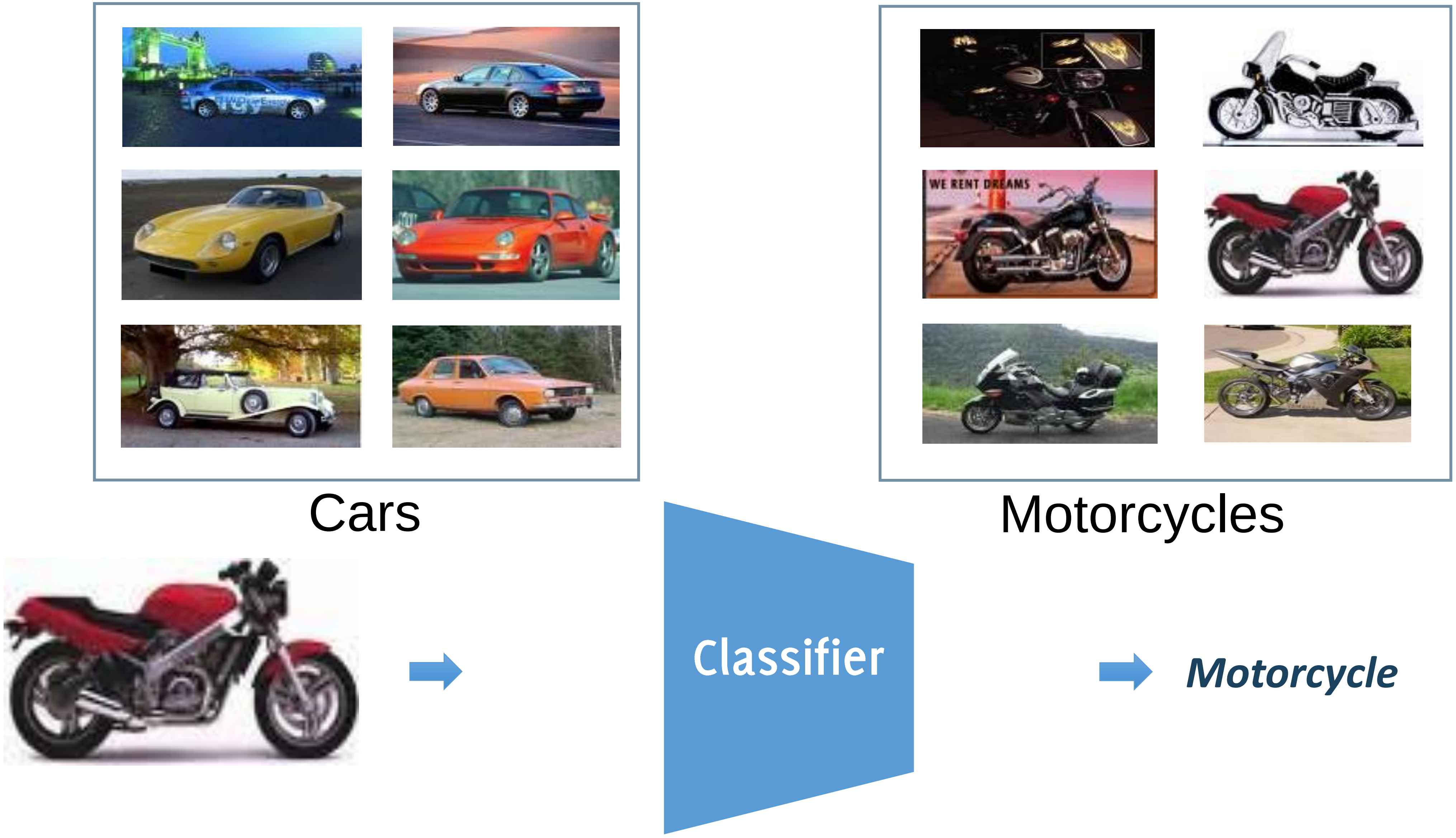


Motorcycles

$$TR = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

- $x_i \in \mathbb{R}^{R \times C \times 3}$ is the input image
- $y_i \in \{"car", "motorcycle"\}$

Inference Using the Trained Classifier



Supervised learning: Regression

12000 $

15000 $

6000 $

2000 $

8000 $

22000 $

4000 $

28000 $

6000 $

35000 $

Training Set for Regression

12000 $

15000 $

6000 $

2000 $

8000 $

22000 $

4000 $

28000 $

6000 $

35000 $

$$TR = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

- $x_i \in \mathbb{R}^{R \times C \times 3}$ is the input image
- $y_i \in \mathbb{R}$

Supervised learning: Regression

12000 $

15000 $

6000 $

2000 $

8000 $

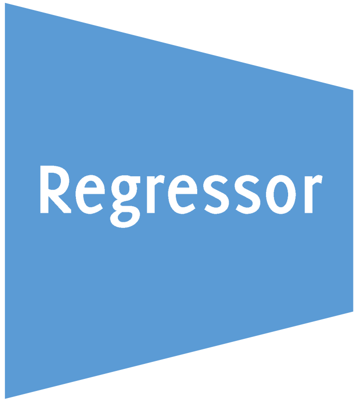
22000 $

4000 $

28000 $

6000 $

35000 $



3800 $

Remarks

- Number of classes can be larger than two (multiclass classification, e.g., {"car", "motorcycle","truck"})
- The input size in general needs to be fixed
- The number of outputs for regression can be larger (multivariate regression, e.g., estimating cost and weight of the vehicle)
- Training a Classifier or a Regressor requires different losses
- Difference between classification or regression is not only on the fact that Λ discrete, but whether it is ordinal
  - Λ categorical (no ordinal) -> classification
  - Λ ordinal (either discrete or continuous) -> regression

Give a few examples of

Classification problem in images

-
-
-
-
-

Regression problems on images

-
-
-
-
-

Unupervised Learning

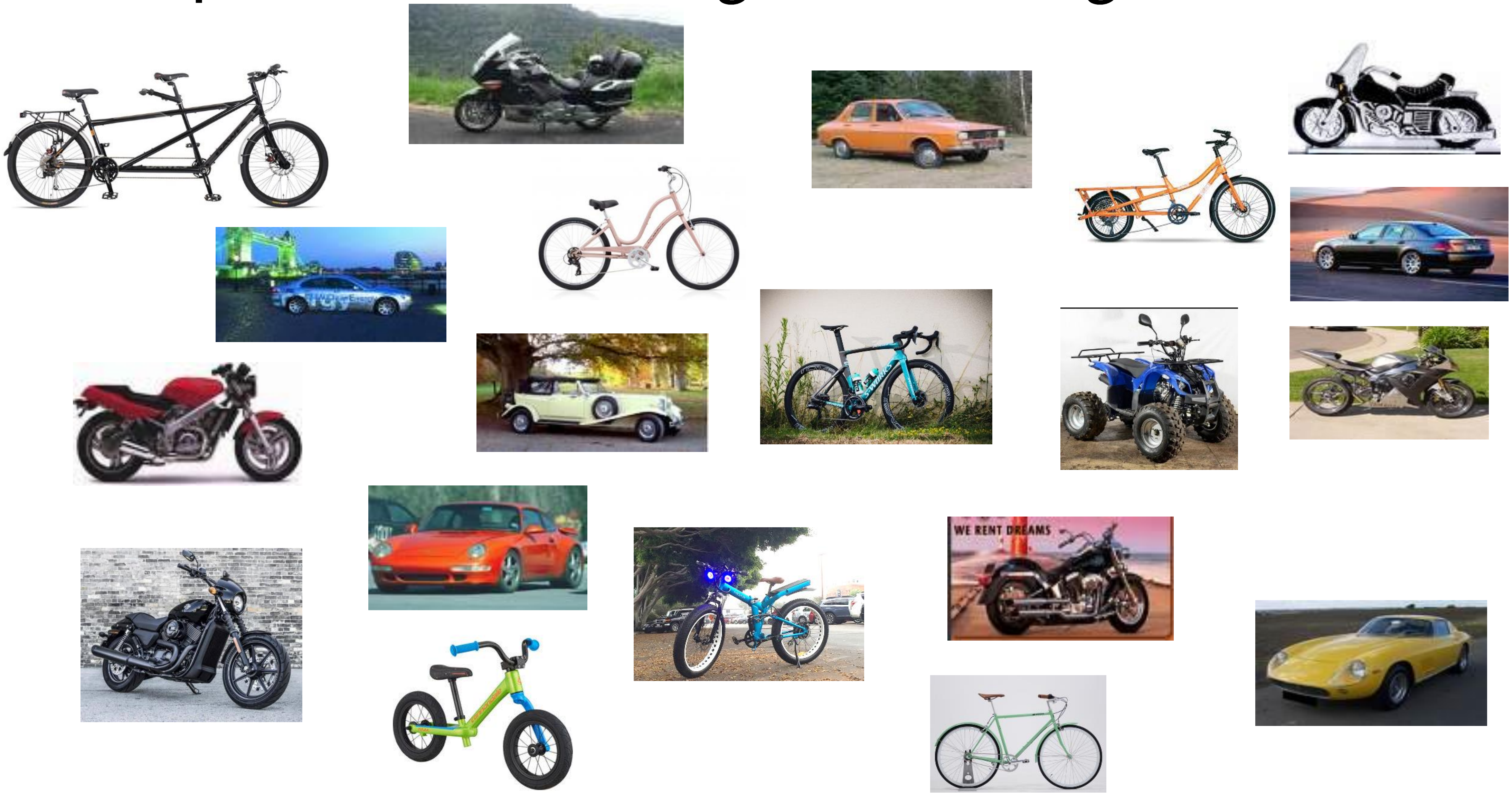
In **Unsupervised Learning**, the training set contains only inputs,

$$TR = \{x_1, \dots, x_n\}$$

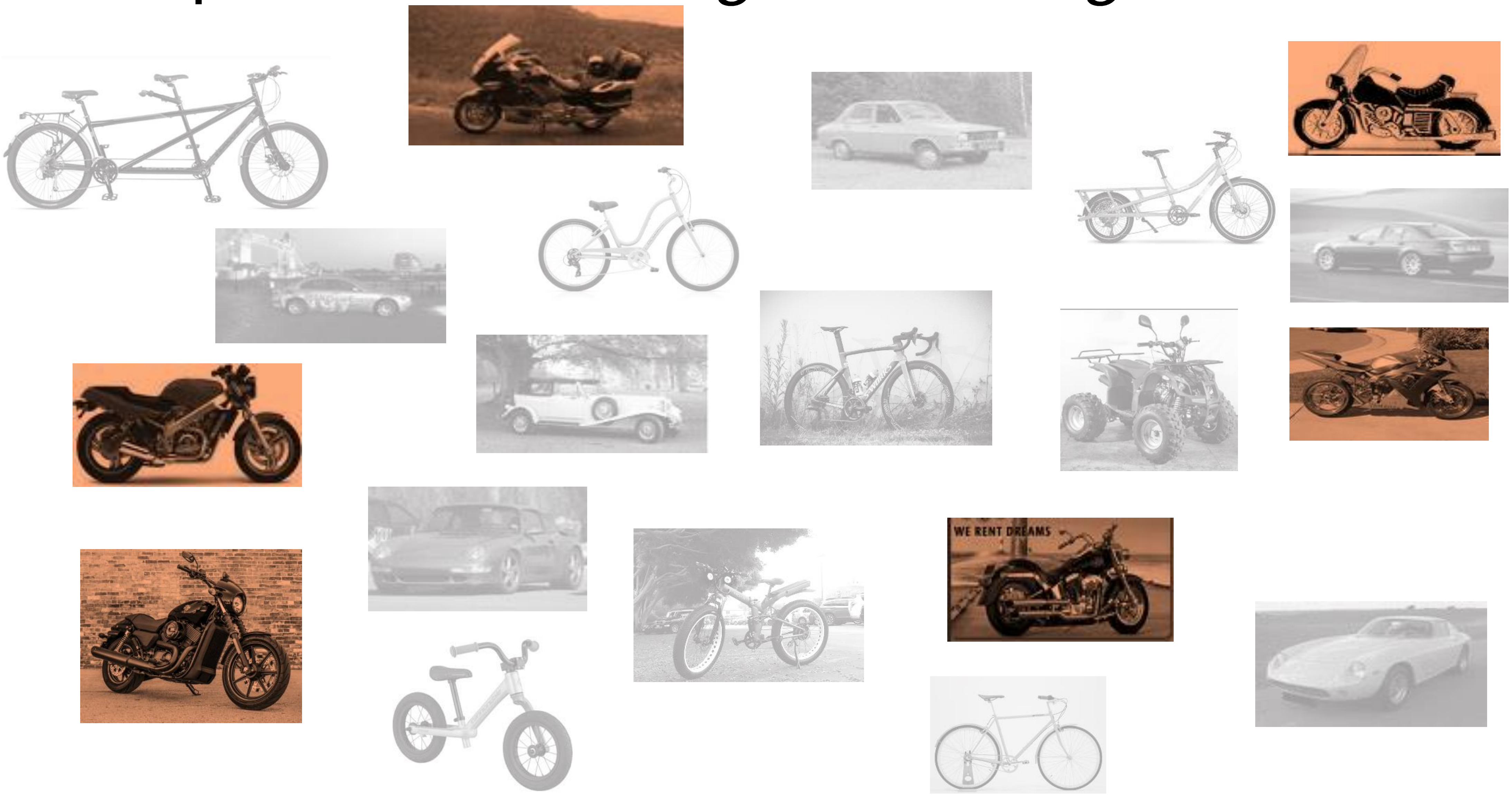
and the goal is to find structure in the data, like

- grouping or clustering of data points
- estimating probability density distribution
- detecting outliers
- …

Unsupervised learning: Clustering

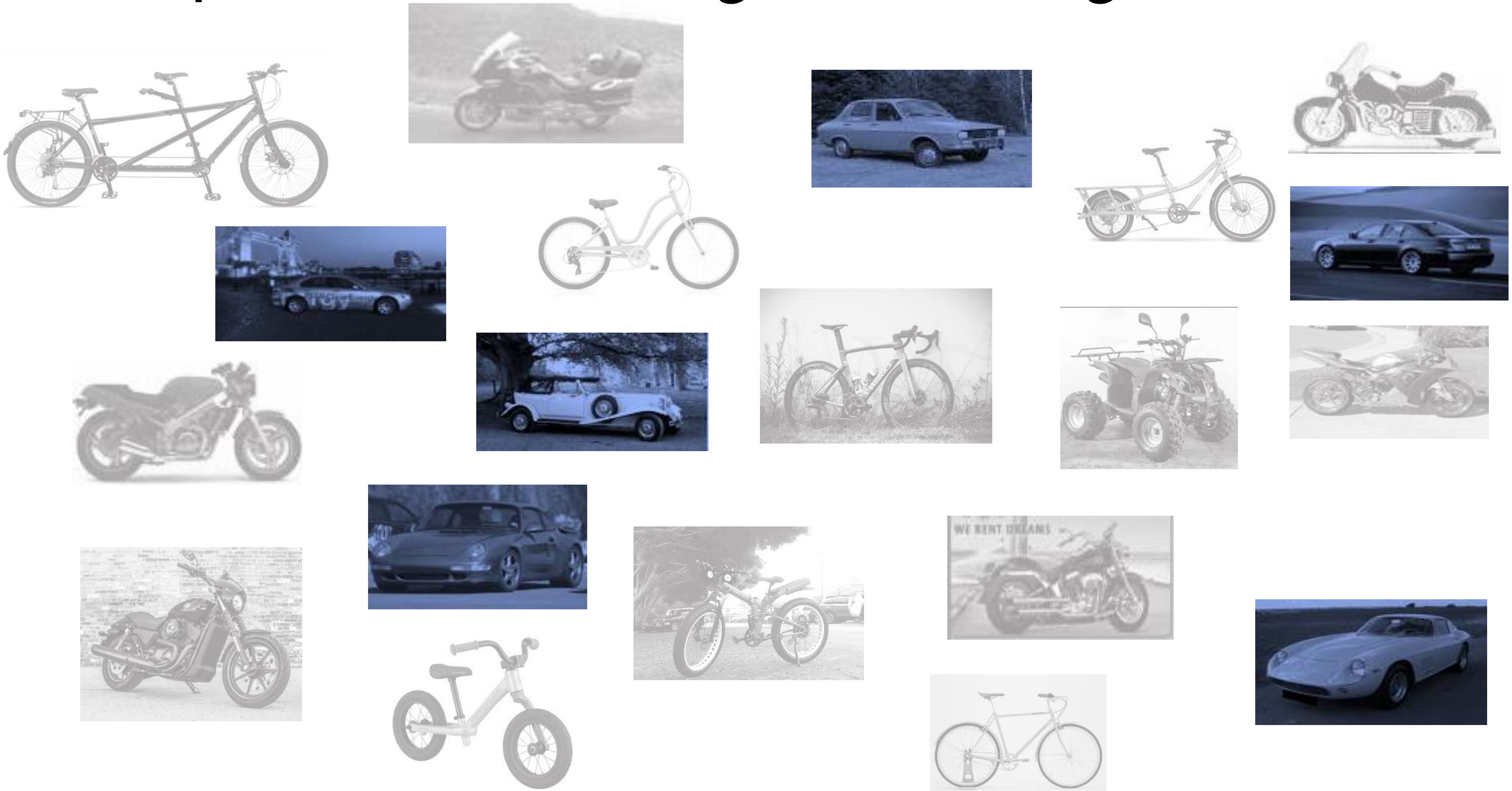


Unsupervised learning: Clustering

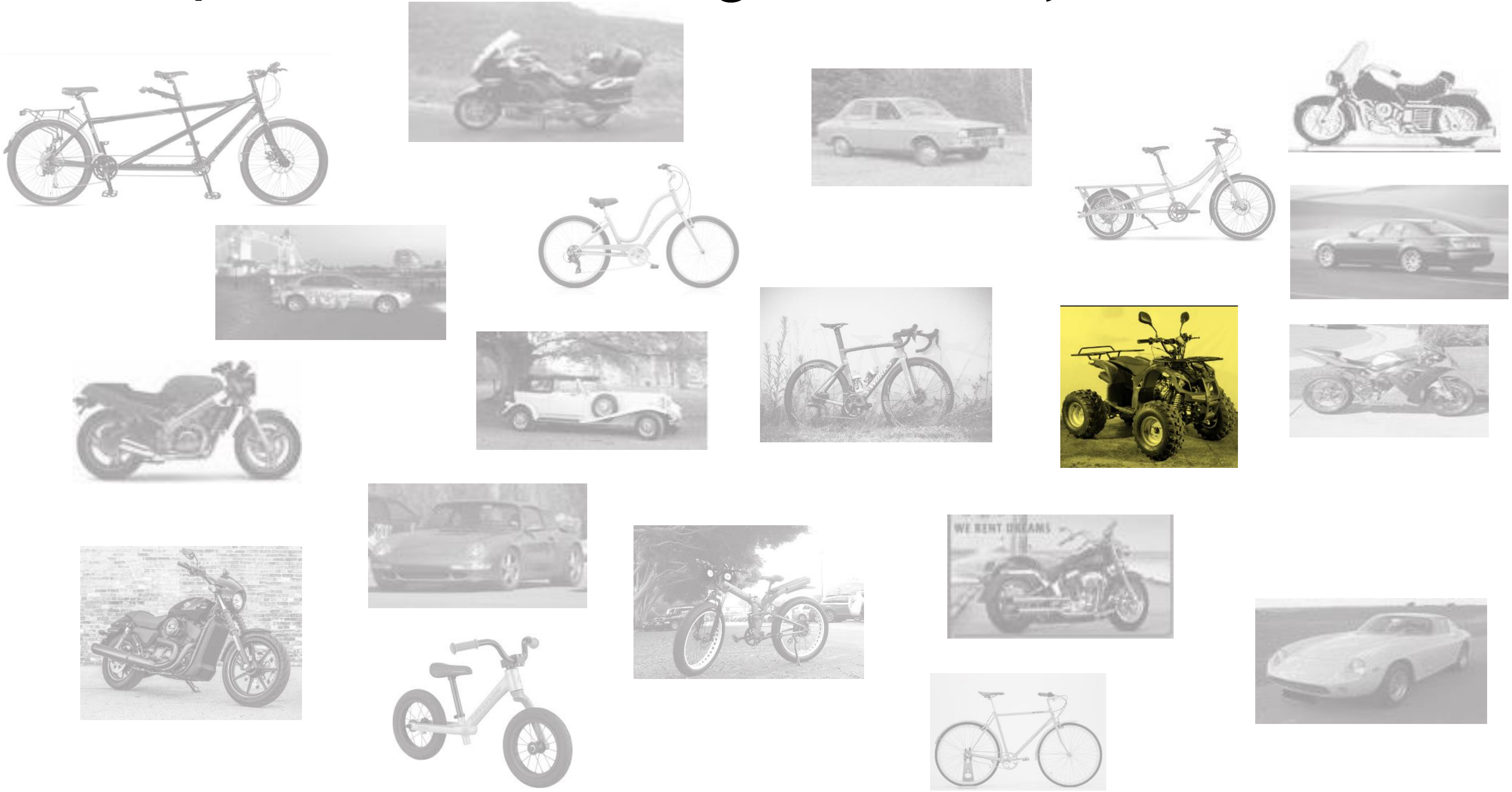


Unsupervised learning: Clustering

Unsupervised learning: Clustering



Unsupervised learning: Anomaly Detection



Unupervised Learning

In **Unsupervised Learning**, the training set contains only inputs,

$$TR = \{x_1, \dots, x_n\}$$

and the goal is to find structure in the data, like

- grouping or clustering of data points
- estimating probability density distribution
- detecting outliers
- …

We will see that in Deep Learning, Unsupervised learning (or self-supervised learning) can also be used to learn meaningful representations of data, to ease classification problem

The Image Classification Problem

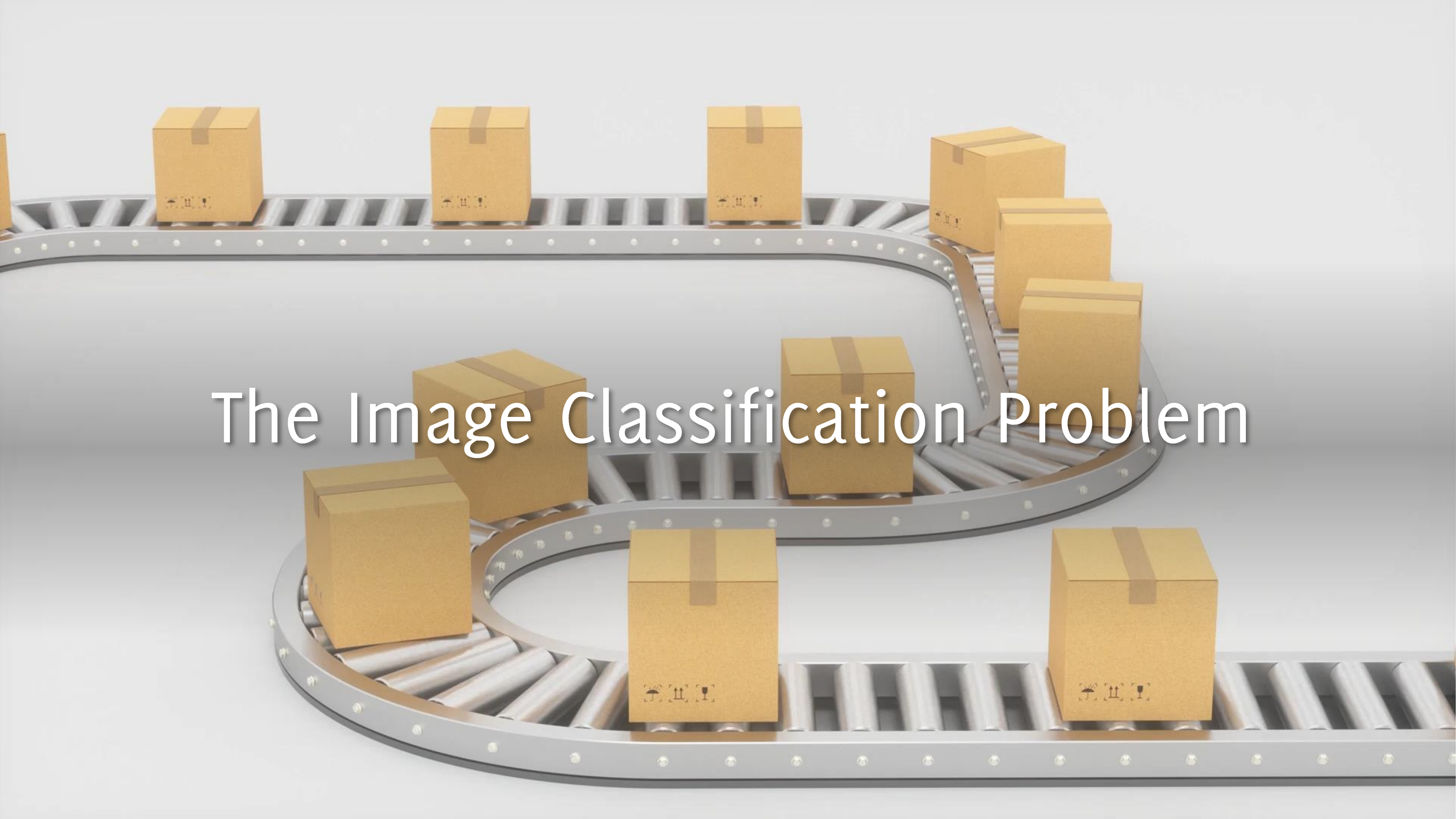


Image Classification

$$\Lambda = \{\text{"wheel"}, \text{"cars"}, \ldots, \text{"castle"}, \text{"baboon"}\}$$

x

"wheel"

x

"castle"

Image Classification

$\Lambda = \{\text{"wheel"}, \text{"cars"}, \ldots, \text{"castle"}, \text{"baboon"}\}$

x

⇨ “wheel” 65%, “tyre” 30%..

x

⇨ “castle” 55%, “tower” 43%..

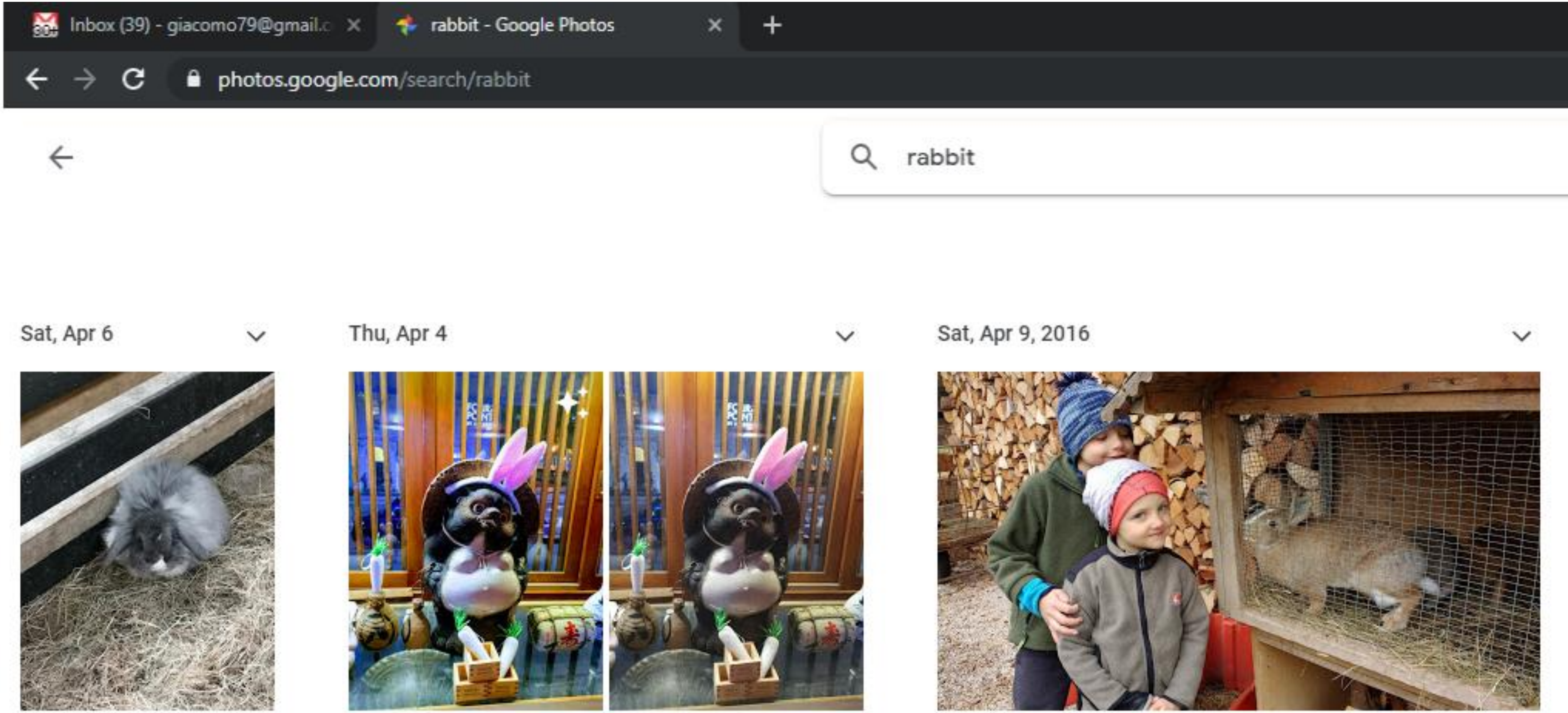
Image Classification, the problem

Assign to an input image $x \in \mathbb{R}^{R \times C \times 3}$:

- a label y from a fixed set of categories $\Lambda = \{\text{"wheel"}, \text{"cars"}, \ldots, \text{"castle"}, \text{"baboon"}\}$

$$x \rightarrow f_\theta(x) \in \Lambda$$

Image Classification Example



Is Image Classification a Challenging Problem?

Yes, it is…

First challenge: dimensionality

Images are very high-dimensional image data

CIFAR-10 dataset

The CIFAR-10 dataset contains 60000 images:

Each image is 32x32 RGB

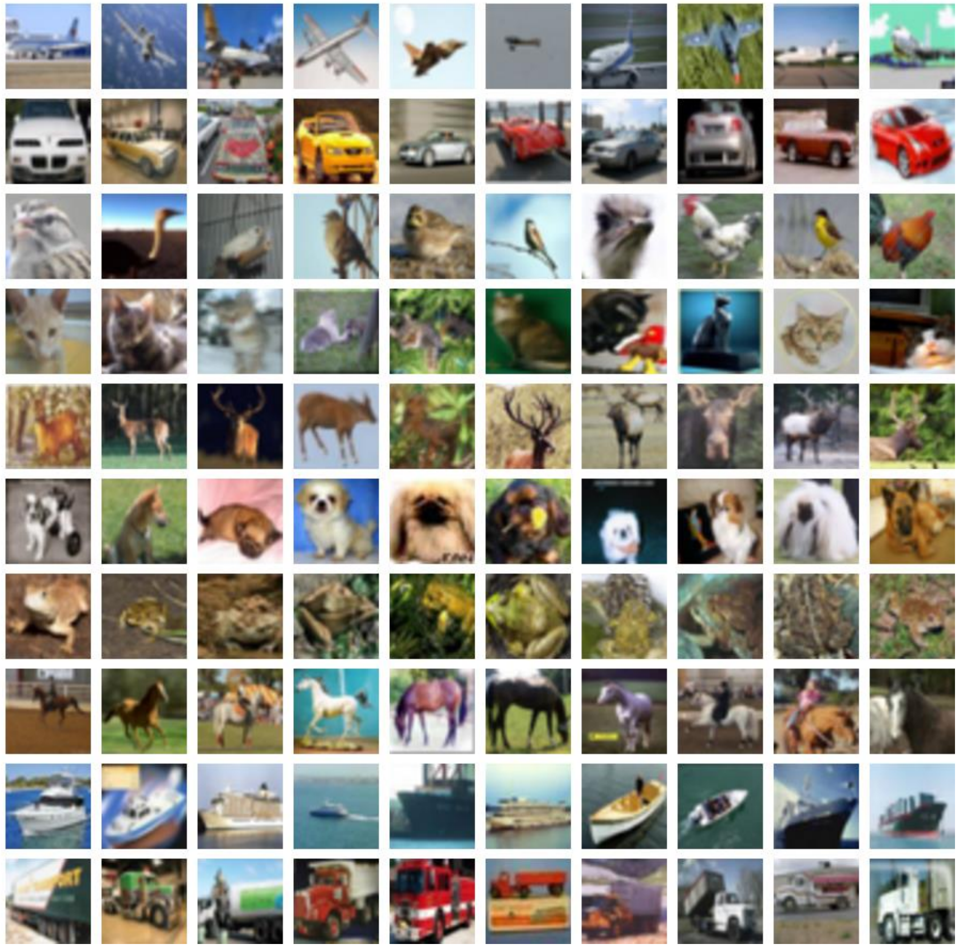
Images are in 10 classes

6000 images per class

Extremely small images, but high-dimensional:

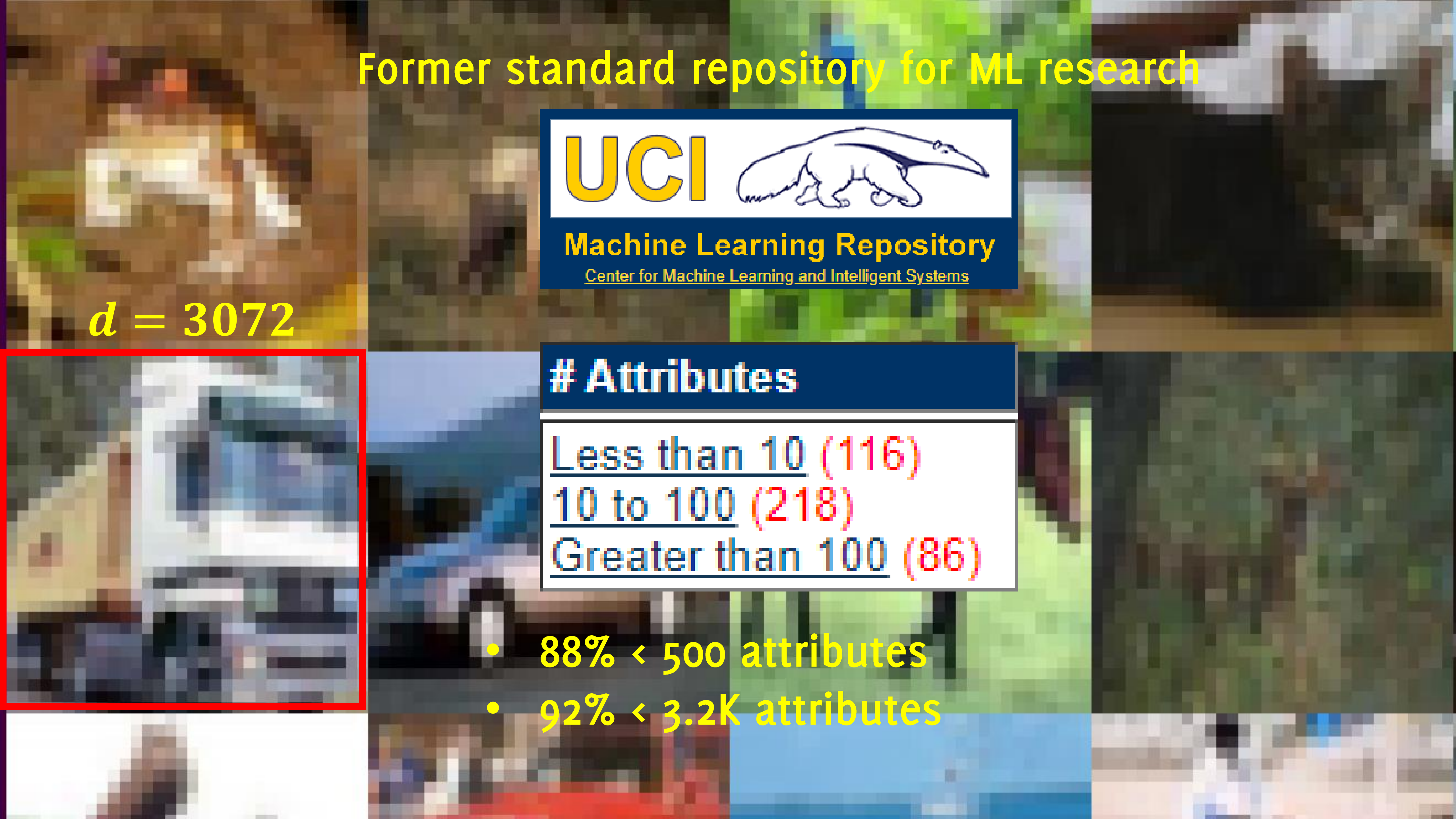
$d = 32 \times 32 \times 3 = 3072$

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck



Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.


This resolution is by far smaller than what we are used to
$d = 3072$


Former standard repository for ML research
$d = 3072$
UCI
Machine Learning Repository
Center for Machine Learning and Intelligent Systems
Attributes
Less than 10 (116)
10 to 100 (218)
Greater than 100 (86)
• 88% < 500 attributes
• 92% < 3.2K attributes

Former standard repo… r ML research

$d = 3072$

Bear in mind how large an image is (in terms of Bytes) when you'll be implementing your CNN… the whole batch and the corresponding activations have to be stored in memory!

…ributes

….2K attributes

Second challenge: label ambiguity

A label might not uniquely identify the image

Second challenge: label ambiguity

Man?

Beer?

Dinner?

Restaurant?

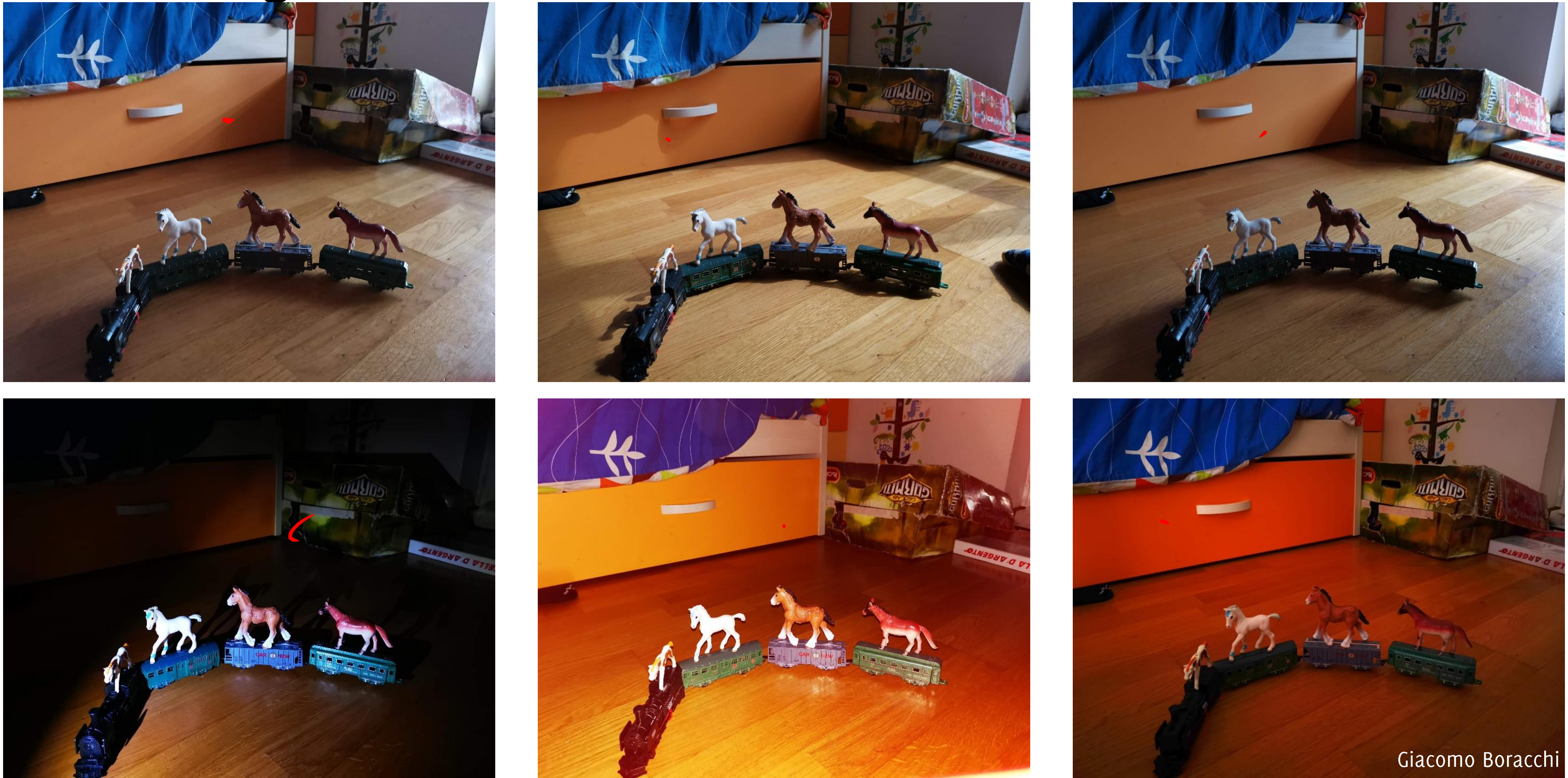
Sausages?

….

Third challenge: transformations

There are many transformations that change the image dramatically, while not its label

Changes in the Illumination Conditions



Giacomo Boracchi

Deformations

Copyright Christine Matthews

© Copyright Patrick Roper

View Point Change

… and many others

Occlusion

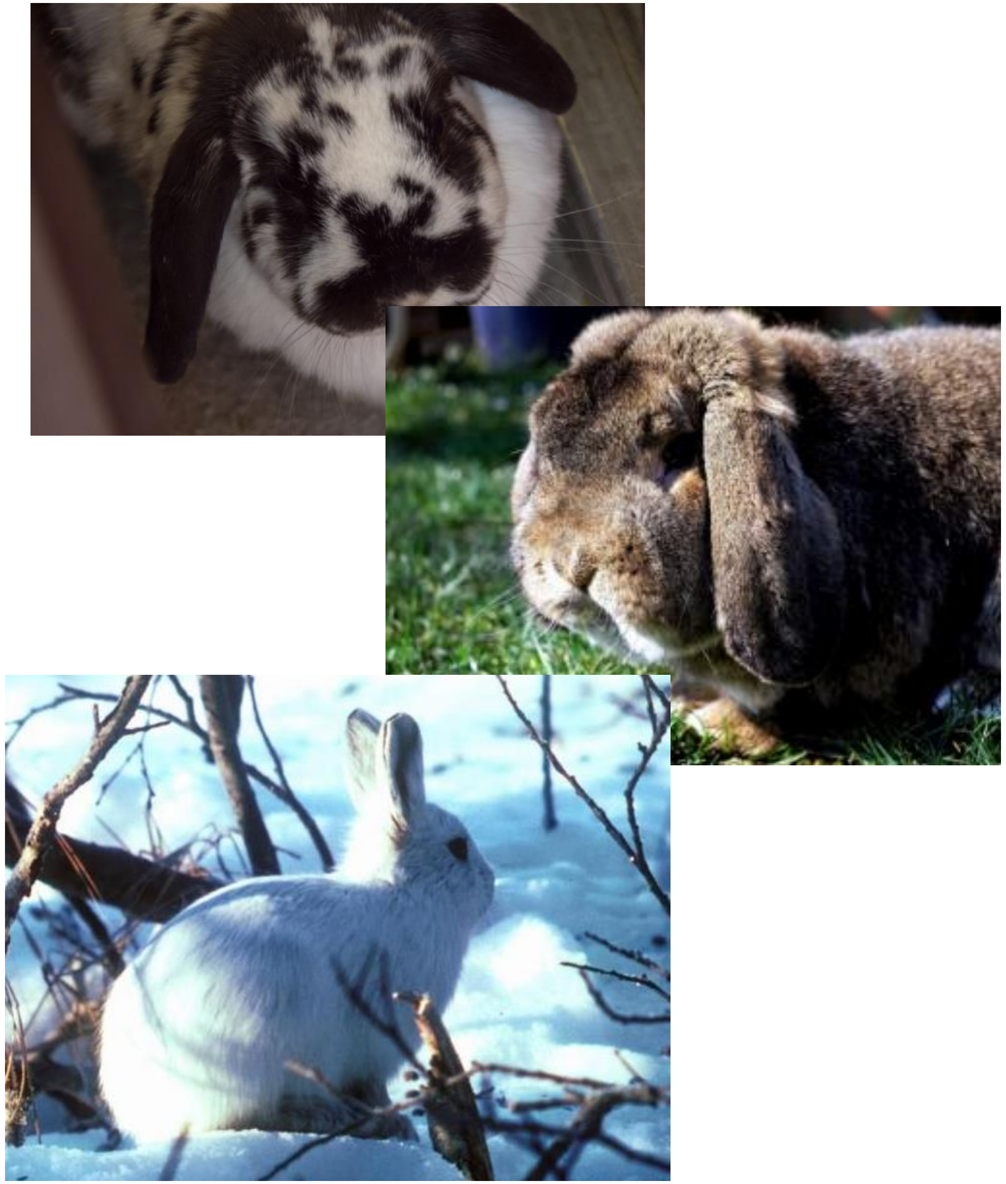
Background clutter

Scale variation

Fourth challenge: inter-class variability

Images in the same class might be dramatically different

Inter-class variability



Fifth problem: perceptual similarity

Perceptual similarity in images is not related to pixel-similarity

Nearest Neighborhood Classifiers for Images

Assign to each test image, **the label of the closest image** in the training set

$$\hat{y}_j = y_{j^*}, \qquad \text{being} \;\; j^* = \underset{i=1\ldots N}{\operatorname{argmin}}\, d(\,\boldsymbol{x}_j, \boldsymbol{x}_i)$$

Distances are typically measured as

$$d(\boldsymbol{x}_j, \boldsymbol{x}_i) = \left\|\boldsymbol{x}_j - \boldsymbol{x}_i\right\|_2 = \sqrt{\sum\nolimits_k \left([\boldsymbol{x}_j]_k - [\boldsymbol{x}_i]_k\right)^2}$$

Or

$$d(\boldsymbol{x}_j, \boldsymbol{x}_i) = \left|\boldsymbol{x}_j - \boldsymbol{x}_i\right| = \sum\nolimits_k \left|[\boldsymbol{x}_j]_k - [\boldsymbol{x}_i]_k\right|$$

Pixel-wise distance among images

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

→ 456

K-Nearest Neighborhood Classifiers for Images

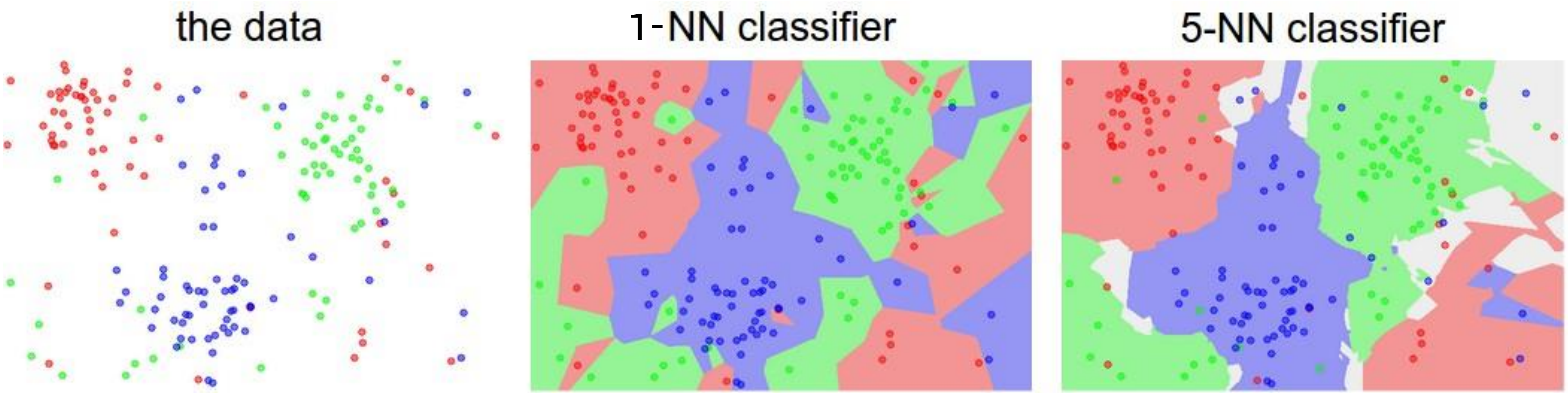
Assign to each test image, **the most frequent label among the $\boldsymbol{K}$ −closest images in the training set**

$$\hat{y}_j = y_{j^*}, \qquad \text{being } \; j^* \text{ the mode of } \mathcal{U}_K(\boldsymbol{x}_j)$$

where $\mathcal{U}_K(\boldsymbol{x}_j)$ contains the K closest training images to $\boldsymbol{x}_j$

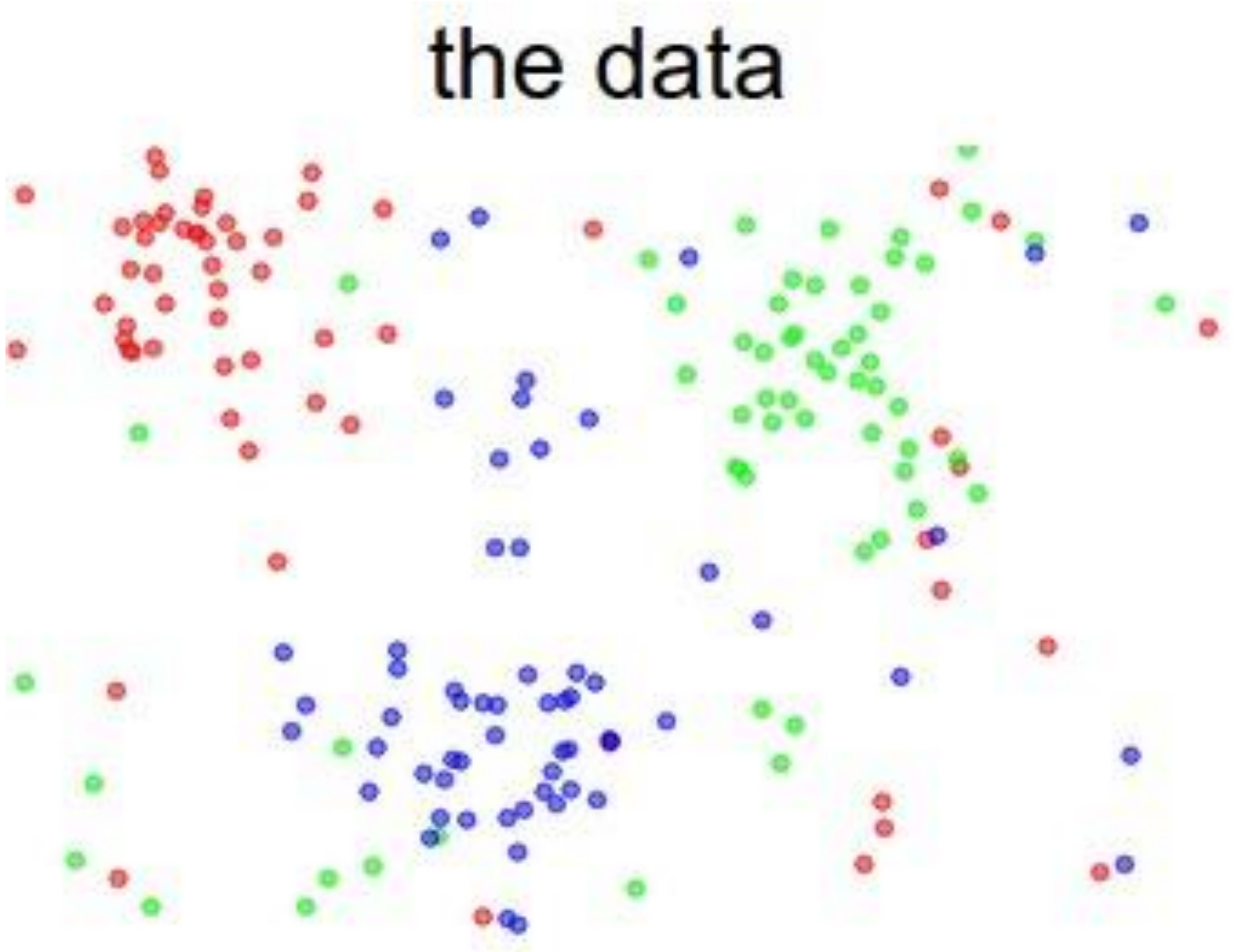
Setting the parameter K and the distance measure is an issue

Nearest Neighborhood Classifier (k-NN) for Images



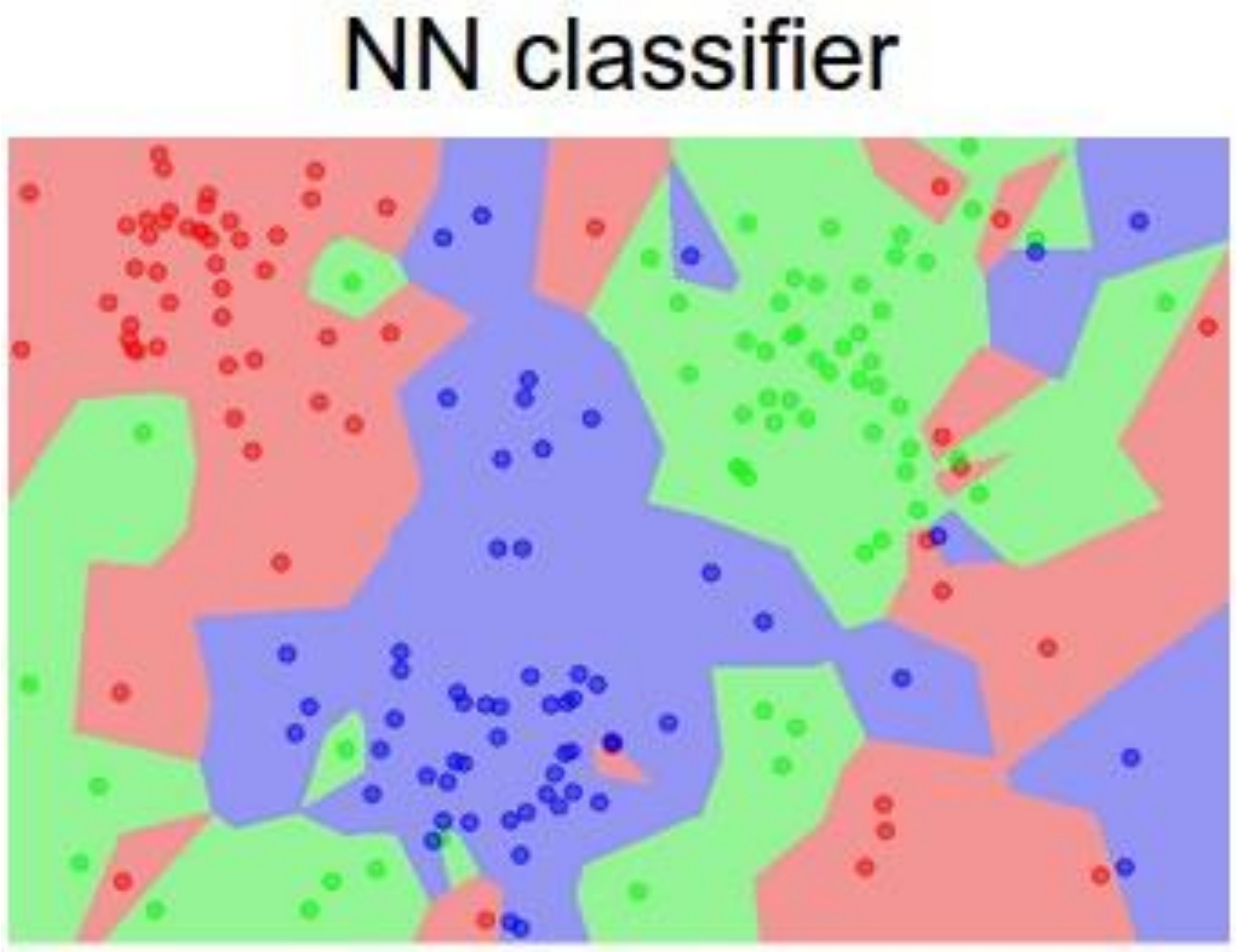
CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

k-NN for Images



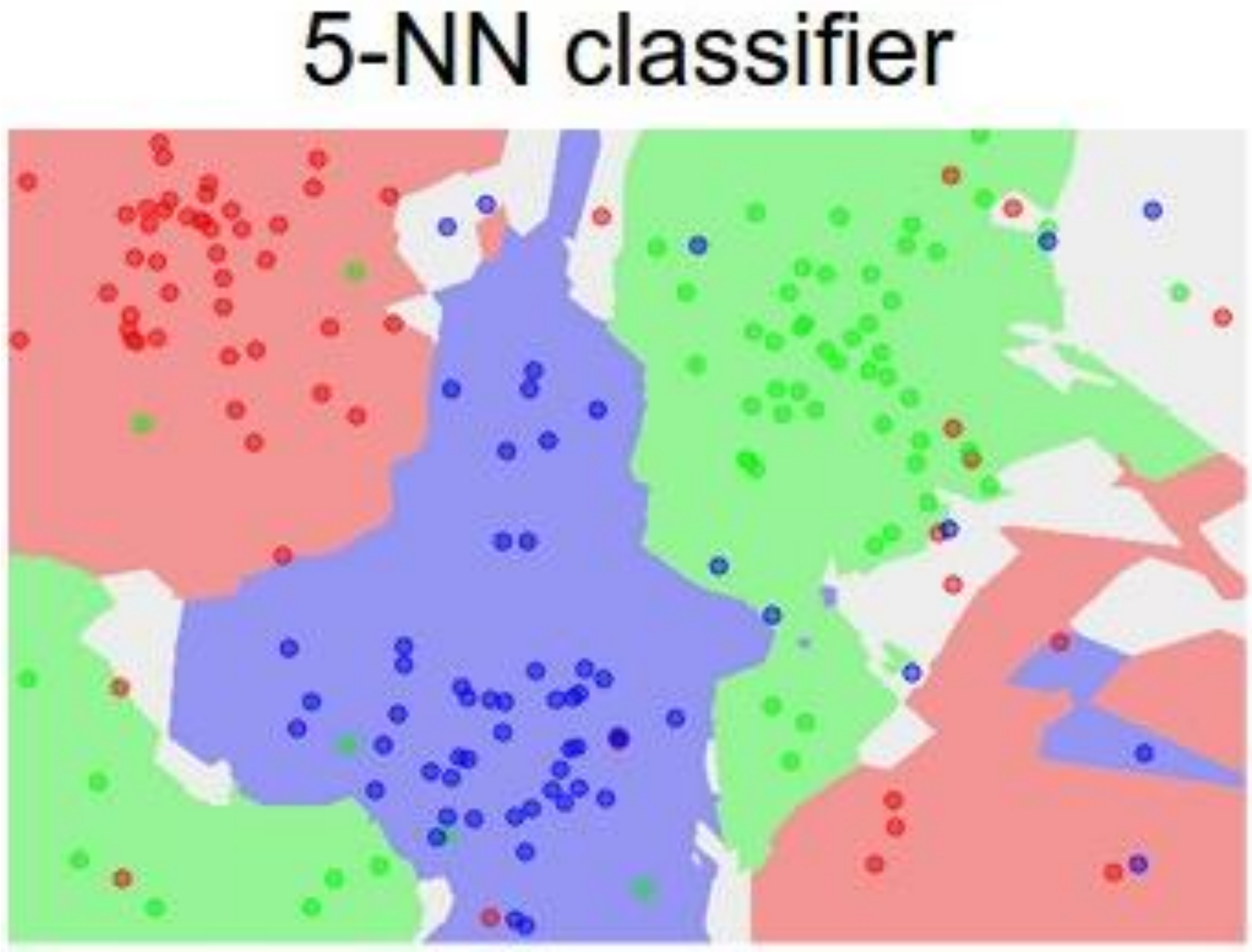
CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

k-NN for Images



CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

k-NN for Images



CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

k-NN for Images

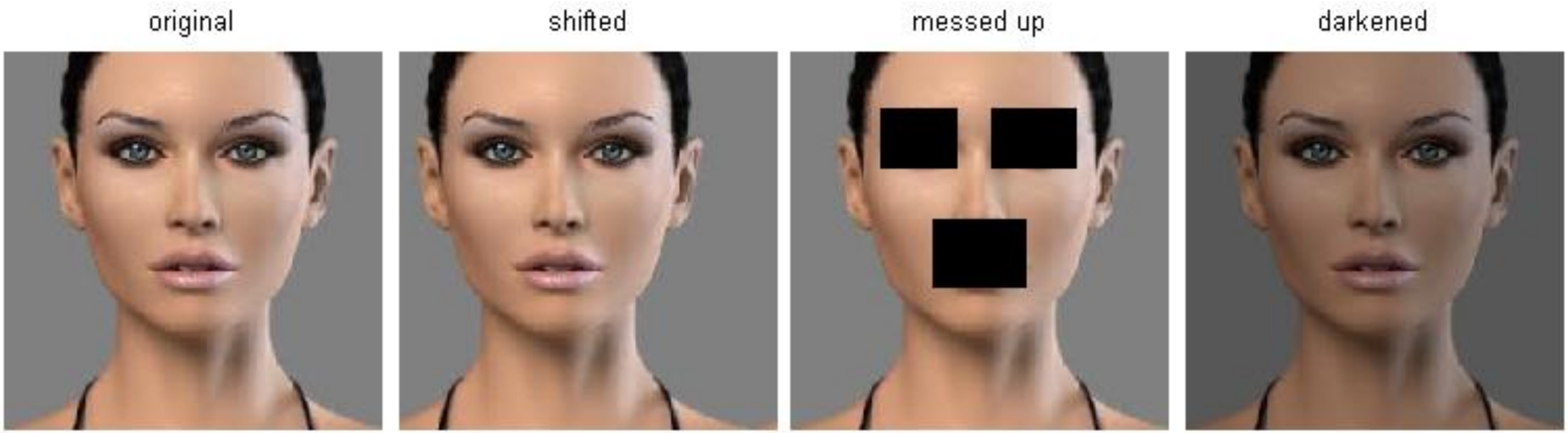
Pros:

- Easy to understand and implement
- It takes no training time

Cons:

- Computationally demanding at test time, when TR is large and d is also large.
- Large training sets must be stored in memory.
- Rarely practical on images: distances on high-dimensional objects are difficult to interpret.

Perceptual Similarity vs Pixel Similarity

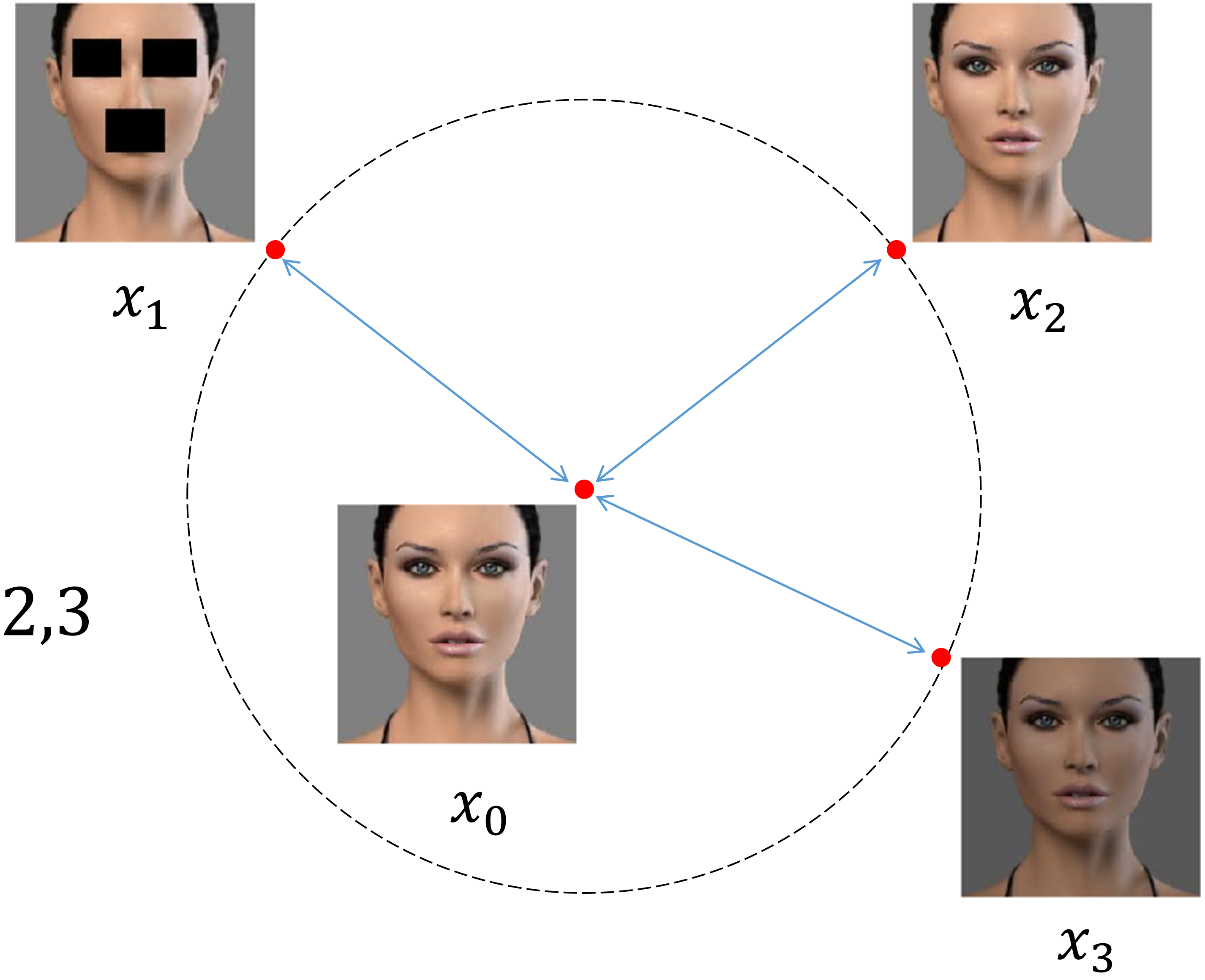


The three images have the same pixel-wise distance from the original one...

...but perceptually they are very different

CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

Perceptual Similarity vs Pixel Similarity

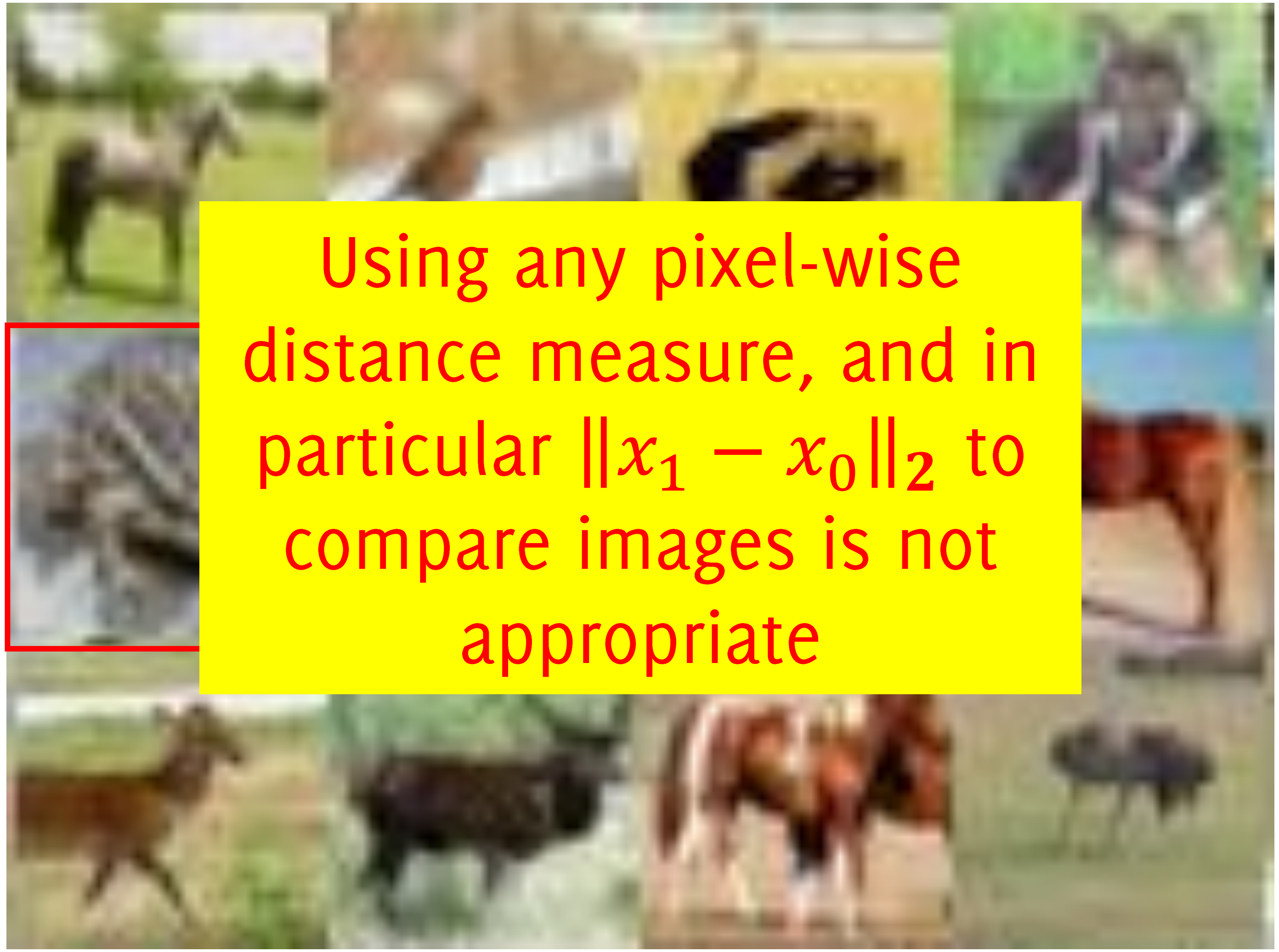


$$\left\| x_j - x_0 \right\|_2 \approx const \; j = 1,2,3$$

Let's see what happens on the whole CIFAR10 using t-SNE

On CIFAR10 we see exactly this problem

On CIFAR10 we see exactly this problem



Using any pixel-wise distance measure, and in particular $||x_1 - x_0||_2$ to compare images is not appropriate

On CIFAR10 we see exactly this problem

Some special model is needed to handle images…
we'll see in the next class!

Hand-Crafted Features

How images / signals were classified before deep learning

Assume you need to automatize this process

Assume you need to automatize this process

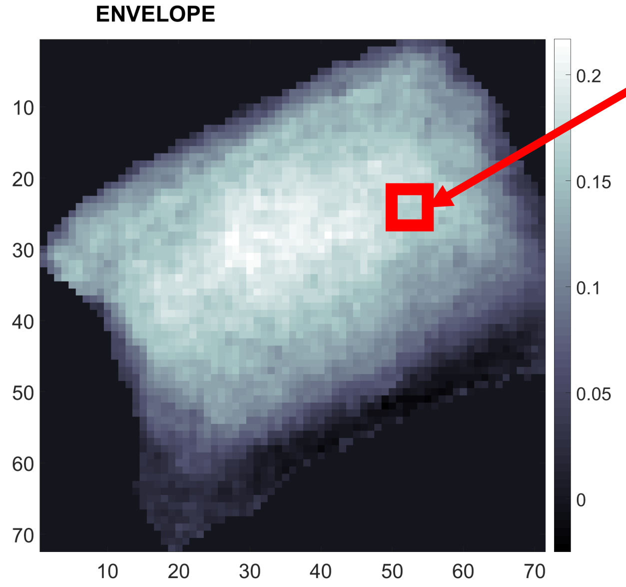
Assume you need to automatize this process

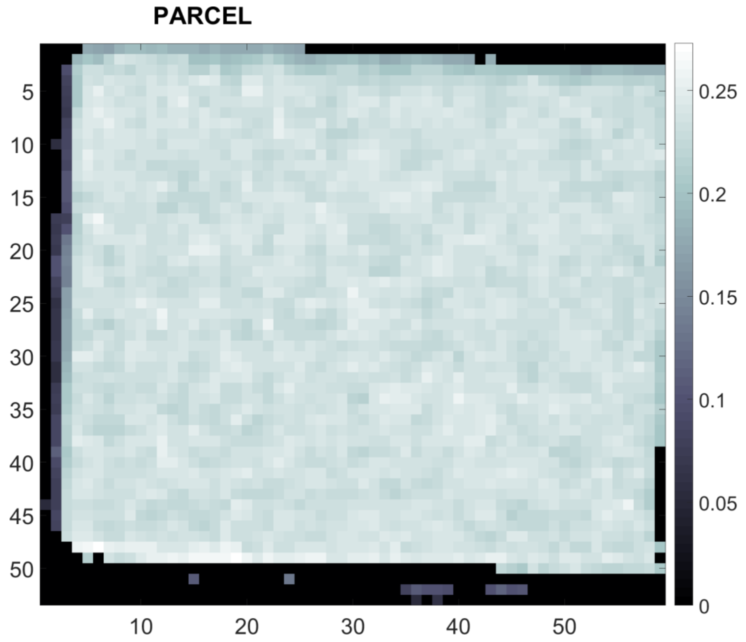
An Illustrative Example: Parcel Classification

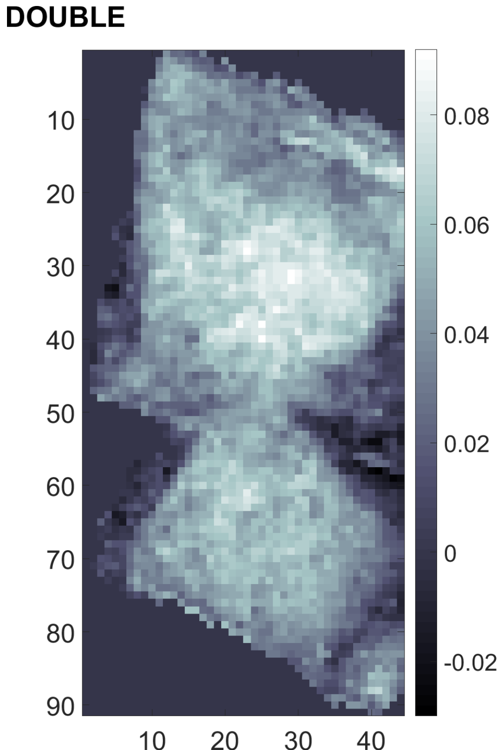
Images acquired from an RGB-D sensor:

- No color information provided
- Images of 3 classes
  - ENVELOPE
  - PARCEL
  - DOUBLE

Envelop height at that pixel



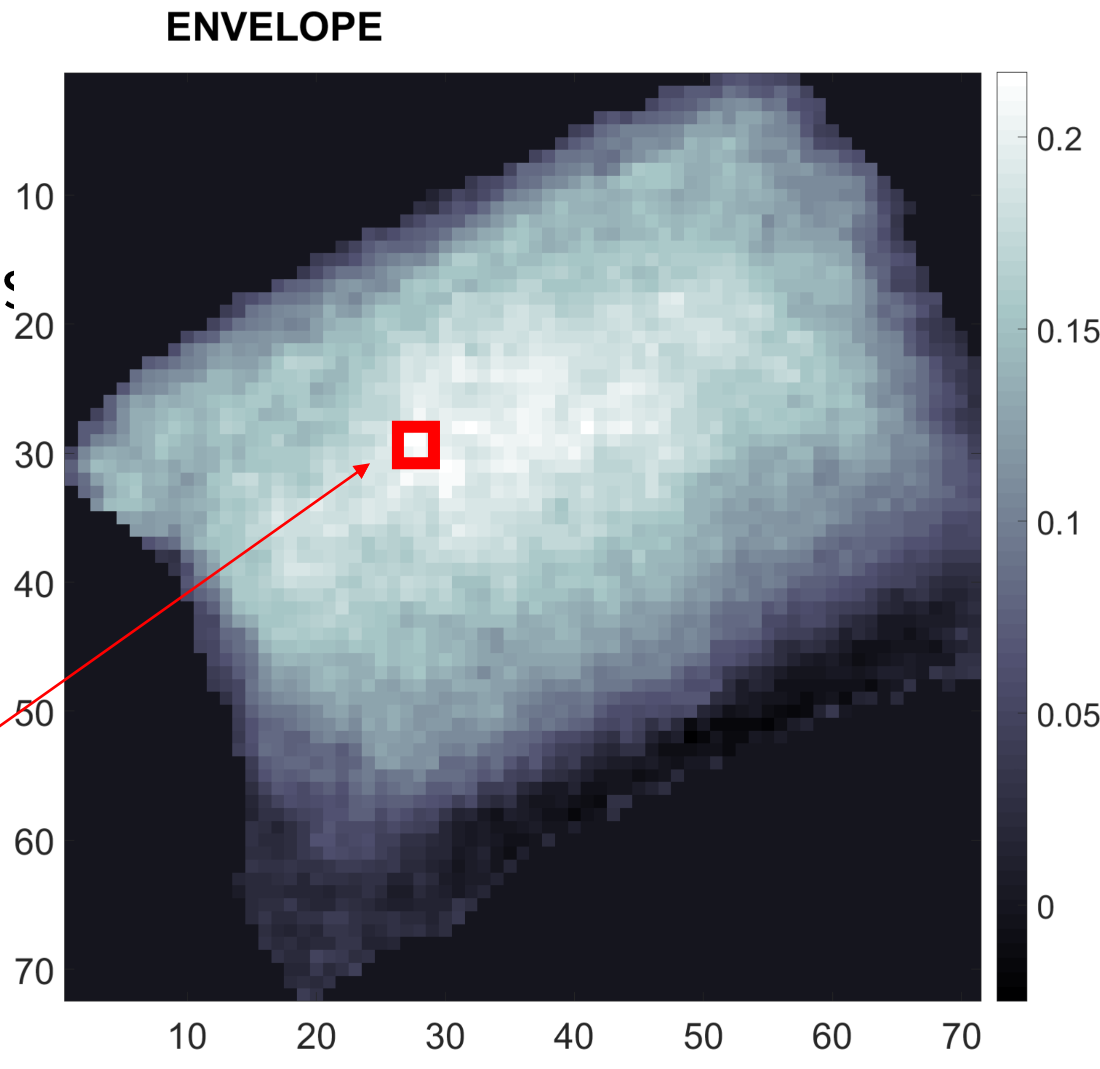




An Illustrative Example: Parcel Classification

Images acquired from a RGB-D sensor:

- No color information provided
- A few pixels report depth measurements
- Images of 3 classes
    - ENVELOPE
    - PARCEL
    - DOUBLE

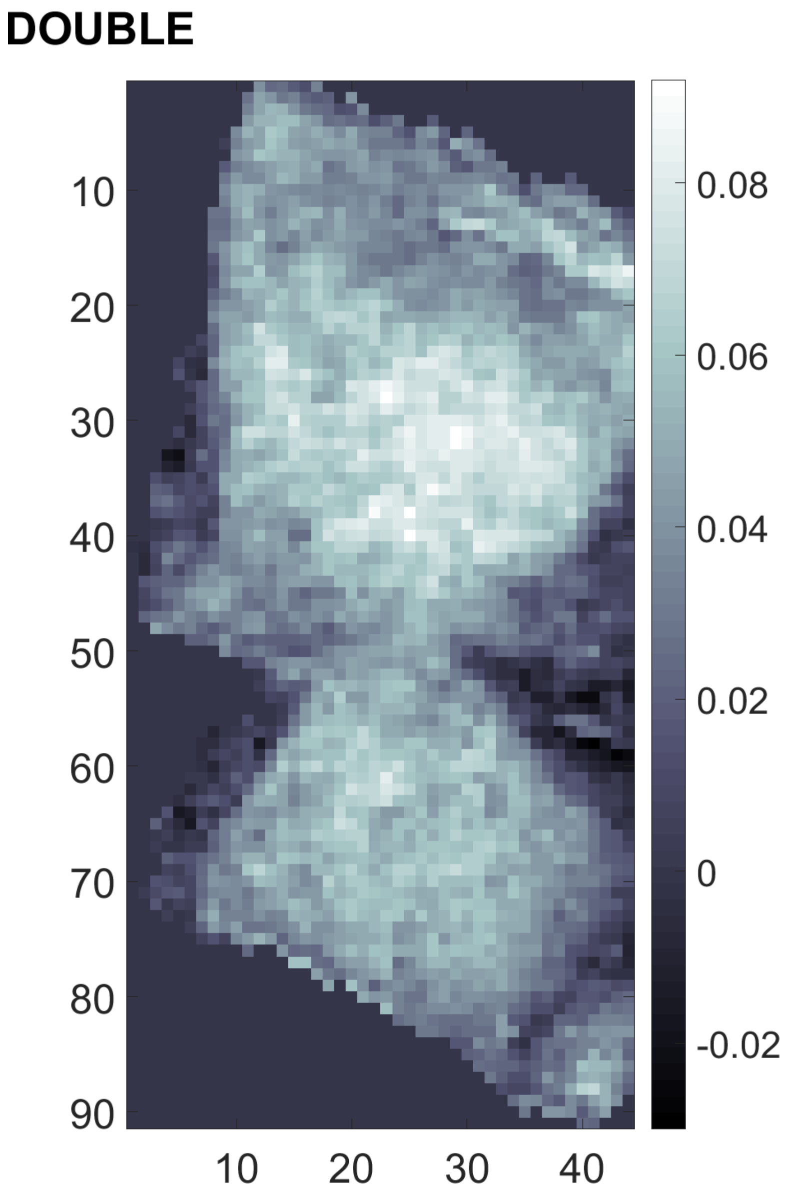


Envelop height at that pixel

An Illustrative Example: Parcel Classification

Images acquired from a RGB-D sensor:

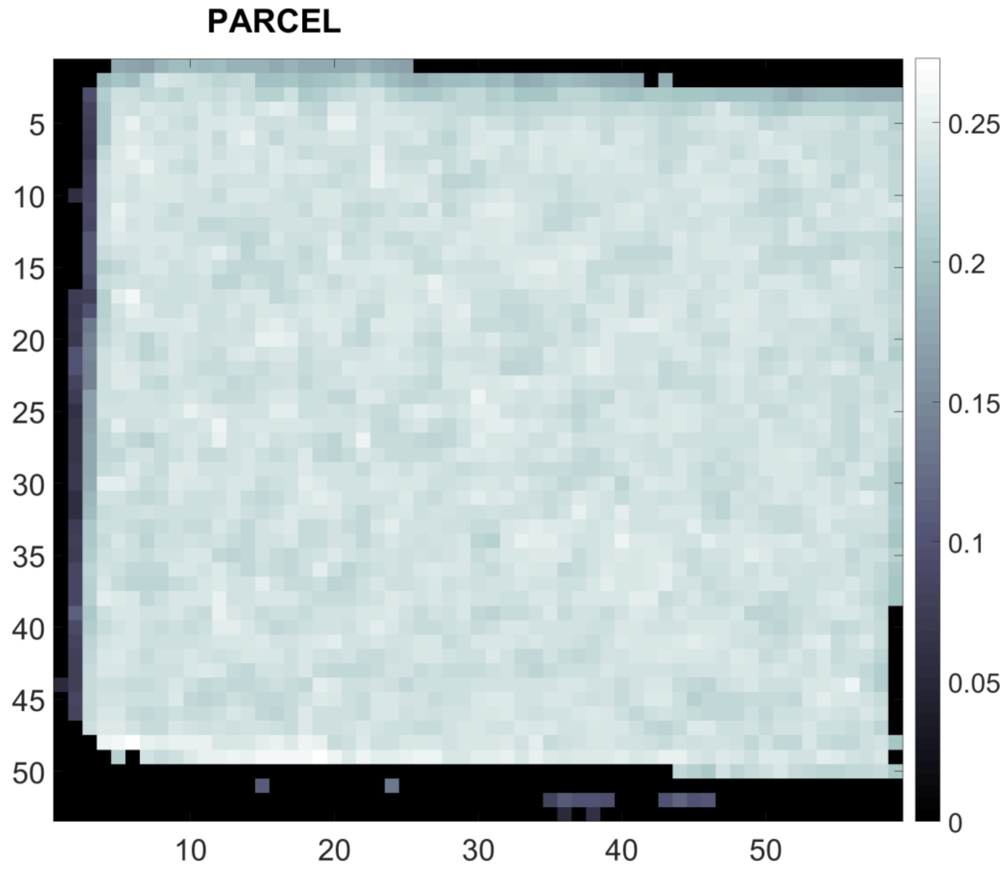
- No color information provided
- A few pixels report depth measurements
- Images of 3 classes
  - ENVELOPE
  - PARCEL
  - DOUBLE



An Illustrative Example: Parcel Classification

Images acquired from a RGB-D sensor:

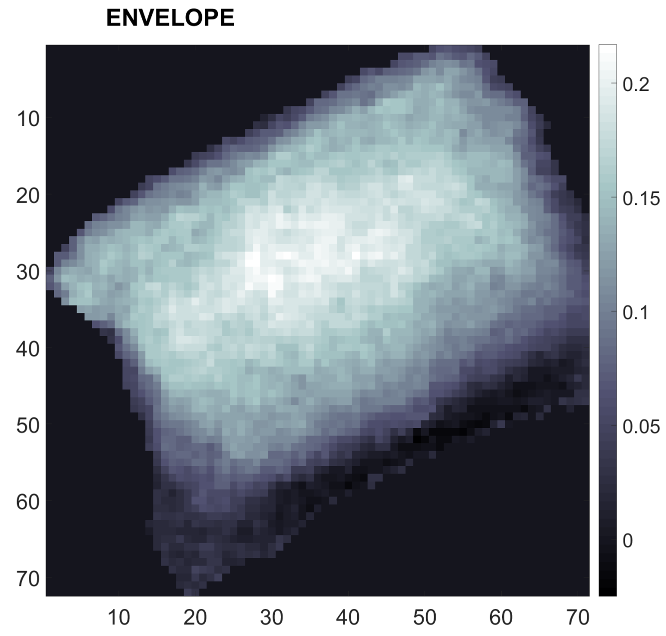
- No color information provided
- A few pixels report depth measurements
- Images of 3 classes
    - ENVELOPE
    - PARCEL
    - DOUBLE

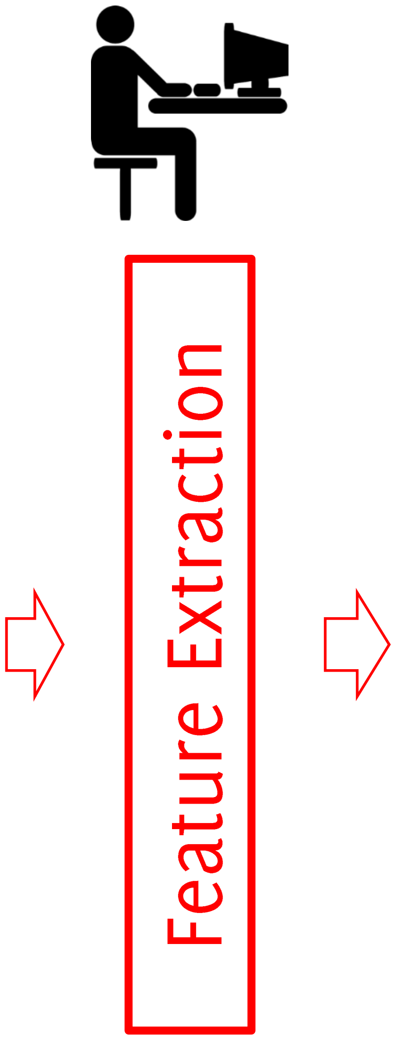


Hand Crafted Featues

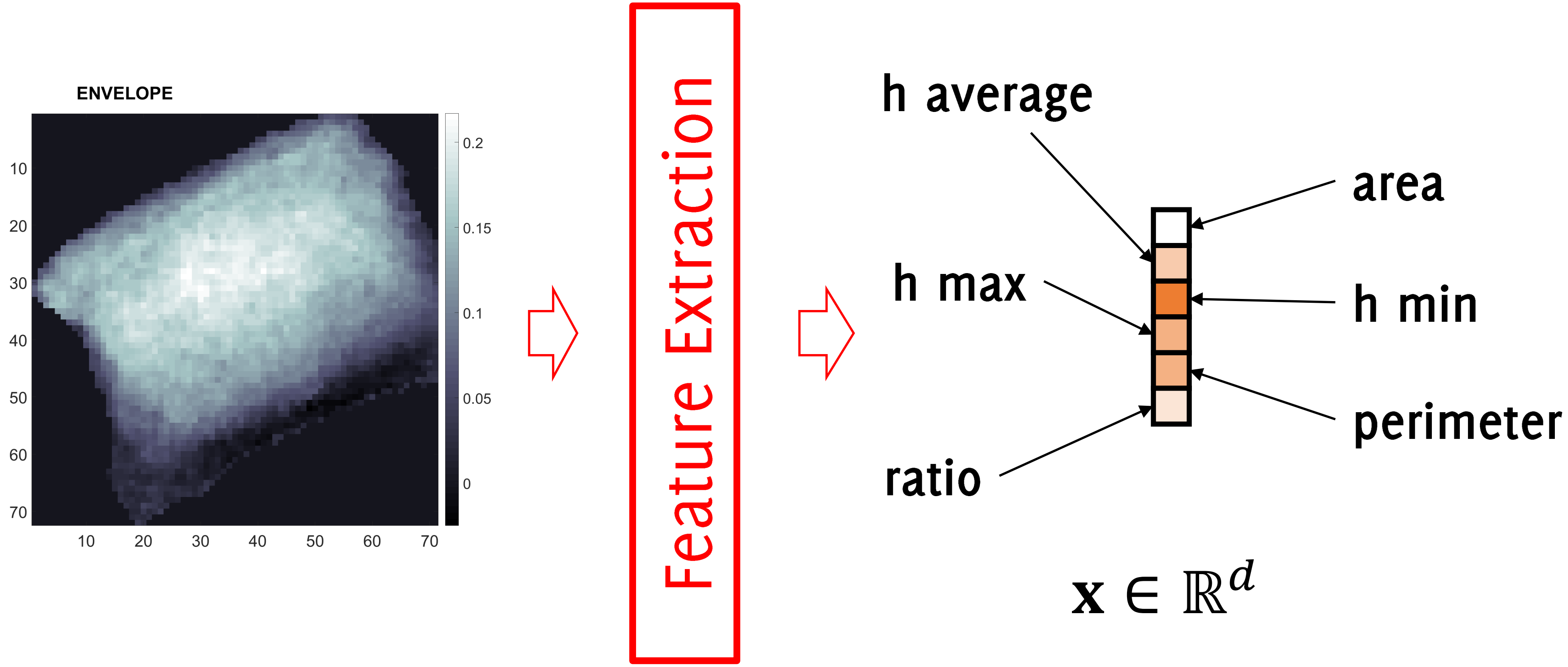
Engineers:

- know what's meaningful in an image (e.g. a specific color/shape, the area, the size)
- can implement algorithms to map this information in a set of measurements, a **feature vector**





Hand Crafted Featues



$\mathbf{x} \in \mathbb{R}^d$

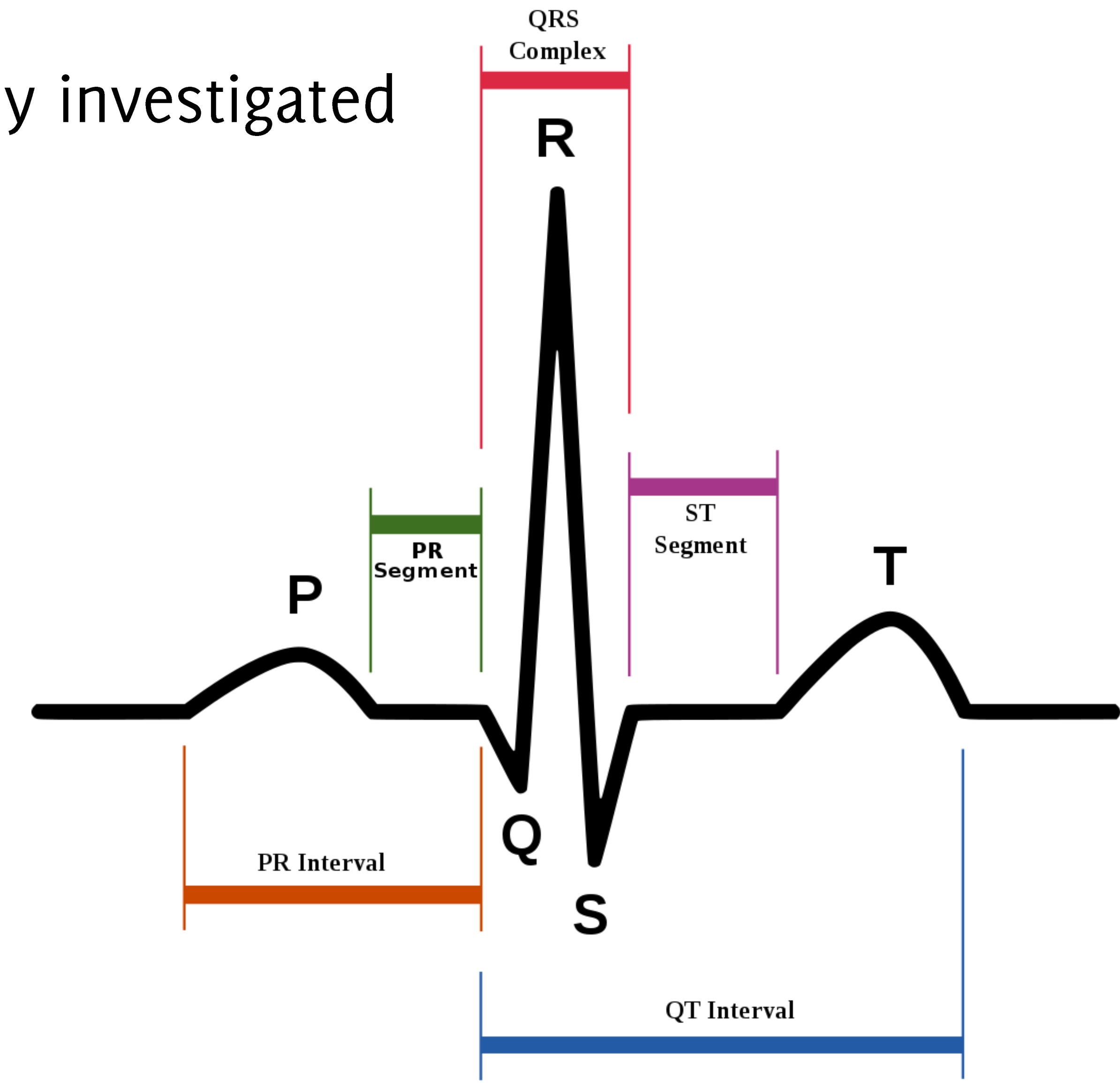
This is exactly what a doctor would to to classify ECG tracings

Heartbeats morphology has been widely investigated

Doctors know which patterns are meaningful for classifying each beat

Features are extracted from landmarks indicated by doctors:

e.g. QT distance, RR distance…



Created by Agateller (Anthony Atkielski), Public Domain, https://commons.wikimedia.org/w/index.php?curid=1560893

The Training Set

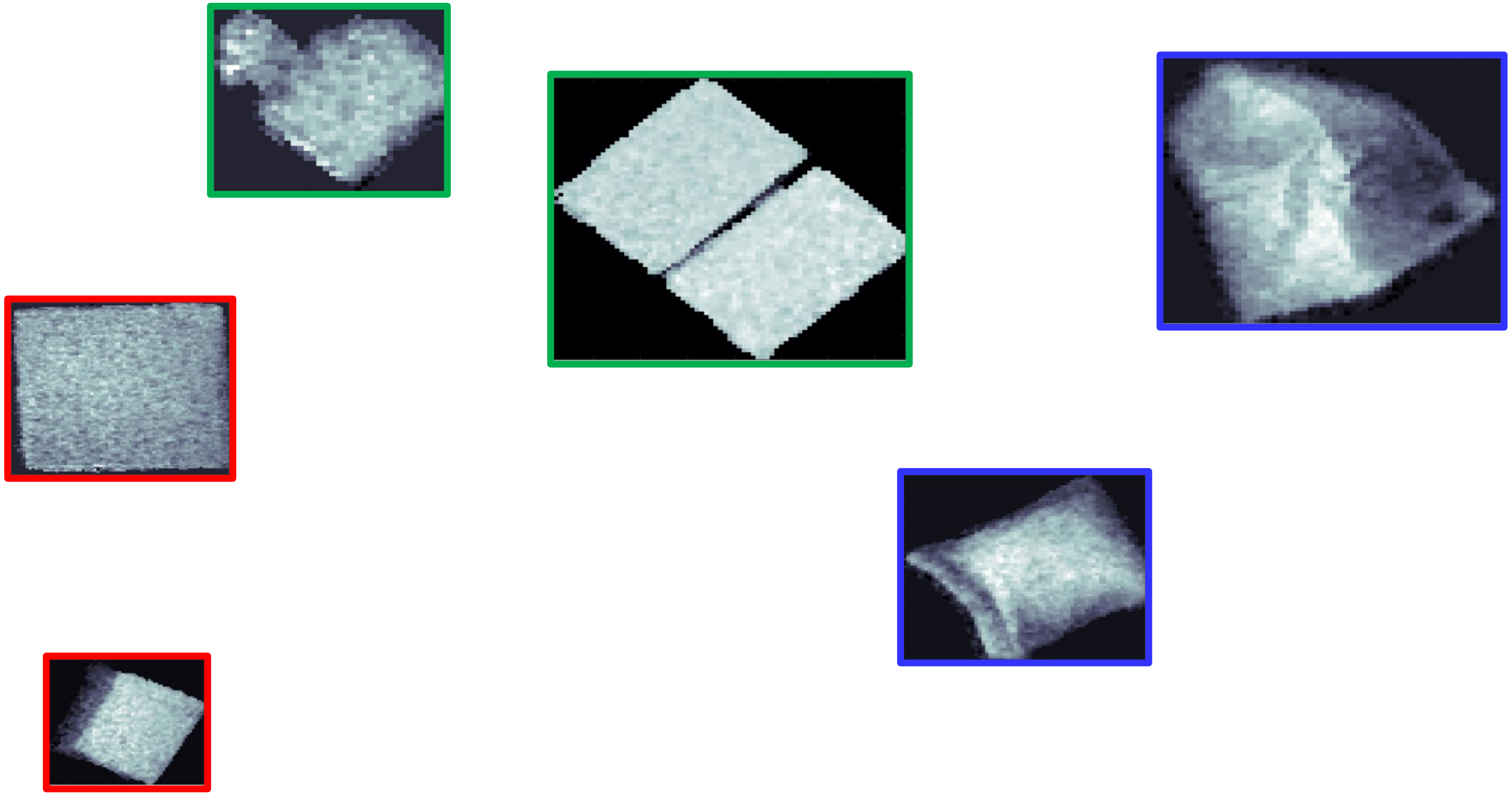
The training set is a set of annotated examples

$$TR = \{(x, y)_i, i = 1, \dots, N\}$$

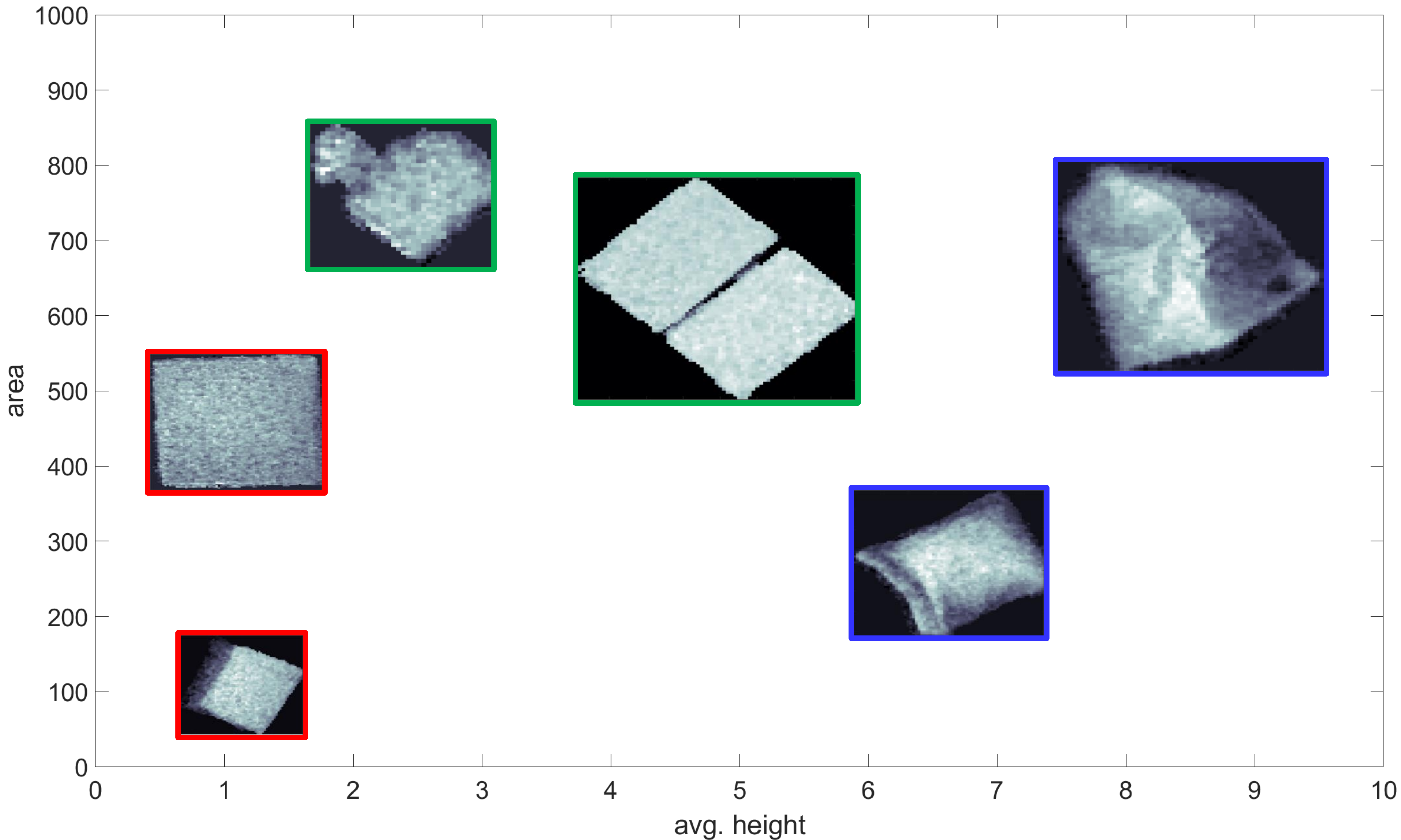
Each couple $(x, y)_i = (x_i, y_i)$ corresponds to:

- an image $x_i \in \mathbb{R}^{R \times C \times 3}$
- the corresponding label $y_i \in \Lambda$

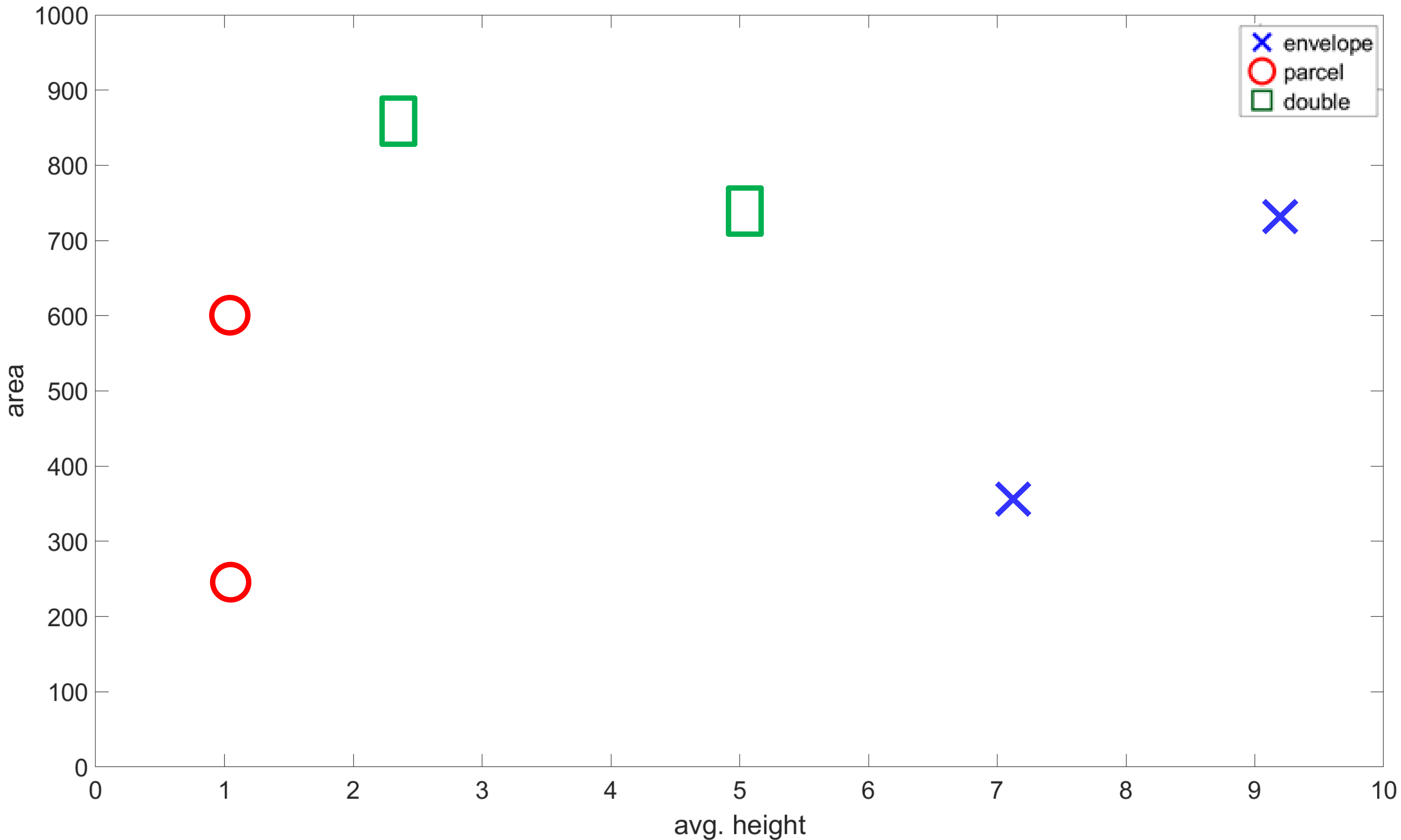
The Training Set: images + labels



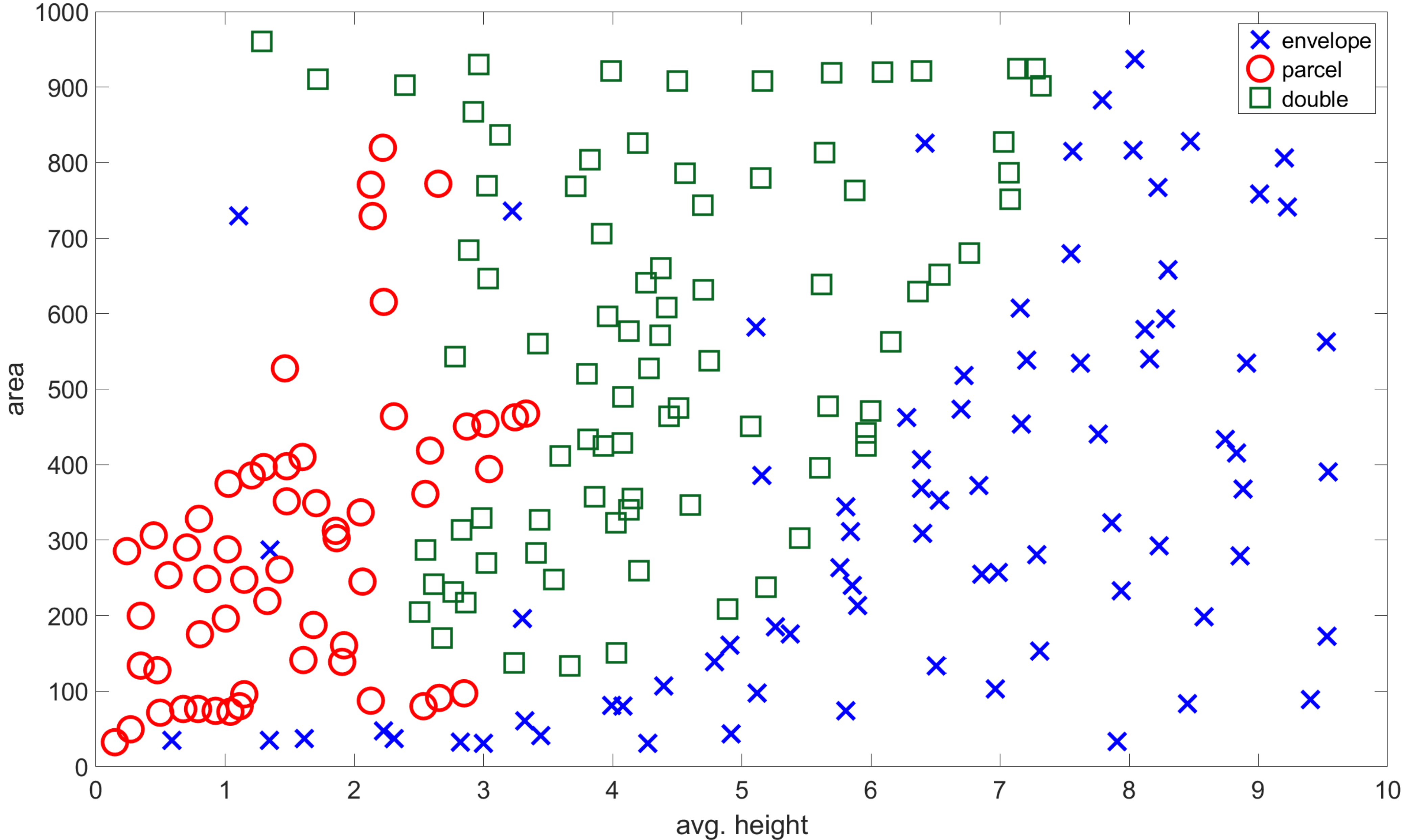
The Training Set: images + labels



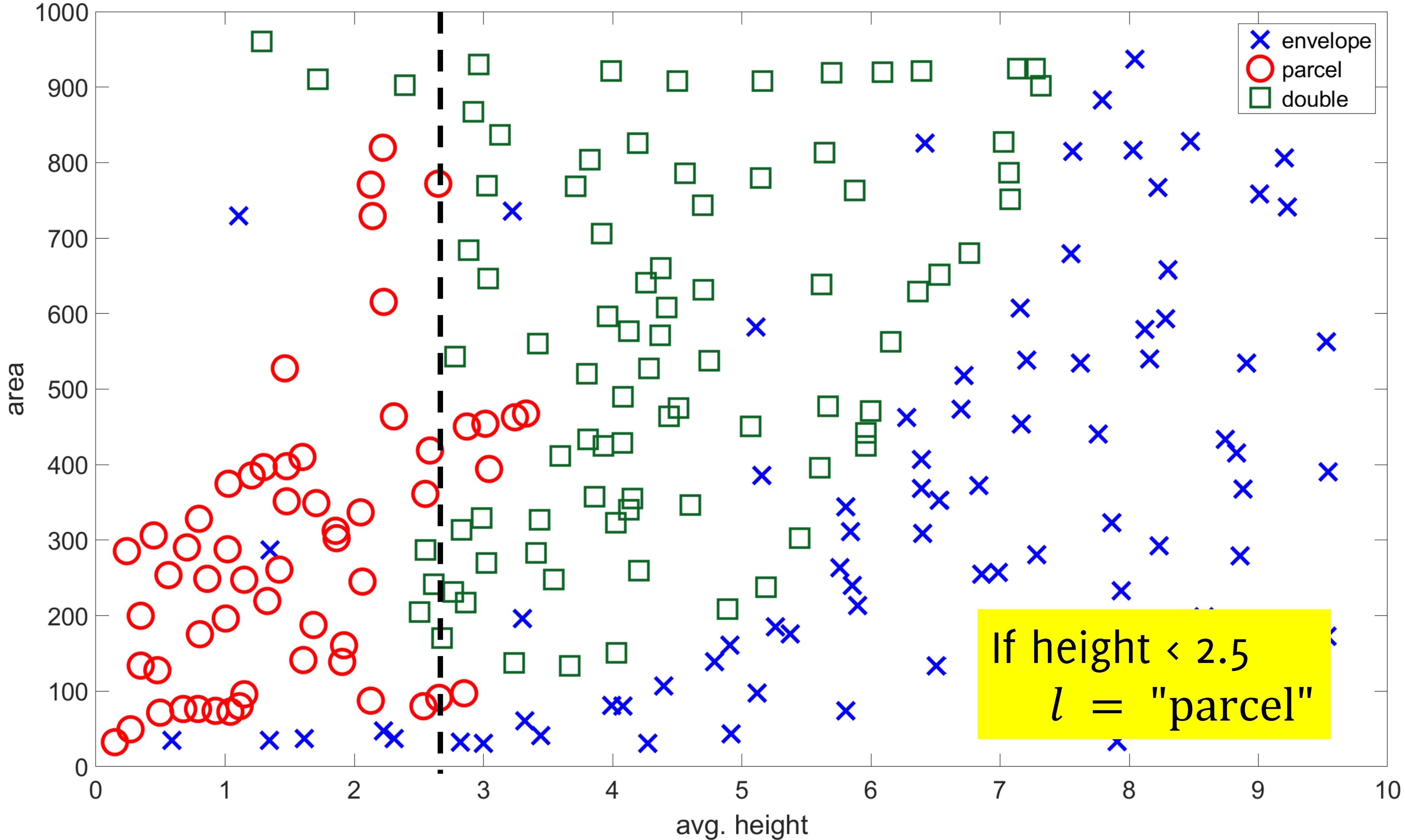
The Training Set: features + labels



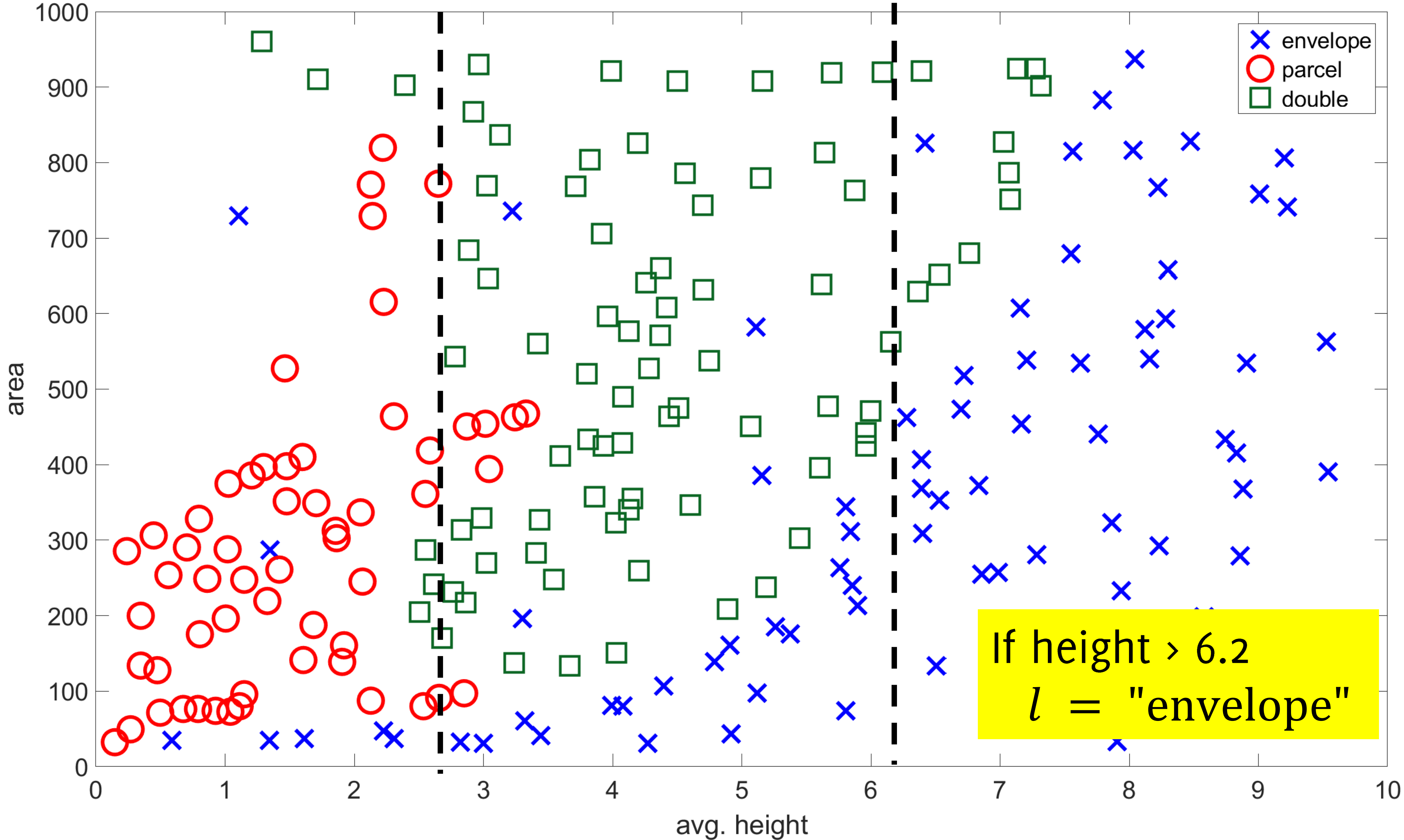
The Training Set



Training Set

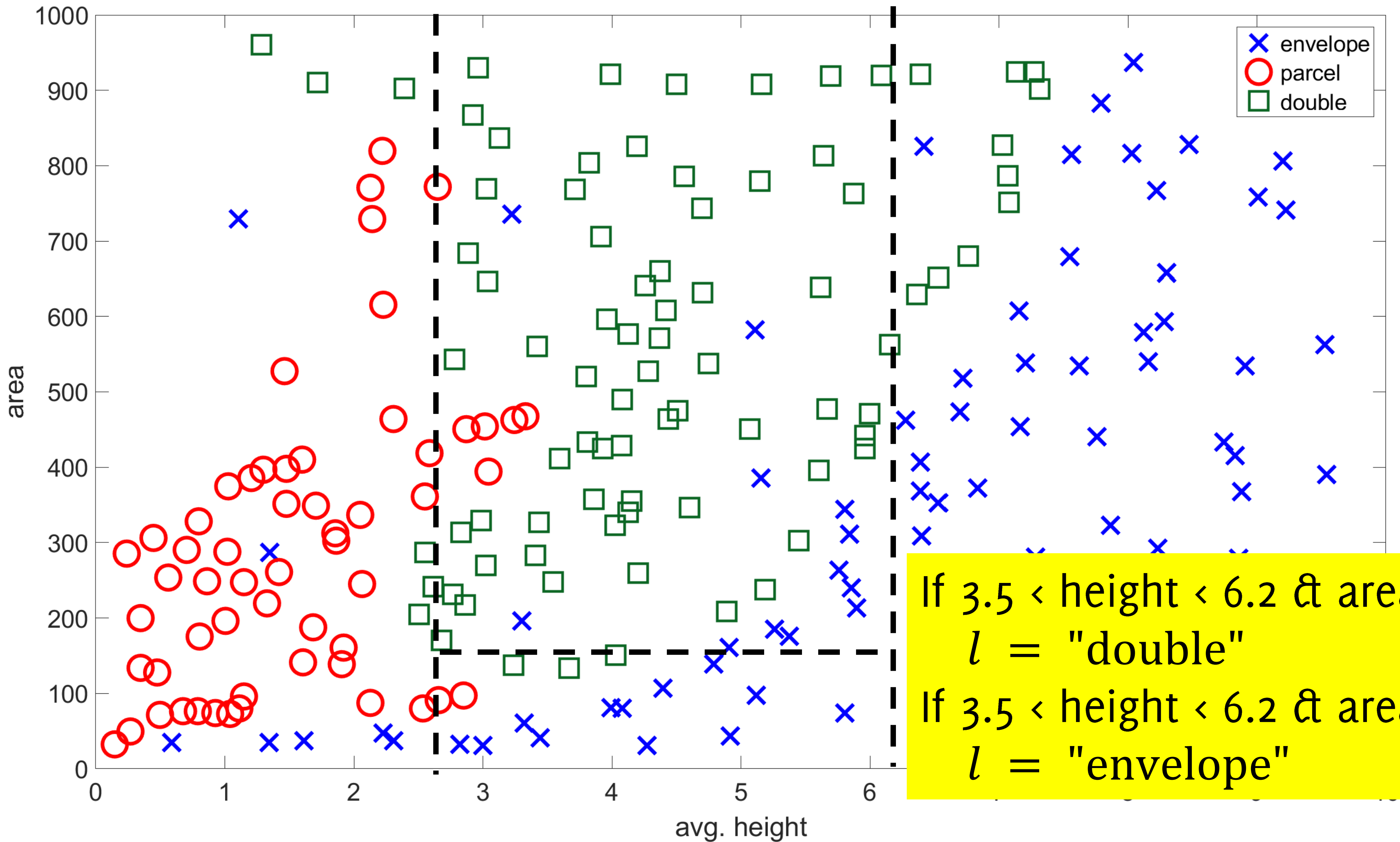


Training Set



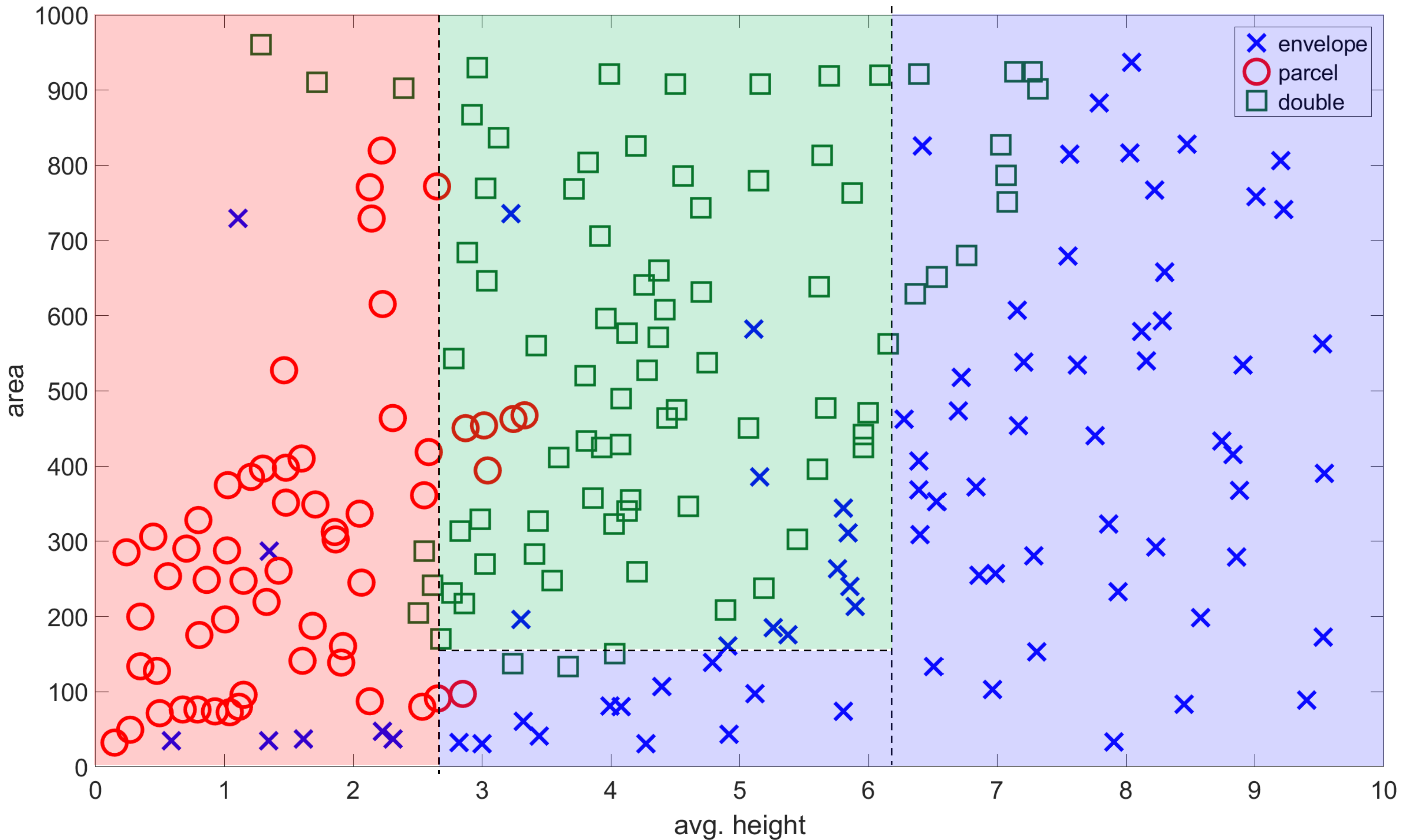
If height > 6.2
l = "envelope"

Training Set



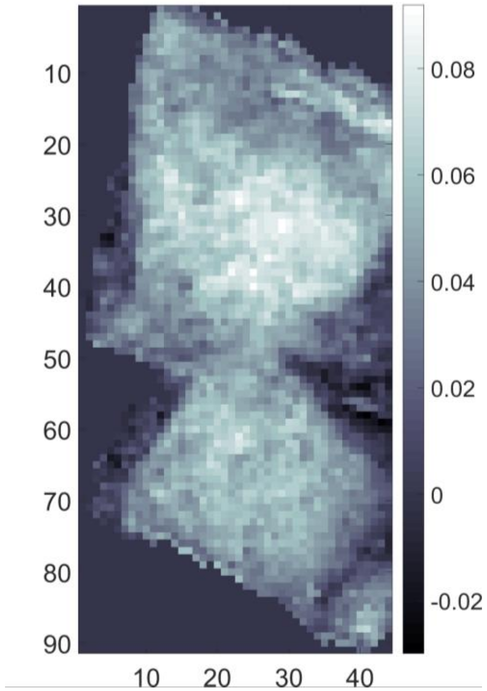
If 3.5 ‹ height ‹ 6.2 & area › 200
l = "double"
If 3.5 ‹ height ‹ 6.2 & area ‹ 200
l = "envelope"

Classifier output

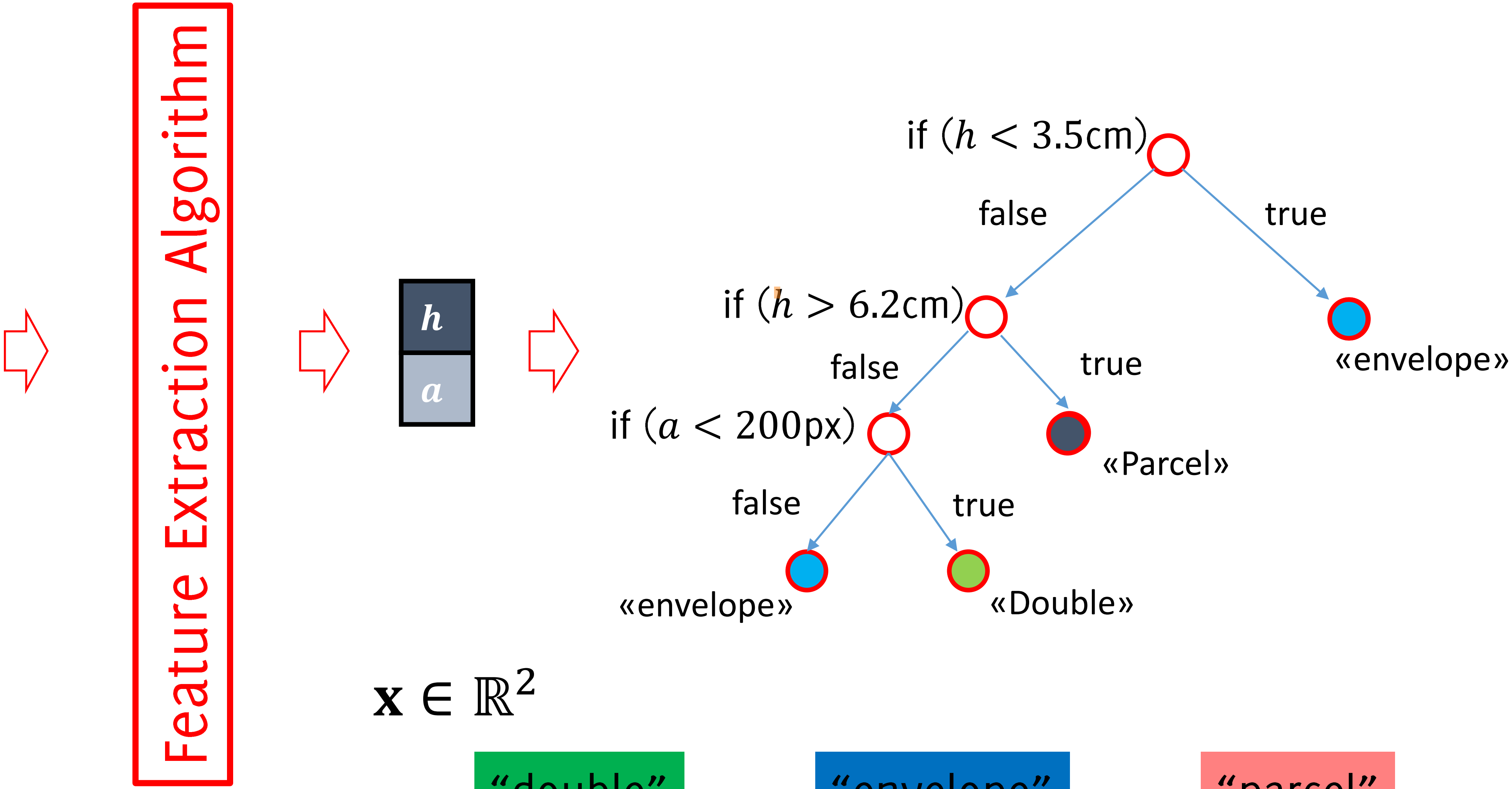


A tree classifying image features

Input image



$I_1 \in \mathbb{R}^{r_1 \times c_1}$



$\mathbf{x} \in \mathbb{R}^2$

"double" "envelope" "parcel"

Limitations of Rule Based Classifier

It is difficult to grasp what are meaningful dependencies over multiple variables (it is also impossible to visualize these)

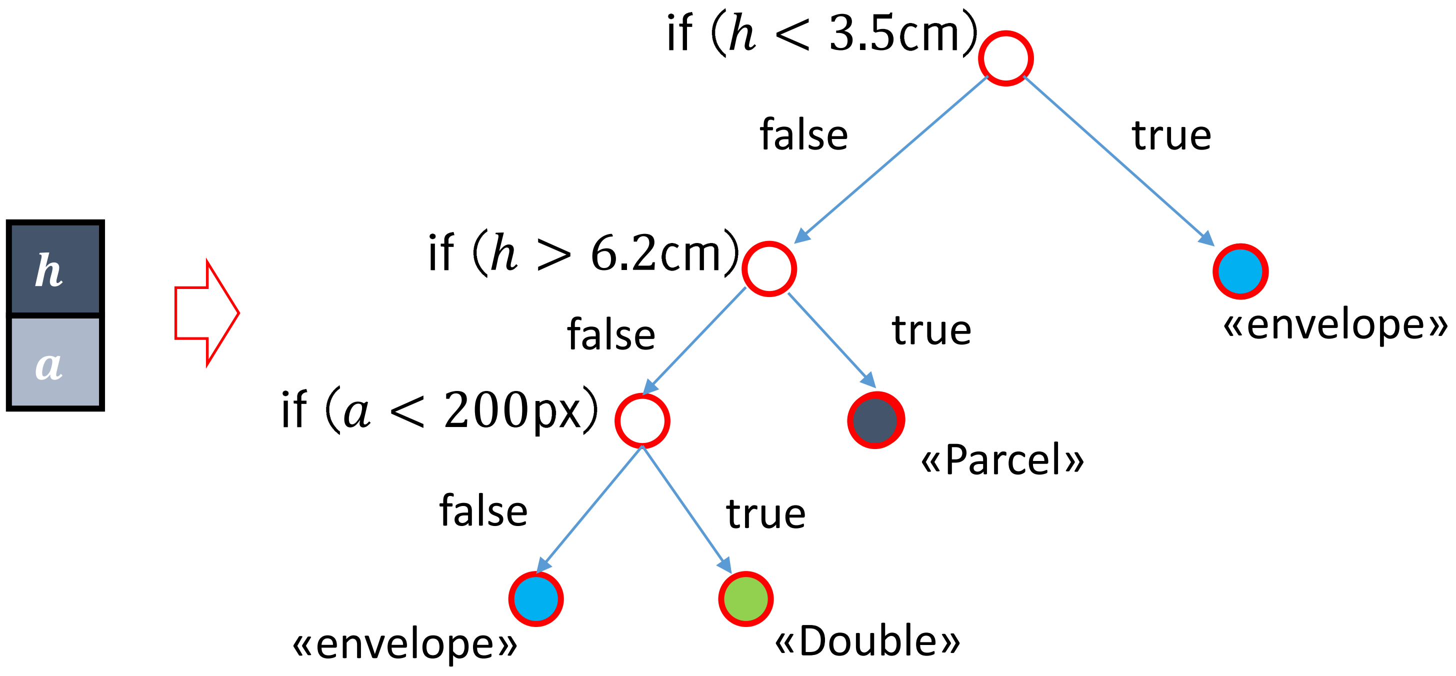
Let's resort to a **data-driven model** for the only task of separating feature vectors in different classes.

How can a classifier achieve better performance?

A tree classifying image features

The classifier has a few patameters:

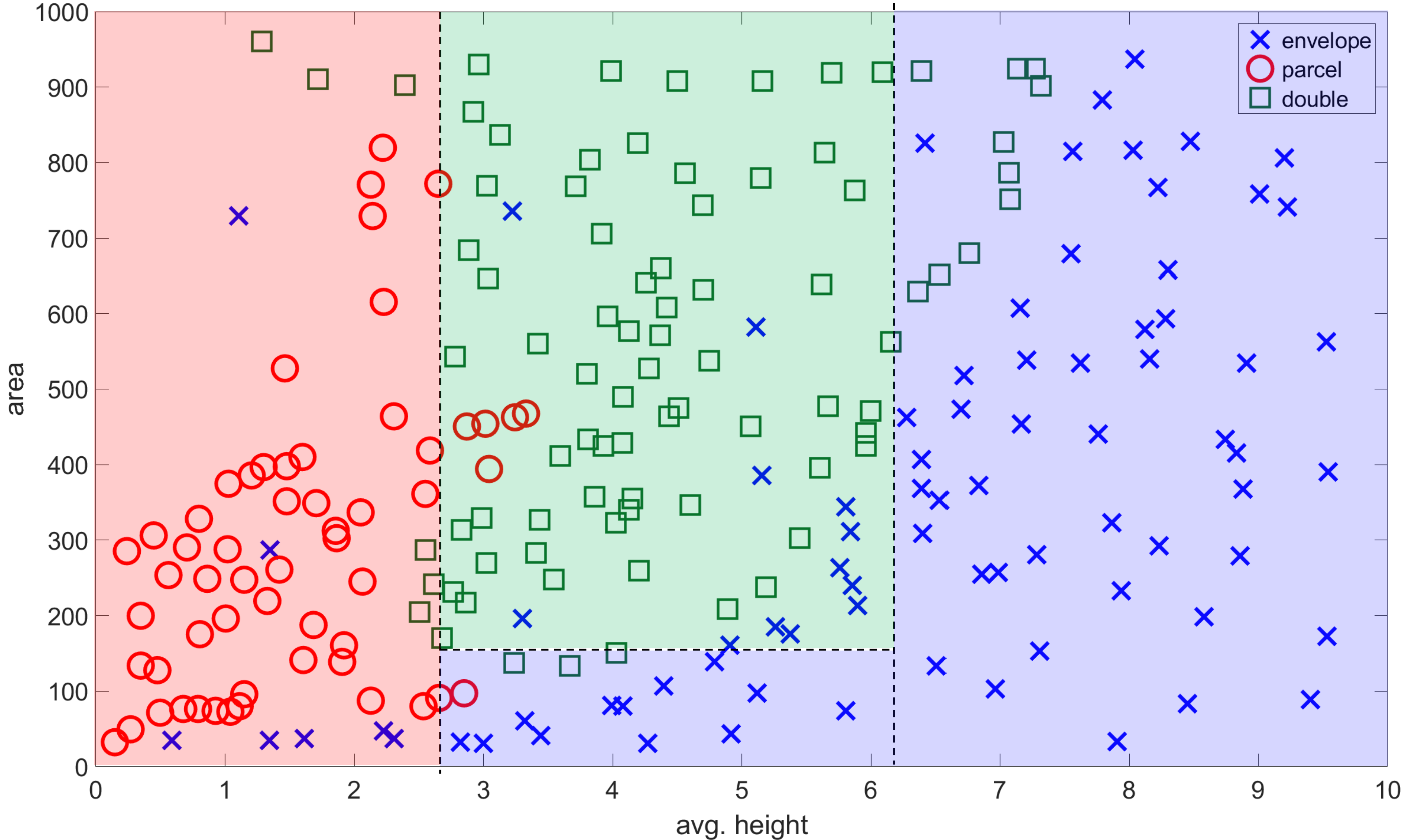
- The **splitting criteria**
- The **splitting thresholds T_i**



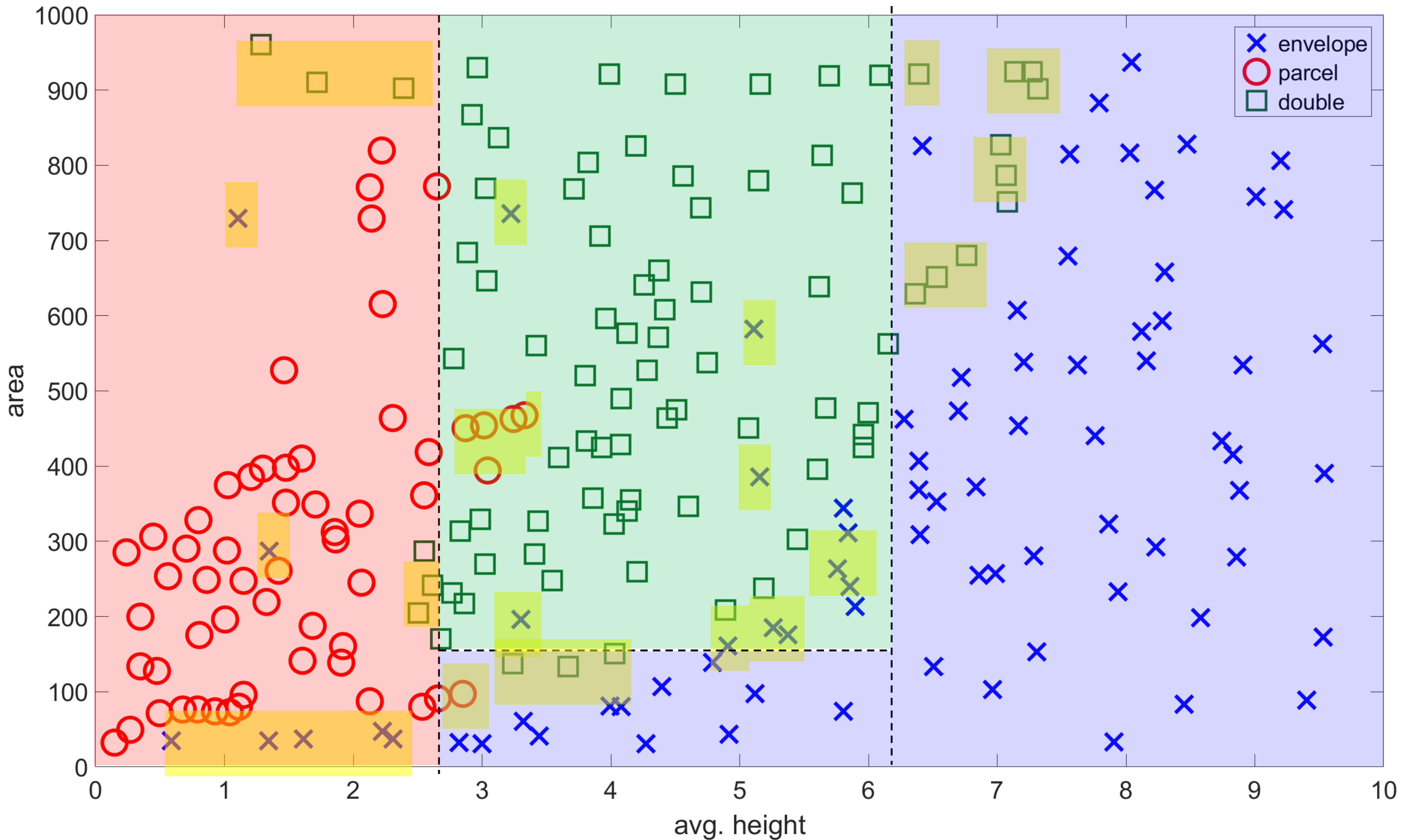
$\mathbf{x} \in \mathbb{R}^2$

"double" "envelope" "parcel"

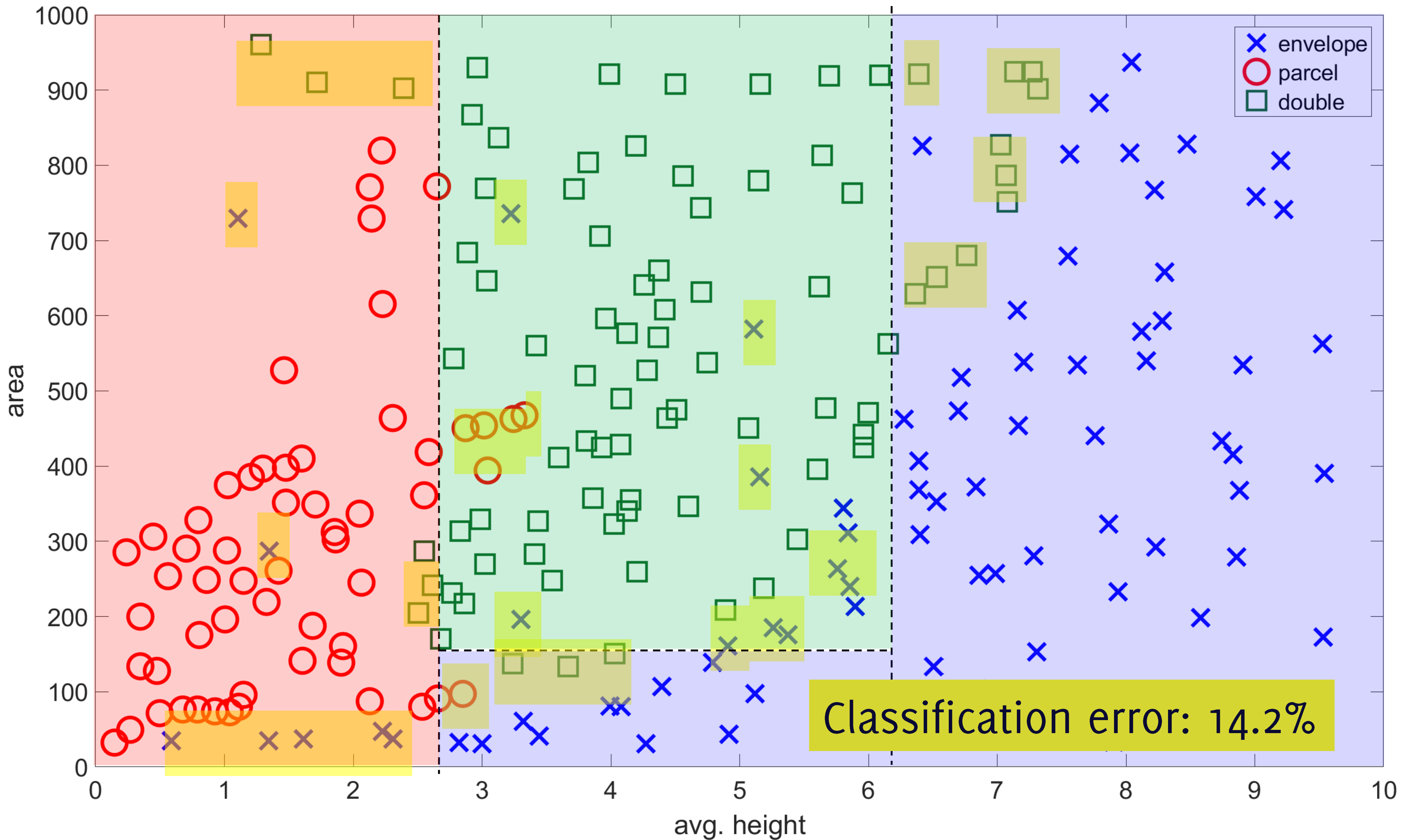
This is our first solution



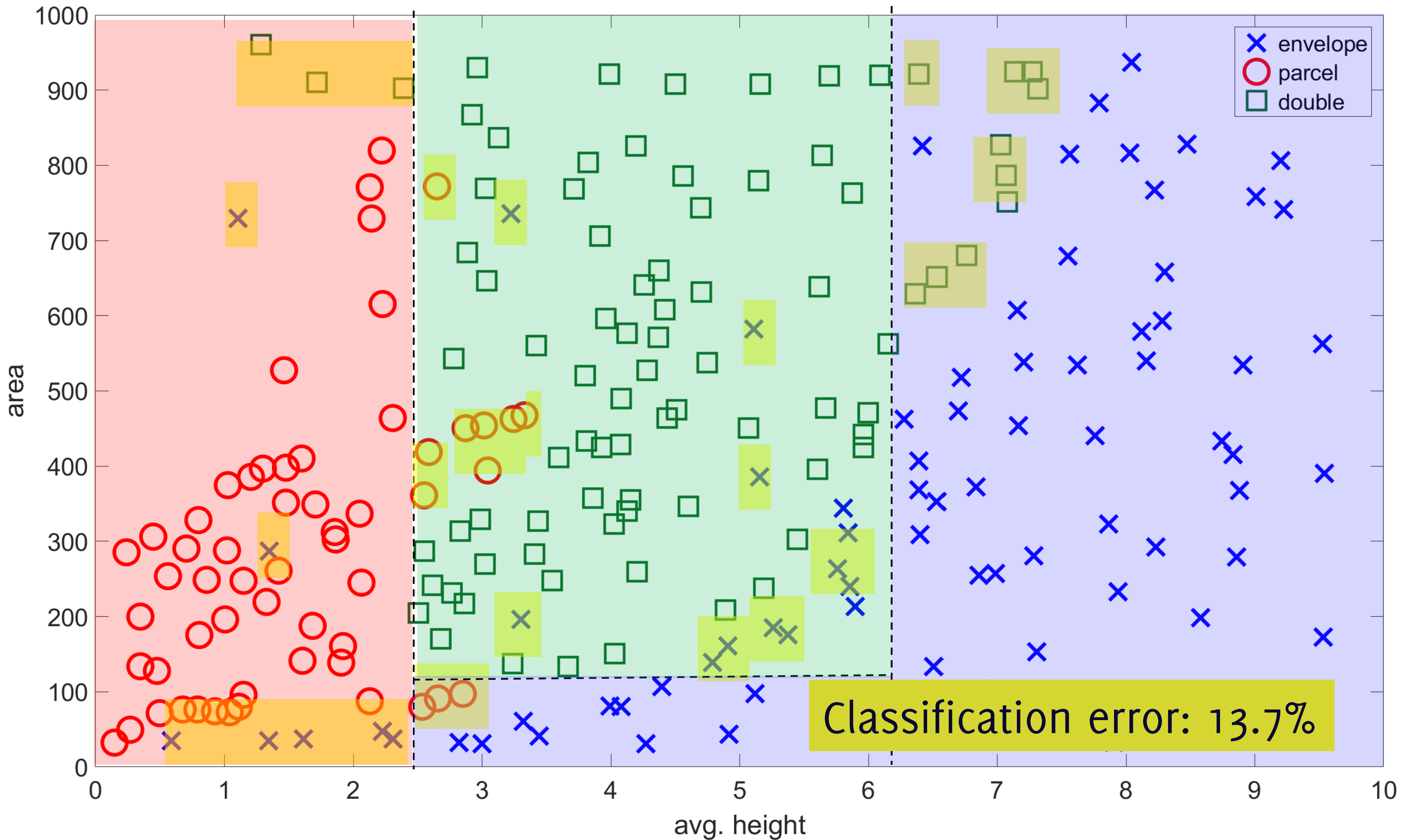
There are a few errors



Can I do better?



Let's try different parameters



Data Driven Models

They are defined from a training set of supervised pairs

$$TR = \{(x, y)_i, i = 1, \dots, N\}$$

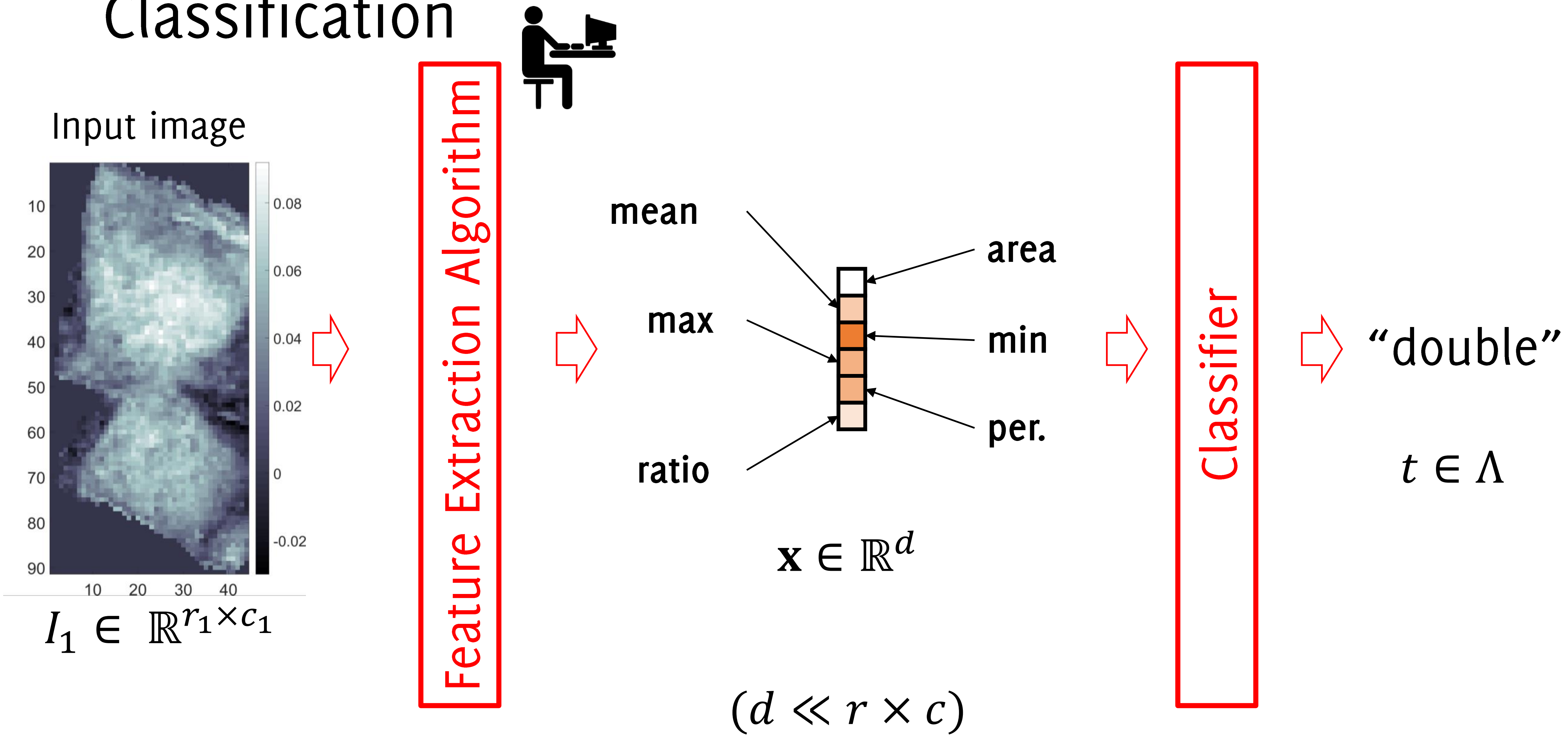
The model parameters (e.g. Neural Network weights) are set to minimize a **loss function** (e.g., the classification error in case of discrete output or the reconstruction error in case of continuous output)

Can definitvely boost the image classification performance

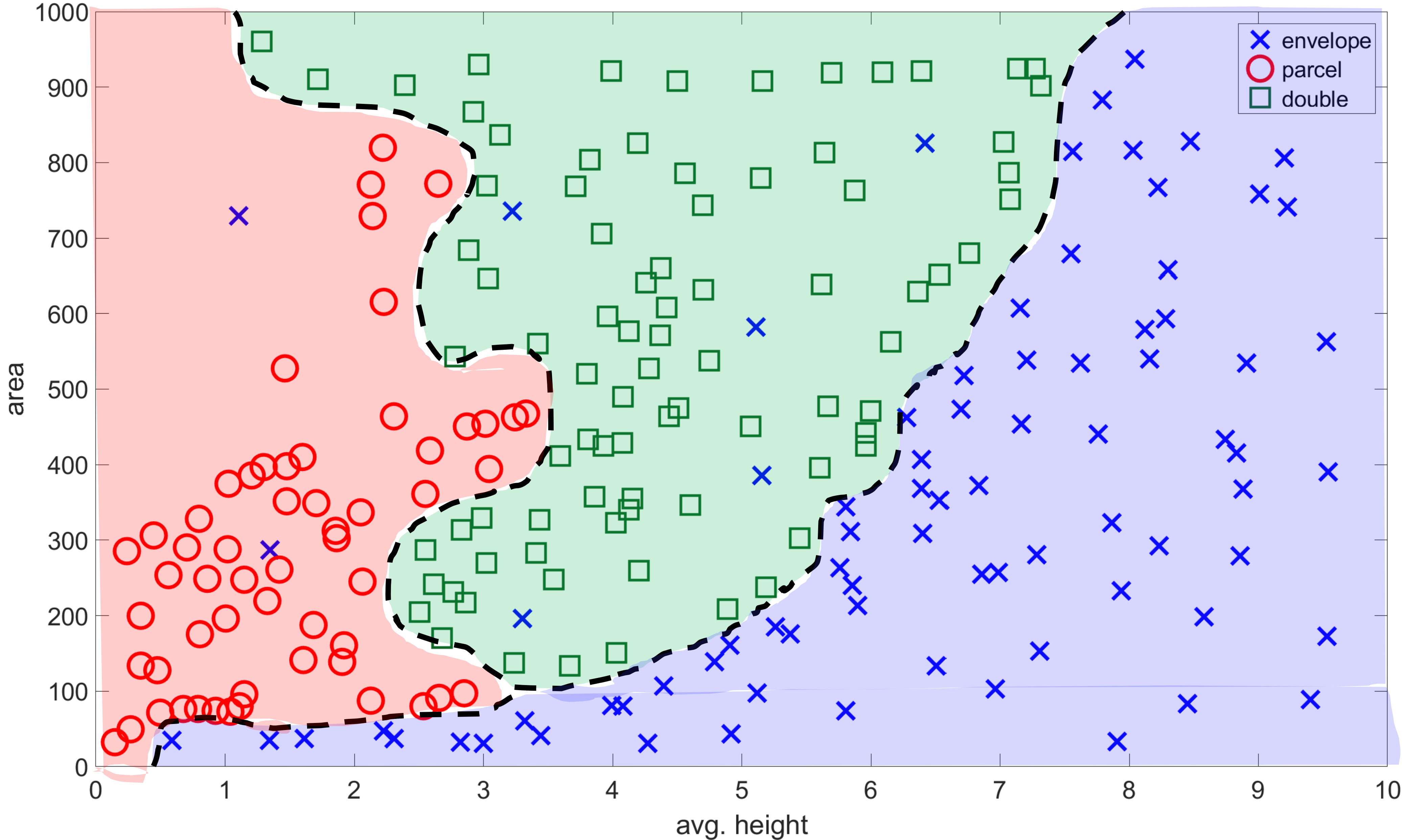
This is how, during training, the computer **learns.**

- Annotated training set is always needed
- Classification performance depends on the training set
- Generalization is not guaranteed

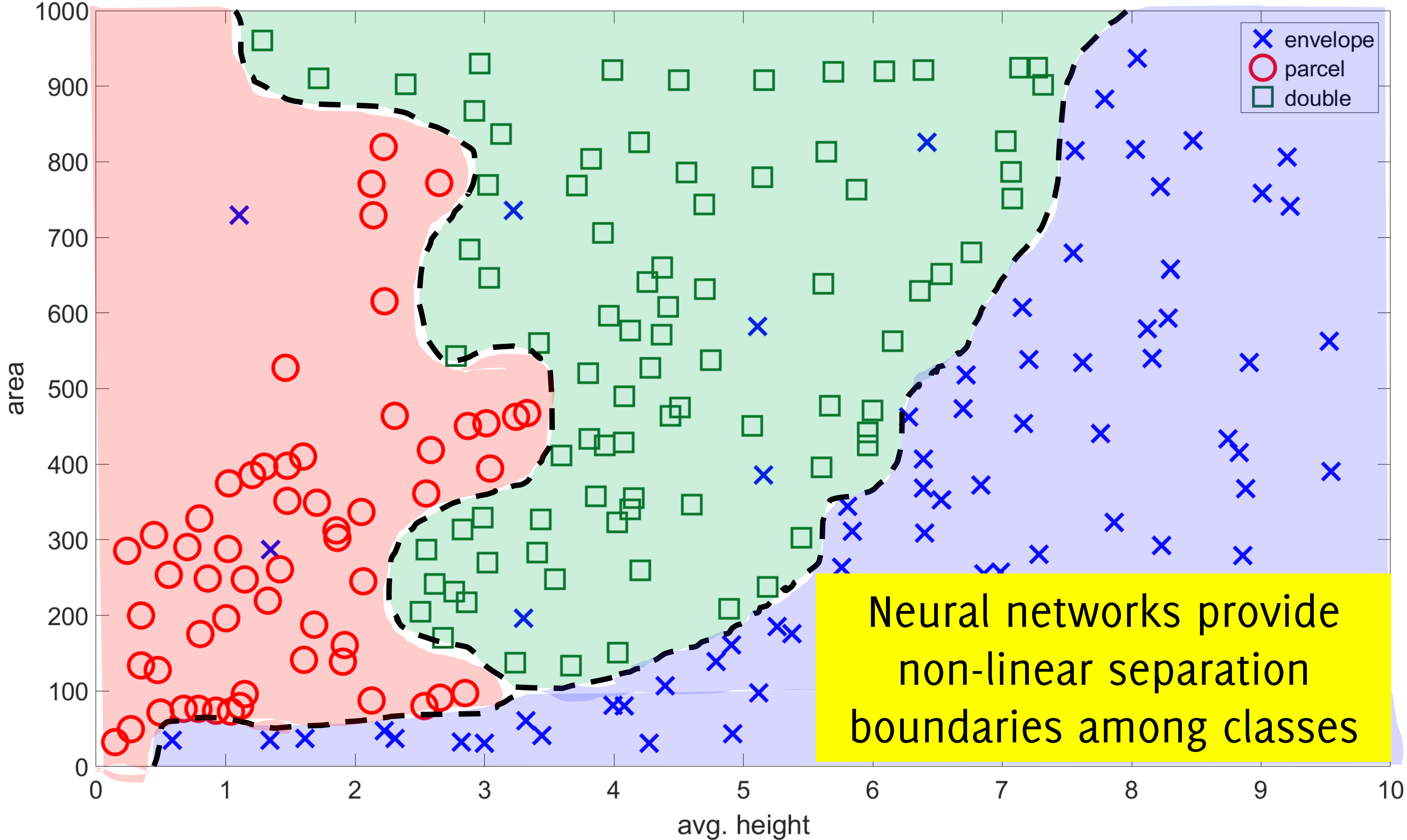
Hand Crafted Feature Extraction, data-driven Classification



Are there better classifiers?

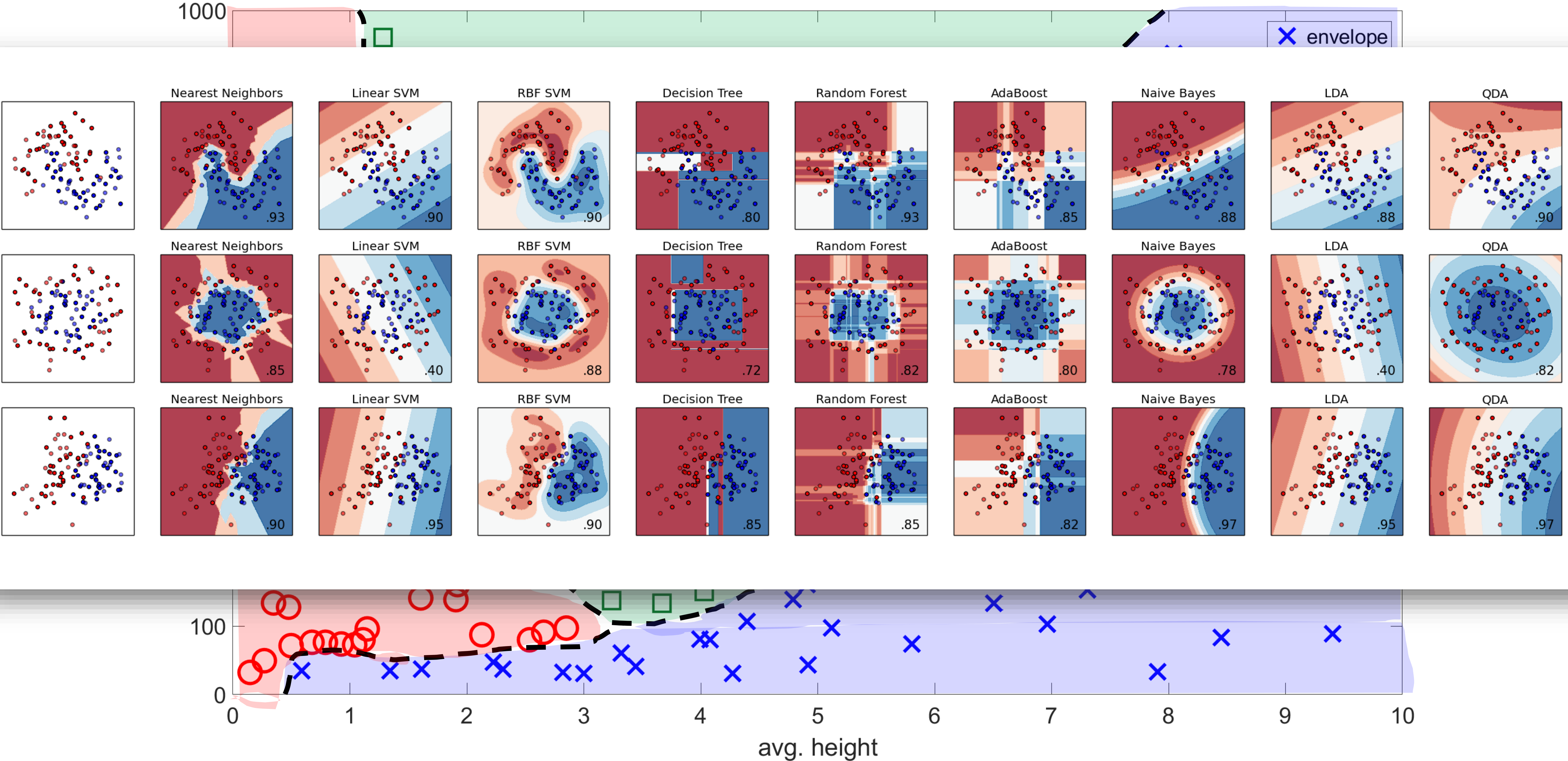


Are there better classifiers?

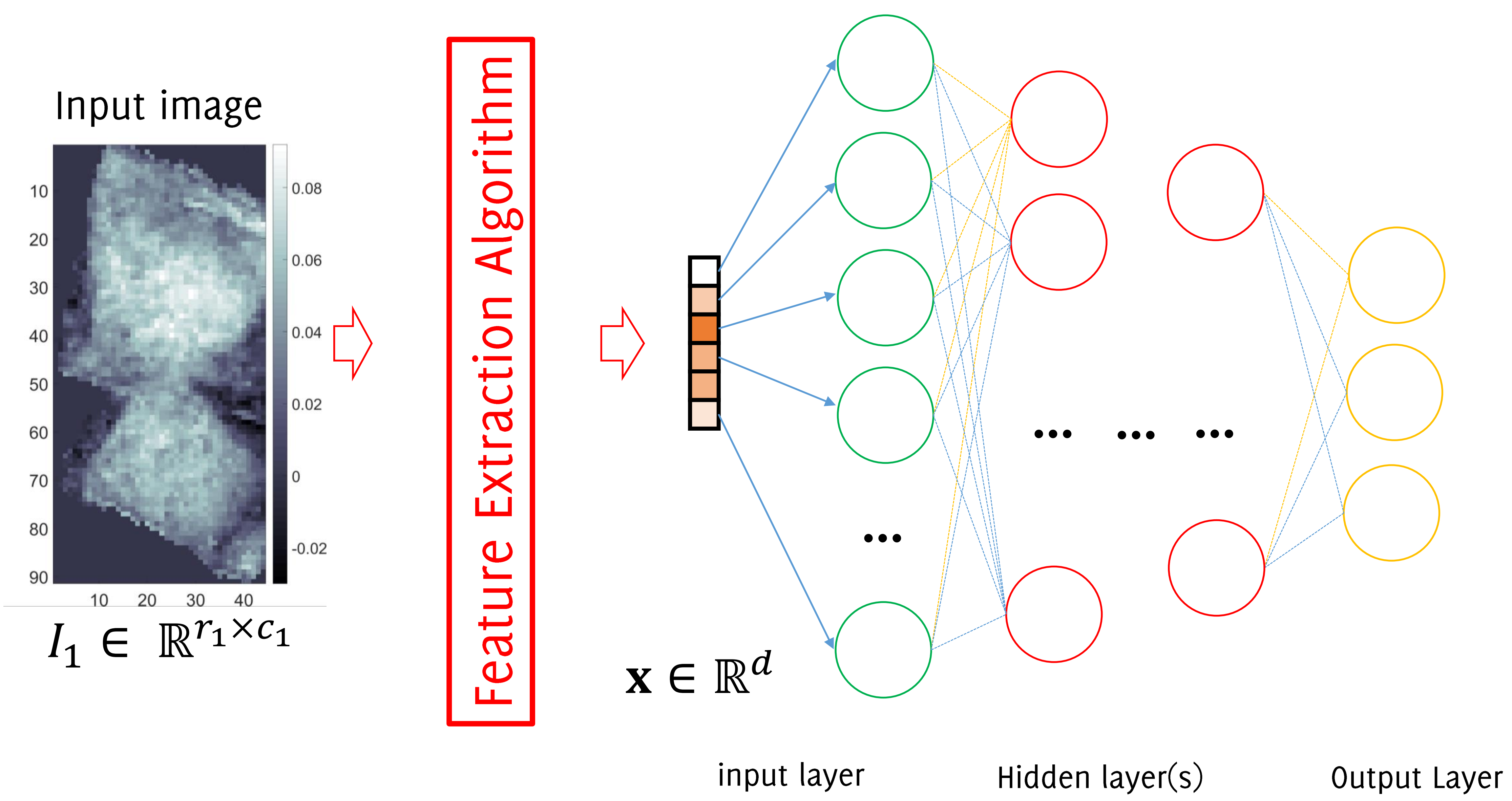


Neural networks provide non-linear separation boundaries among classes

And Neural Networks are not the only..



Neural Networks for Feature Classification



A Short Recap on Neural Network

Giacomo Boracchi

giacomo.boracchi@unibocconi.it

February 14th 2024

UEM, Maputo

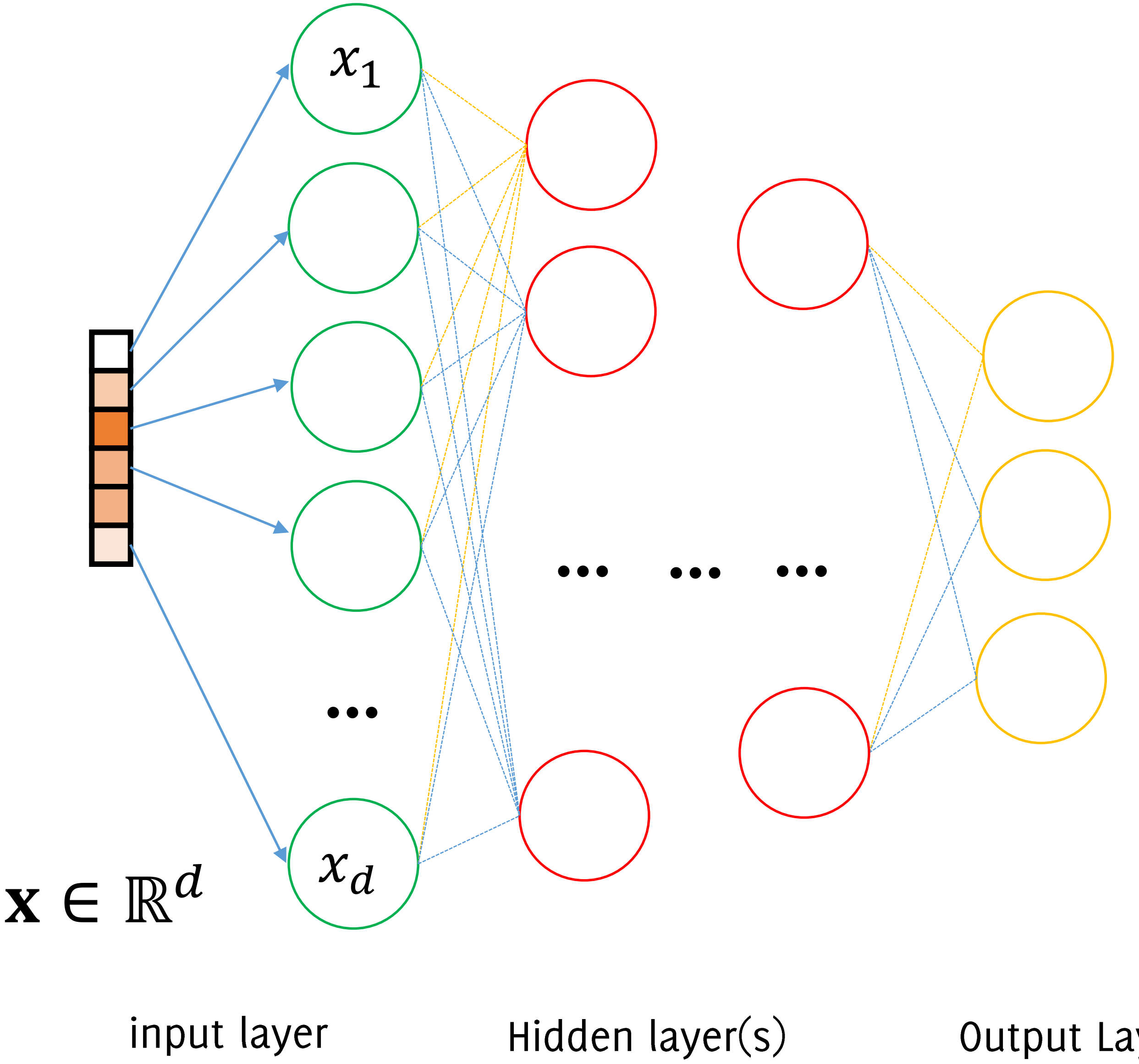
https://boracchi.faculty.polimi.it

Neural Networks

- The input layer has the same number of neurons as the number of inputs

- This is not a hyperparameter!

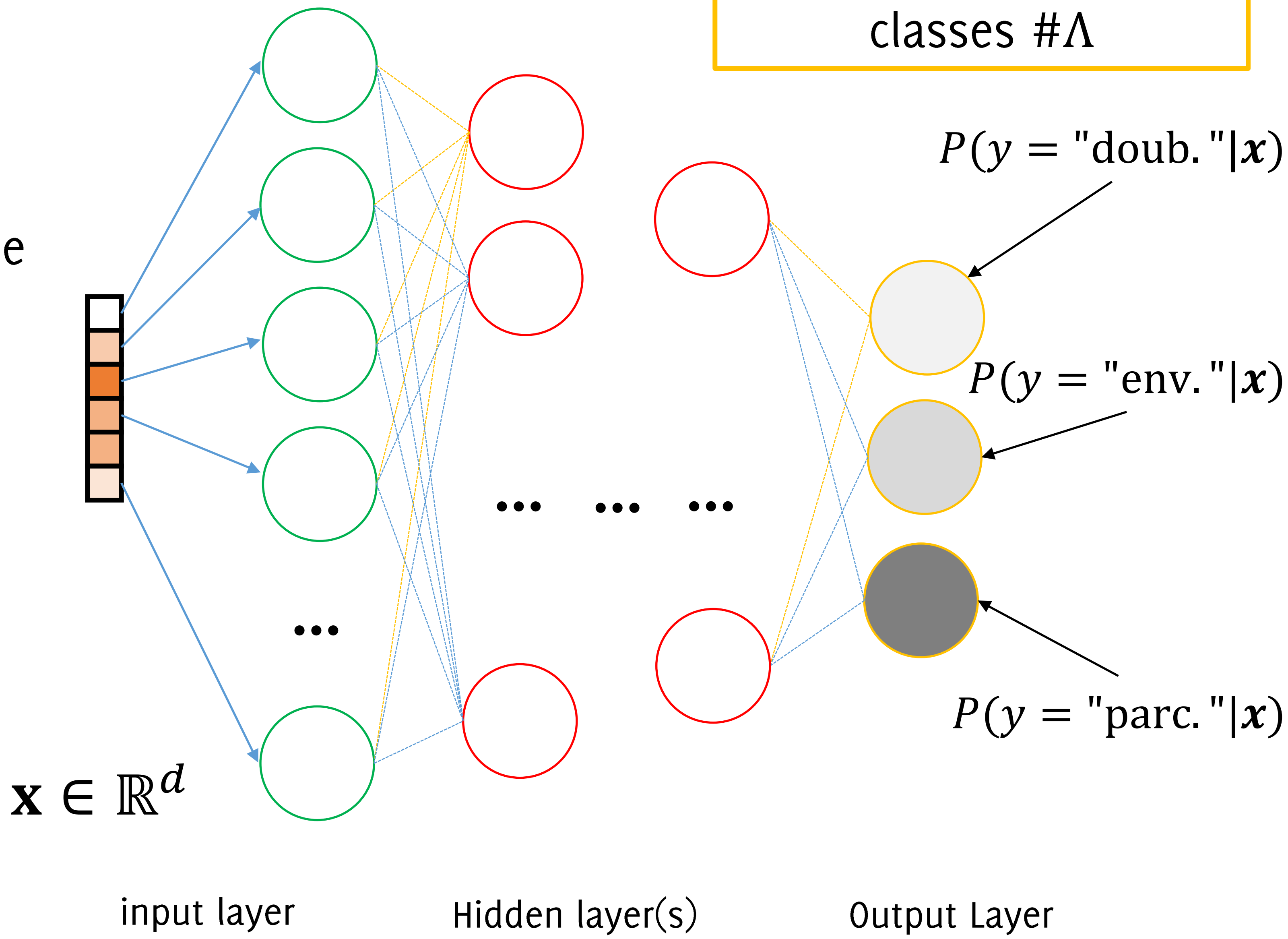
Input layer: Same size of the feature vector



Neural Networks

- The output size depends on the number of classes to be predicted (or the number of outputs in case of regression).
- This is not a hyperparameter, this is defined by the task!
- In case of classification, the output are probabilities, in case of regression these are real values

Output layer: Same size as the number of classes $\#\Lambda$

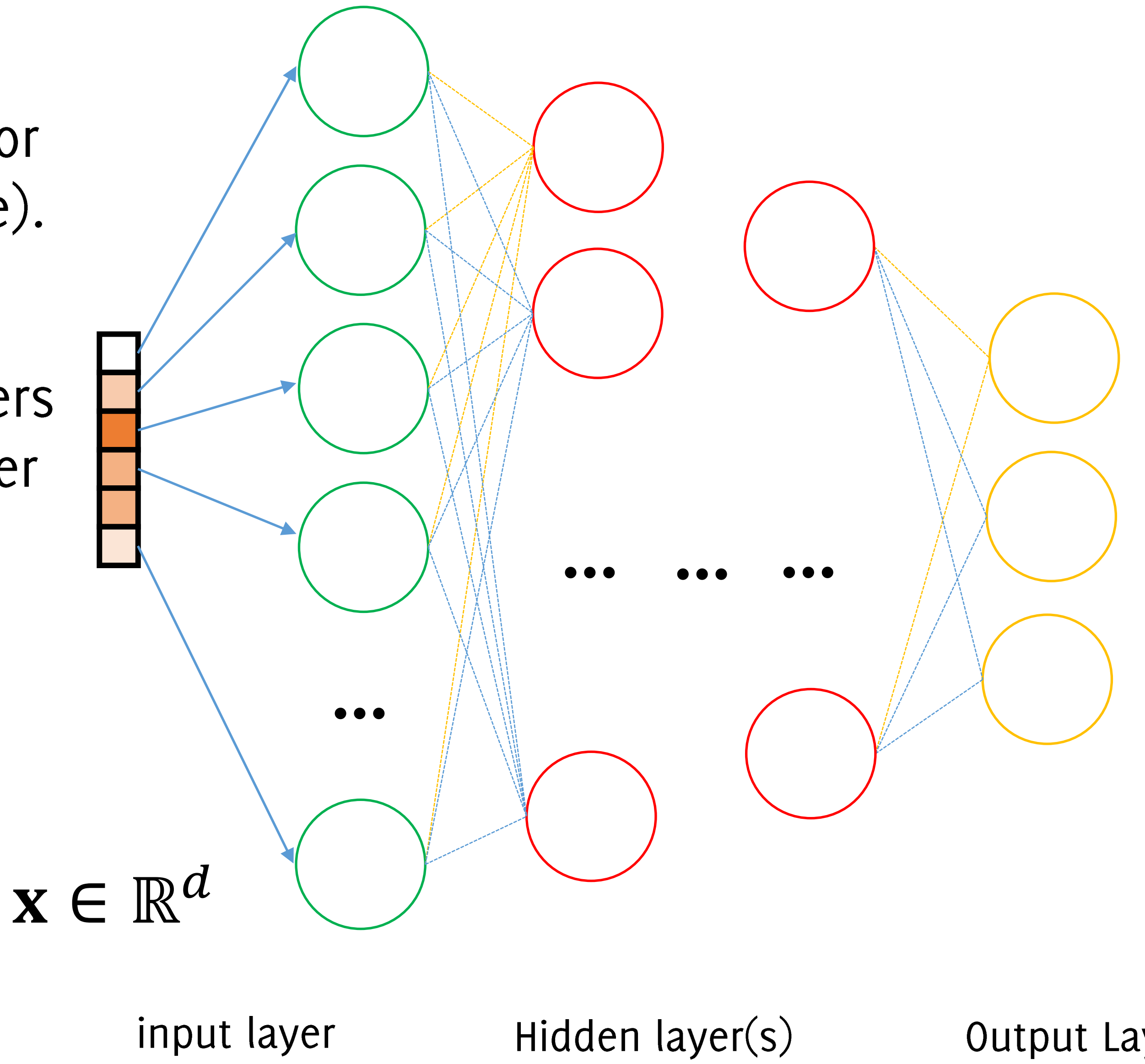


Neural Networks

Hidden layers: arbitrary size

- Hidden layers are not directly connected input or output (hence their name).

- The design of hidden layers (number of layers, number or neurons) is a hyperparameter of the network.



Inside Neural Networks

Each connection is associated to a weight

$$w_{i,j}^{k} \in \mathbb{R}$$

This weight connects:

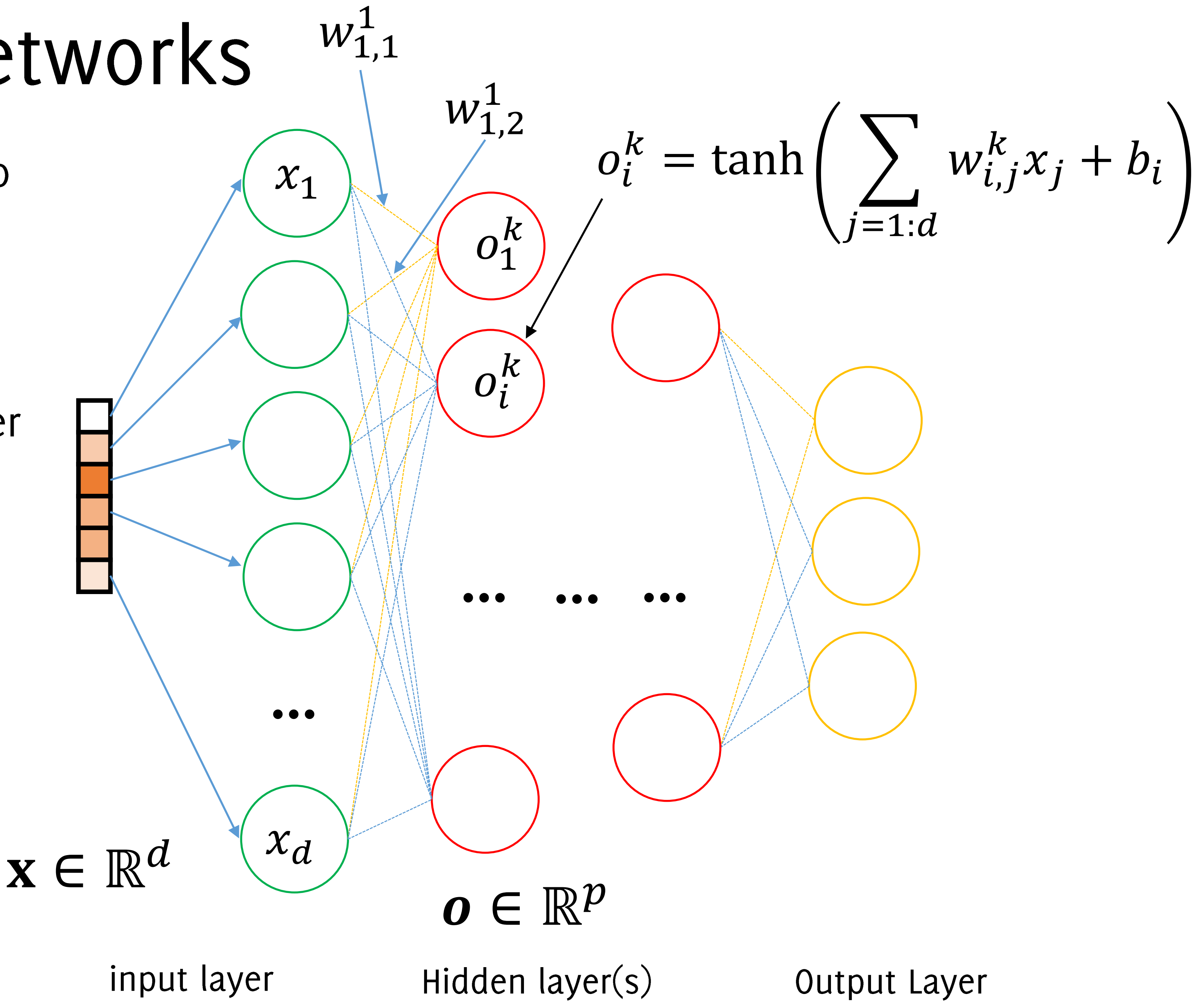
- The i^{th} input neuron of layer $(k-1)$
- The j^{th} output neuron of layer k^{th}

On top of weights there are biases, one bias per neuron

$$\{b_i^k\}_{i,k}$$

The parameters of the network are:

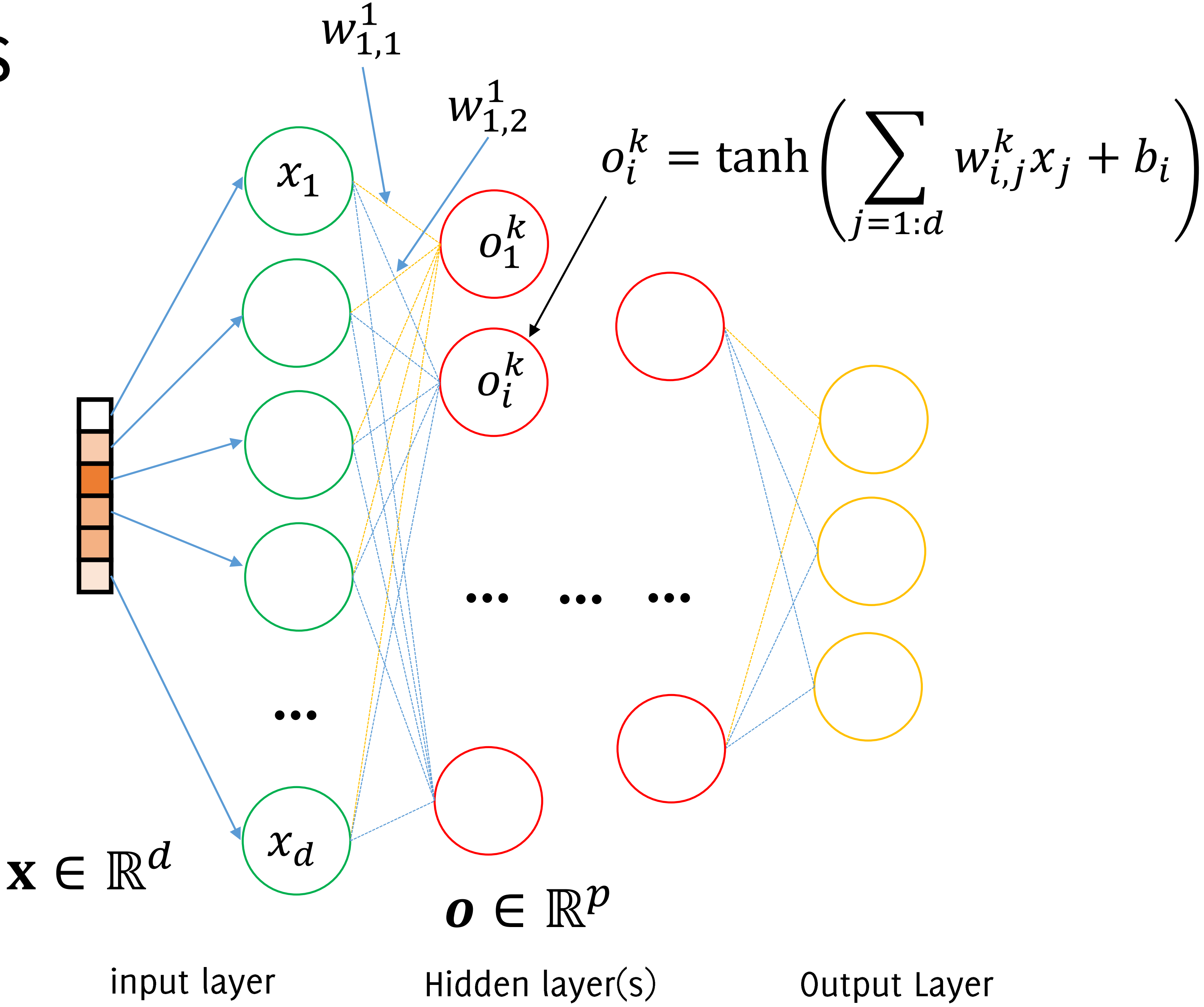
$$\{w_{i,j}^{k}\}_{i,j,k}, \{b_i^k\}_{i,k}$$



Neural Networks

Each neuron:

- Computes a linear combination of its inputs
- Applies a nonlinear, scalar function (here $\tanh(\cdot)$)



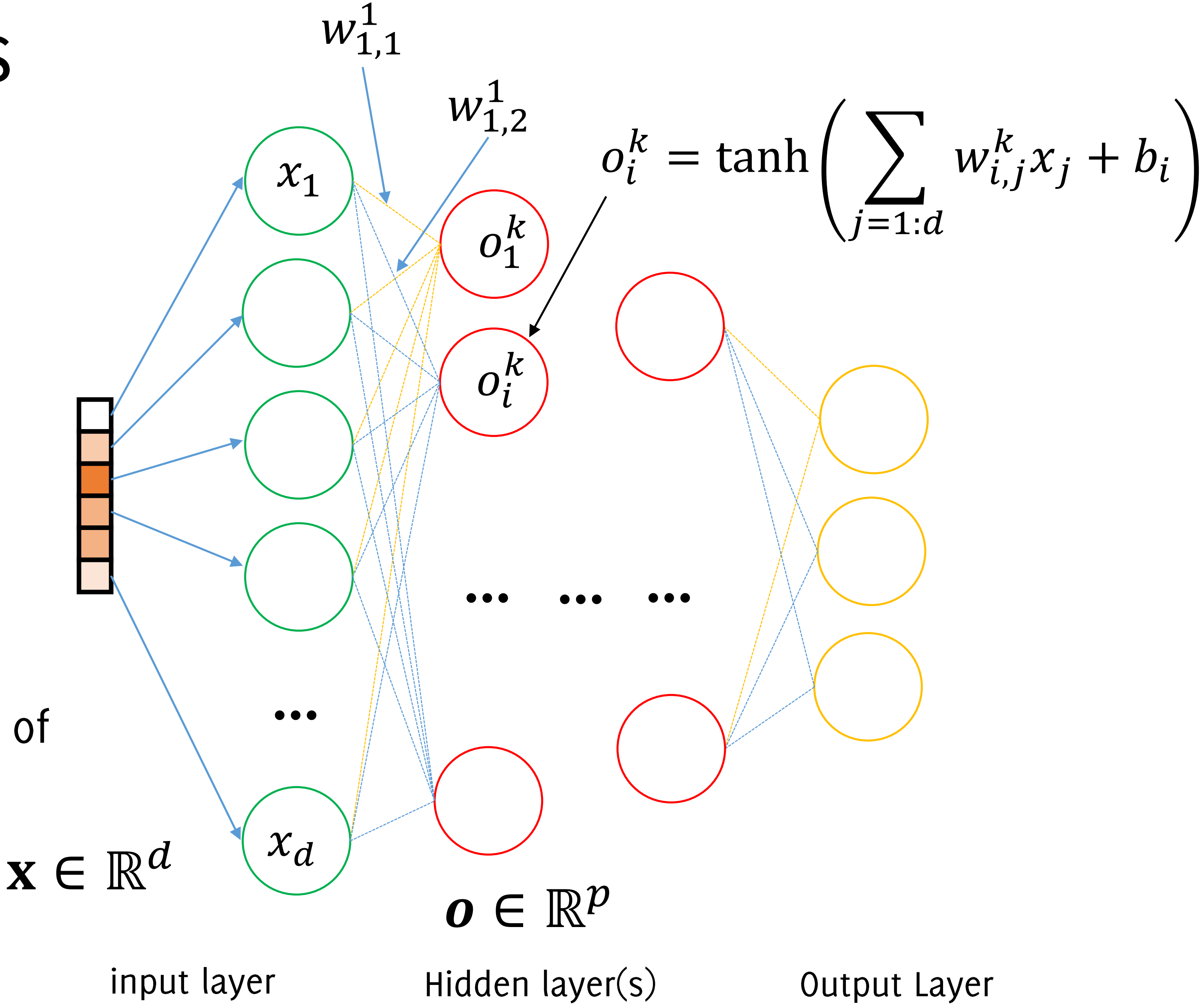
Neural Networks

Each neuron:

- Computes a linear combination of its inputs
- Applies a nonlinear, scalar function (here $\tanh(\cdot)$)

Nonlinearity is mandatory, otherwise everything will become a linear combination of a linear combination…

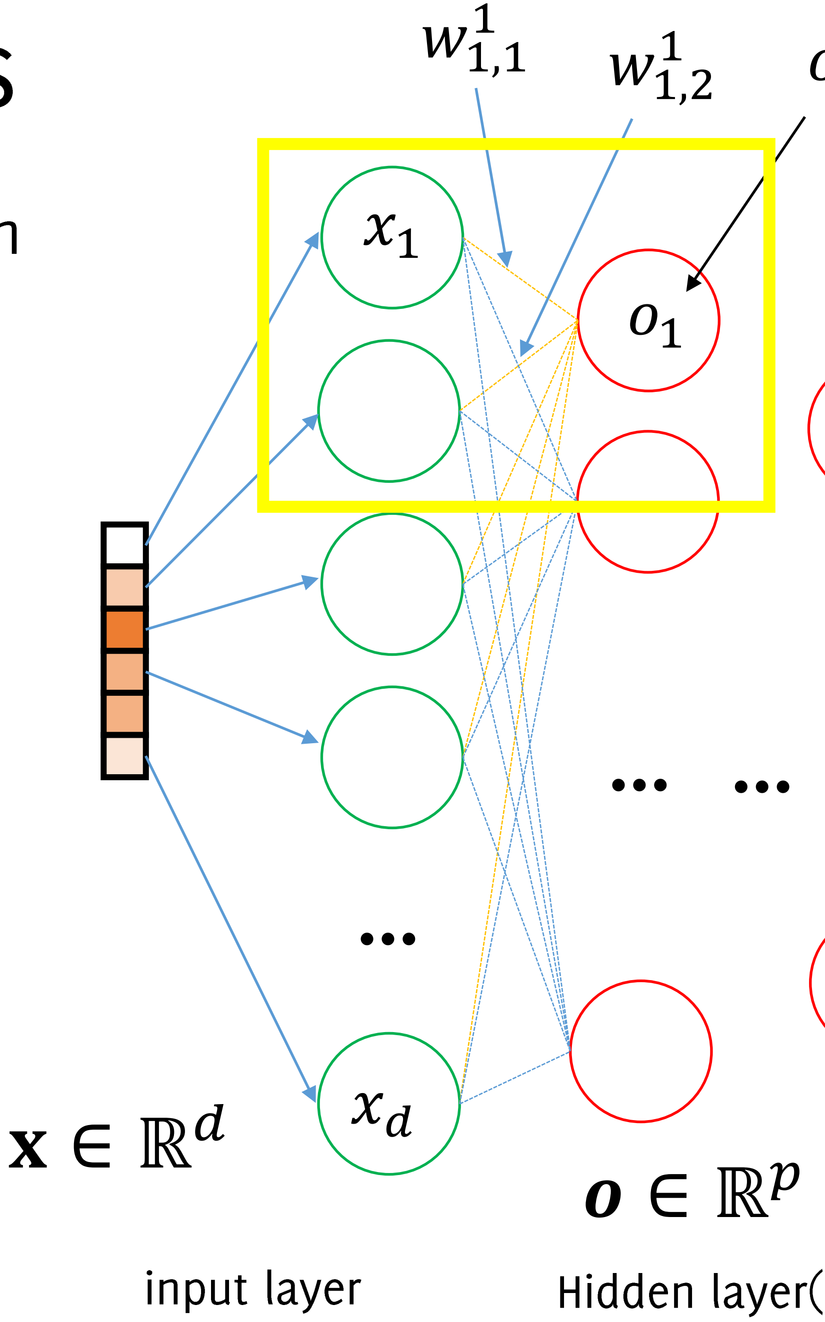
Thus, equivalent to a linear classifier!



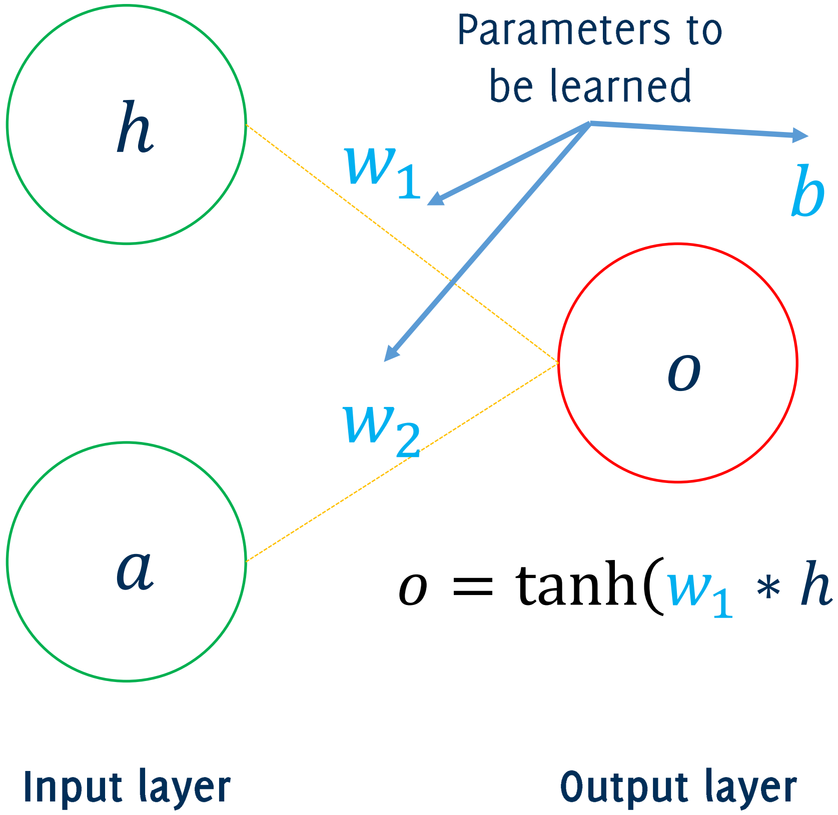
Neural Networks

Let's focus on a single neuron and see what happens while learning

$$o_1 = \tanh\left(\sum_{j=1:d} w_{1,j}^1 x_j + b_1\right)$$



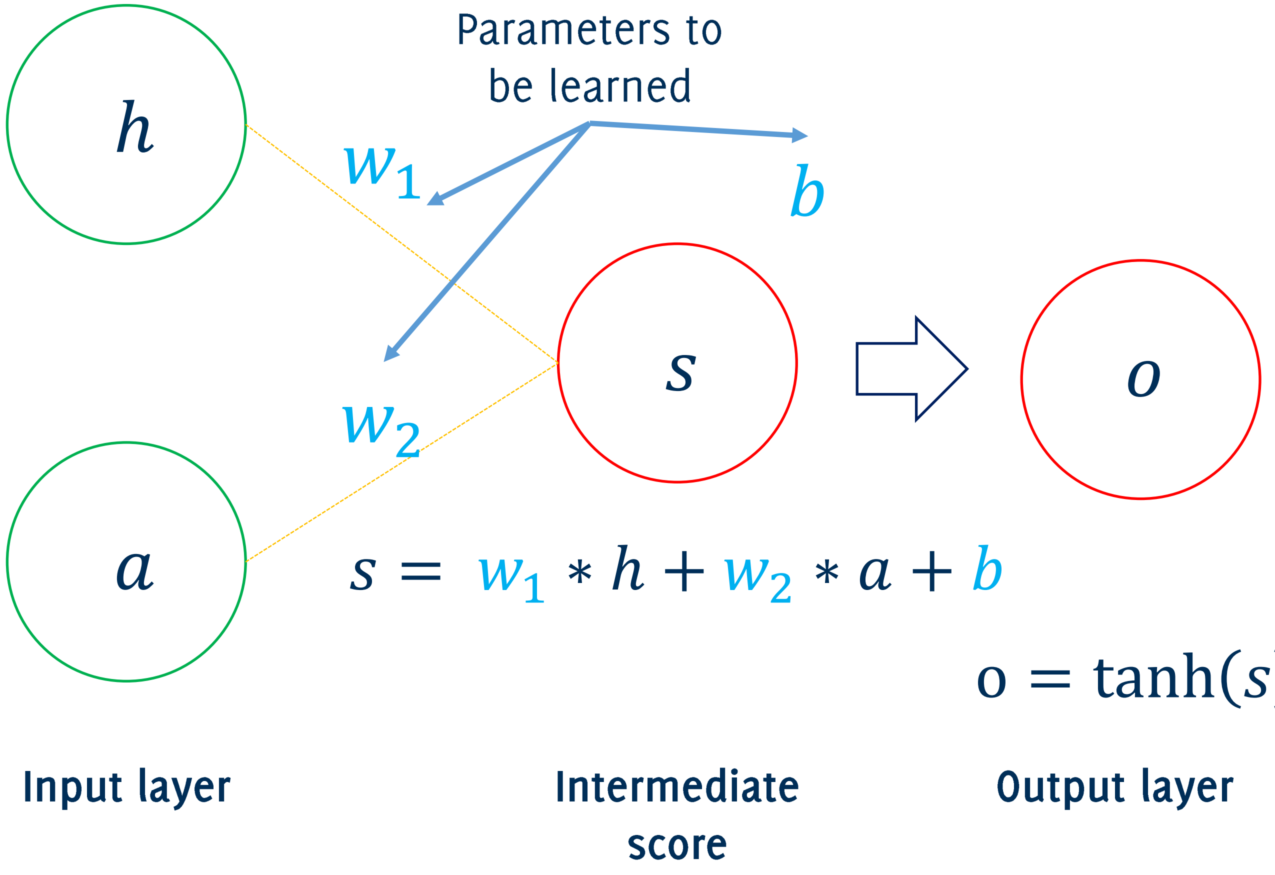
At the core of NN: Linear Combinations



$$o = \tanh(w_1 * h + w_2 * a + b)$$

At the core of NN: Linear Combinations

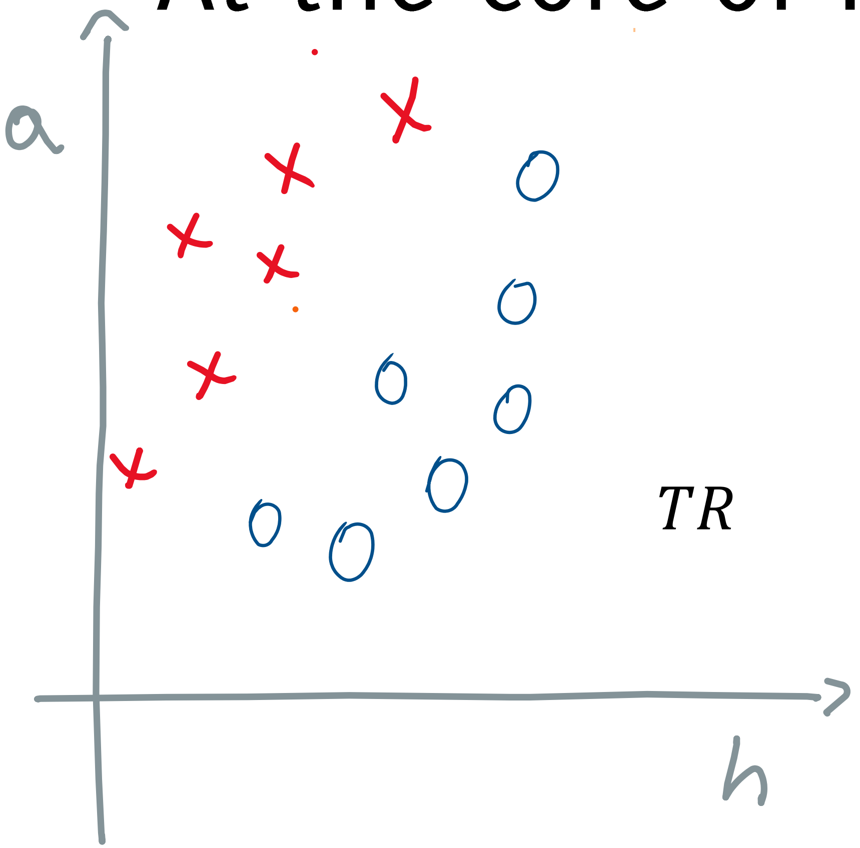
Let us ignore the nonlinearity for a while, as this is not relevant for a single layer

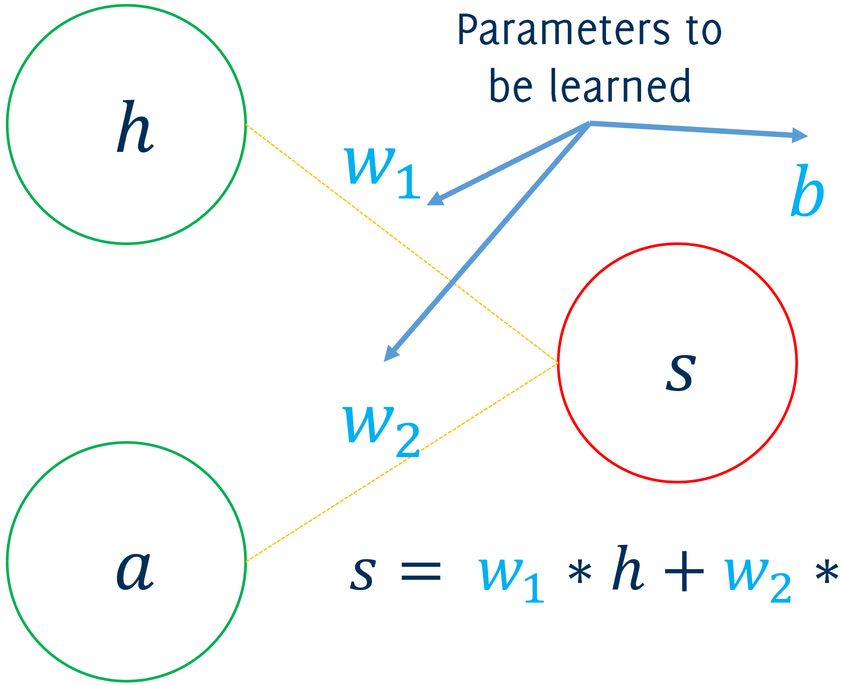


$$s = w_1 * h + w_2 * a + b$$

$$o = \tanh(s)$$

At the core of NN: Linear Combinations

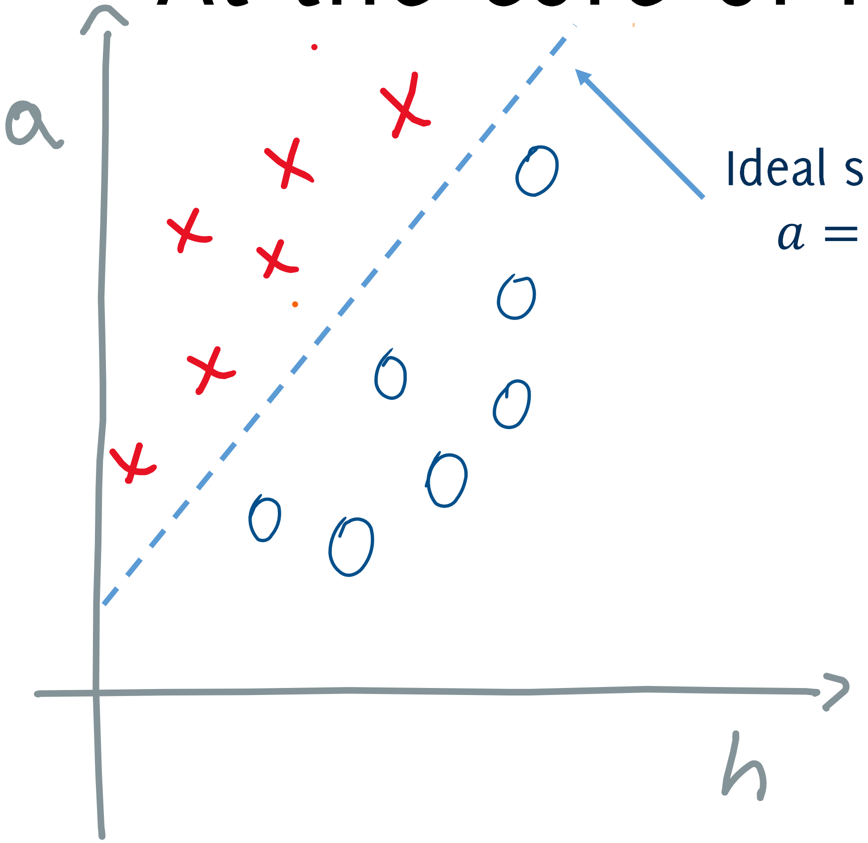




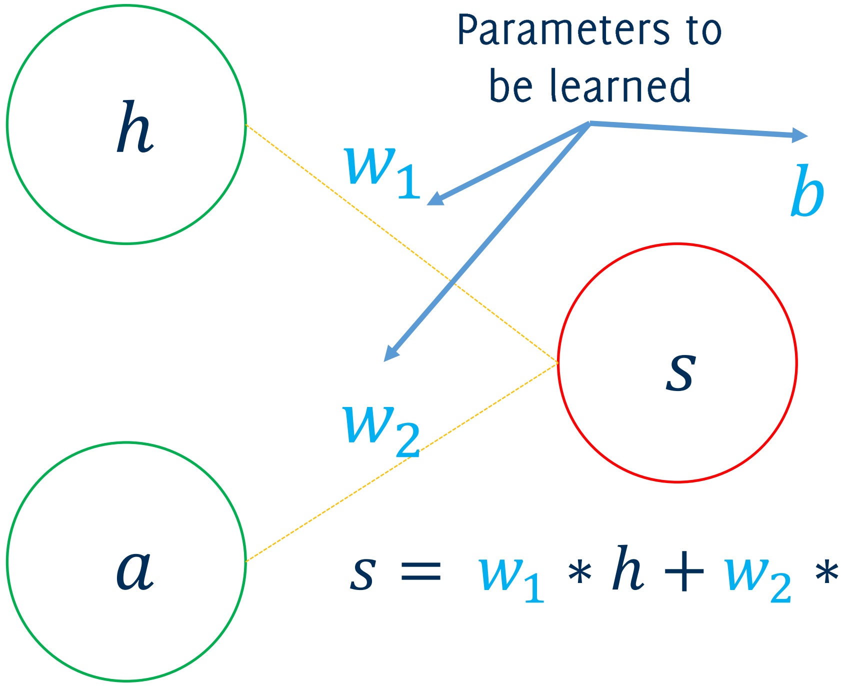
$$s = w_1 * h + w_2 * a + b$$

What parameters would the classifier learn from this training set?

At the core of NN: Linear Combinations



Ideal separation line
$a = 1.5\,h + 23$



$s = w_1 * h + w_2 * a + b$

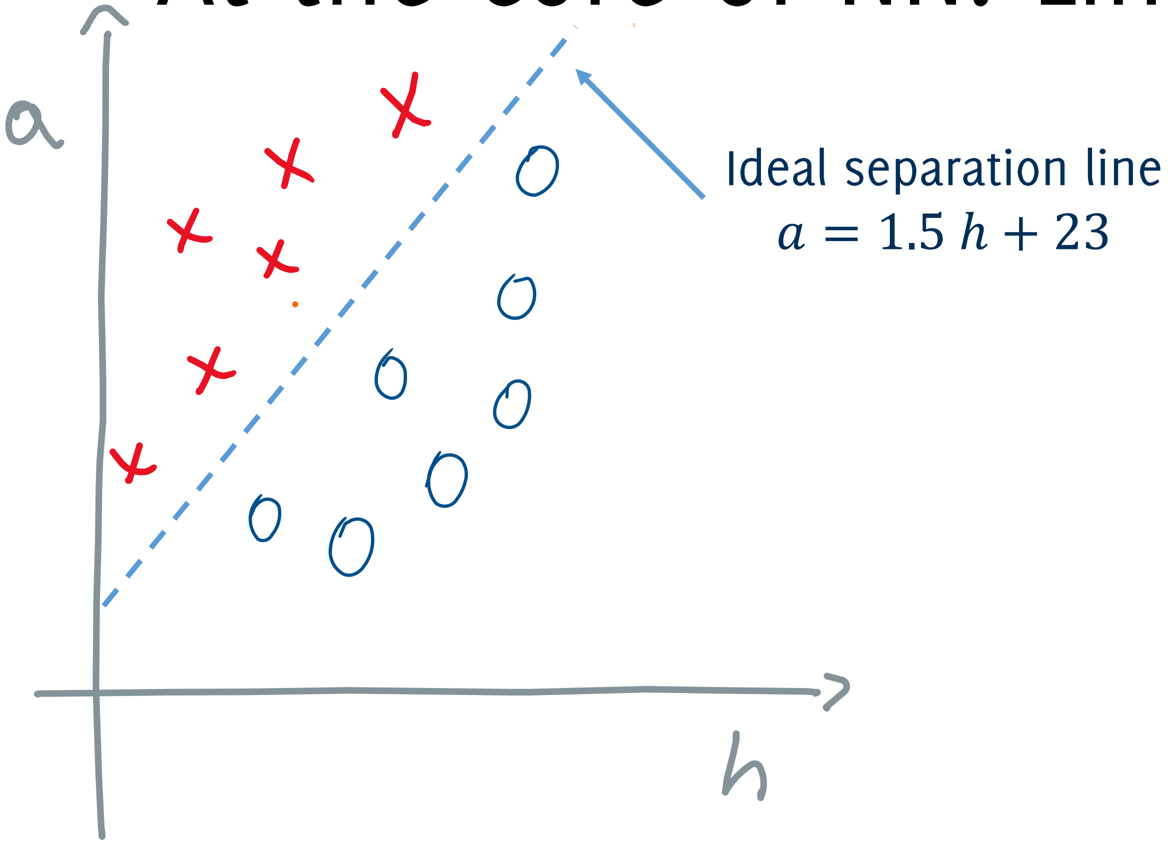
Thus, the ideal parameters are

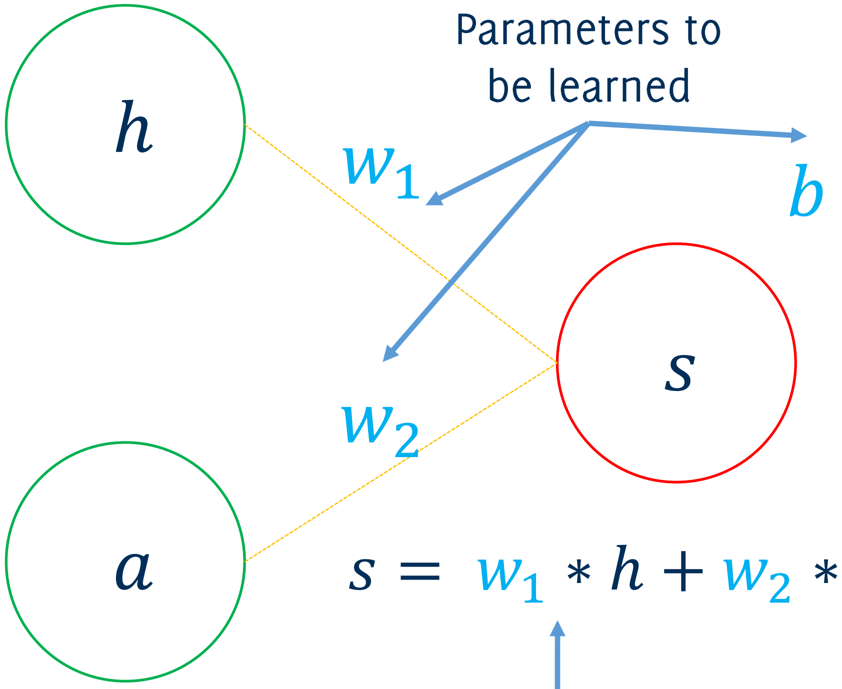
$$w_1 = 1.5, w_2 = -1 \text{ and } b = 23$$

To define the ideal score function function

$$s = 1.5 * h - a + 23$$

At the core of NN: Linear Combinations



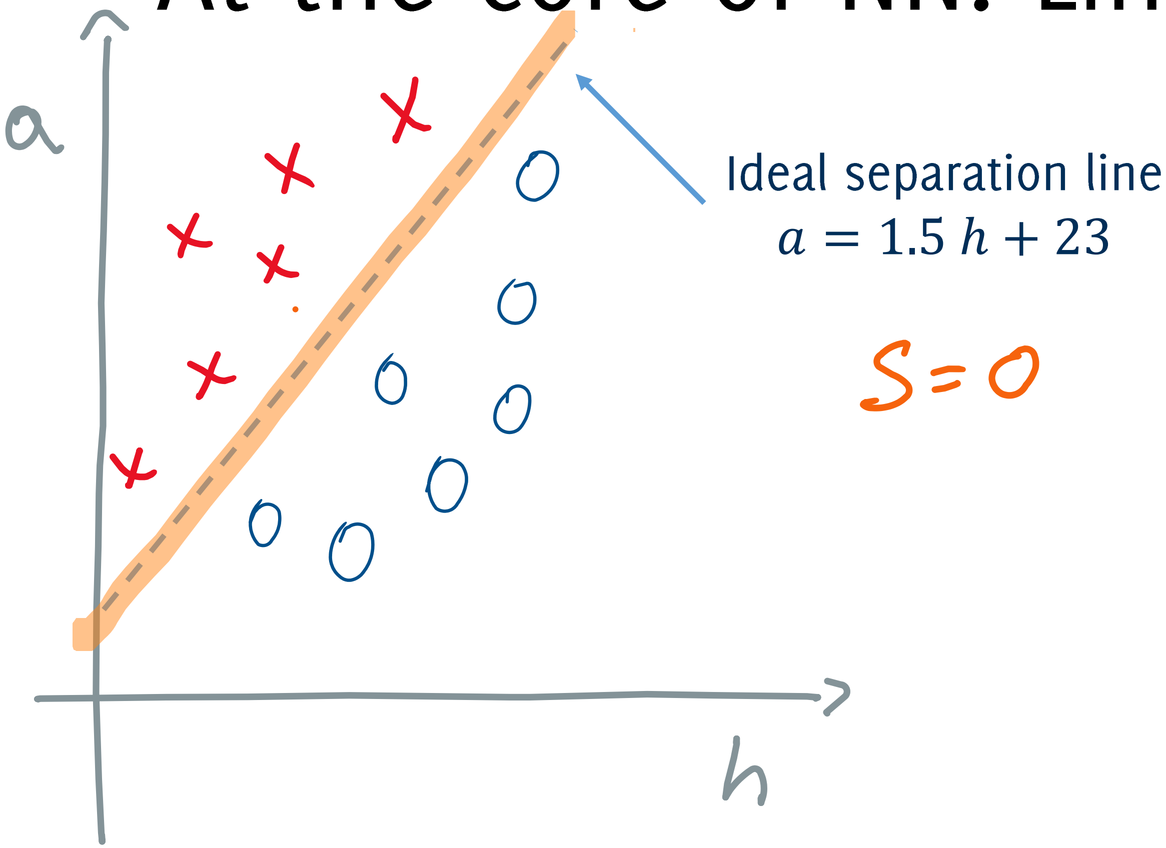


$$s = w_1 * h + w_2 * a + b$$

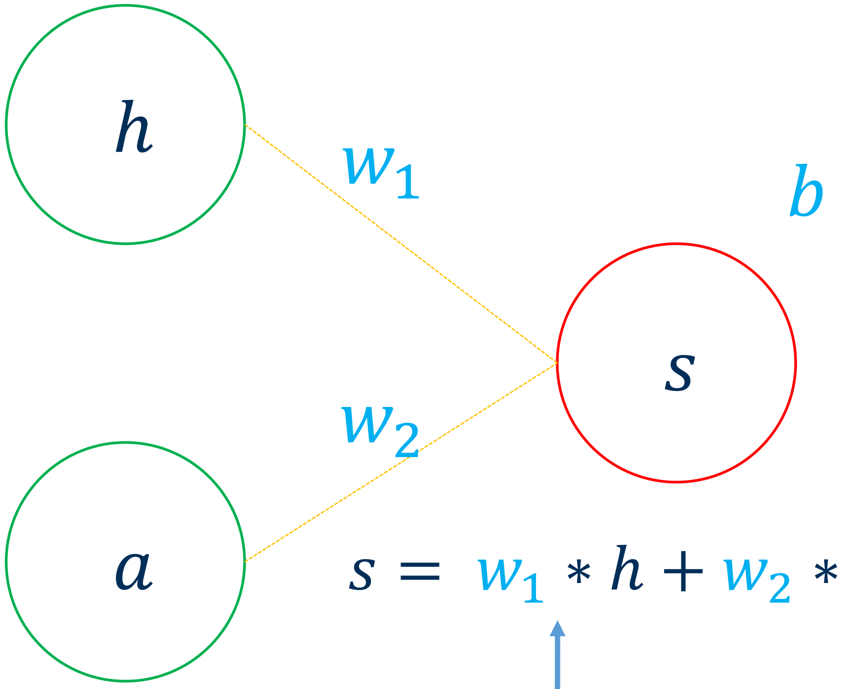
If the training is successful, the parameters will be

$w_1 = 1.5, w_2 = -1, b = 23$

At the core of NN: Linear Combinations



$$s = 1.5 * h - a + 23$$

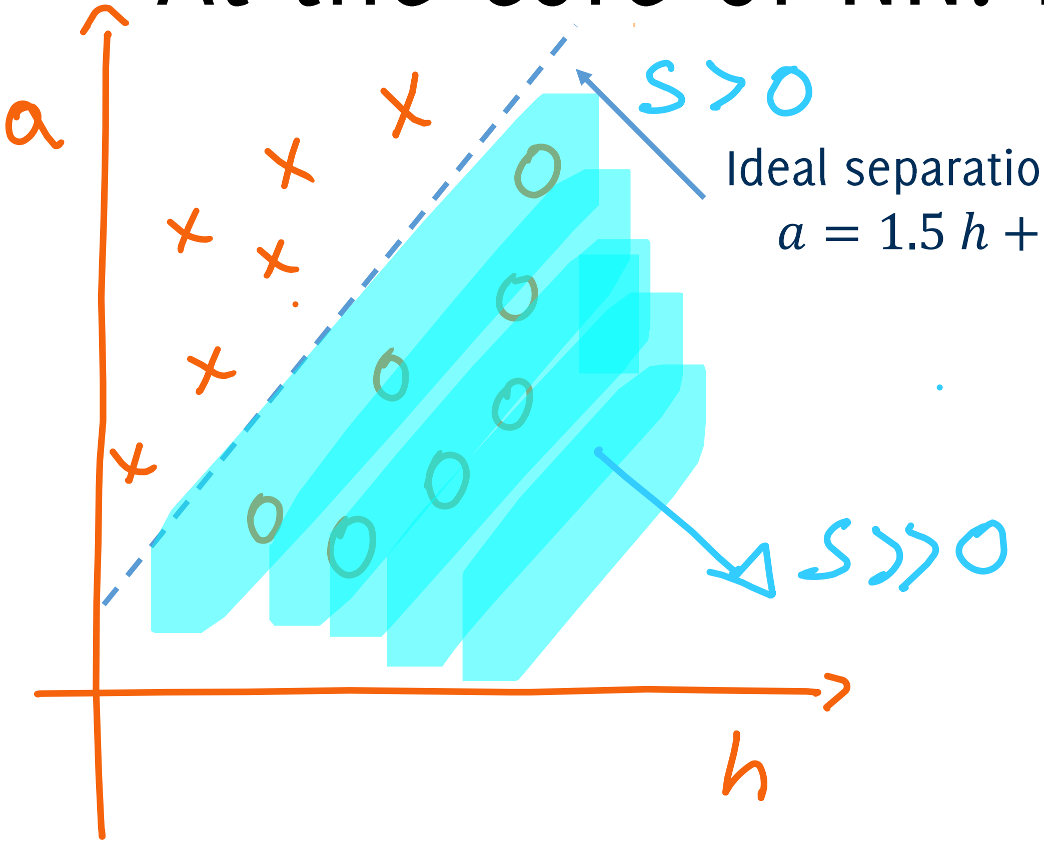


$$s = w_1 * h + w_2 * a + b$$

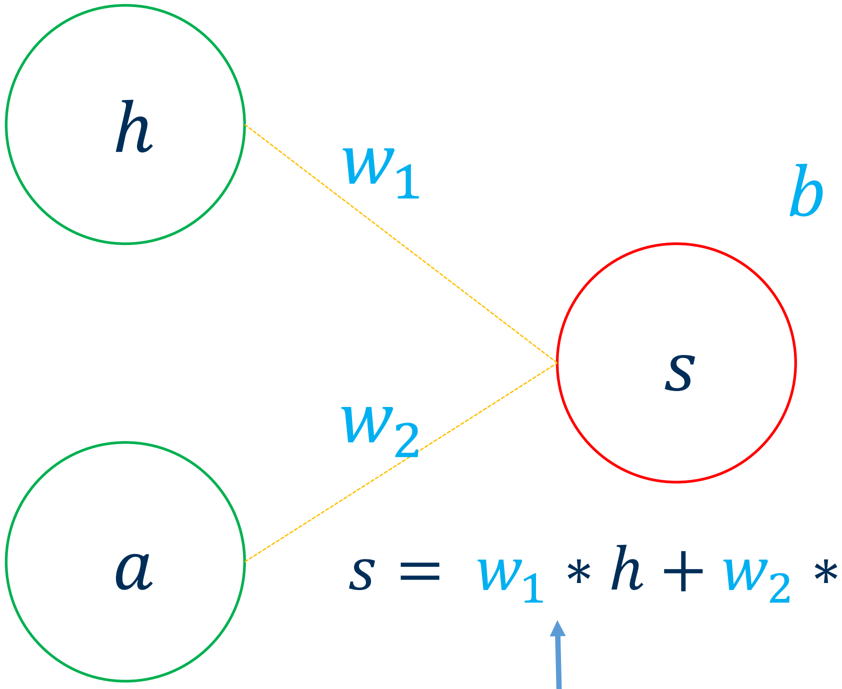
If the training is successful, the parameters will be

$$w_1 = 1.5, w_2 = -1, b = 23$$

At the core of NN: Linear Combinations



$$s = 1.5 * h - a + 23$$

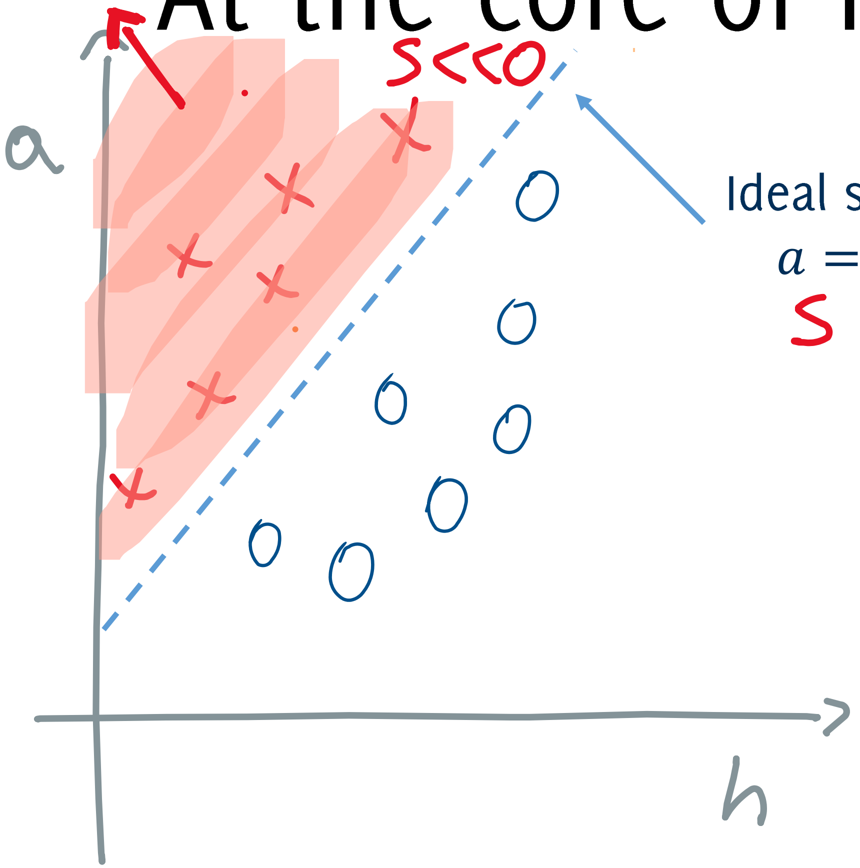


$$s = w_1 * h + w_2 * a + b$$

If the training is successful, the parameters should be

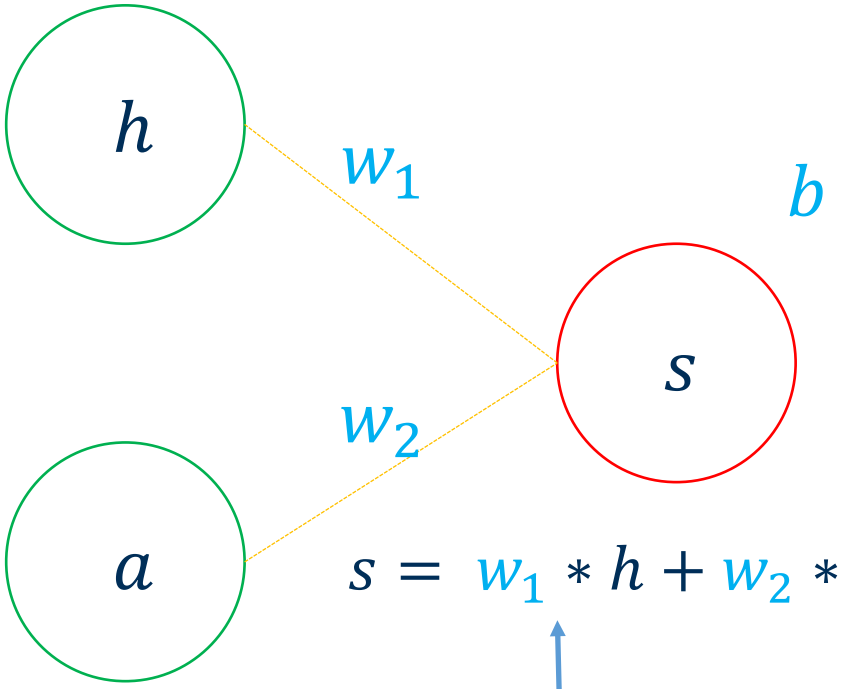
$$w_1 = 1.5, w_2 = -1, b = 23$$

At the core of NN: Linear Combinations



Ideal separation line
$a = 1.5\,h + 23$

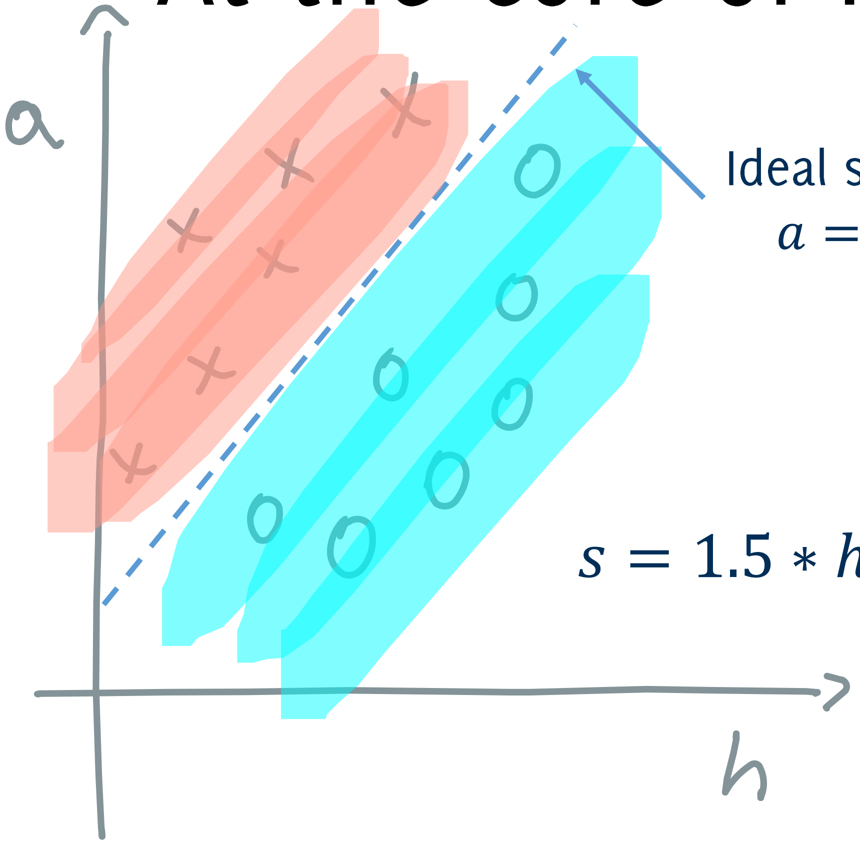
$$s = 1.5 * h - a + 23$$



$$s = w_1 * h + w_2 * a + b$$

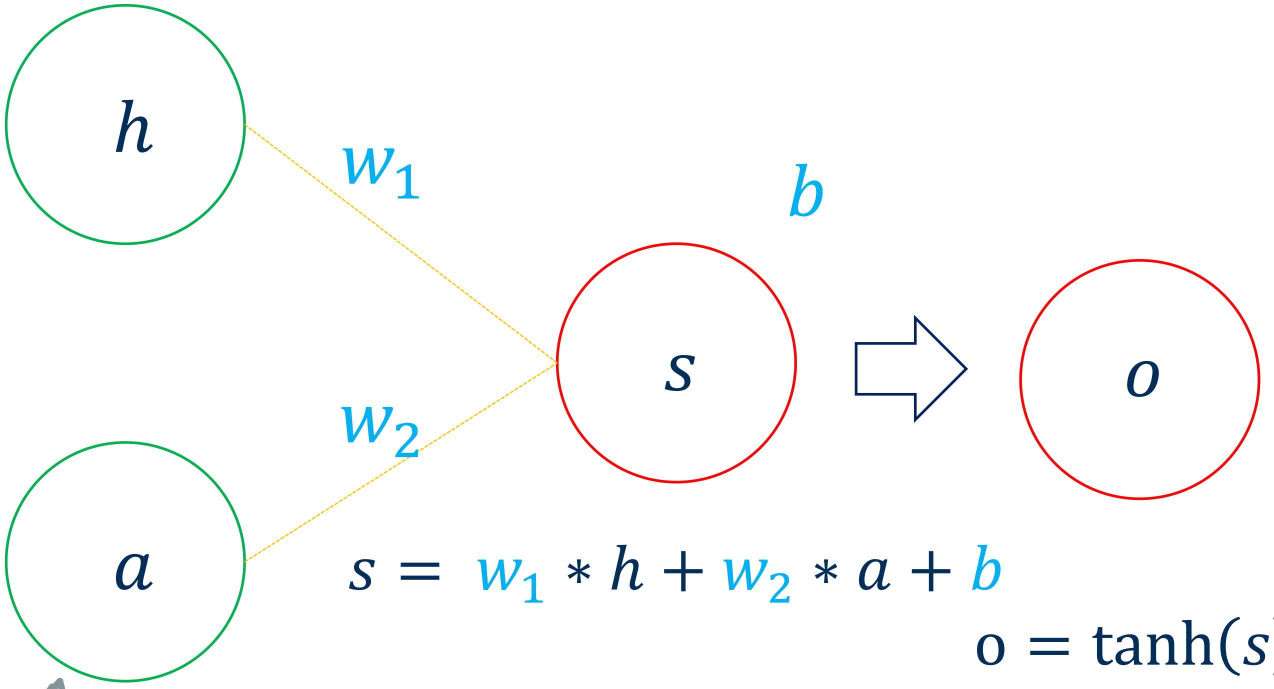
If the training is successful,
the parameters should be
$w_1 = 1.5, w_2 = -1, b = 23$

At the core of NN: Linear Combinations



Ideal separation line
$a = 1.5\,h + 23$

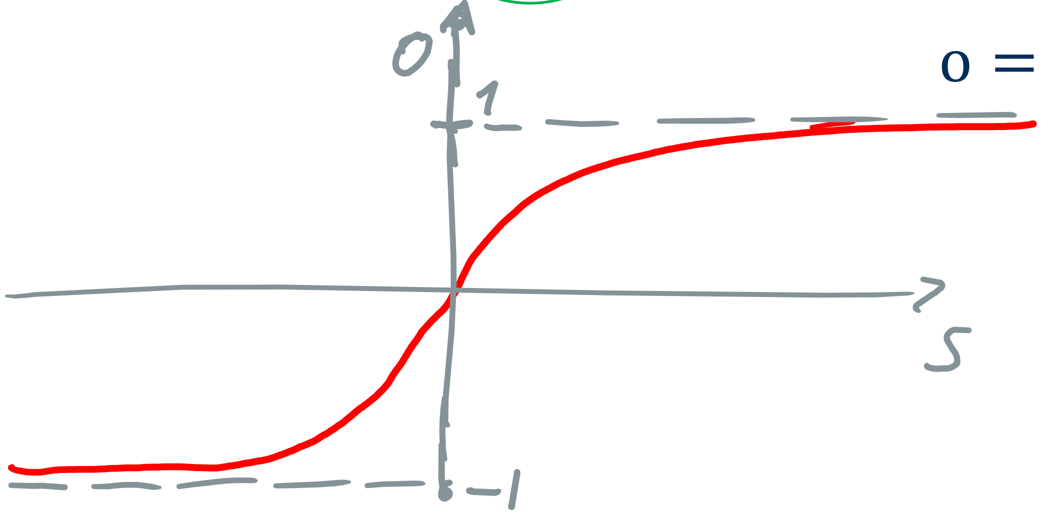
$$s = 1.5 * h - a + 23$$



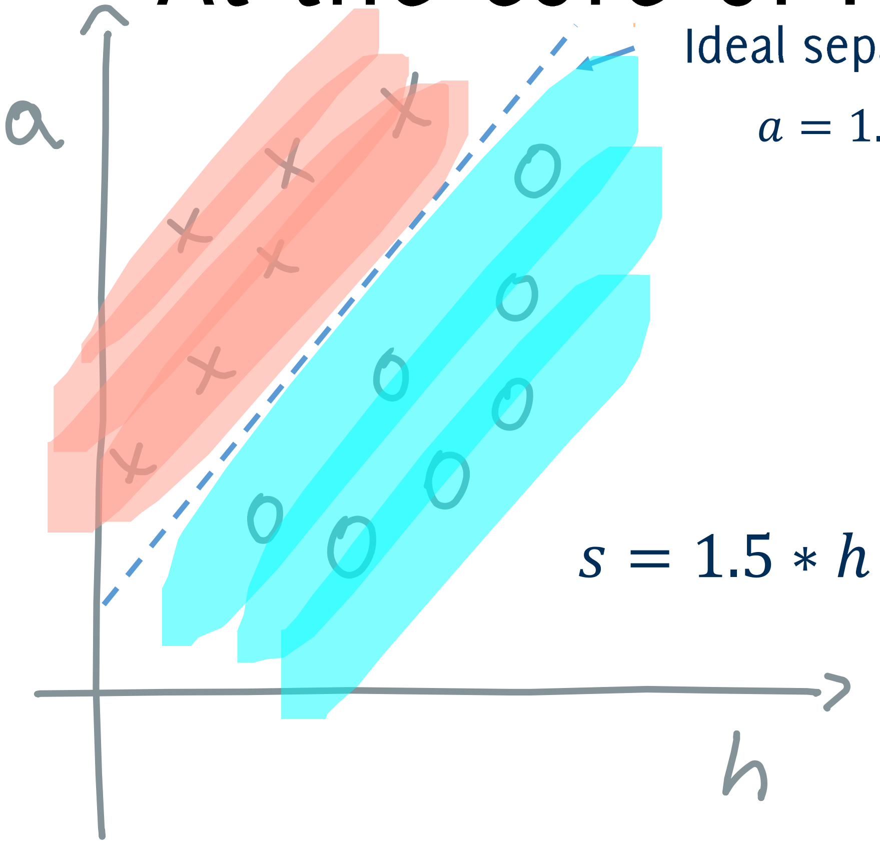
$$s = w_1 * h + w_2 * a + b$$

$$o = \tanh(s)$$

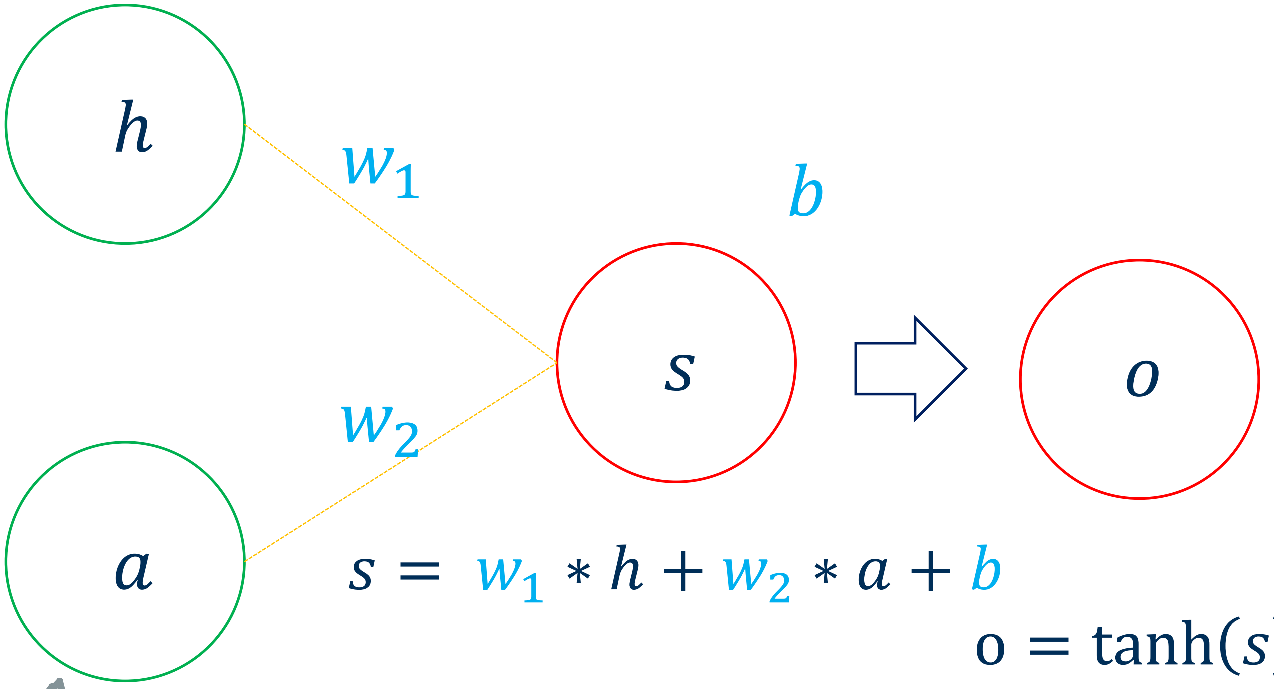
$$o = \tanh(s)$$



At the core of NN: Linear Combinations

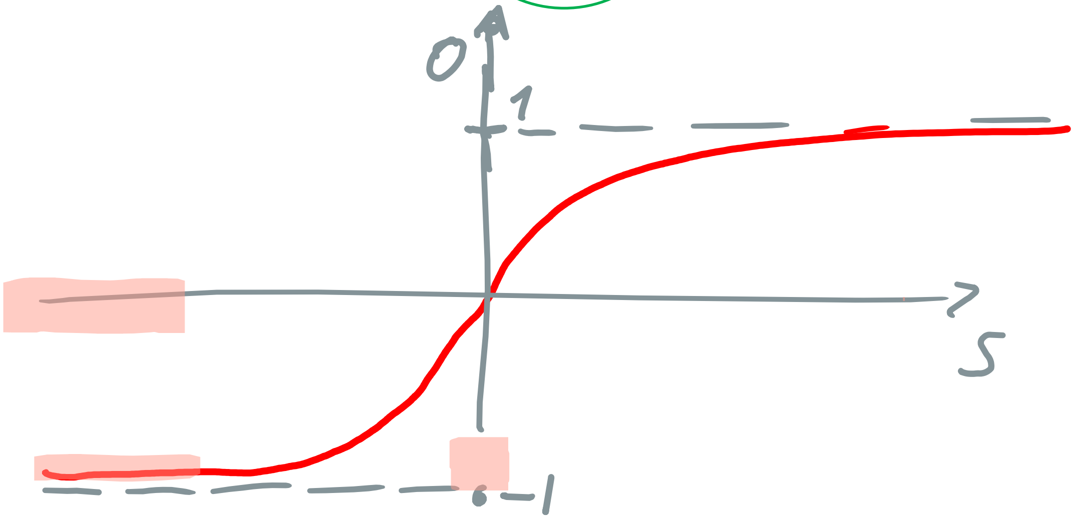


$$s = 1.5 * h - a + 23$$



$$s = w_1 * h + w_2 * a + b$$

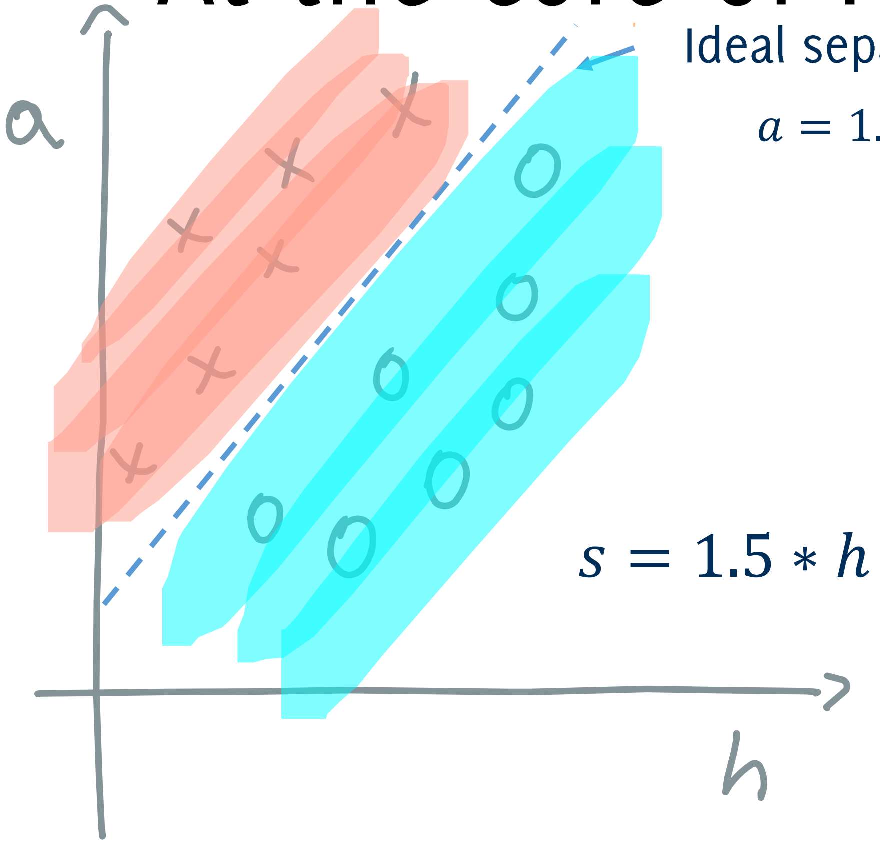
$$o = \tanh(s)$$



$s << 0$

$o \approx -1$

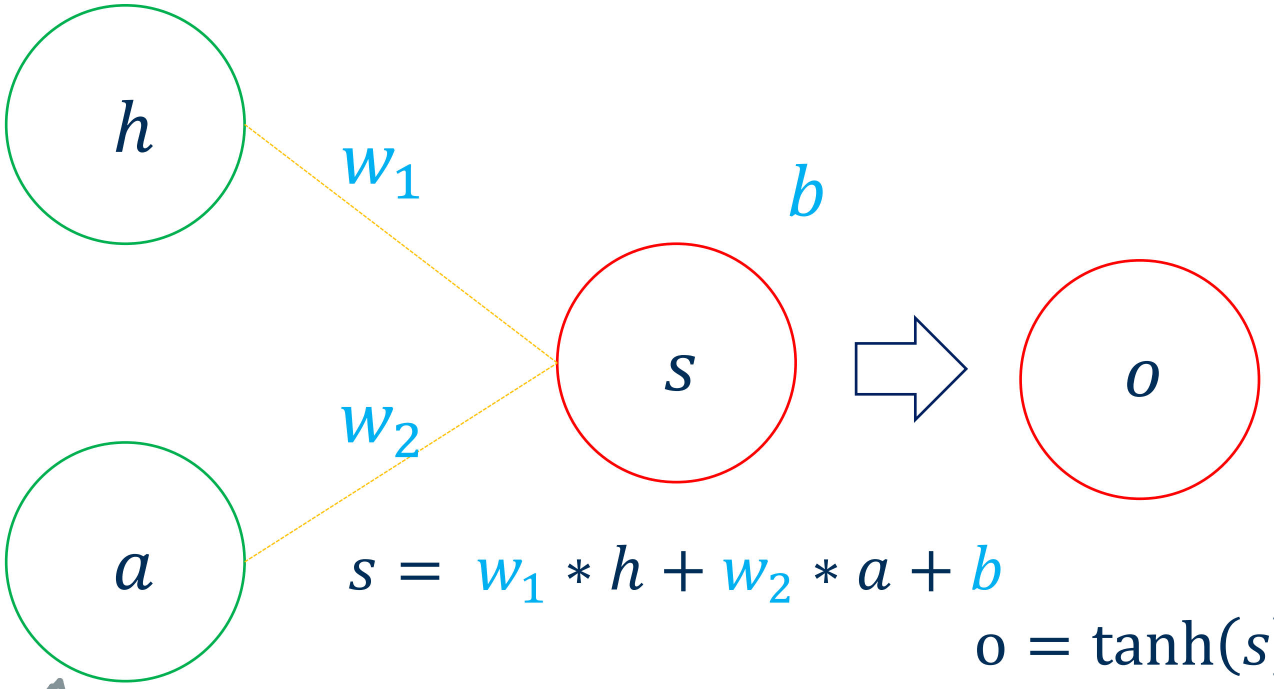
At the core of NN: Linear Combinations



Ideal separation line

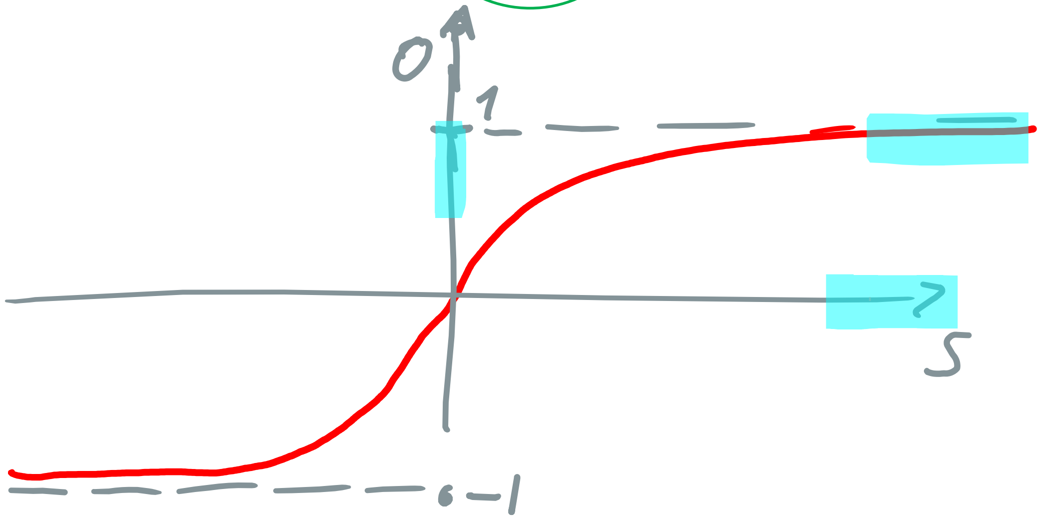
$a = 1.5\, h + 23$

$s = 1.5 * h - a + 23$



$s = w_1 * h + w_2 * a + b$

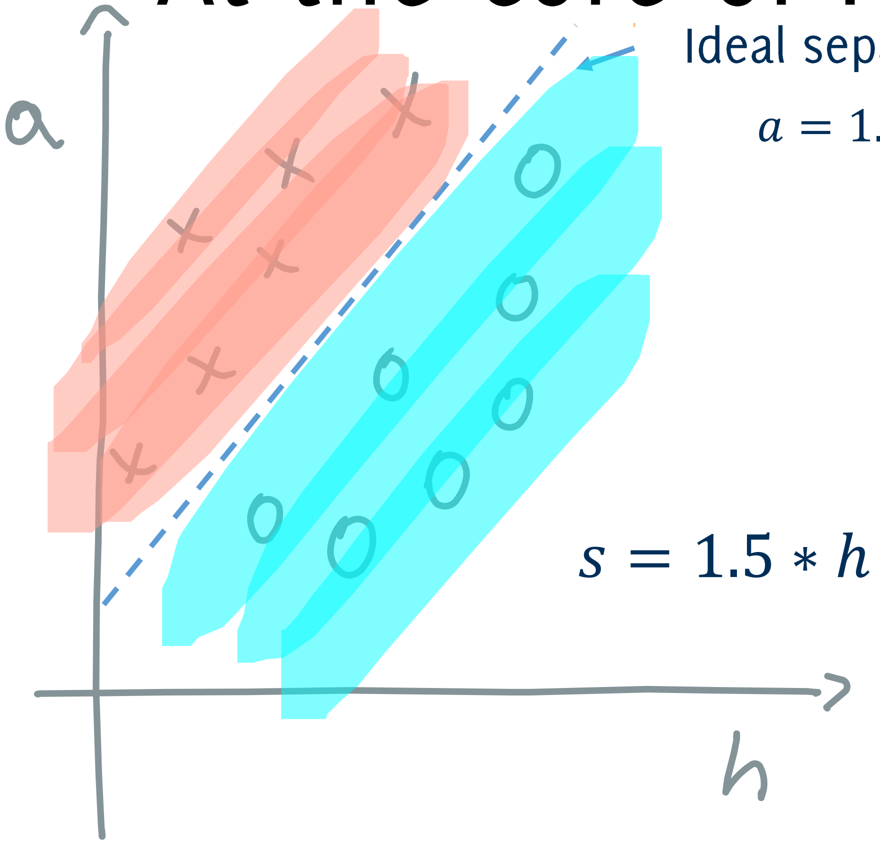
$\mathrm{o} = \tanh(s)$



$s >> 0$

$o \approx 1$

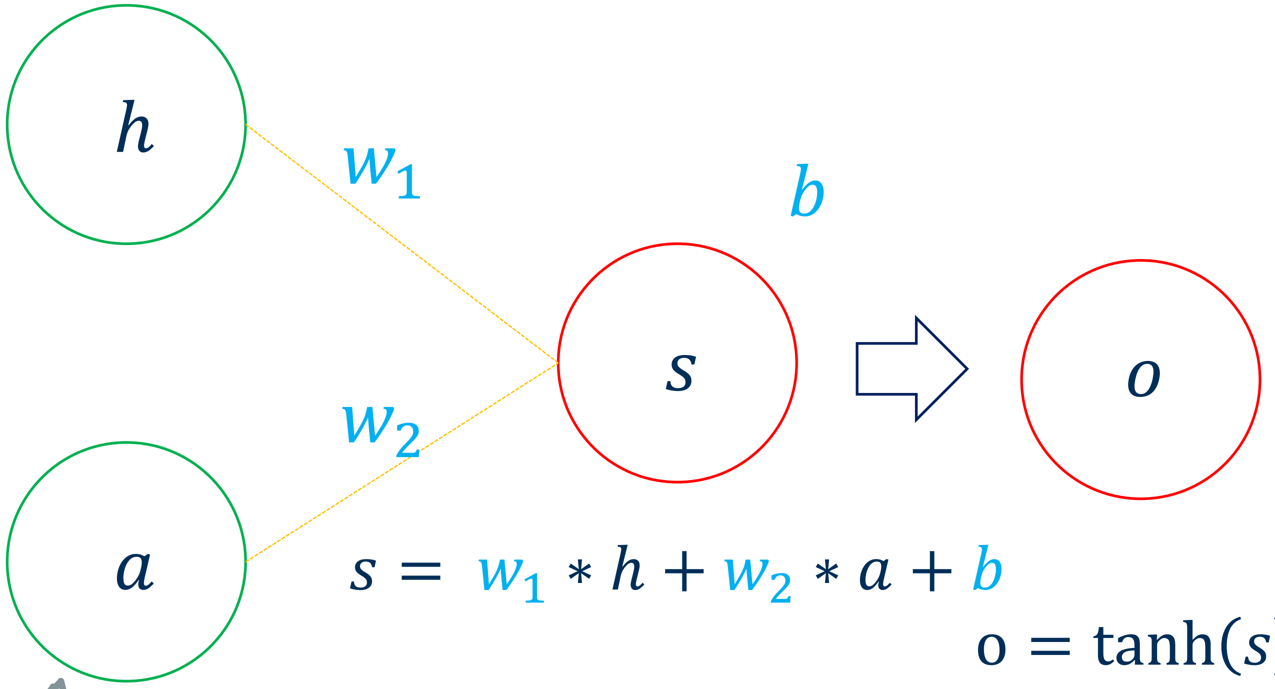
At the core of NN: Linear Combinations



Ideal separation line

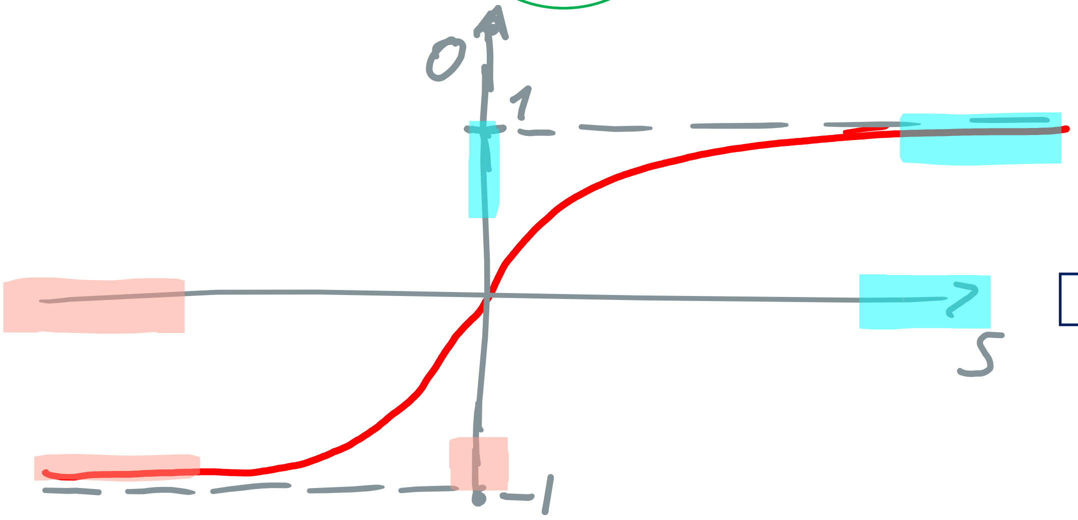
$a = 1.5\,h + 23$

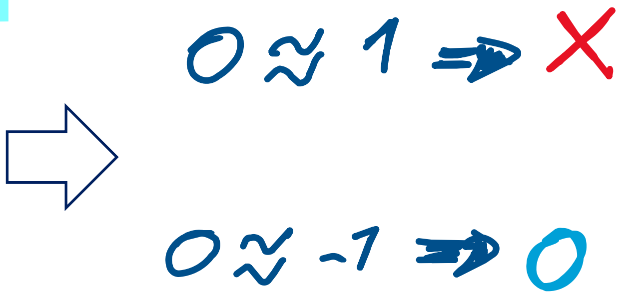
$$s = 1.5 * h - a + 23$$



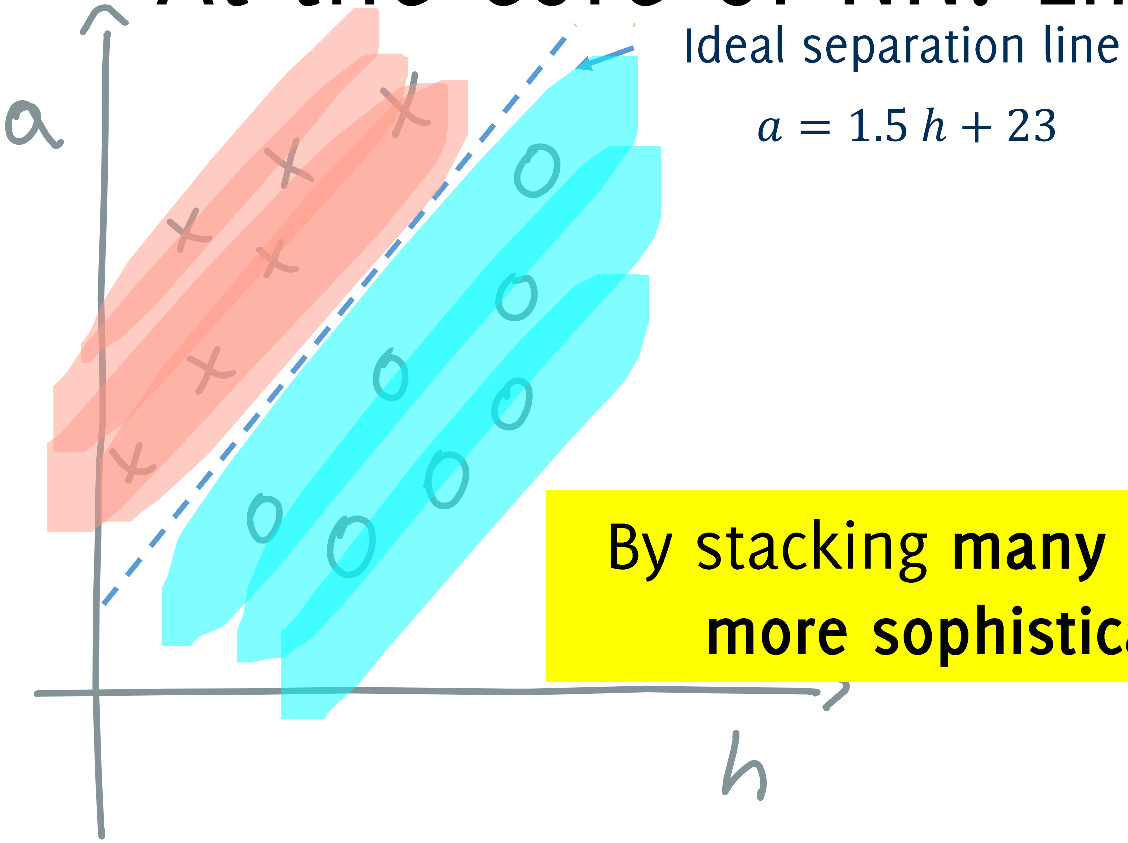
$$s = w_1 * h + w_2 * a + b$$

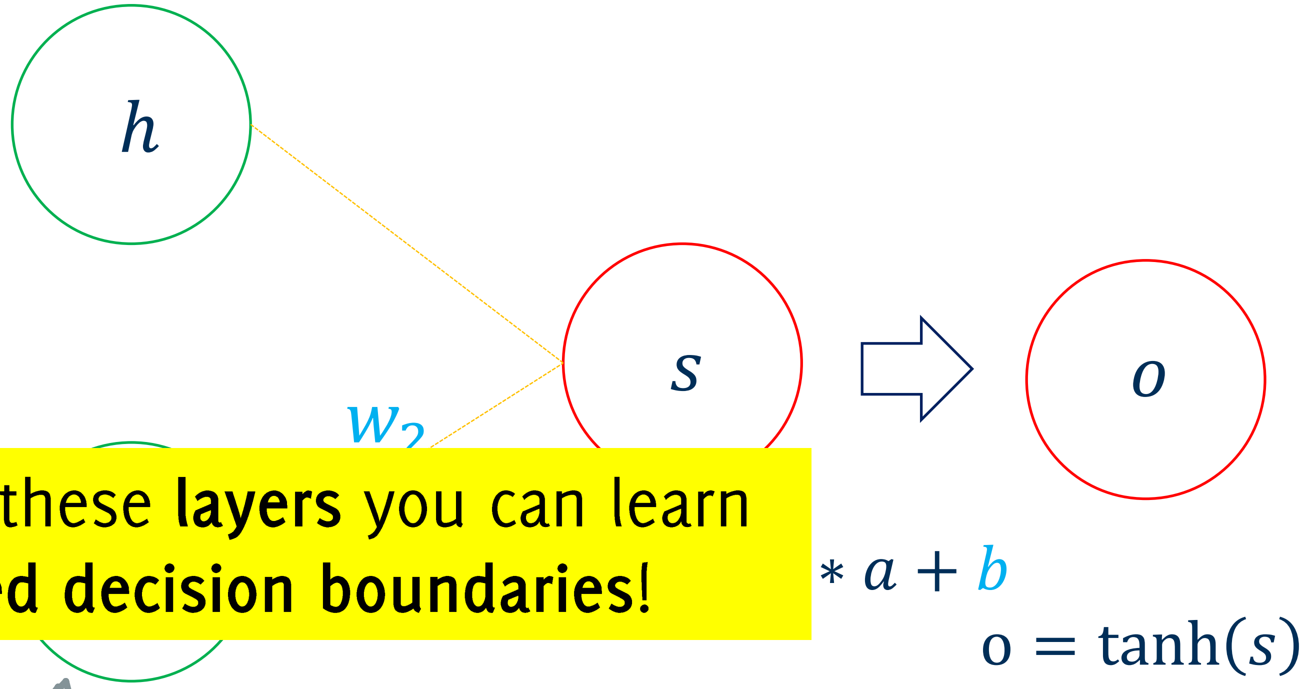
$$o = \tanh(s)$$

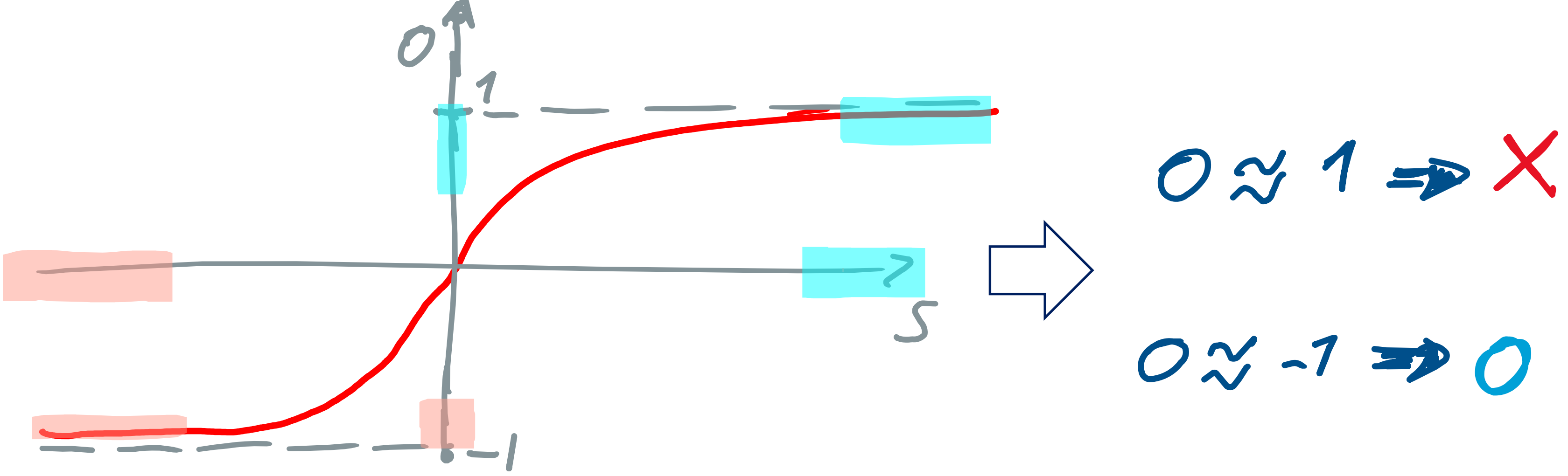




At the core of NN: Linear Combinations







By stacking **many** of these **layers** you can learn **more sophisticated decision boundaries!**

Neural Network Training

Training

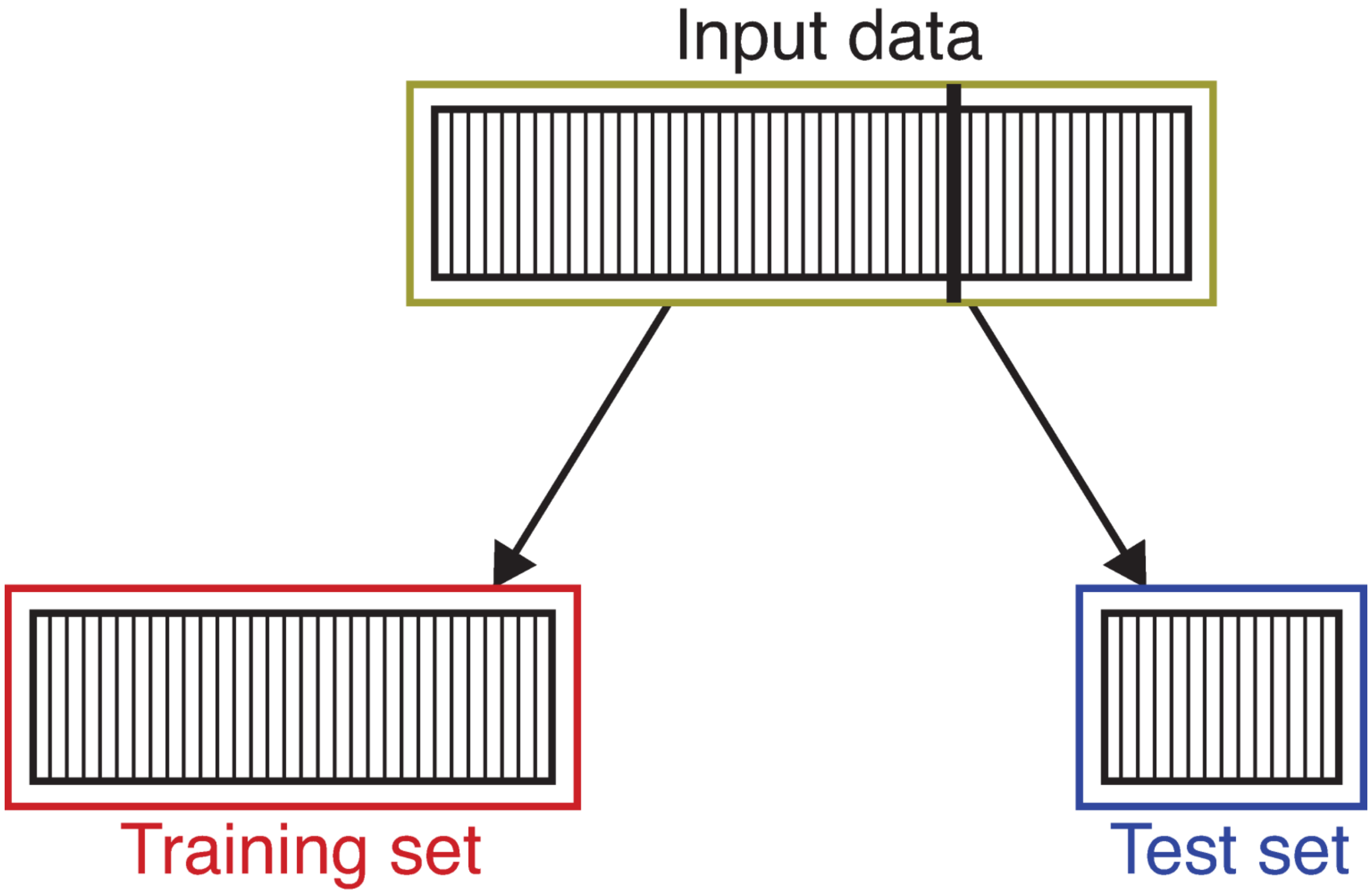
The process of taking a NN that's been initialized with default or random values and gradually improving it so that it "generalize" well.

Training, testing

Training set: the data used to learn the model parameters

Test set: used only at the end to perform final model assessment

$$TR = \{(\boldsymbol{x}, \boldsymbol{t})_i, i = 1, \dots, N\}$$



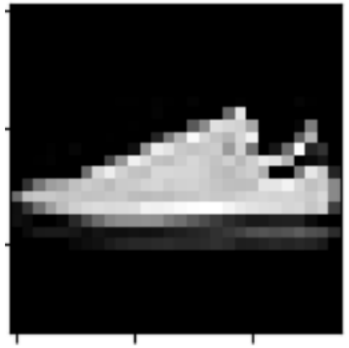
Training

Given:

- the training set $TR = \{(\boldsymbol{x}, \boldsymbol{t})_i, i = 1, \dots, N\}$,
- a Neural Network $f(\boldsymbol{x}, W)$ that depends on a collection of parameters W,

the training optimizes the values of W such that f "learns" the correct values on the training set.

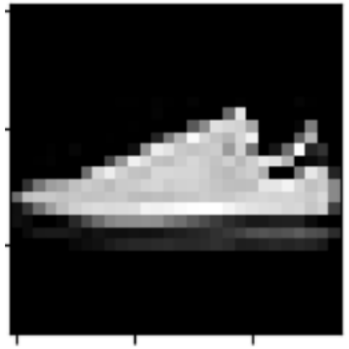
Before training



$\boldsymbol{x}_i$

$f(\boldsymbol{x}, W_{\text{rand}})$ → pullover

After training



$\boldsymbol{x}_i$

$f(\boldsymbol{x}, W_{\text{opt}})$ → sneaker

$\boldsymbol{t}_i$

Training

In practice, networks learn by minimizing their mistakes encoded in a a loss function (the lower the more accurate f is in predicting the target values $\boldsymbol{t}$).

For example (mean squared error)

$$L(W, \boldsymbol{x}_i, \boldsymbol{t}_i) = \frac{1}{N}(f(W, \boldsymbol{x}_i) - \boldsymbol{t}_i)^2$$

The training (hopefully) returns the parameters W of the weights that minimize the loss (the mistakes on the training set)

Training

However we don’t care very much on mistakes on the Training Set, we want that our network can correctly predict labels on unseen data.
We assess our model on the Test Set.

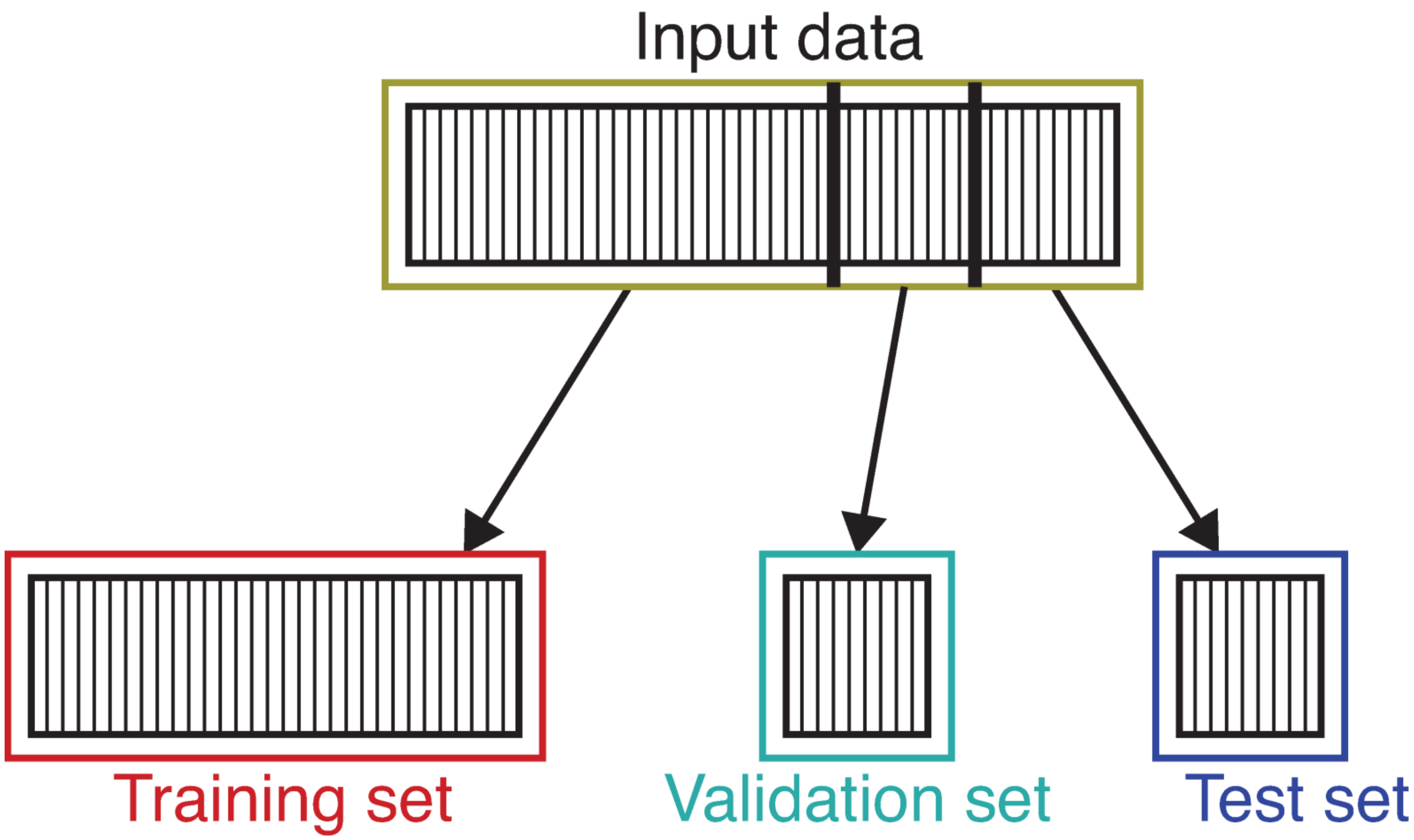
In the metaphor of learning, it is the same difference as «parroting» the lesson, or really understanding what one has studied.

Training, testing and validation

Training set: the data used to learn the model parameters

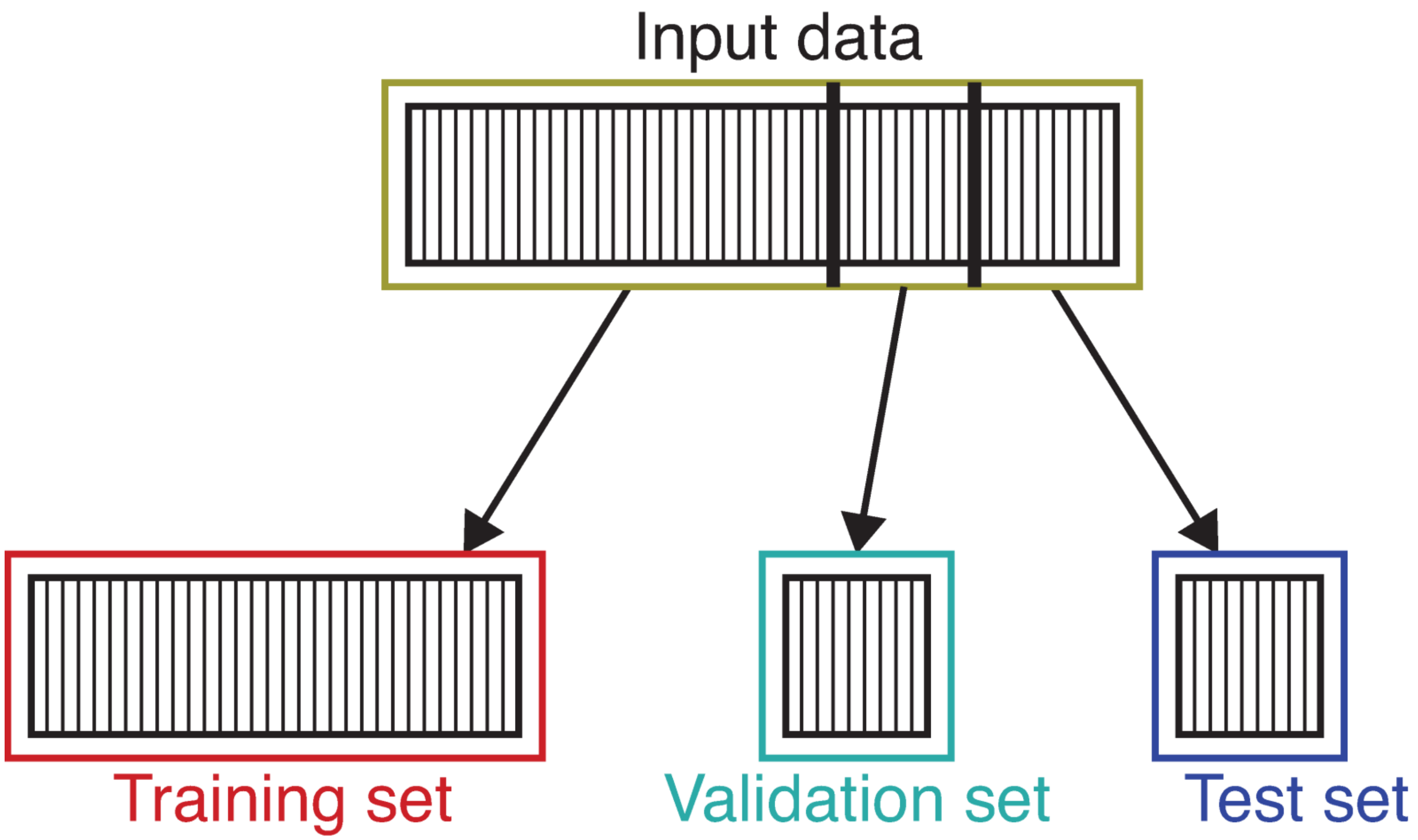
Test set: used only at the end to perform final model assessment

Validation set: the data used to perform "model selection". The validation set is also used to assess stopping criteria during training.

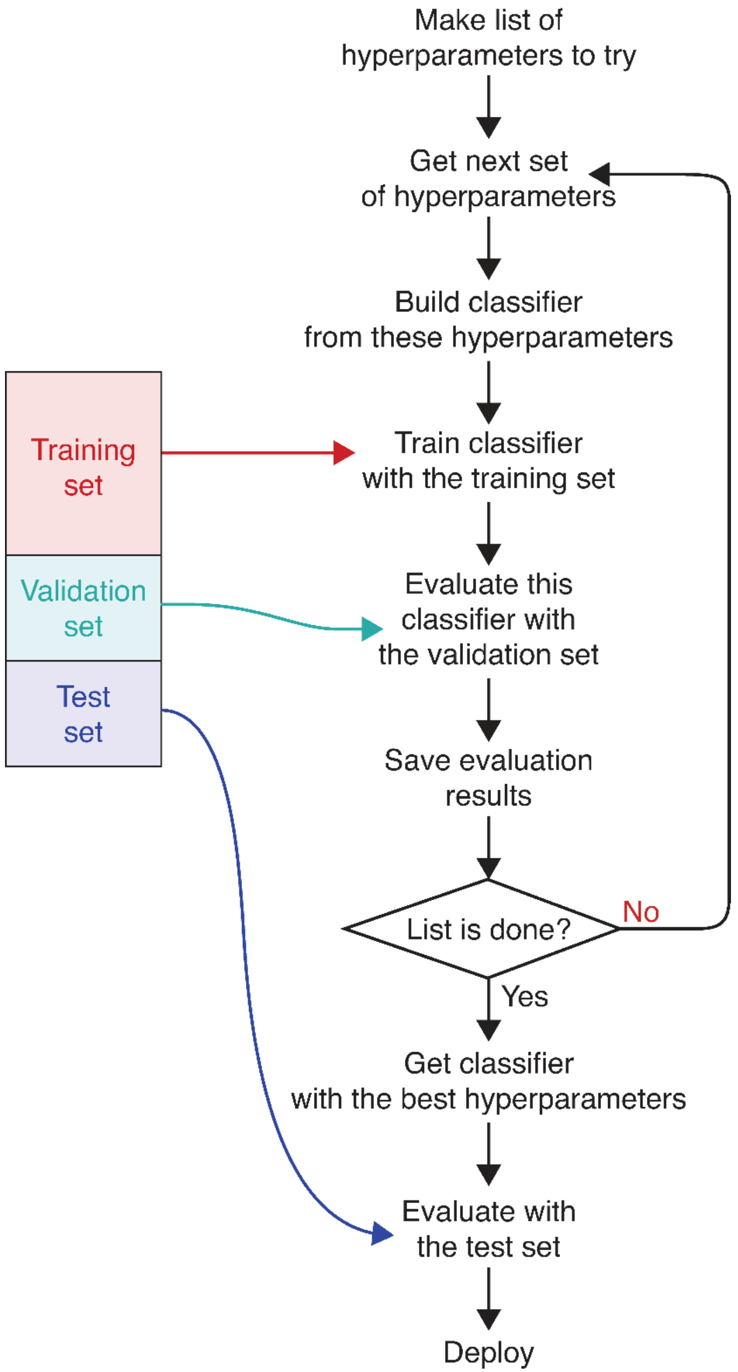


Training, testing and validation

We want that all the splits have the same distribution of the input data.

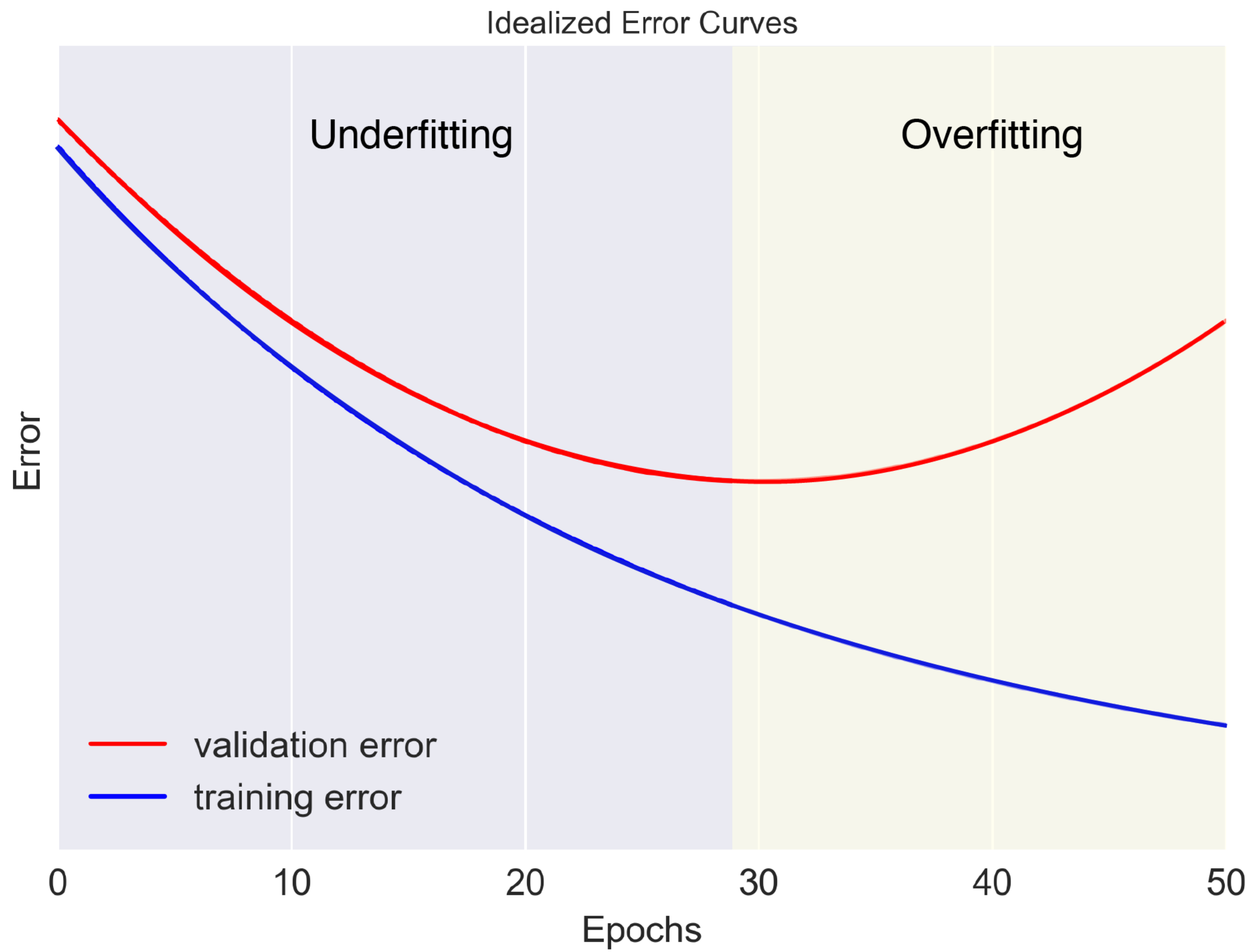


Validation data

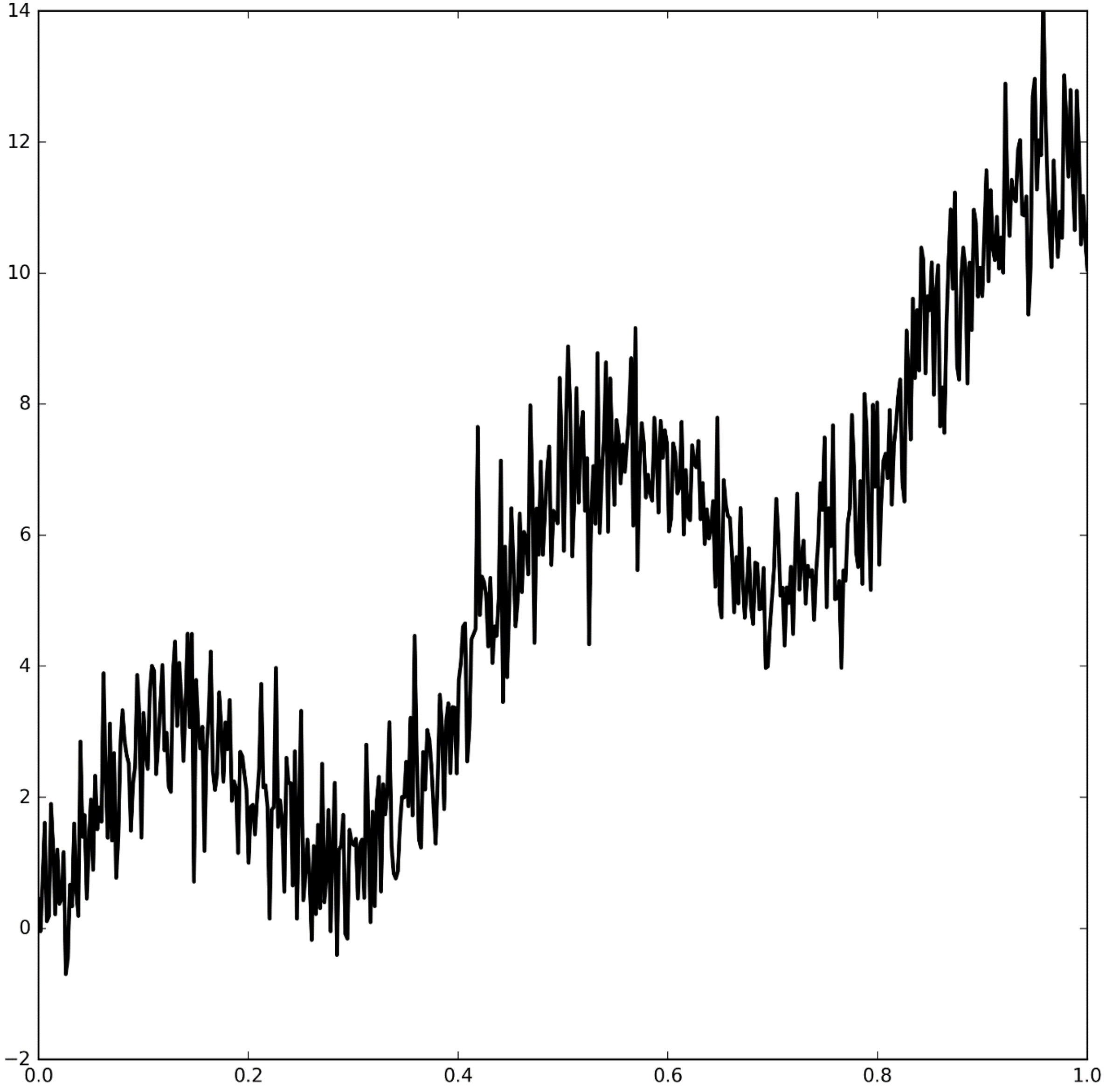


A good proxy of the real-world data we can use to deploy the system to test different hyperparameters and perform model selection.

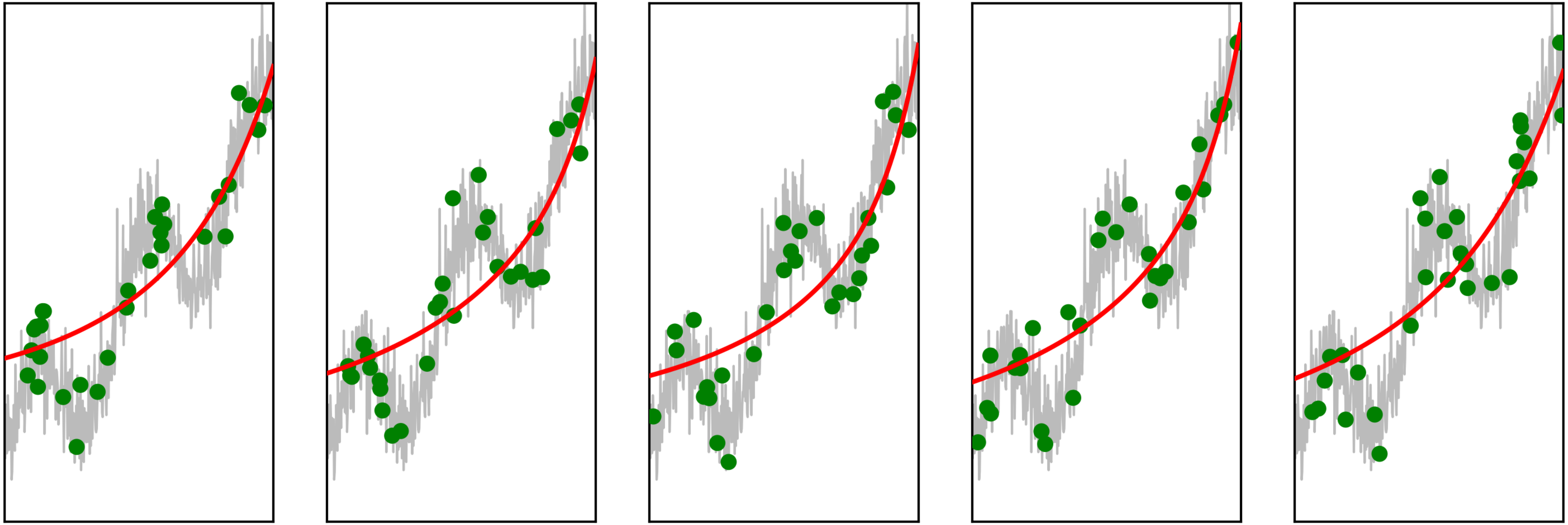
Underfitting and overfitting



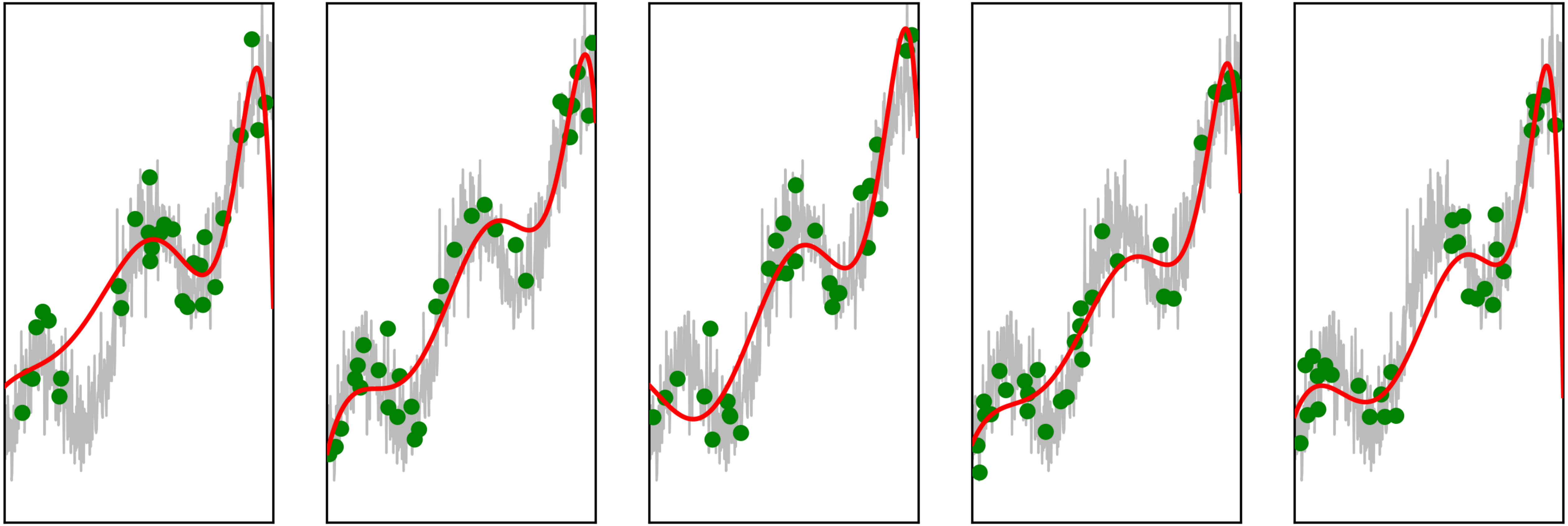
Occam's razor



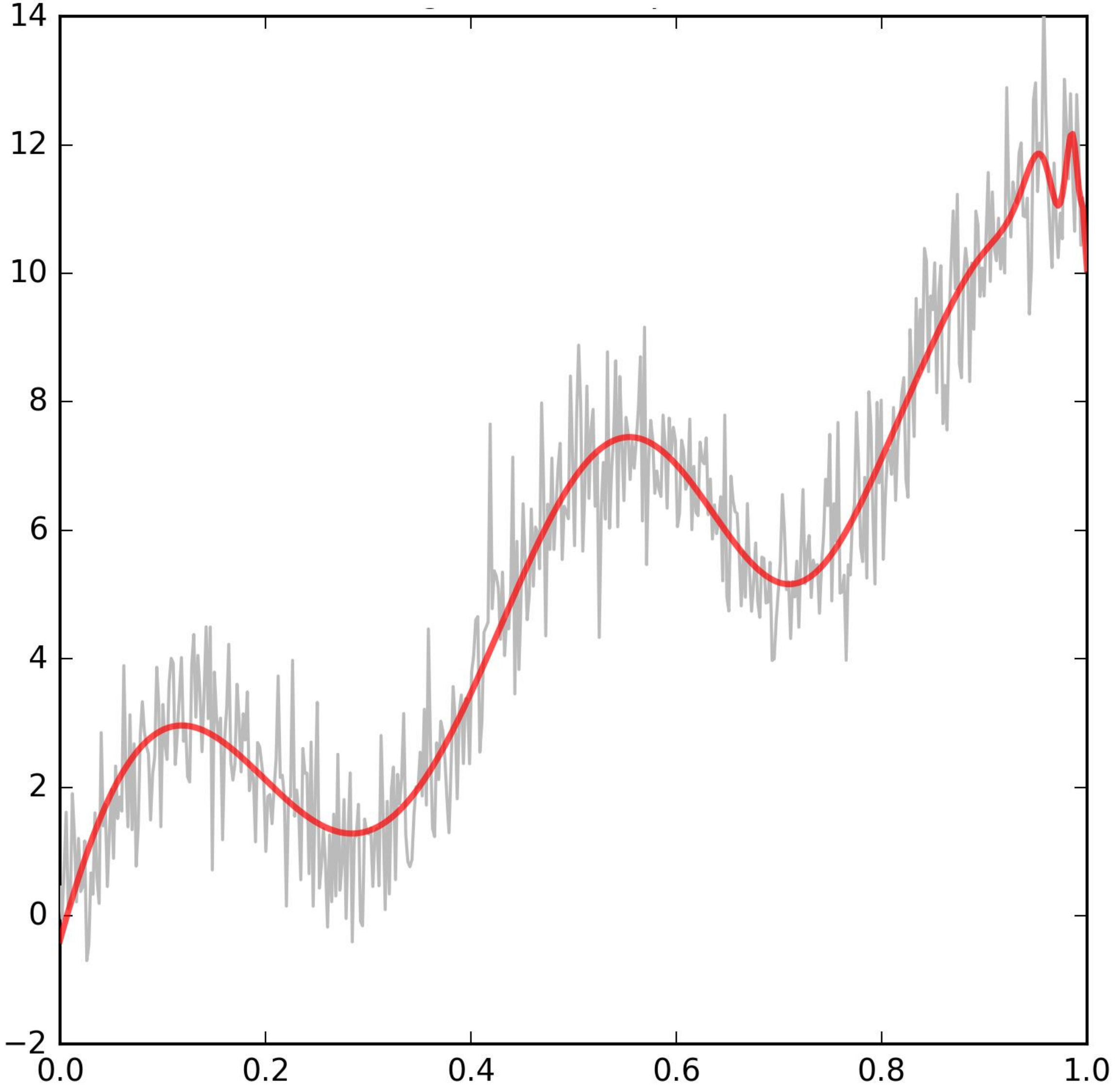
Under-fitting



Over-fitting



Occam's razor



How to prevent overfitting?

- early stopping
- add a regularization in the loss
- drop-out

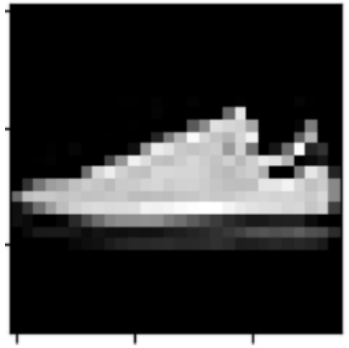
Network Training

Given:

- the training set $TR = \{(x, y)_i, i = 1, \dots, N\}$,
- a Neural Network f_θ that depends on a collection of parameters θ,

the training optimizes the values of θ such that f "learns" the correct values on the training set.

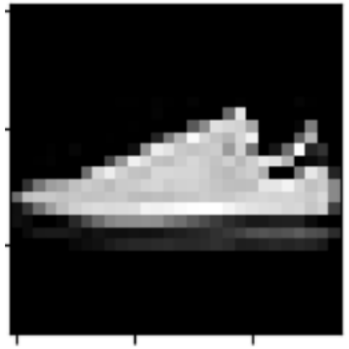
Before training



$\boldsymbol{x_i}$

$f_{\theta_{\text{rand}}}(\boldsymbol{x})$ → pullover

After training



$\boldsymbol{x_i}$

$f_{\theta_{\text{opt}}}(\boldsymbol{x})$ → sneaker

$\boldsymbol{t_i}$

Training in Supervised Settings

Networks learn by minimizing a loss function over the training set

$$TR = \{(x, y)_i, i = 1, \dots, N\},$$

The loss function

$$\mathcal{L}(\theta, TR) \in \mathbb{R}$$

returns a number that *is low* when f_θ *is good at predicting* the target y over the entire TR. The loss function accounts of all the errors on TR.

Network training is an optimization process:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \, \mathcal{L}(\theta, TR)$$

Namely, finding the parameters θ of the weights that minimize the loss

An Important Benefit of Neural Networks

- Losses used can be written and derived w.r.t. the network parameters.
- You do not simply know "the value of $\mathcal{L}(\theta, TR)$" for a given value of θ, but you also know $\nabla\mathcal{L}(\theta, TR)$, which tells you how to modify θ to reduce the value of the loss.
- Network training (namely parameters optimization) can be performed by **Gradient Descent**

$$\theta^{(i+1)} = \theta^{(i)} - \gamma\nabla\mathcal{L}\left(\theta^{(i)}, TR\right)$$

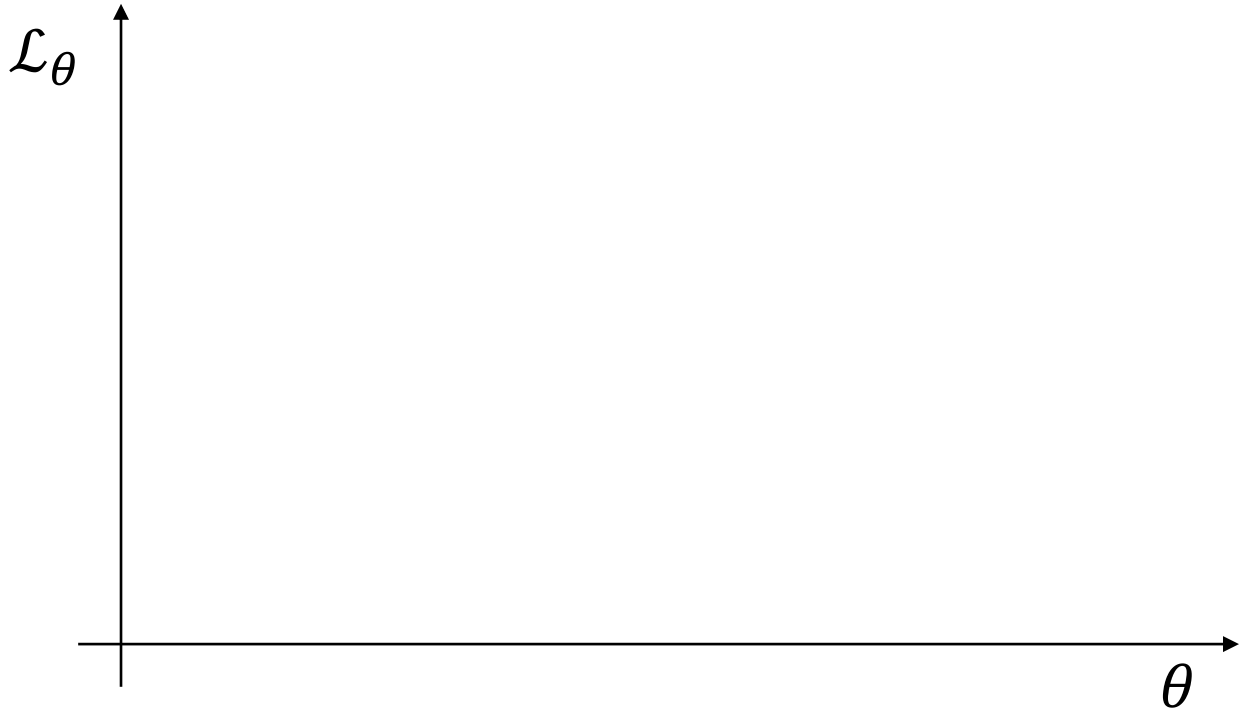
This iterative procedure converges to a local minima of the loss function (no guarantees of hitting the global minima). The $\gamma > 0$ parameter regulates the convergence speed and needs to be carefully adjusted to prevent the procedure to diverge

The Network Training

It’s an optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure**: gradient descent.

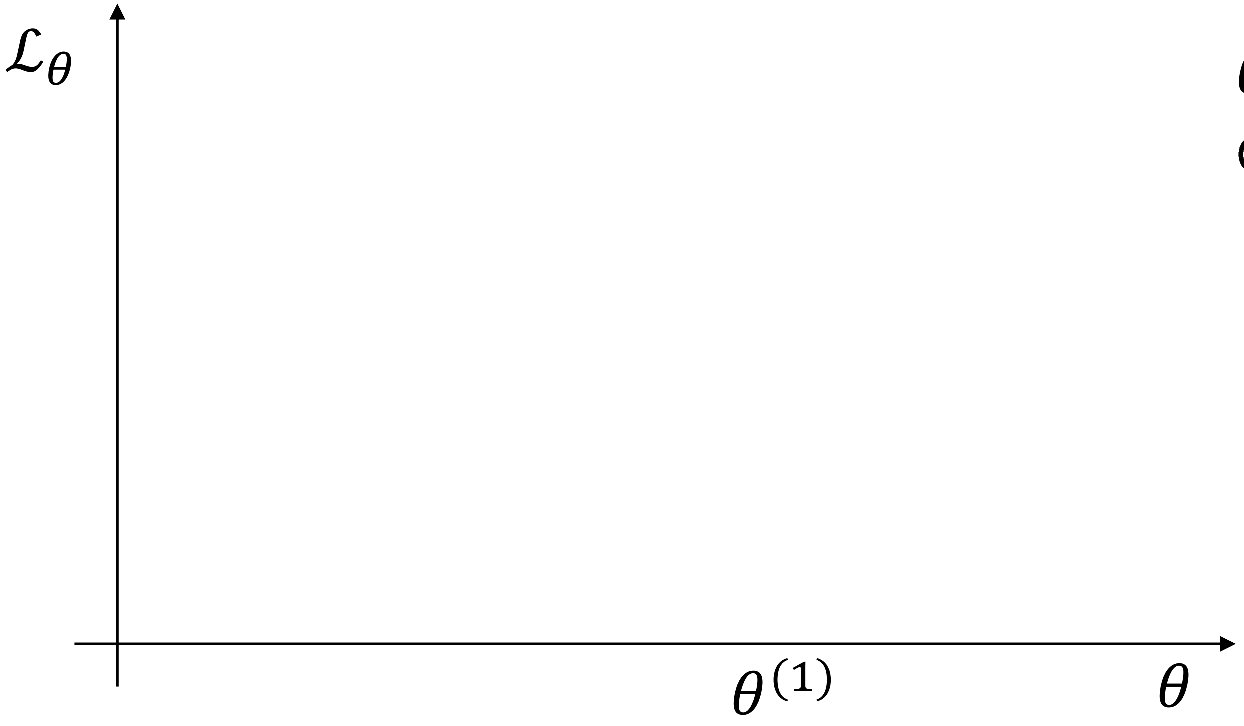


The Network Training

It's an optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure**: gradient descent.



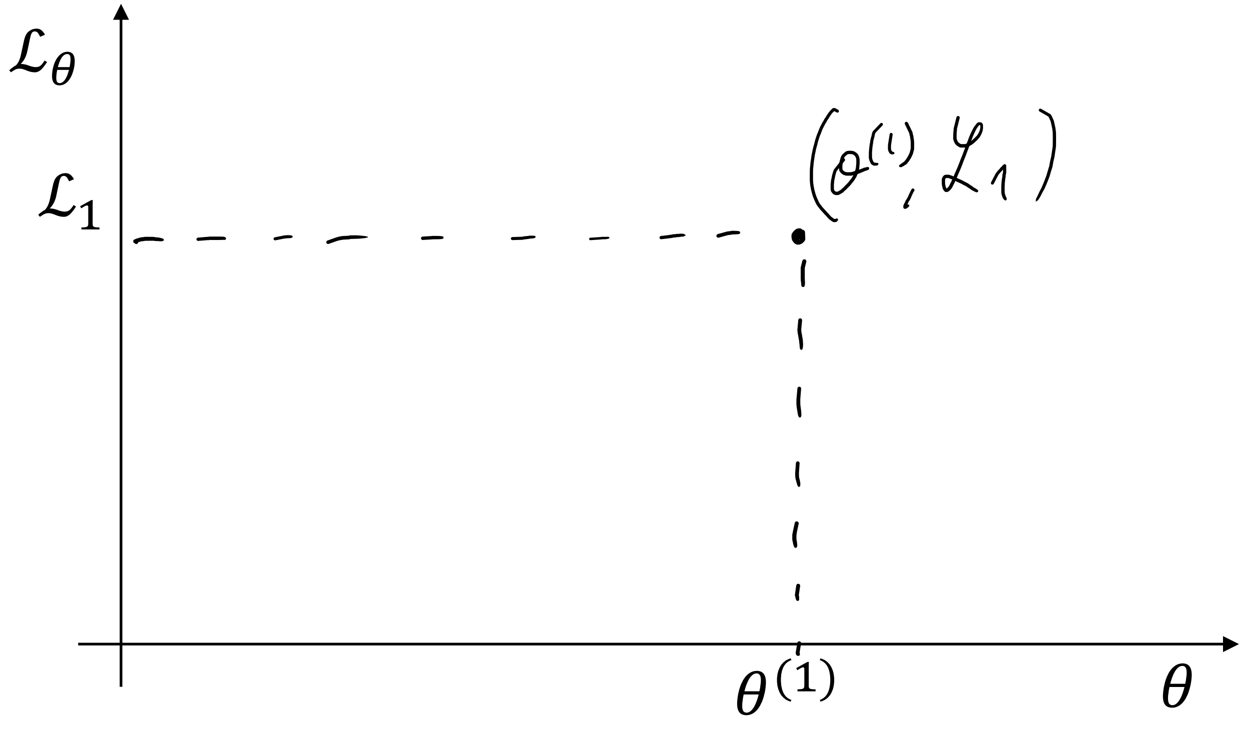
θ^1 inizialized at random or by special procedures

The Network Training

It's an optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure**: gradient descent.



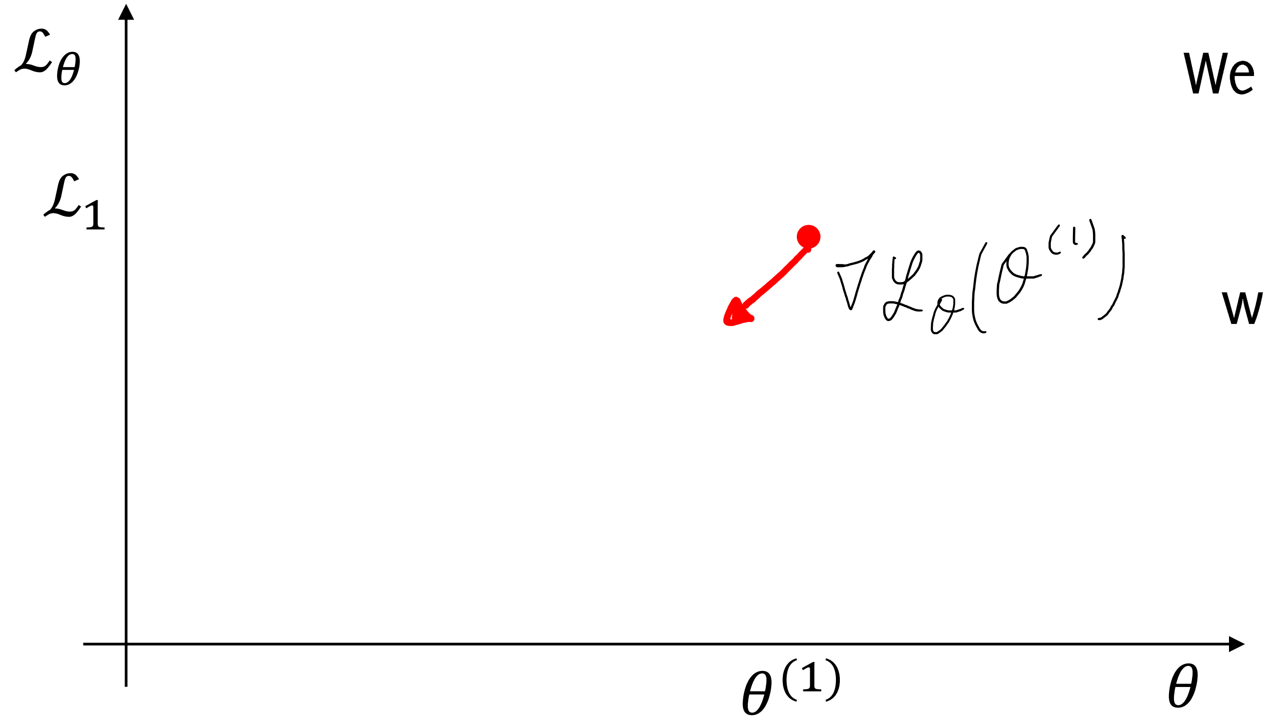
Test many images and compute the loss at θ^1, namely $\mathcal{L}_1 = \mathcal{L}(\theta^1, TR)$

The Network Training

It's an optimization problem

$$\theta^* = \underset{\theta}{\text{argmin}}\, \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure:** gradient descent.



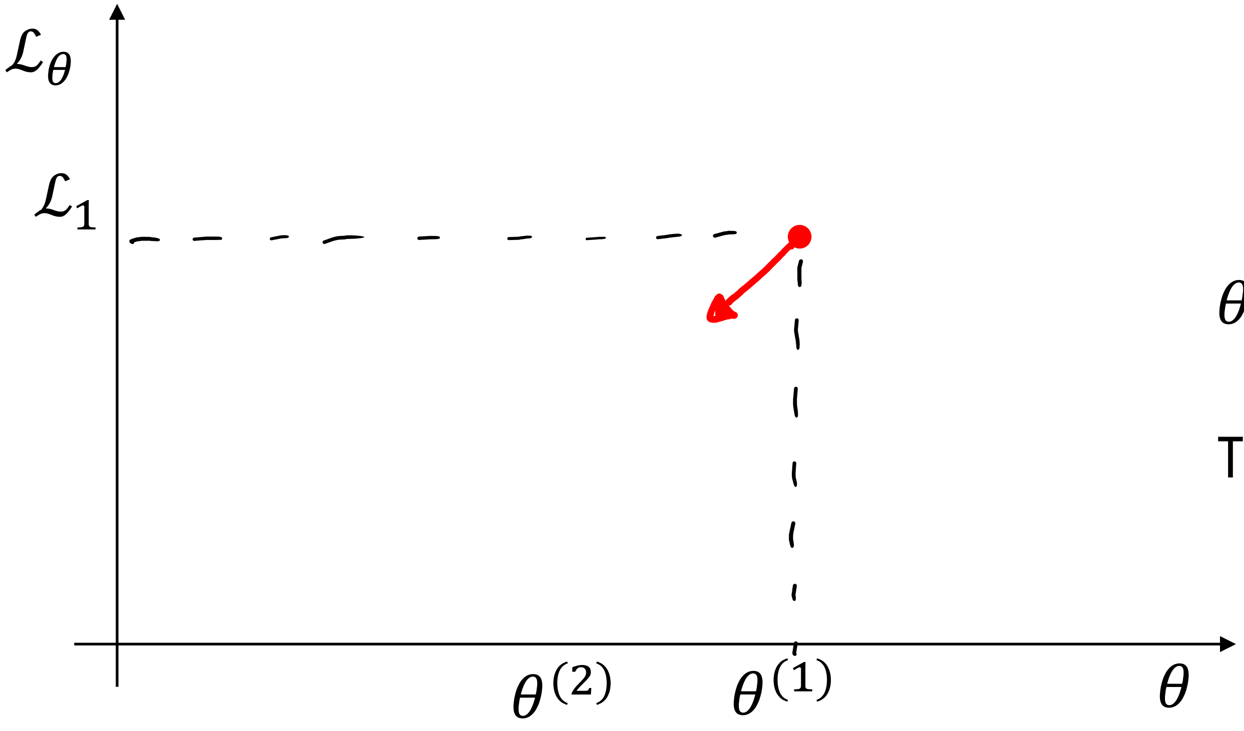
We also get the gradient for this value of the loss $\nabla \mathcal{L}(\theta^1, TR)$ which indicates in which direction the loss will decrease

The Network Training

It’s an optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure:** gradient descent.



Next parameter, θ^2 is chosen accordingly

$$\theta^2 = \theta^1 - \gamma \, \nabla\mathcal{L}(\theta^1, TR)$$

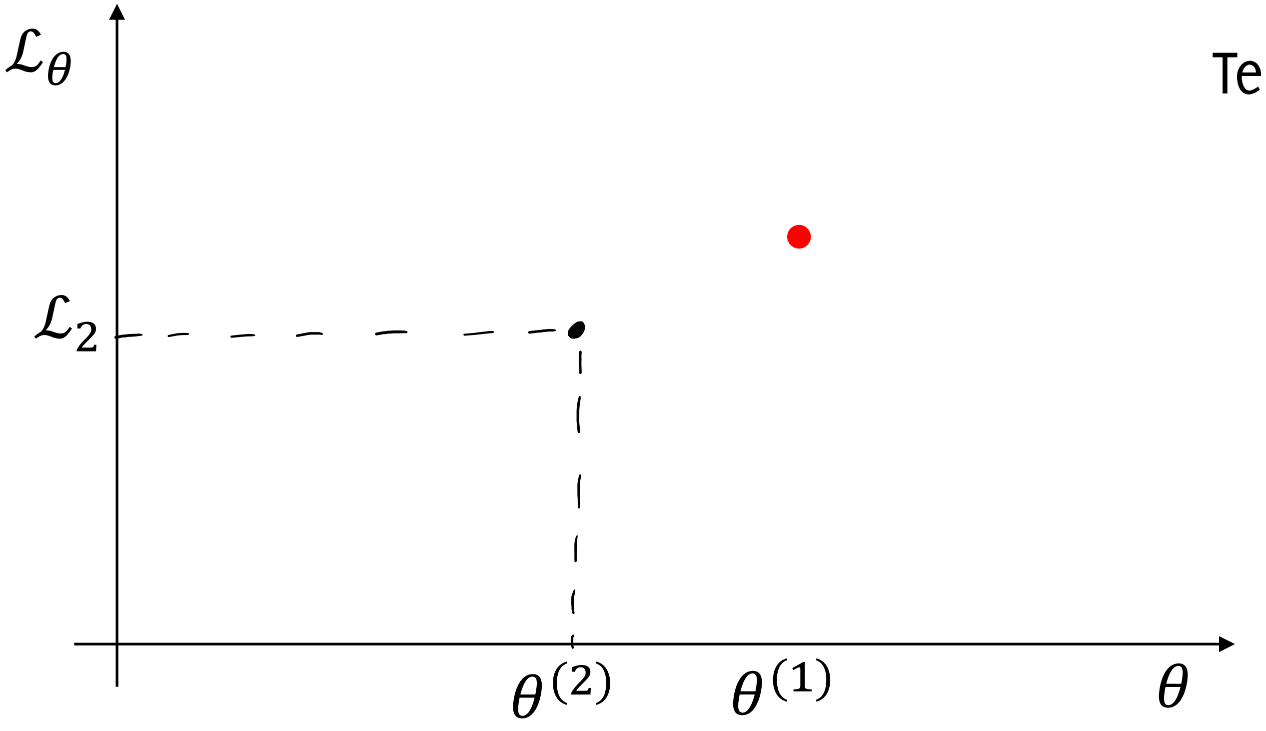
This is **gradient descent,** γ is the learning rate

The Network Training

It's an optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure**: gradient descent.



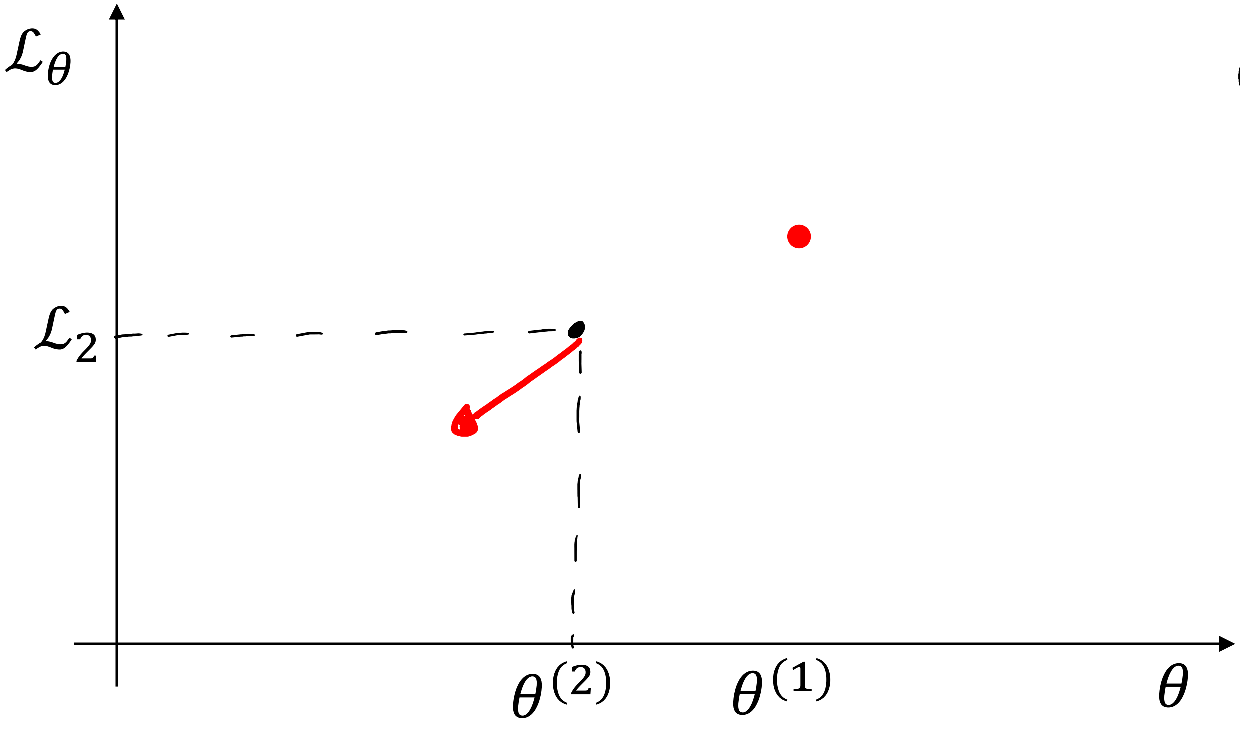
Test images and compute the loss $\mathcal{L}_2$
$\mathcal{L}(\theta^2, TR)$

The Network Training

It’s an optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure:** gradient descent.



Get the gradient at $\theta^{(2)}$

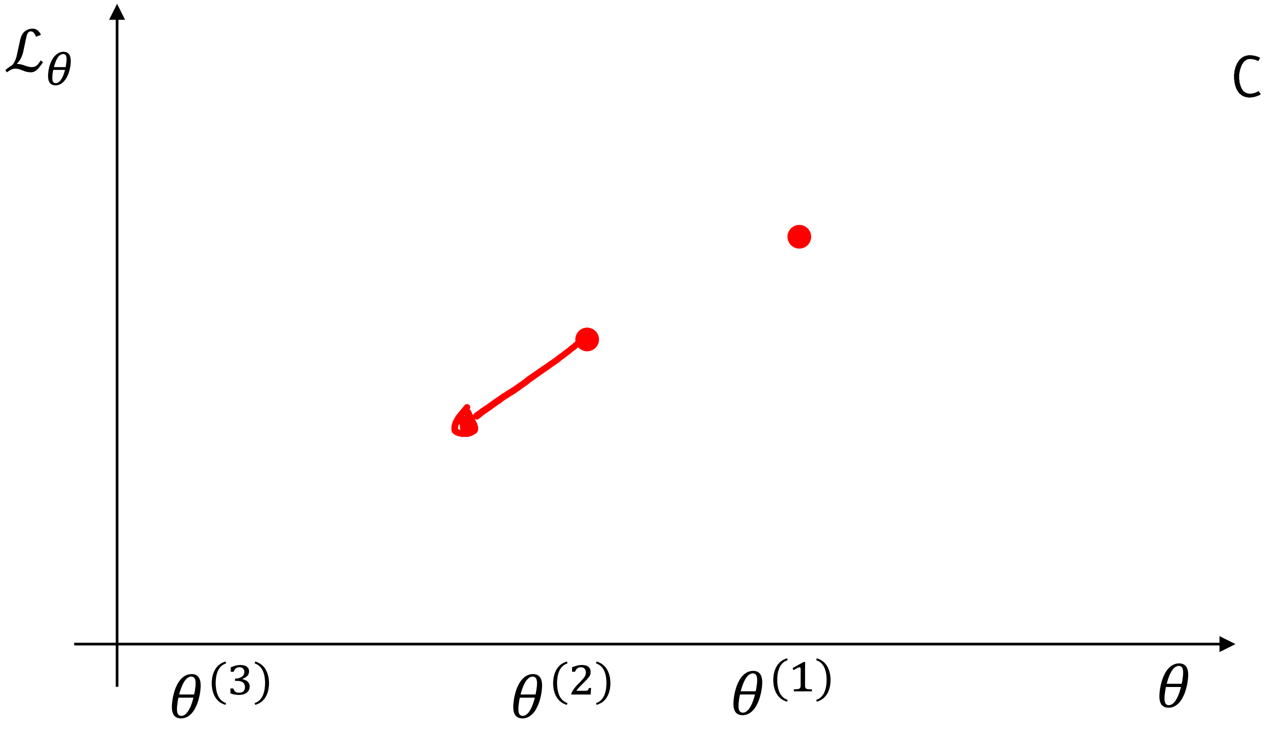
$\nabla \mathcal{L}(\theta^2, TR)$

The Network Training

It's an optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure**: gradient descent.



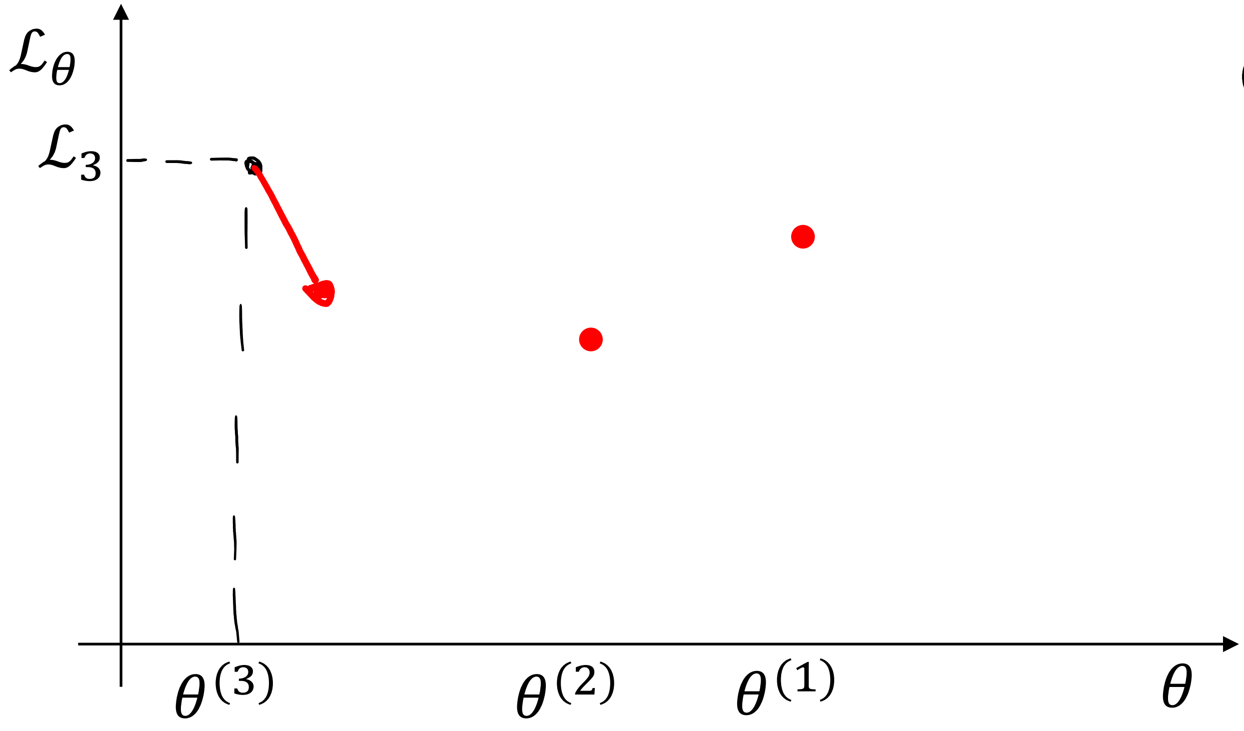
Choose $\theta^{(3)}$ accordingly

The Network Training

It's an optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure**: gradient descent.



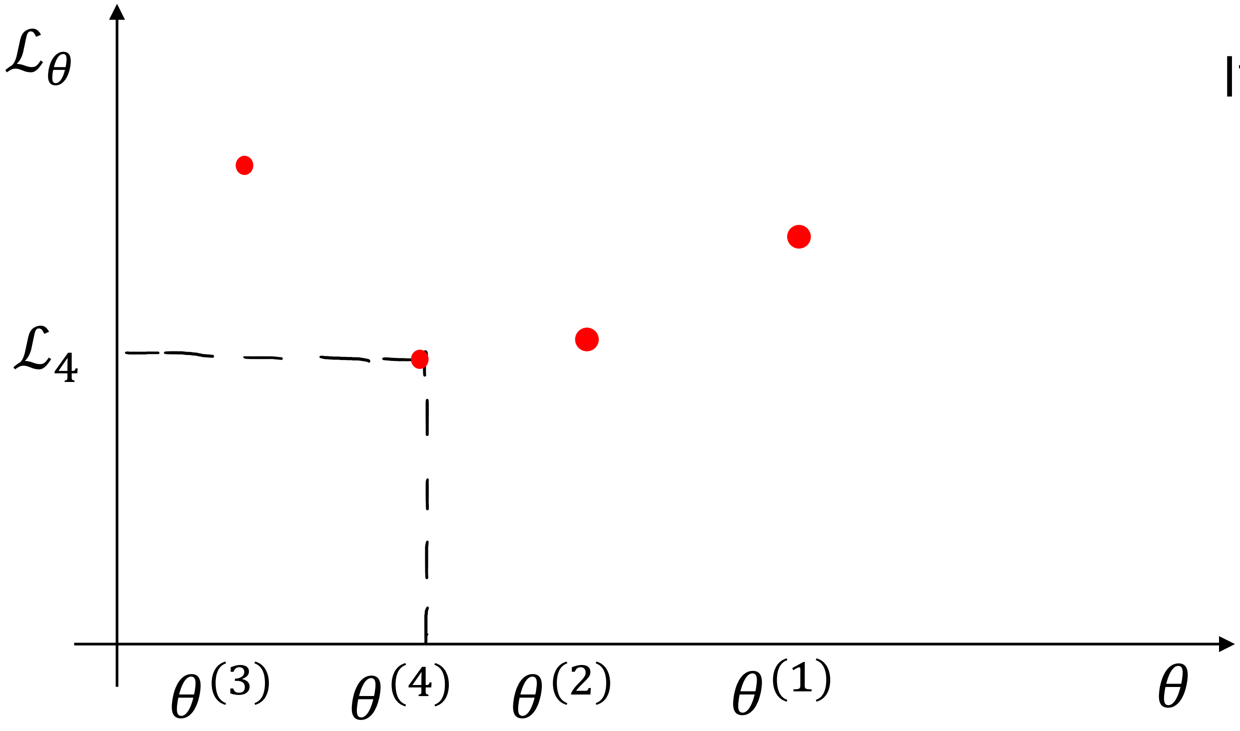
Get the gradient at $\theta^{(3)}$

The Network Training

It's an optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure**: gradient descent.



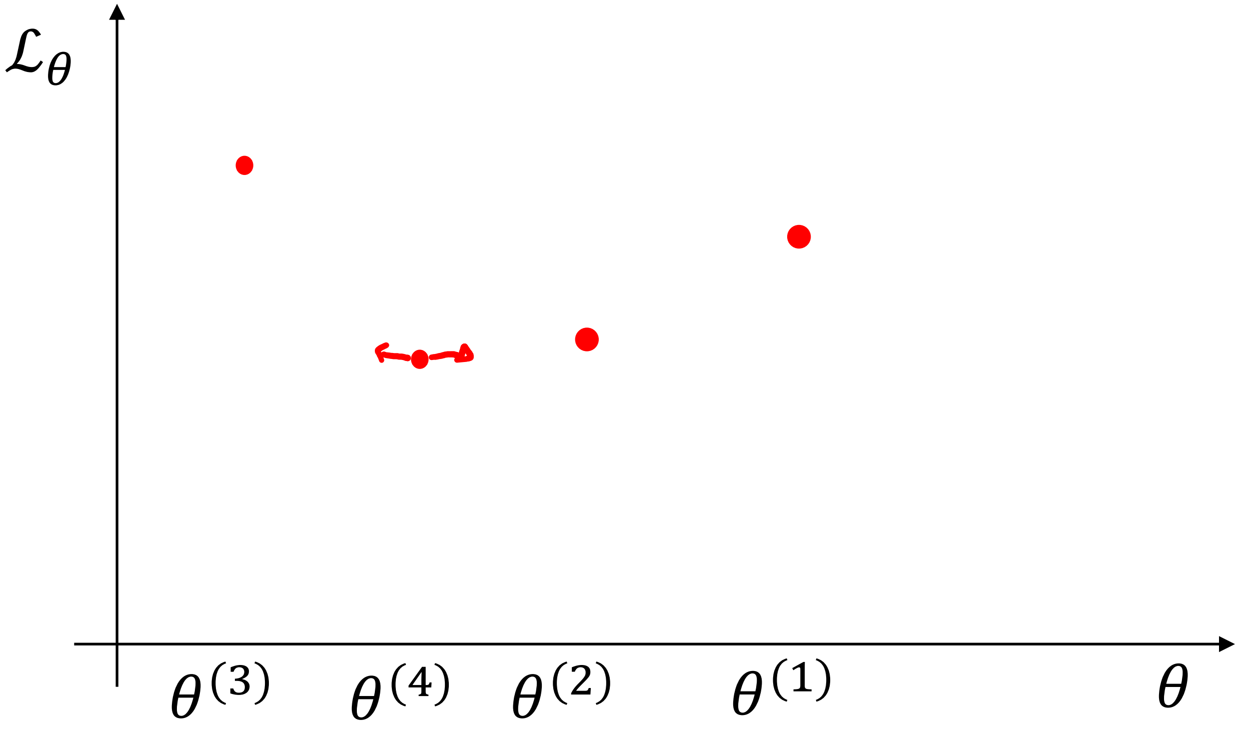
Iterate $\theta^{(4)}$ and possibly many times

The Network Training

It’s an optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure**: gradient descent.



Once you get to a point where gradient is zero, stop!

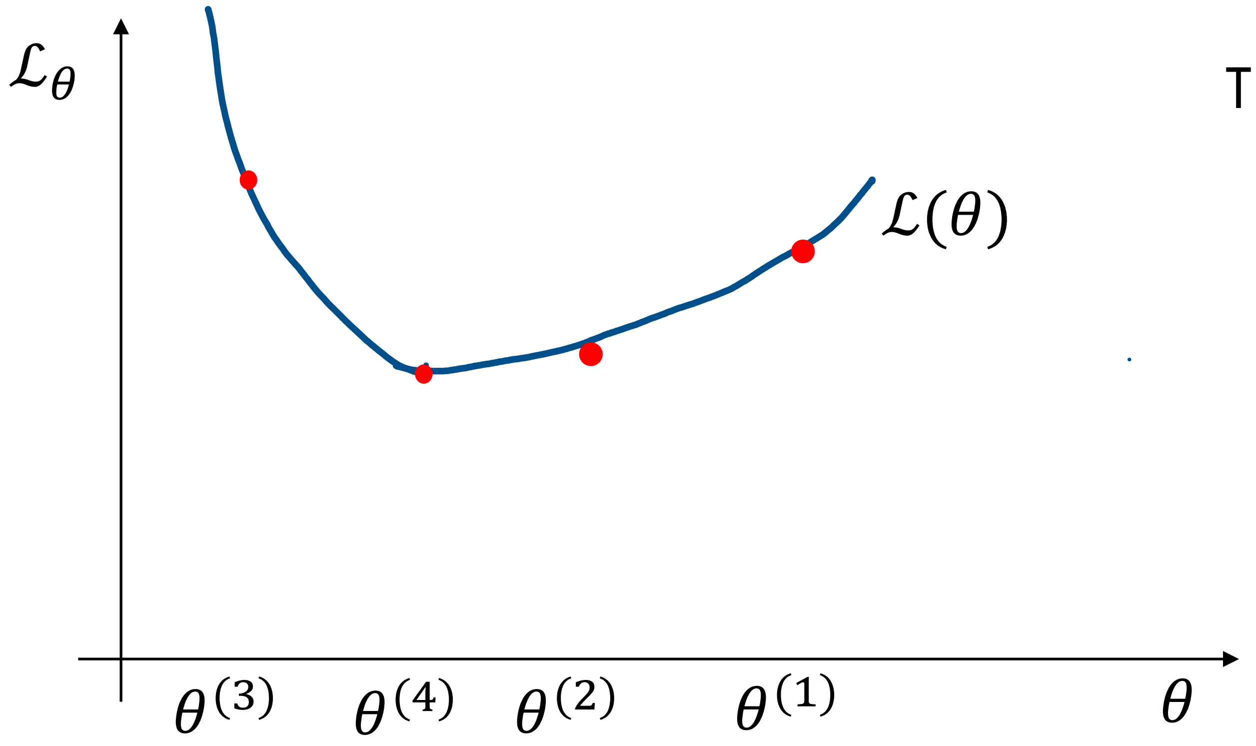
$$\|\nabla \mathcal{L}(\theta^n, TR)\| \approx 0$$

The Network Training

It's an optimization problem

$$\theta^* = \underset{\theta}{\text{argmin}}\, \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure:** gradient descent.



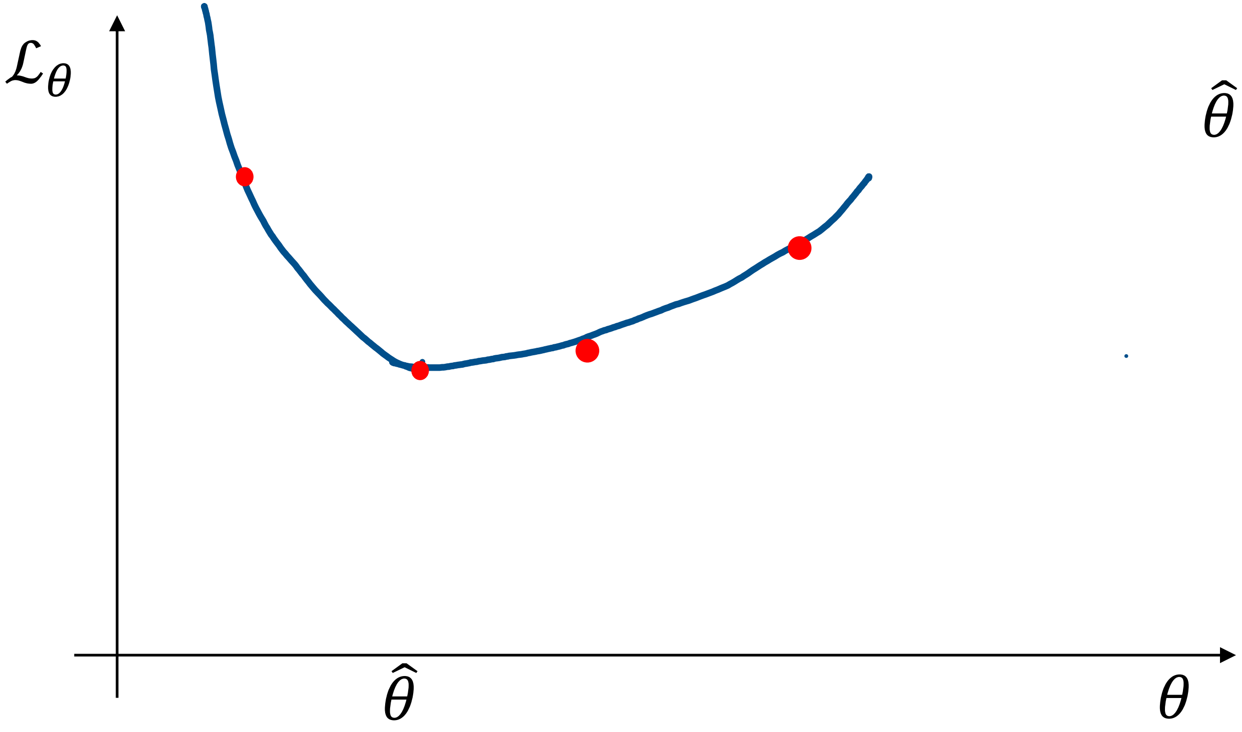
This is how we minimize the loss function

The Network Training

It's an optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta, TR)$$

This is solved by an **iterative procedure:** gradient descent.



$\hat{\theta}$ is the network parameter

Do I need to take care of this process?

Of couse not!

```
learning_rate = 0.5
optimizer = tfk.optimizers.SGD(learning_rate)
```

The optimization process adjusts the learning rate γ, which is how much to trust the gradient in each iteration.

Do I need to take care of this process?

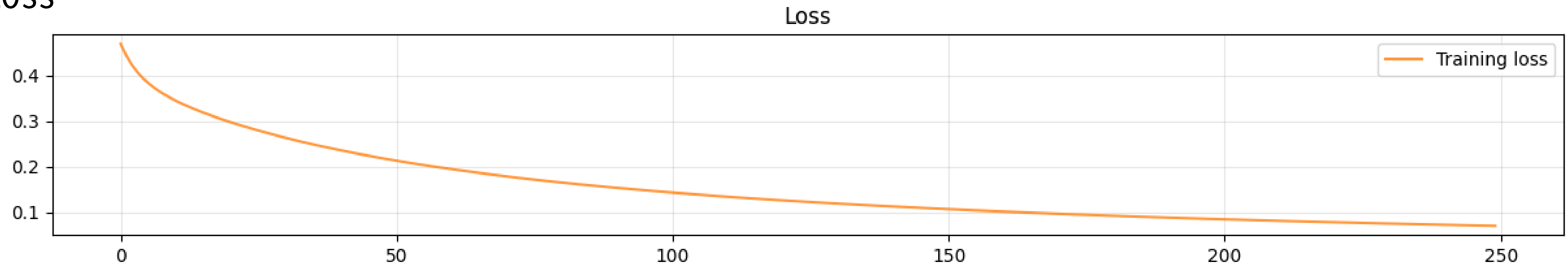
There are optimizers implemented that can adjust the step size to prevent the procedure to diverge, adopt momentum etc..

The most popular one is Adam optimizer

```
learning_rate = 1e-3
opt = tfk.optimizers.Adam(learning_rate)
```

Loss during training

Loss



epochs

(the number of times the entire training set is being scanned)

Training Losses

Training in Supervised Settings

The **MSE** (Mean Squared Error) is the most popular loss for **regression:**

$$\mathcal{L}(\theta, TR) = \frac{1}{N}\sum_{i=1}^{N}(f_\theta(x_i) - y_i)^2$$

The loss measures how far the predictions $f_\theta(x_i)$ are from the corresponding target y_i

In keras: **`tfk.losses.MeanSquaredError()`**

Training in Supervised Settings

The most famous classification losses are different

Binary Cross-entropy (when $y \in \{0,1\}$)

$$\mathcal{L}(\theta, TR) = \frac{1}{N}\sum_{i=1}^{N}\Big(y_i \, \log\big(f_\theta(x_i)\big) + (1 - y_i)\log\big(1 - f_\theta(x_i)\big)\Big)$$

To minimize the loss, you want to minimize each summand, thus

- $f_\theta(x_i) \approx 0$ when $y_i = 0$
- $f_\theta(x_i) \approx 1$ when $y_i = 1$

In keras: **`tfk.losses.BinaryCrossentropy()`**

Training in Supervised Settings

In case of multi-class classification we have the

Categorical Cross-entropy, when $\#\Lambda > 2$:

$$\mathcal{L}(\theta, TR) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j}^{\#\Lambda}[\boldsymbol{y}_i]_j \log\big([f_\theta(x_i)]_j\big)$$

Where $[\boldsymbol{y}_i]_j$ is the j^{th} component of the vector $\boldsymbol{y}_i$

This means you want the network to return a vector $f_\theta(x_i)$ having

- $[f_\theta(x_i)]_j \approx 0$ when $[y_i]_j = 0$, i.e., low probability to the wrong class
- $[f_\theta(x_i)]_j \approx 1$ when $[y_i]_j = 1$, i.e., high probability to the correct class

In keras: **`tfk.losses.CategoricalCrossentropy()`**

Performance Assessment

Training

However we don’t care very much on mistakes on the Training Set, we want that our network can correctly predict labels on unseen data.
We assess our model on the Test Set.

In the metaphor of learning, it is the same difference as «parroting» the lesson, or really understanding what one has studied.

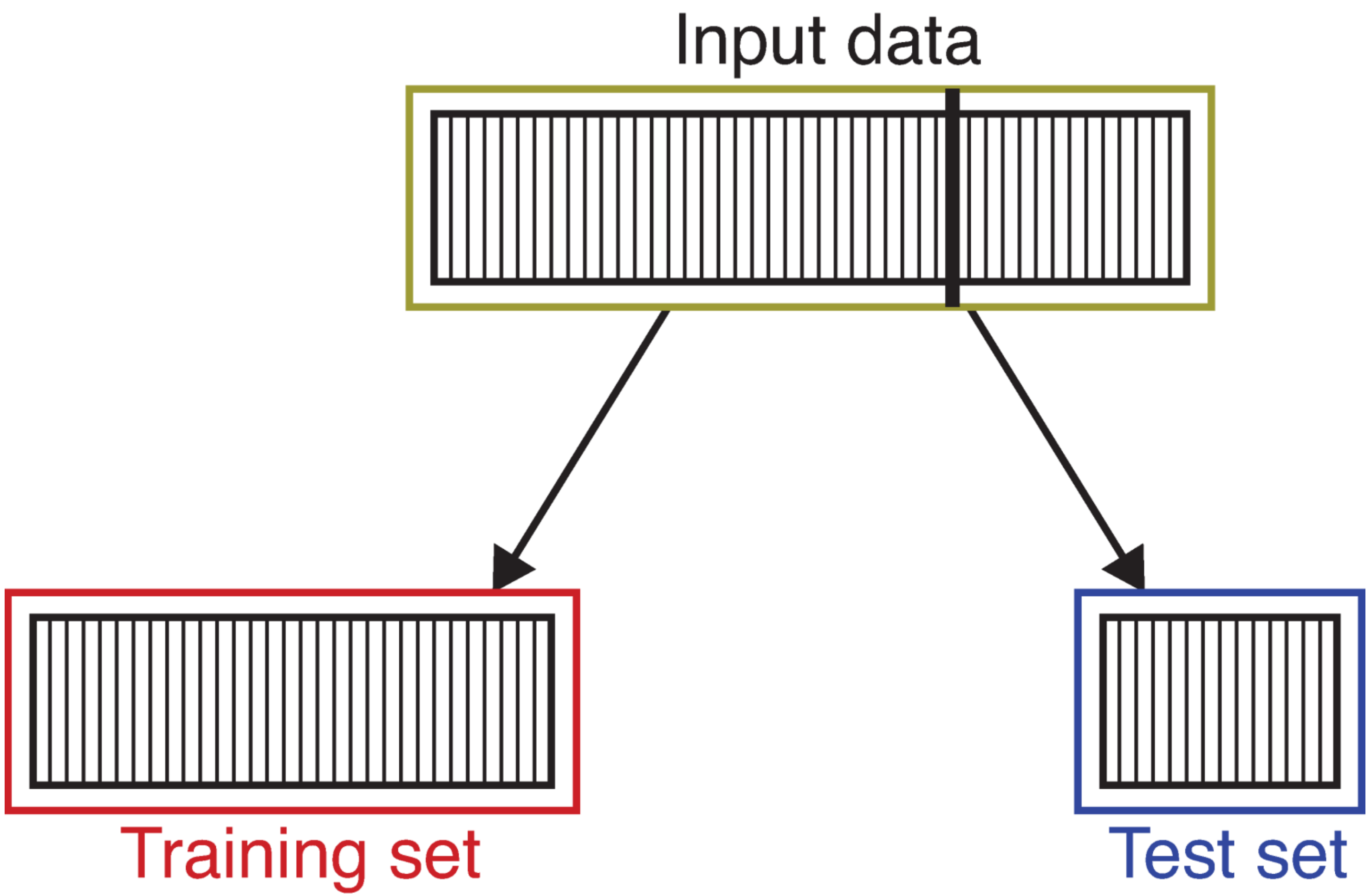
Training, testing

Training set: the data used to learn the model parameters

Test set: used only at the end to perform final model assessment

The test should be used only when all the parameters are fixed, to assess how good the model can generalize

$$TR = \{(x, y)_i, i = 1, \dots, N\}$$

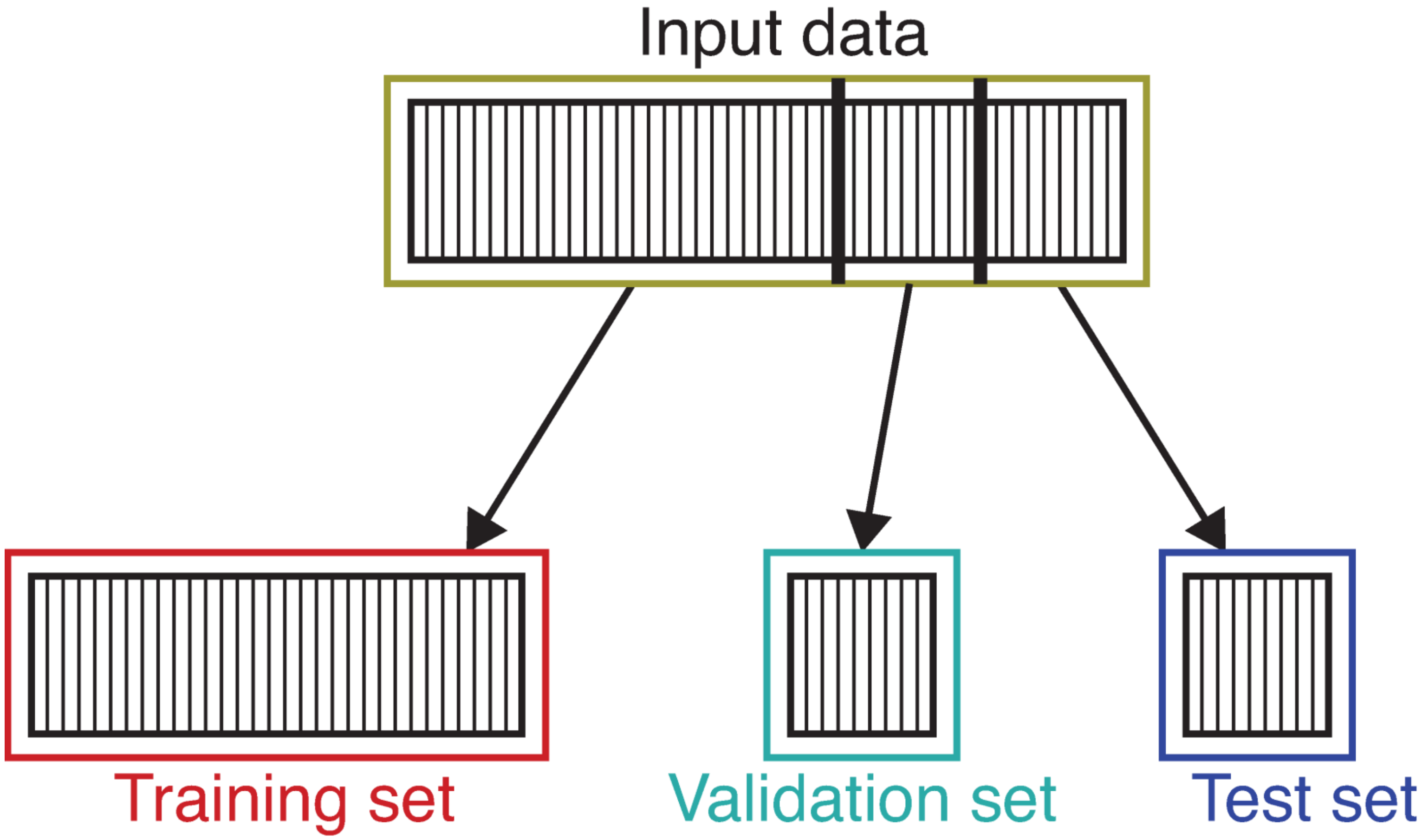


Training, testing and validation

Training set: the data used to learn the model parameters

Test set: used only at the end to perform final model assessment

Validation set: the data used to perform "model selection"



Training, testing and validation

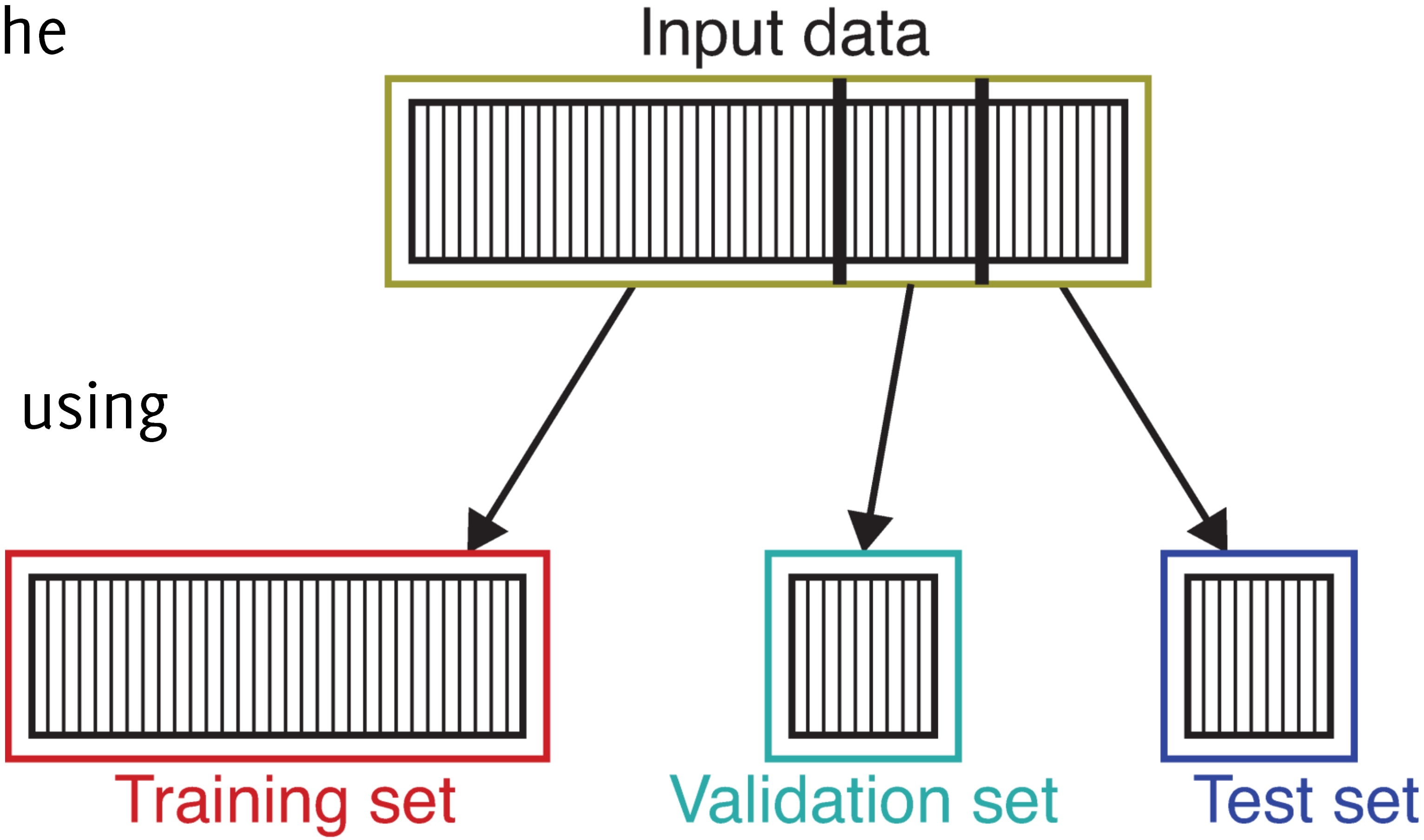
We want that all the splits have the same distribution of the input data.

Cross-Validation:

Parameters are optimized using

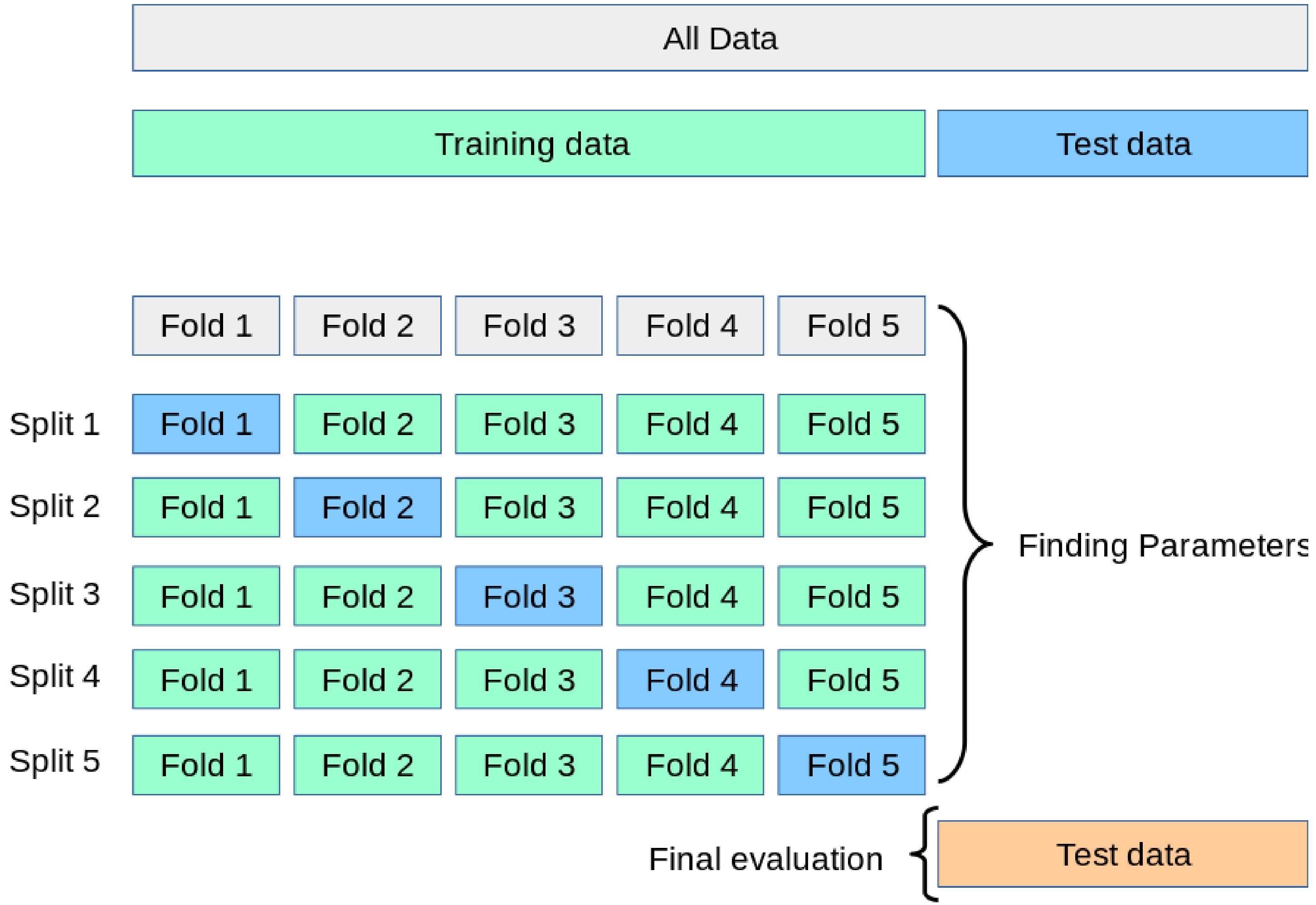
- Training set
- Validation set

Network performance is assessed on the independent test set



K-fold Cross-Validation

This is meant to use the entire dataset for performance assessment



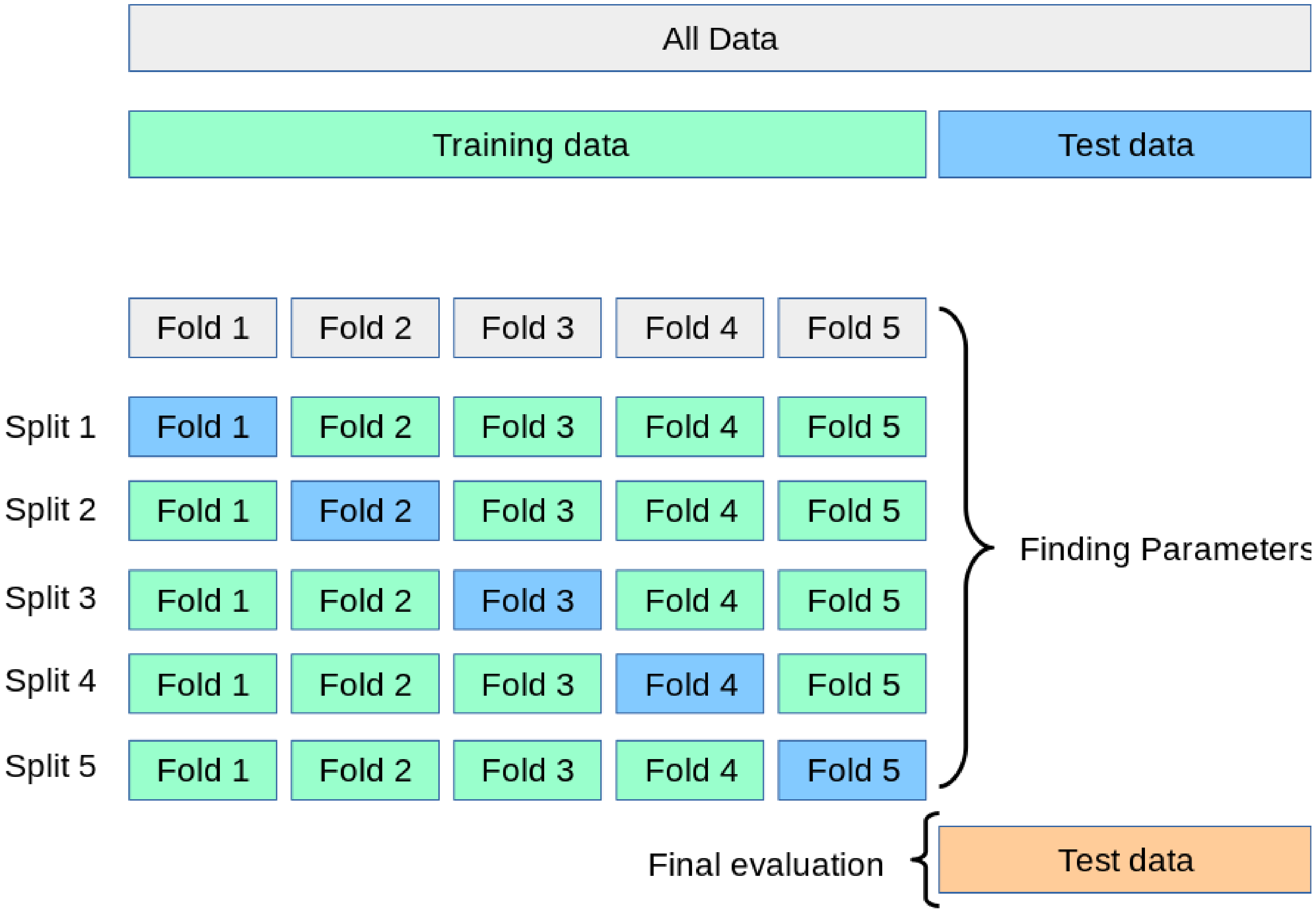

```
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=seed)
```

K-fold Cross-Validation

This is meant to use the entire dataset for performance assessment

K-fold cross validation can be extended in two directions:

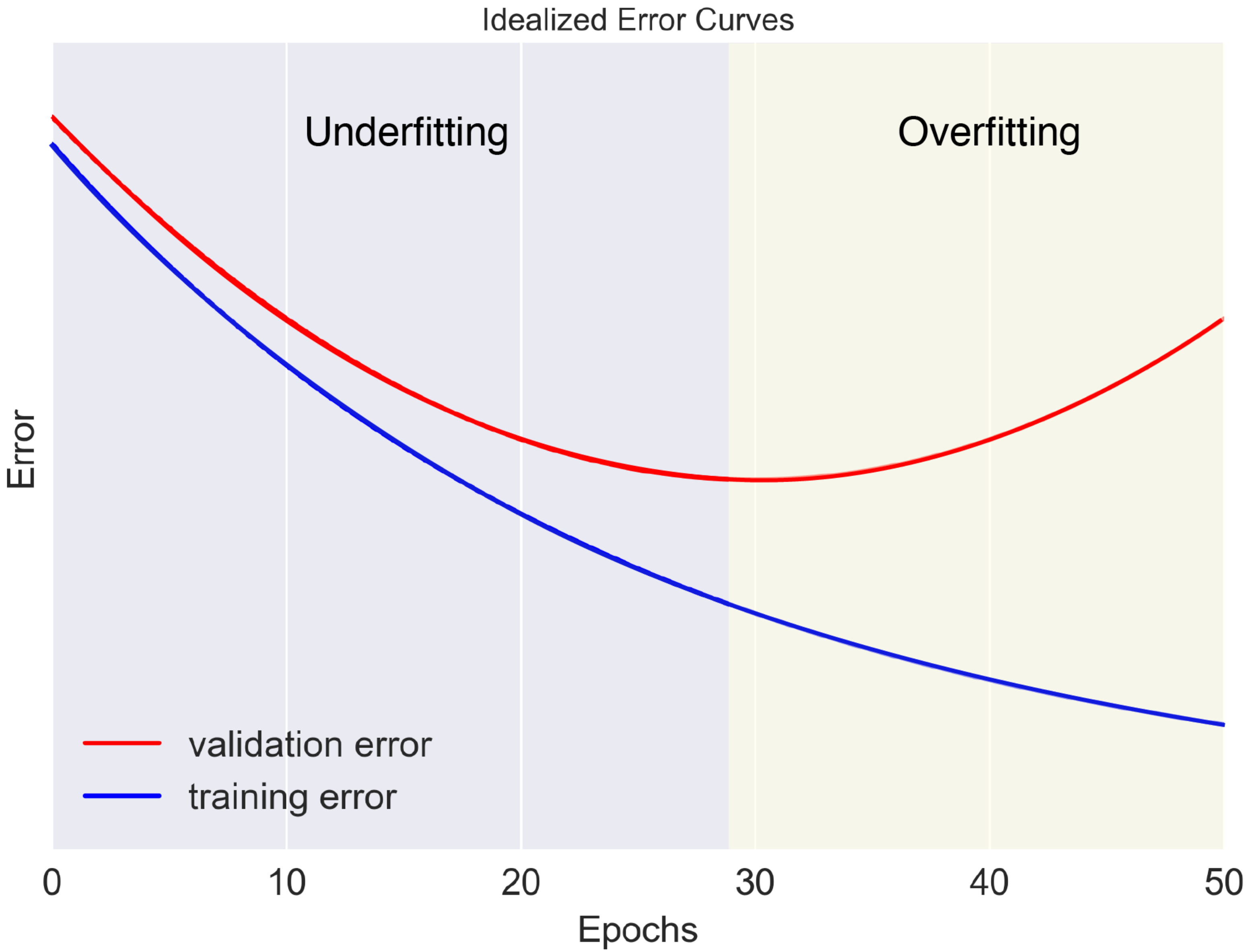
- Leave-one-out cross validation, where you use as test set a single sample (and train $N-1$ models)
- Split is ruled by specific criteria rather than random to assess the generalization performance: e.g., stratified cross-validation, leave-one-patient-out




```
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=seed)
```

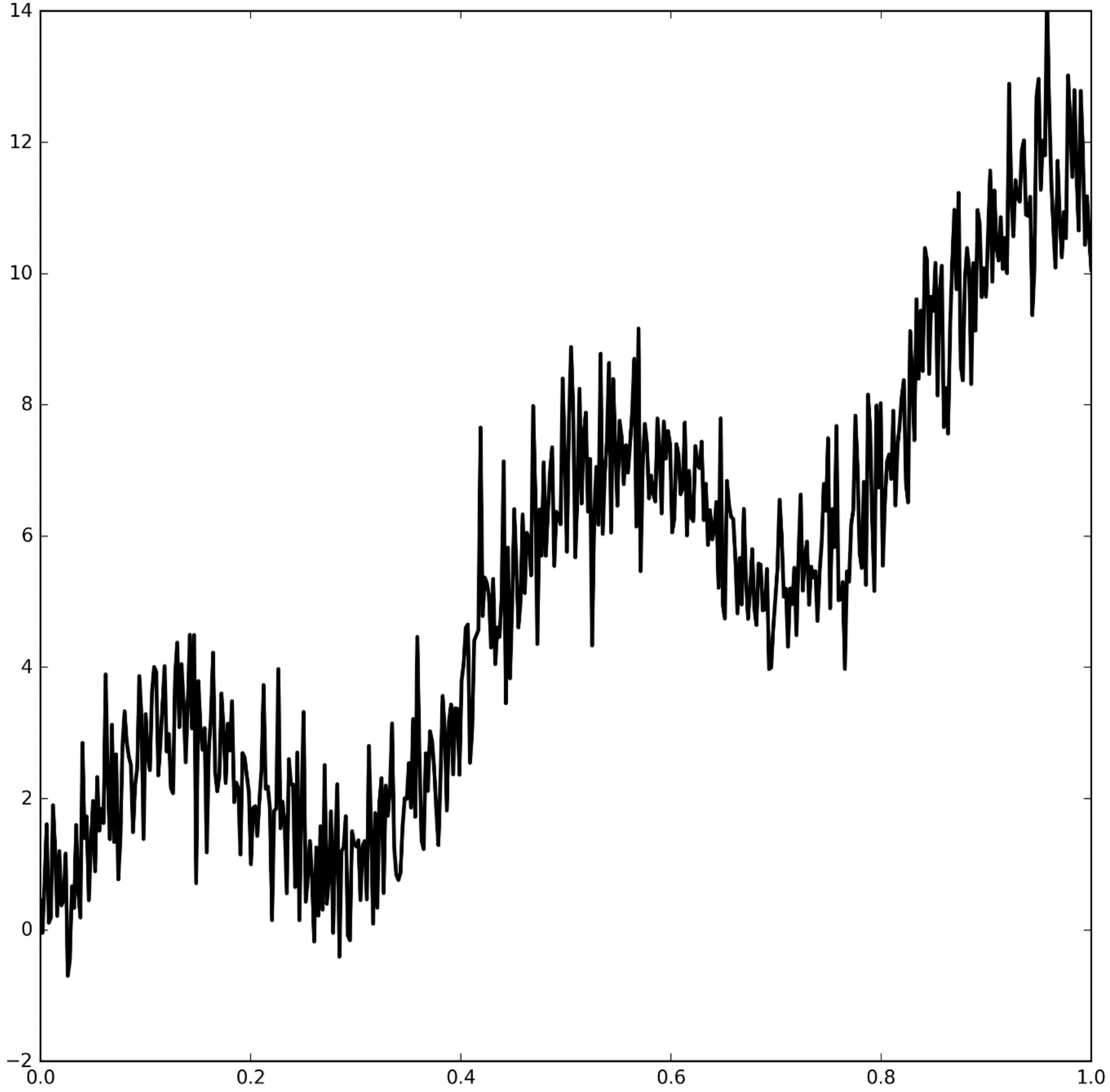
Overfitting and Countermeasures

Underfitting and overfitting

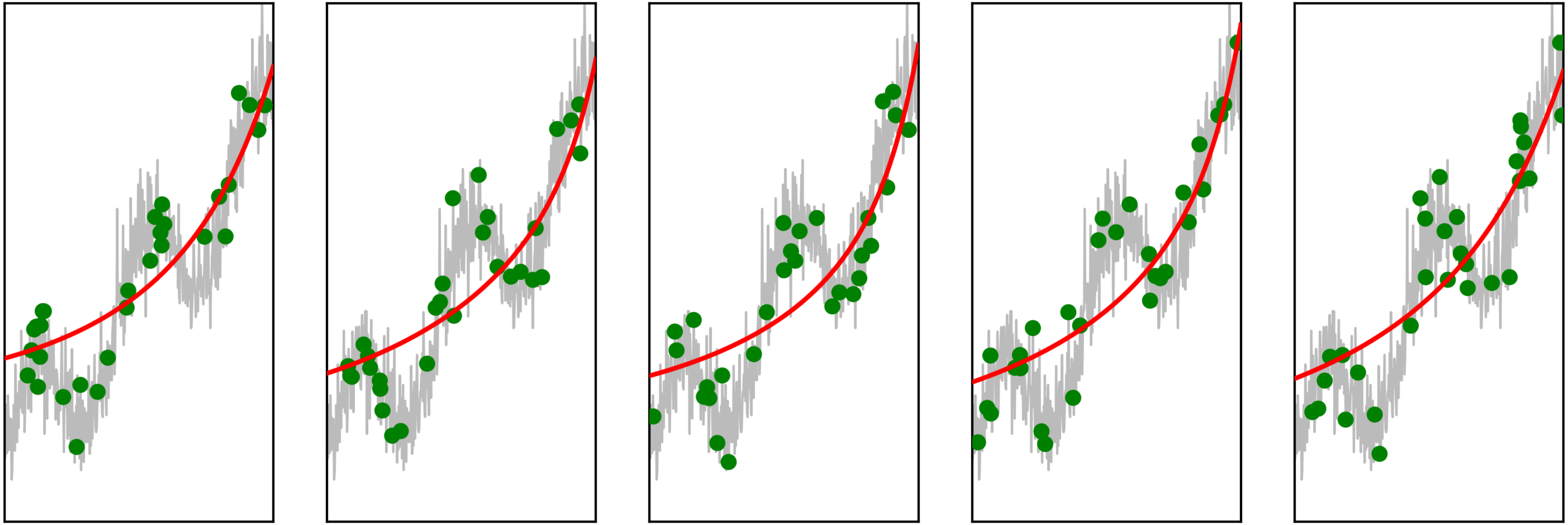


Overfitting networks show a monotone training error trend (on average with SGD) as the number of gradient descent iterations, but they lose generalization at some point …

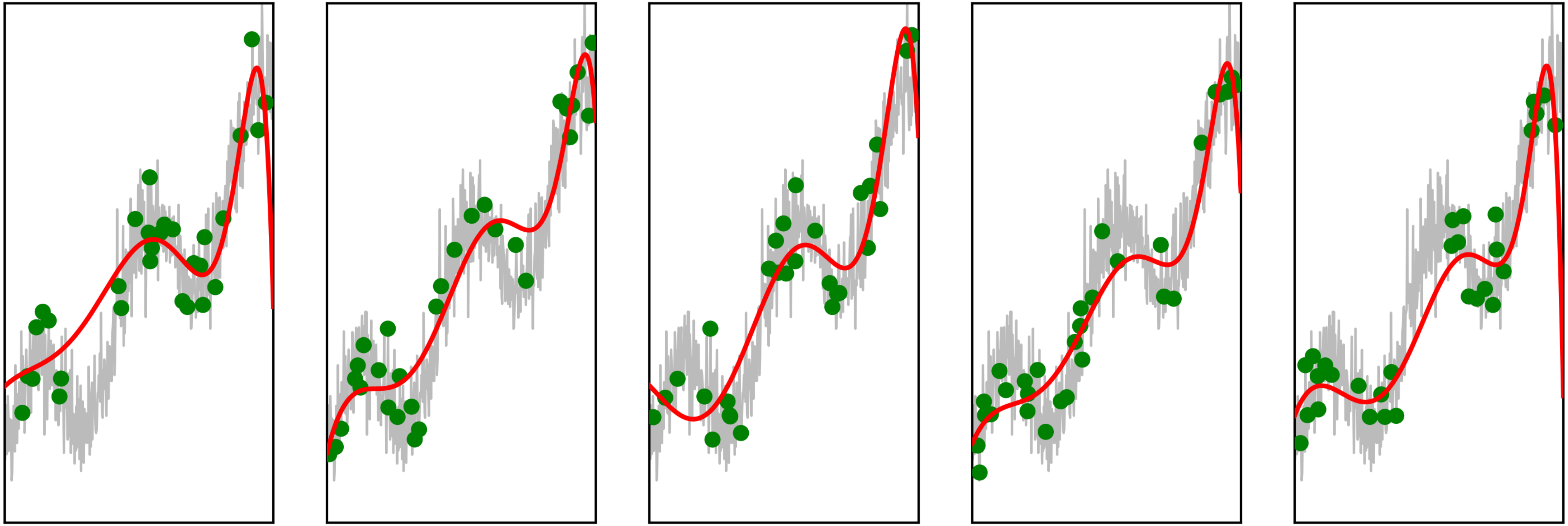
What happens with the data?



Under-fitting



Over-fitting



Solution to Prevent Overfitting

The most common strategies to prevent overfitting when training NN:

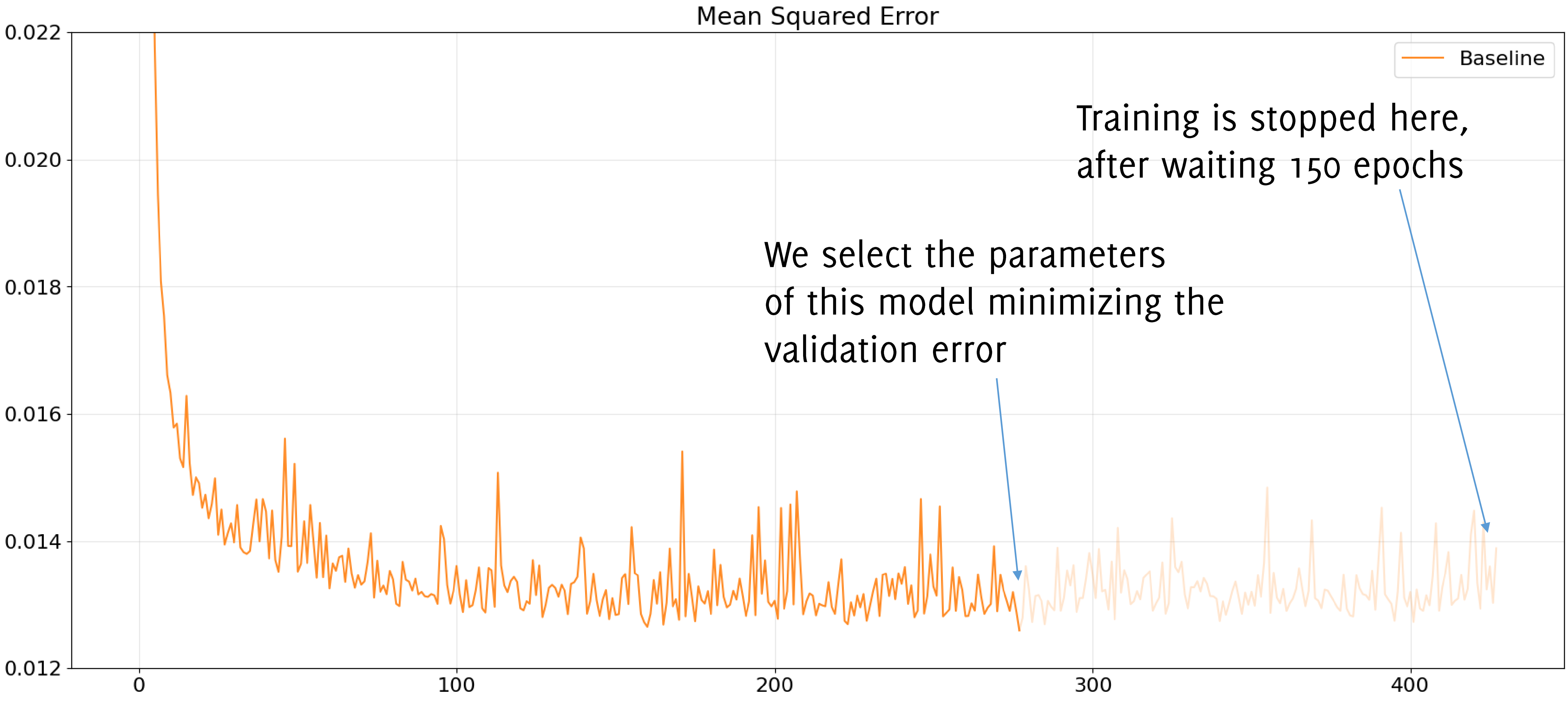
- early stopping
- add a regularization term to the loss
- drop-out

Early Stopping

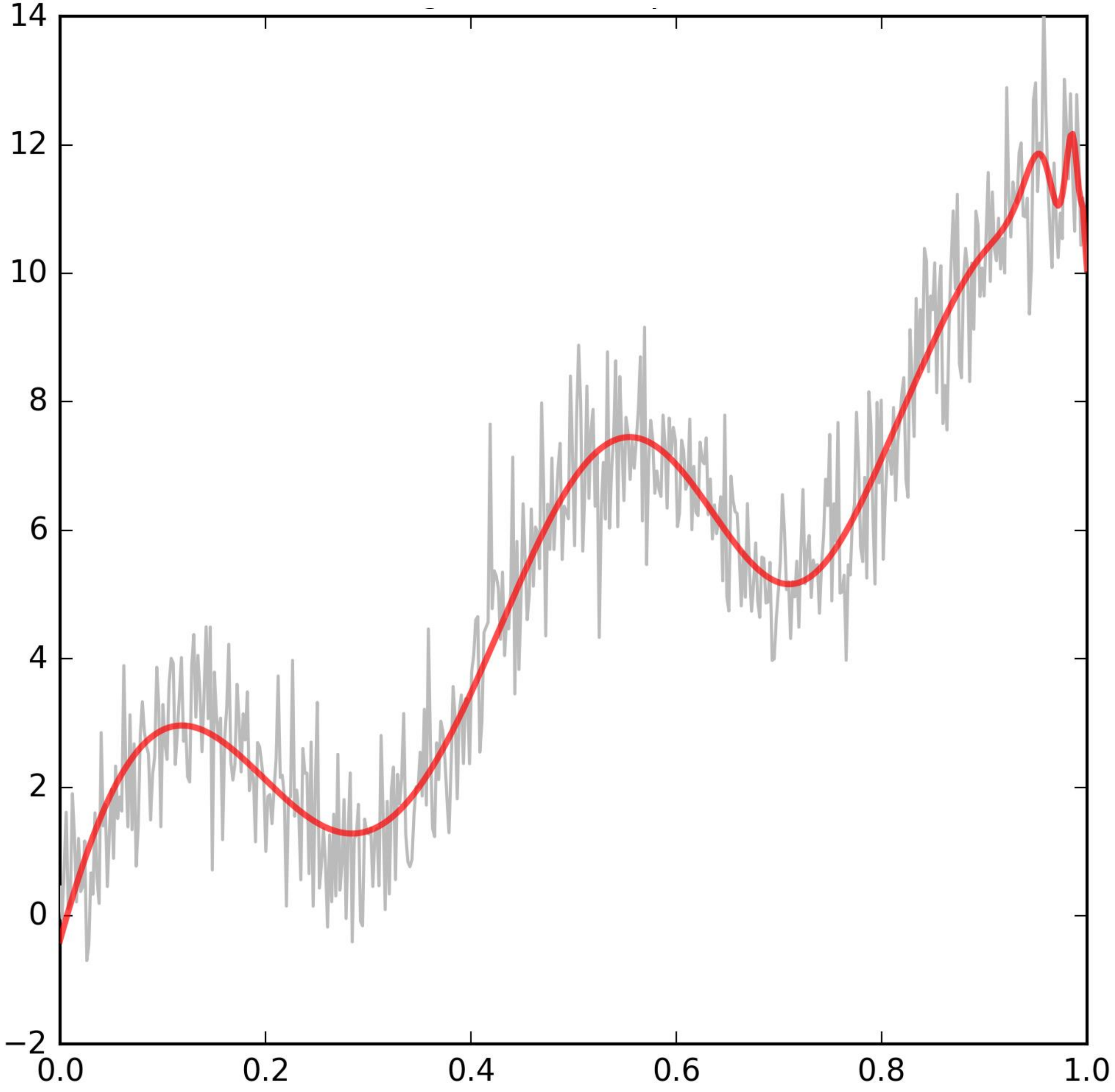
Stop the training process when the validation error stops decreasing


```
patience = 150 #number of epochs to wait

early_stopping = tfk.callbacks.EarlyStopping(monitor='val_mse',
mode='min', patience=patience,restore_best_weights=True)

callbacks = [_stopping ]
```

Early Stopping



Regularization (on the loss side)



Regularization loss

Loss seen so far includes only ***data-fidelity term***, thus tend to return models that can explain data at best.

$$\mathcal{L}(\theta, TR) = \frac{1}{N}\sum_{i=1}^{N}(f_\theta(x_i) - y_i)^2$$

This promotes ***overly complex*** models.

Add a term to the loss to penalize model complexity

$$\mathcal{L}(\theta, TR) = \frac{1}{N}\sum_{i=1}^{N}(f_\theta(x_i) - y_i)^2 + \lambda\,\mathcal{R}(\theta)$$

Popular Regularizer

Ridge regression

$$\mathcal{L}(\theta, TR) = \frac{1}{N}\sum_{i=1}^{N}(f_\theta(x_i) - y_i)^2 + \lambda||\theta||_2^2$$

In gradient descent, $\theta^{(i+1)} = \theta^{(i)} - \gamma\nabla\mathcal{L}\left(\theta^{(i)}, TR\right)$ this adds a term $-2\lambda\theta$, which implies that weights tend towards zero. Therefore, this procedure is also called **weight decay**.

In keras, you need to add this parameter to each layer

```
output_layer = tfkl.Dense(units=1,name='Output',
kernel_regularizer=tf.keras.regularizers.l2(l2_lambda))(
hidden_activation)
```

Popular Regularizer

Lasso

$$\mathcal{L}(\theta, TR) = \frac{1}{N}\sum_{i=1}^{N}(f_\theta(x_i) - y_i)^2 + \lambda||\theta||_1$$

This tend to have **sparse solutions**, where many parameters (or network weights) are zero, and few are not.

In keras, you need to add this parameter to each layer

```
output_layer = tfkl.Dense(units=1,name='Output',
kernel_regularizer=tf.keras.regularizers.l1(l1_lambda))(
hidden_activation)
```

Popular Regularizer

Elastic Net

$$\mathcal{L}(\theta, TR) = \frac{1}{N}\sum_{i=1}^{N}(f_\theta(x_i) - y_i)^2 + \lambda\|\theta\|_1 + \mu\|\theta\|_2^2$$

This tend to have yet **sparse solutions** but with a smoother loss function ($\|\cdot\|_1$ is not differentiable in zero).

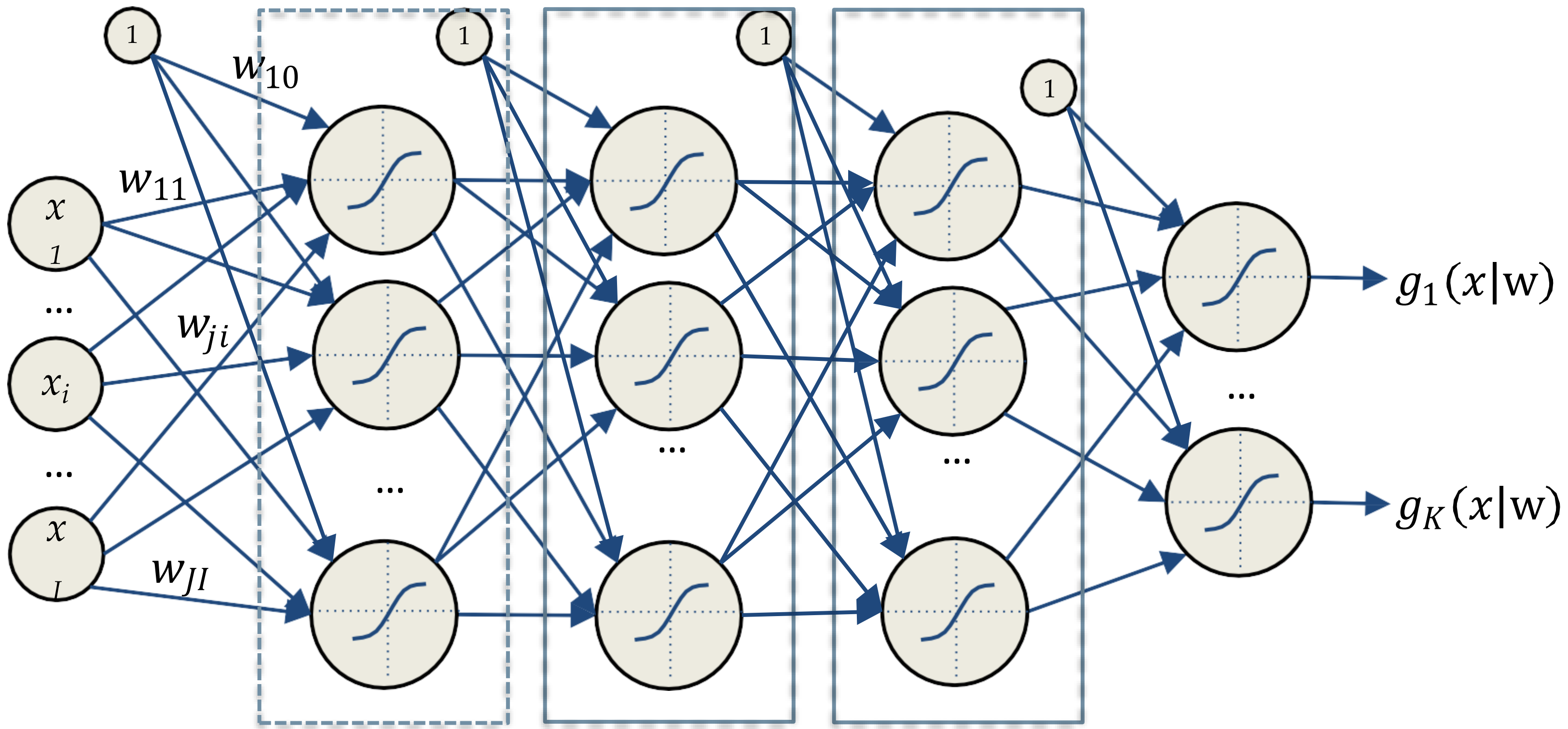
In keras, you need to add this parameter to each layer

```
output_layer = tfkl.Dense(units=1,name='Output',
kernel_regularizer=tf.keras.regularizers.L1L2(l1_lambda,
l2_lambda))(hidden_activation)
```

Dropout: Stochastic Regularization

By turning off randomly some neurons we force to learn an independent feature preventing hidden units to rely on other units (co-adaptation):

- Each hidden unit is set to zero with p_j probability, e.g., $p_j = 0.3$

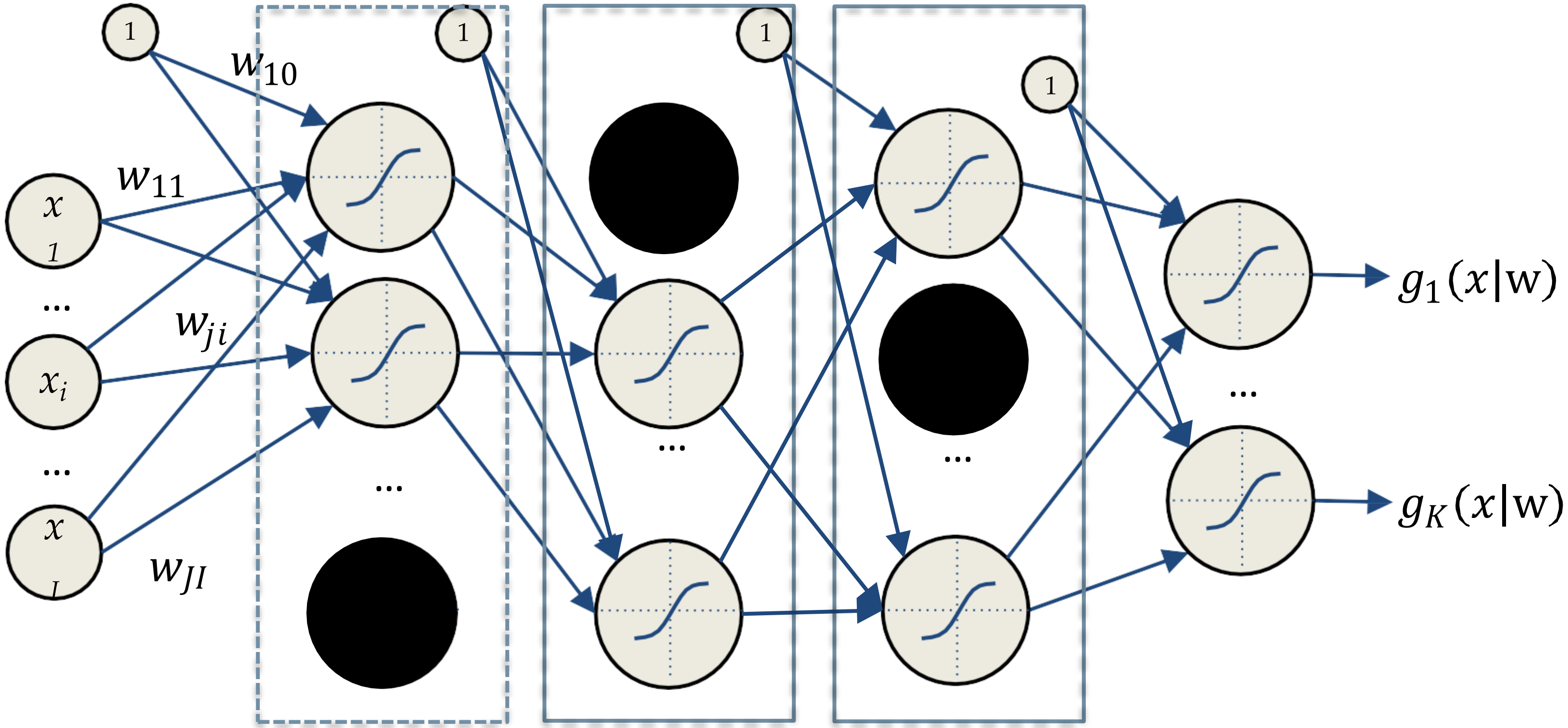


Slide credits Prof. Matteucci

Dropout: Stochastic Regularization

By turning off randomly some neurons we force to learn an independent feature preventing hidden units to rely on other units (co-adaptation):

- Each hidden unit is set to zero with p_j probability, e.g., $p_j = 0.3$

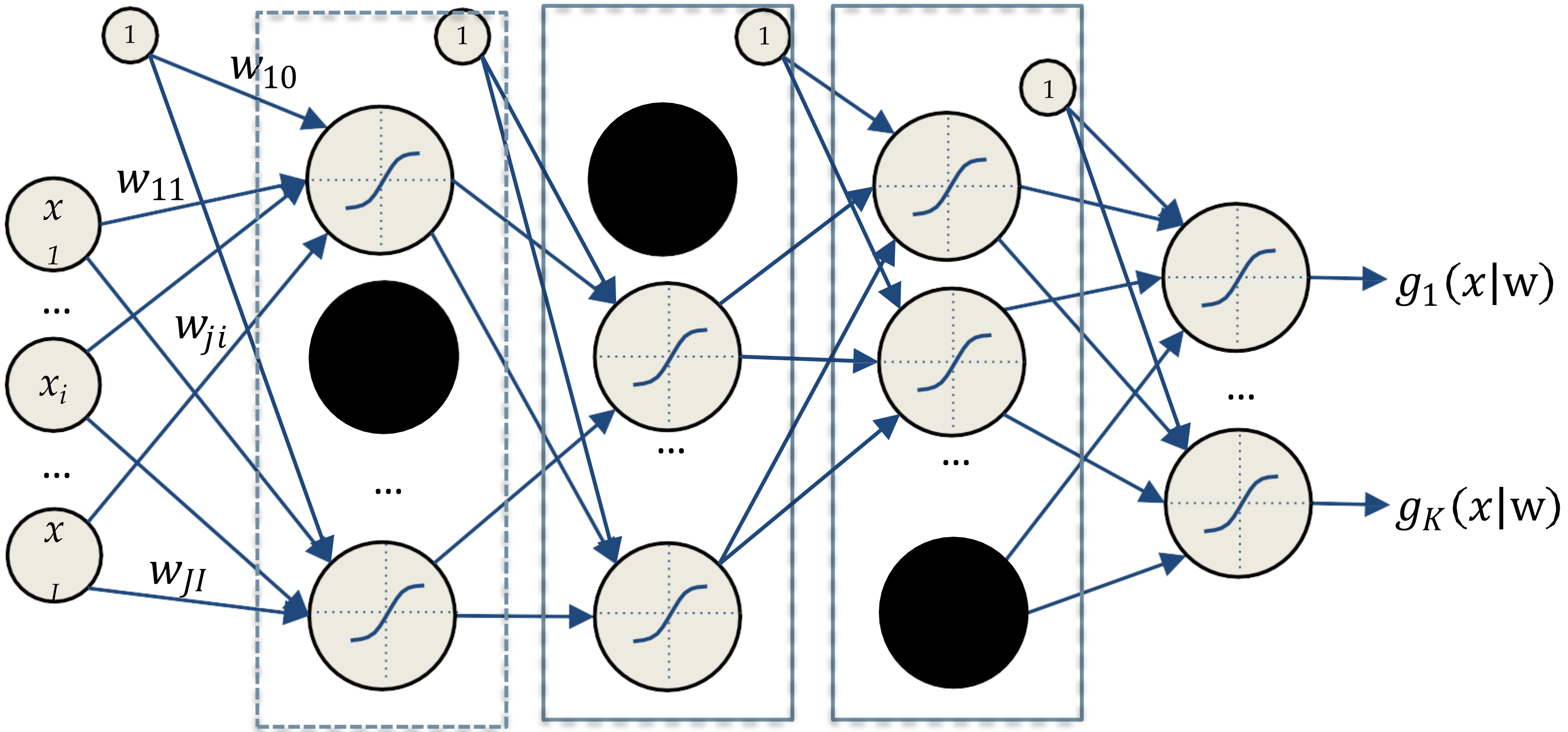


Slide credits Prof. Matteucci

Dropout: Stochastic Regularization

By turning off randomly some neurons we force to learn an independent feature preventing hidden units to rely on other units (co-adaptation):

- Each hidden unit is set to zero with p_j probability, e.g., $p_j = 0.3$

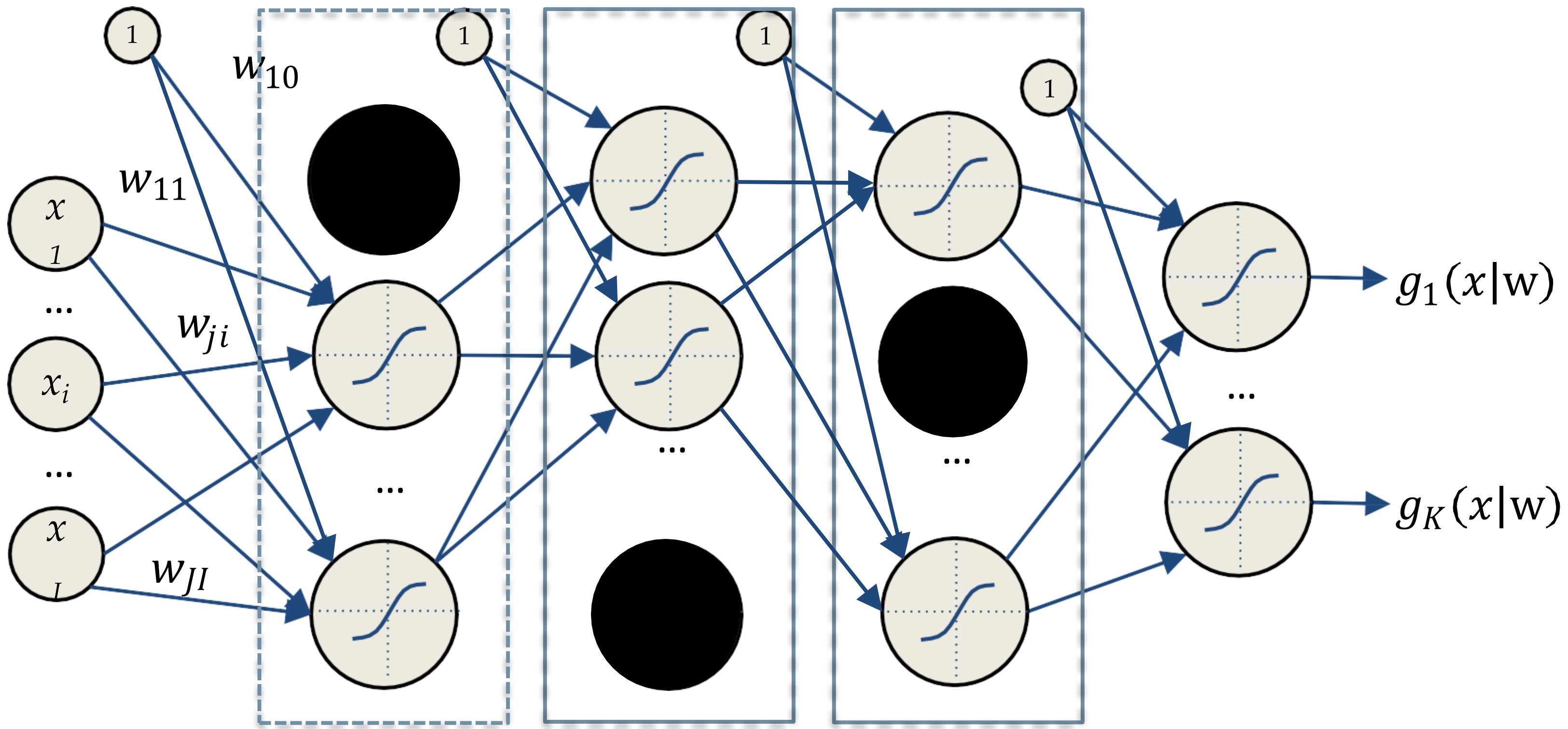


Slide credits Prof. Matteucci

Dropout: Stochastic Regularization

By turning off randomly some neurons we force to learn an independent feature preventing hidden units to rely on other units (co-adaptation):

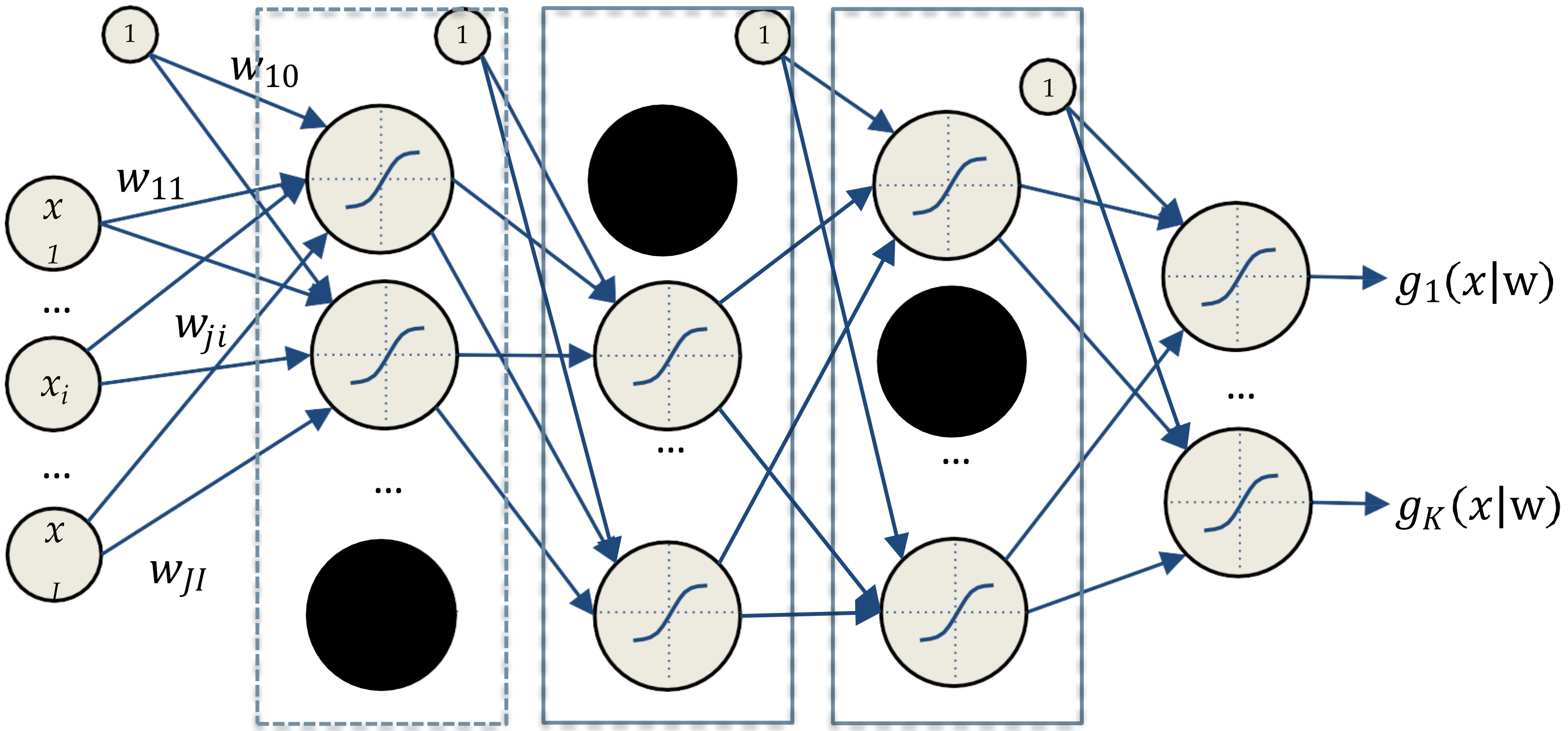
- Each hidden unit is set to zero with p_j probability, e.g., $p_j = 0.3$



Slide credits Prof. Matteucci

Dropout: Stochastic Regularization

Dropout trains weaker classifiers, on different mini- batches and then at test time we implicitly average the responses of all ensemble members.

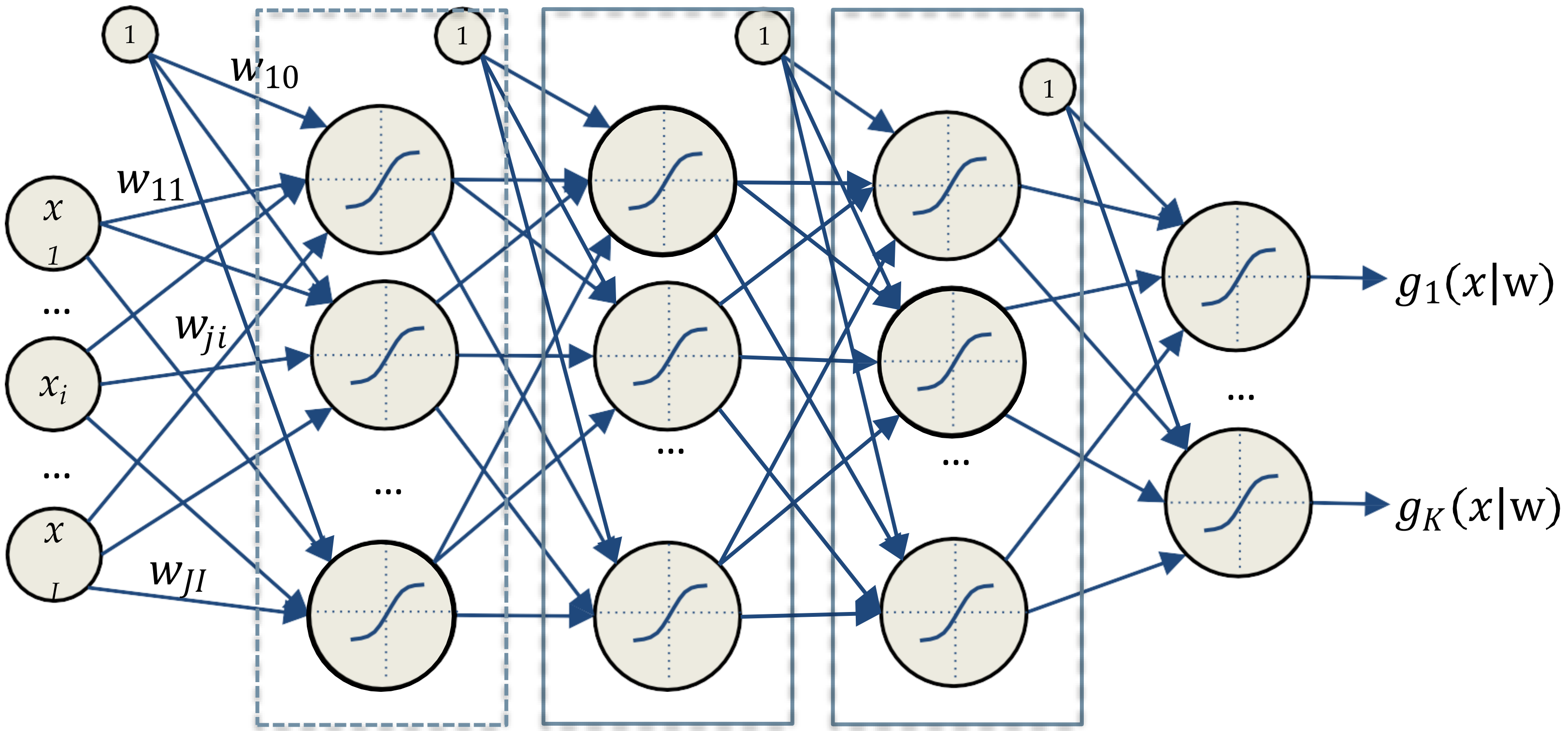


Slide credits Prof. Matteucci

Dropout: Stochastic Regularization

Dropout trains *weaker classifiers*, on different mini- batches and then at test time we implicitly average the responses of all ensemble members.

At testing time we remove masks and average output (by weight scaling)



Behaves as an ensemble method

Slide credits Prof. Matteucci

Dropout: Stochastic Regularization

Dropout complements the other regularization methods.

In keras, you just add a layer to the network

```
dropout = tfkl.Dropout(dropout_rate,
seed=seed)(hidden_activation)
```

Data Preprocessing

Preprocessing

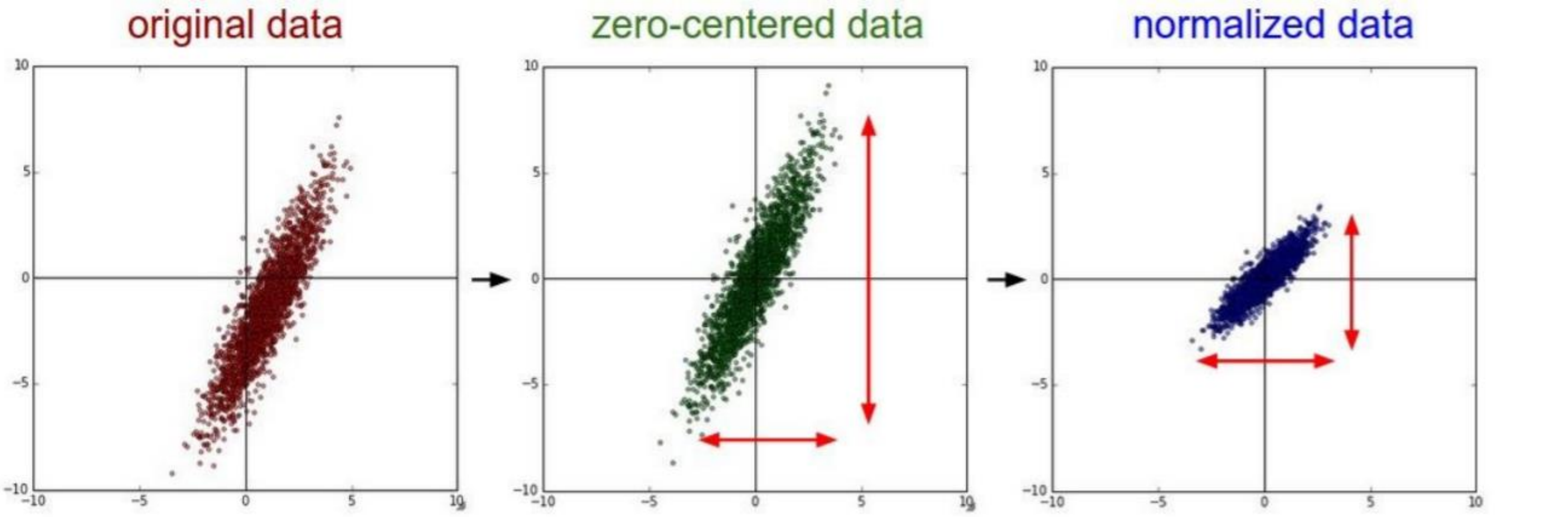
In general, normalization can improve convergence of gradient-based optimizers.

Normalization is meant to **bring training data “around the origin”** and possibly further rescale the data

In practice, **optimization on pre-processed data is made easier** and results are less sensitive to perturbations in the parameters

There are several options

There are different form of preprocessing




```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

This option brings the data to zero mean and unitary variance along each component

There are different form of preprocessing

This option brings the data to the range $[-1,1]$ in each component

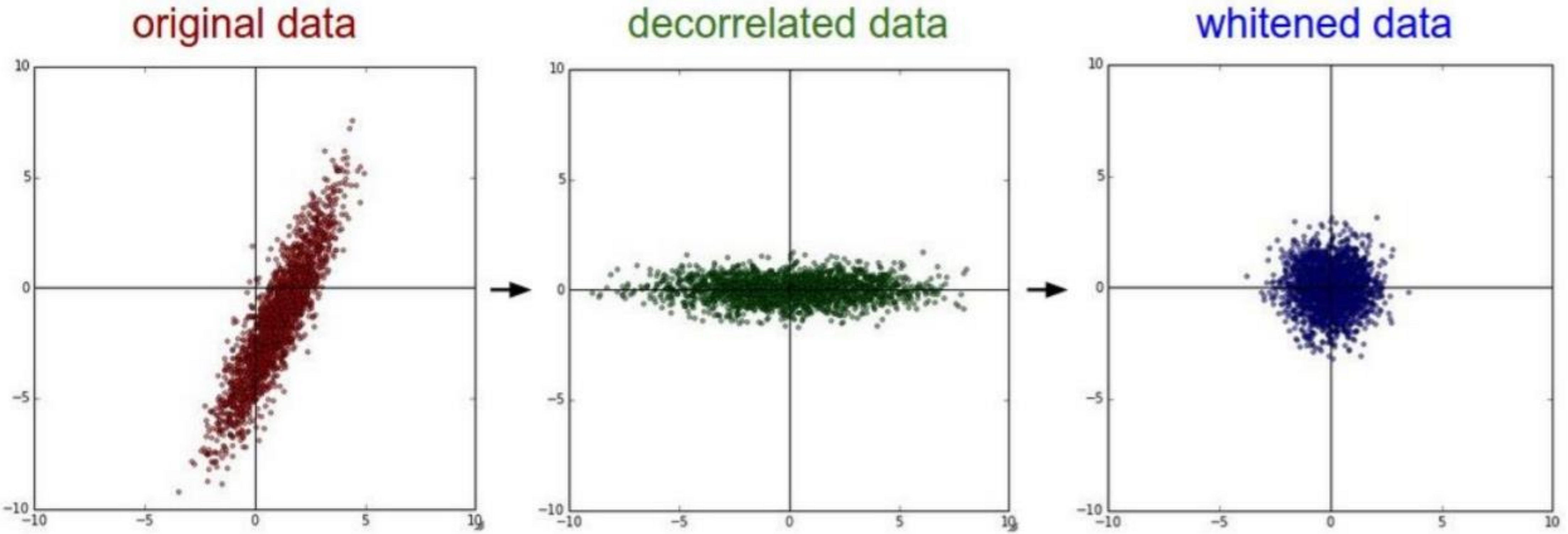
```
max_df = X_train.max()
min_df = X_train.min()

X_train_val = (X_train_val - min_df)/(max_df -
min_df)
X_train = (X_train - min_df)/(max_df - min_df)
```

Watch out:

- You might want to scale also the target in case of regression, as too large component might dominate when computing the error.
- This normalization might heavily suffer of outliers!

PCA – based preprocessing

This is performed after having «zero-centered» the data



diagonal covariance matrix

covariance = identity

Preprocessing and Training

- Any preprocessing statistics (e.g. the data mean) must be computed on training data, and applied to the validation / test data.
- Do not normalize first and then split in training, validation, test
- Normalization statistics are parameters of your ML model

TODO:

Colab Notebooks

First Colab Notebook is

[Feedforward Neural Network.ipynb]

This is already prepared notebook to show you

- how to assemble Neural Networks (MLP) for classifying tabular data (IRIS DATA)
- How to train Neural Networks on tabular data
- How to assess performance of Neural Networks

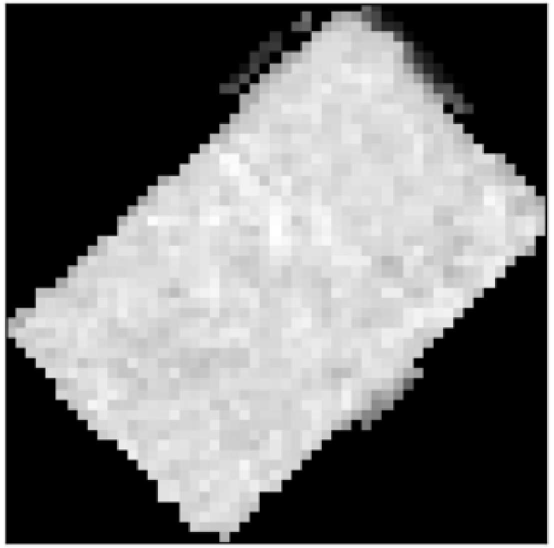
You will then be asked to replicate the same on penguin dataset

Colab Notebooks

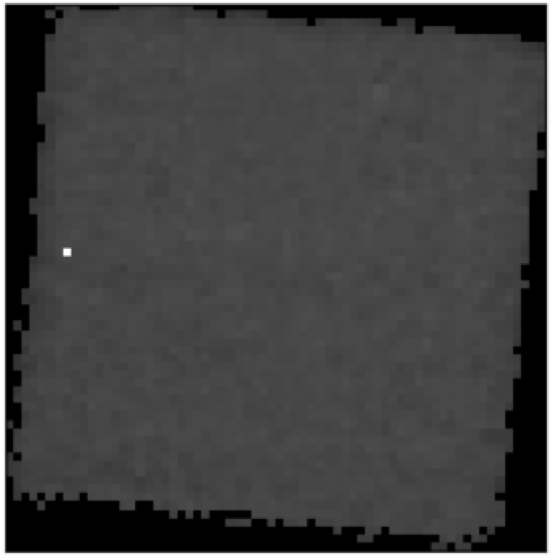
The second Colab Notebook implements the parcel classification problem: 2023_Lez_03_handcrafted_feature_classifier_parcel.ipynb

```
Training image index 181 has shape (53, 53) and label PARCEL
Training image index 91 has shape (67, 66) and label PARCEL
Training image index 149 has shape (57, 77) and label DOUBLE
Training image index 116 has shape (65, 62) and label DOUBLE
Training image index 228 has shape (74, 64) and label DOUBLE
Training image index 73 has shape (39, 34) and label PARCEL
Training image index 138 has shape (68, 93) and label DOUBLE
Training image index 94 has shape (69, 51) and label PARCEL
```

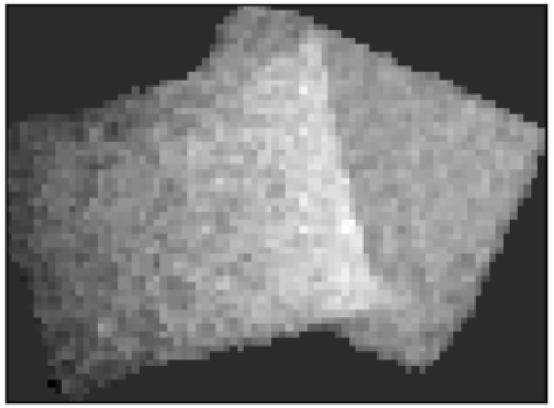
PARCEL



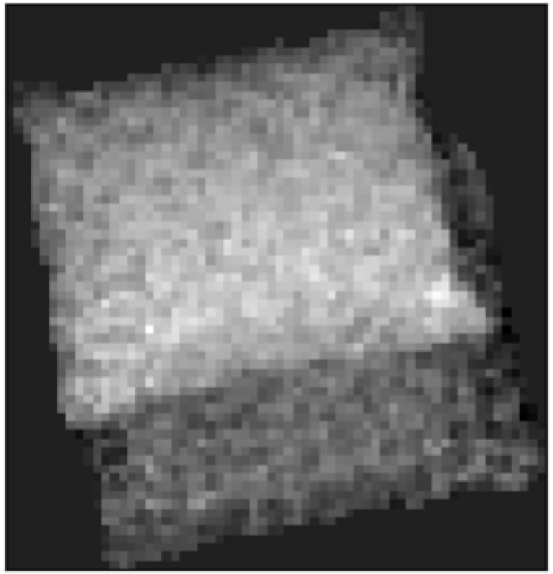
PARCEL



DOUBLE



DOUBLE



Colab Notebooks

The script is operational, but:

- Implement additional hand-crafted features in the function `makefeatures`
- Implement one of the following classifiers
    - Neural Network (refer to the notebook on feed-forward NN)
    - k-nearest neighbor
    - Decision Three

Let's go back to Image Classification

Giacomo Boracchi

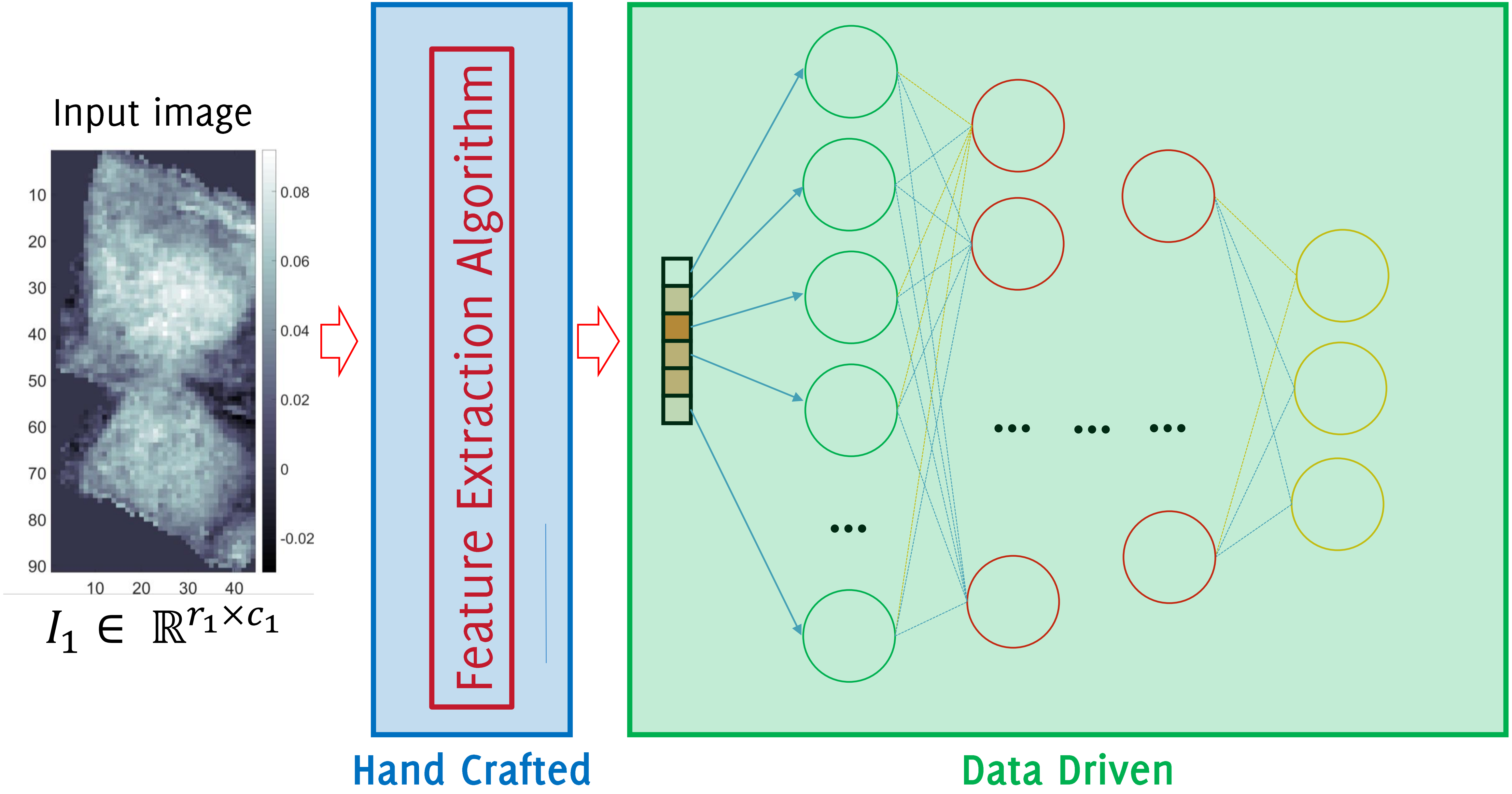
giacomo.boracchi@unibocconi.it

February 14th 2024

UEM, Maputo

https://boracchi.faculty.polimi.it

Image Classification by Hand Crafted Features



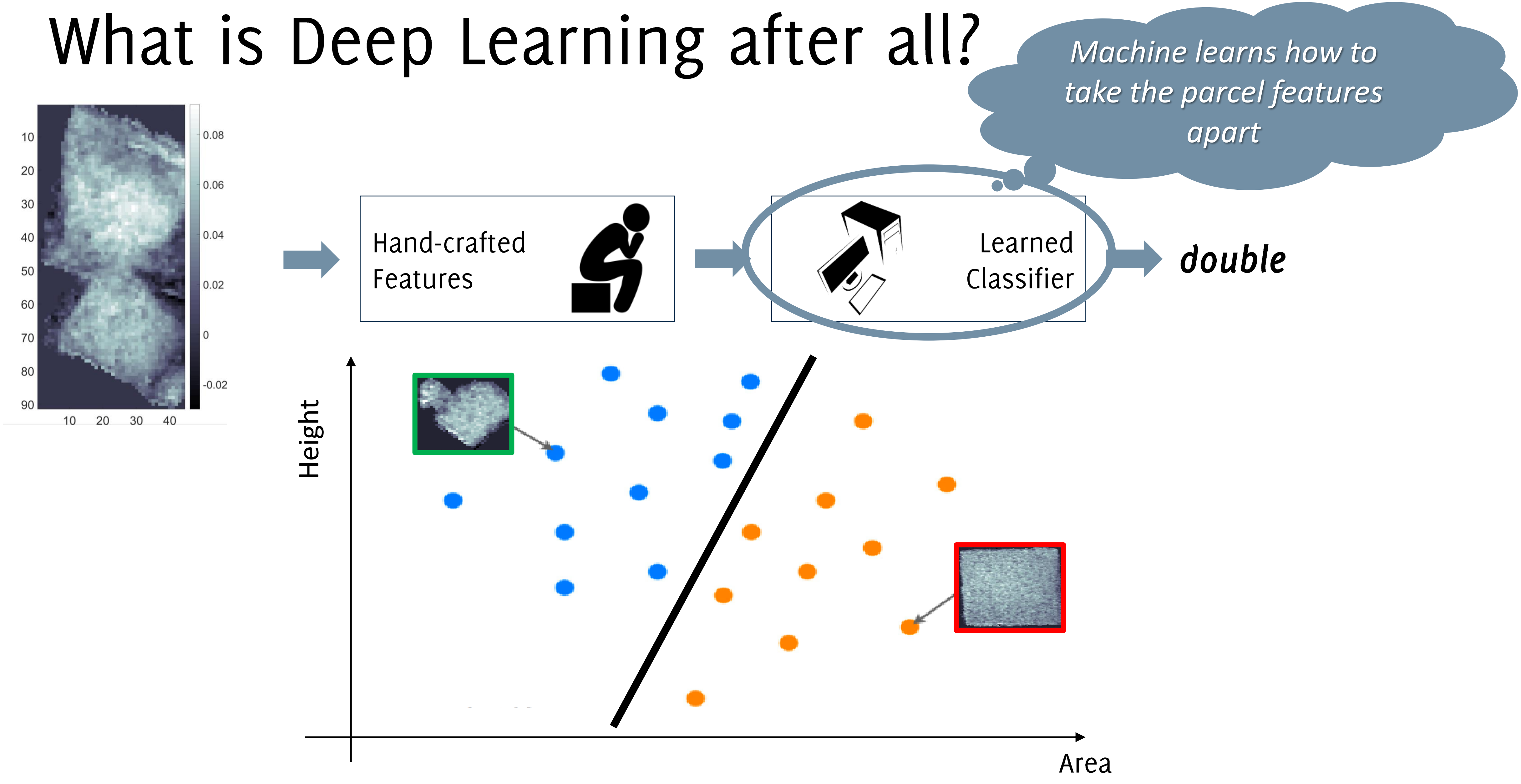
Hand Crafted Featues, pros:

- **Exploit a priori** / expert **information**
- Features are **interpretable** (you might understand why they are not working)
- You can **adjust features** to improve your performance
- **Limited amount of training data** needed
- You can give more relevance to some features

Hand Crafted Featues, cons:

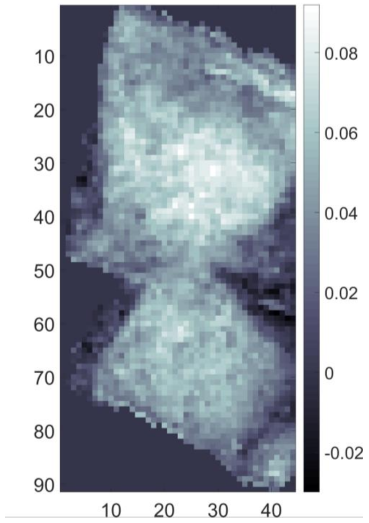
- Requires a lot of **design/programming efforts**
- **Not viable** in many **visual recognition** tasks that are easily performed by humans (e.g. when dealing with natural images)
- **Risk of overfitting** the training set used in the feature design
- **Not very general and "portable"**

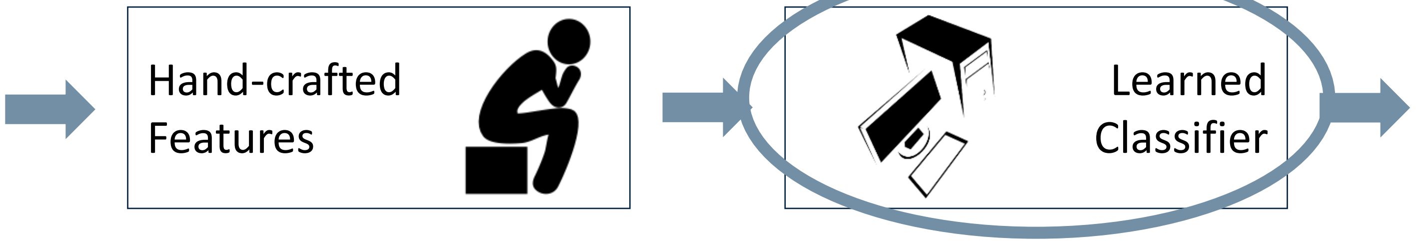
What is Deep Learning after all?

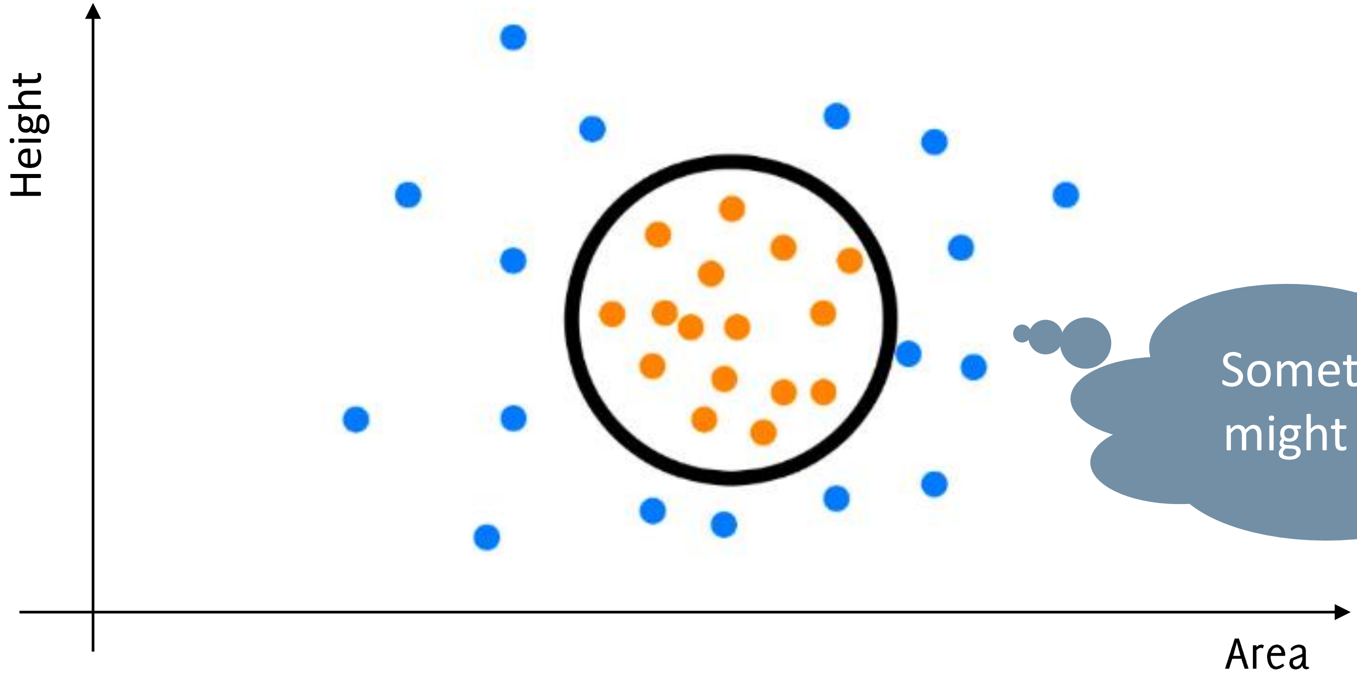


What is Deep Learning after all?

Machine learns how to take the parcel features apart



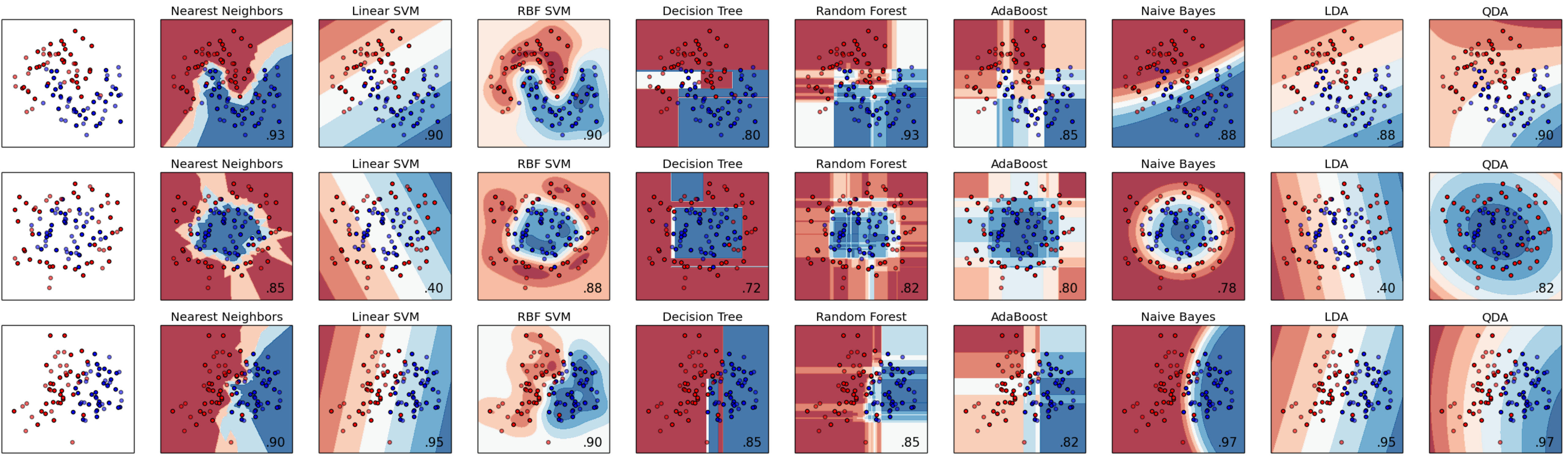




Sometimes the decision might be more complex

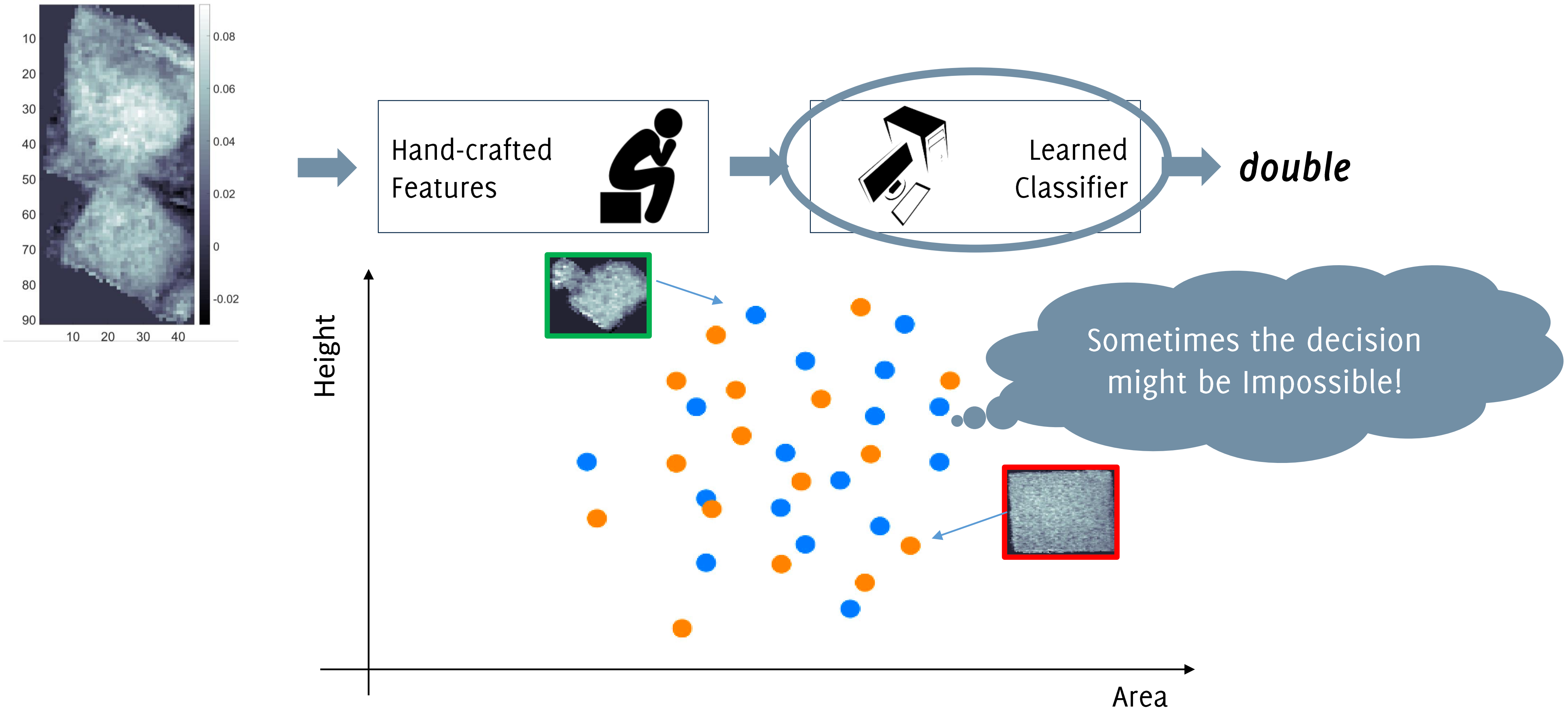
What is Deep Learning after all?

Machine learns how to take the Iris apart

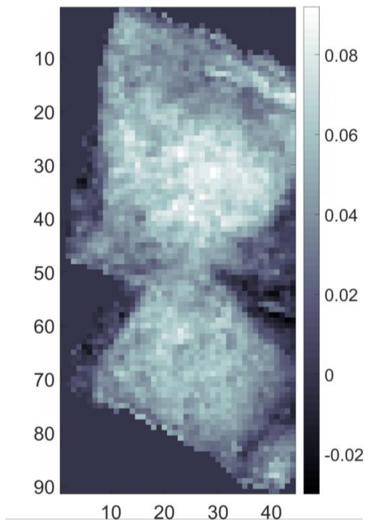


Sepal Lenght

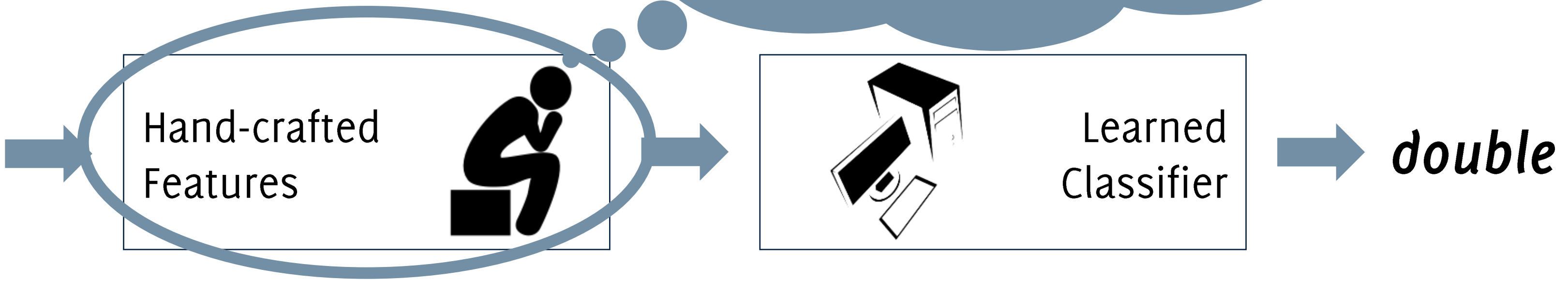
What is Deep Learning after all?

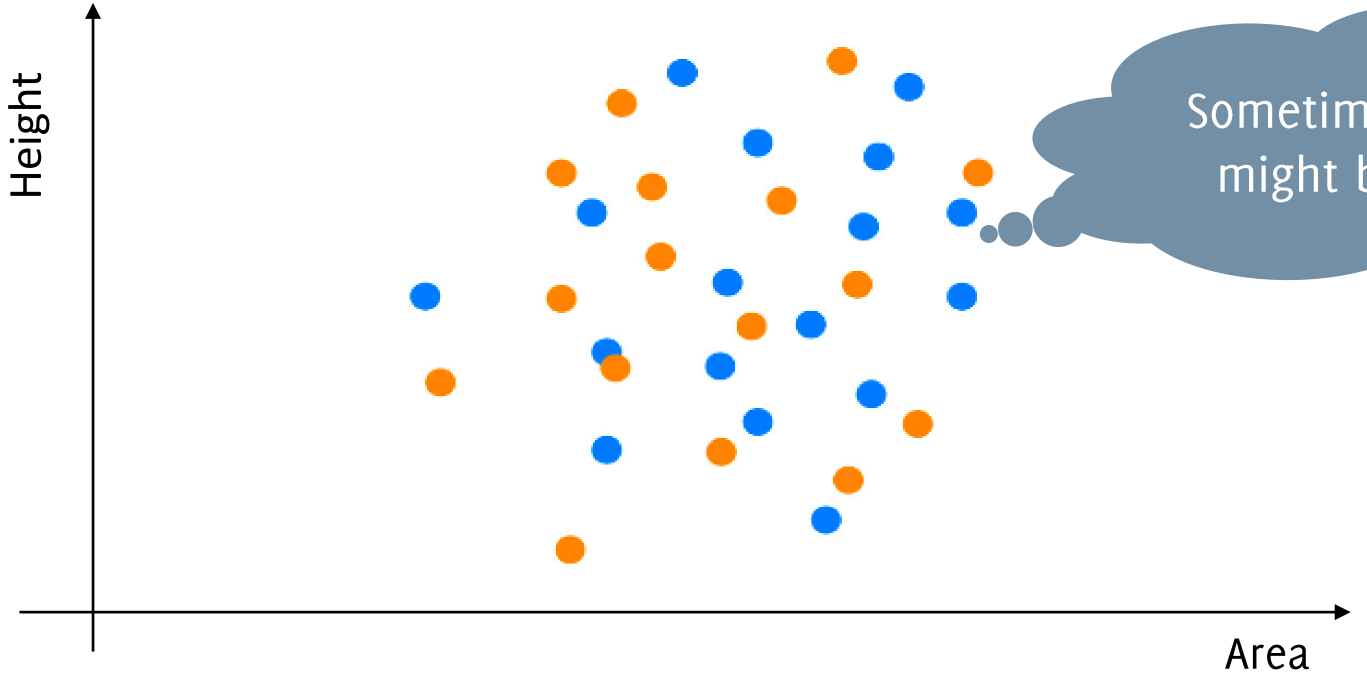


What is Deep Learning



This happens if you do not know which features to extract!!!





Sometimes the decision might be Impossible!

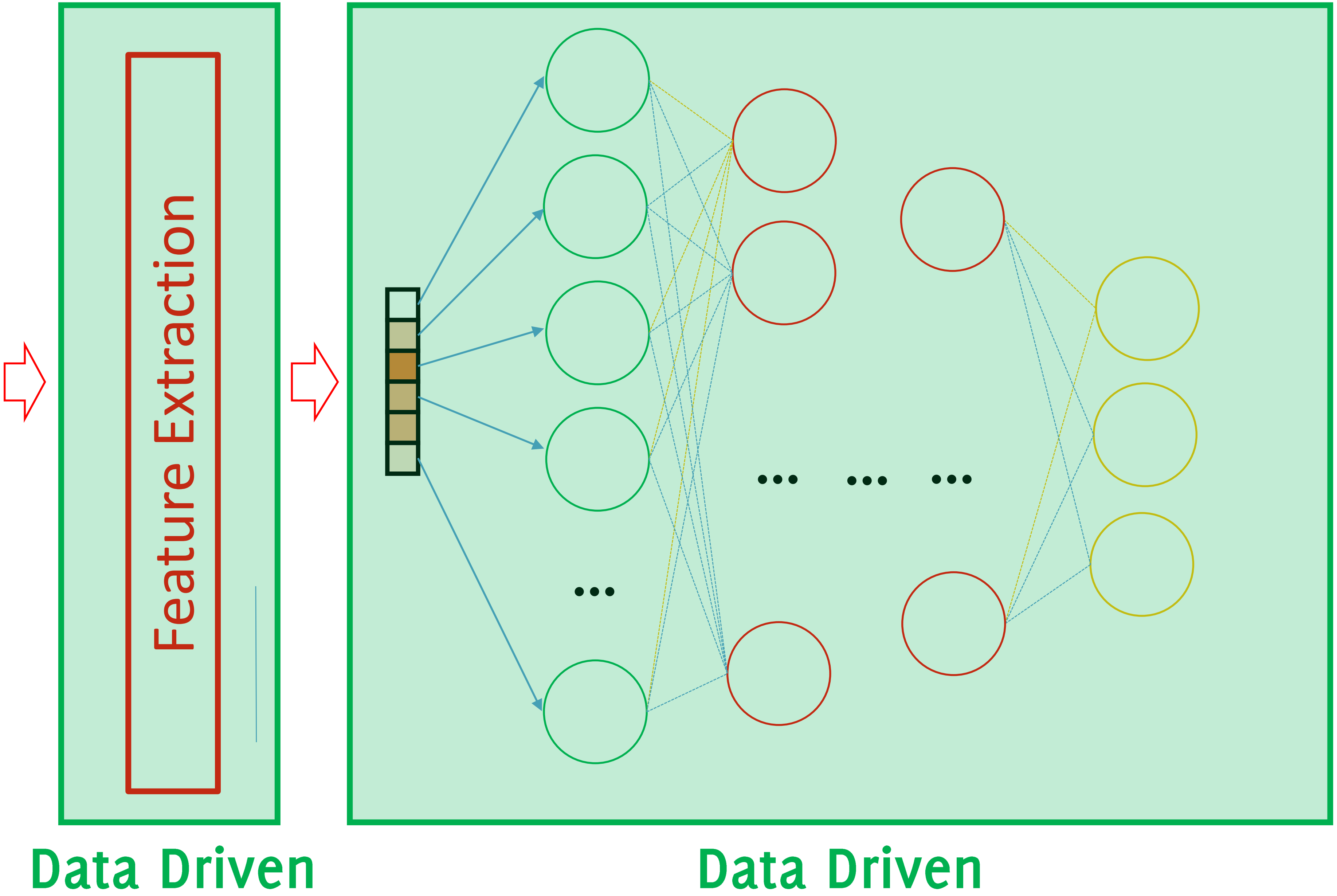
Data Driven Features

… the advent of Deep Learning

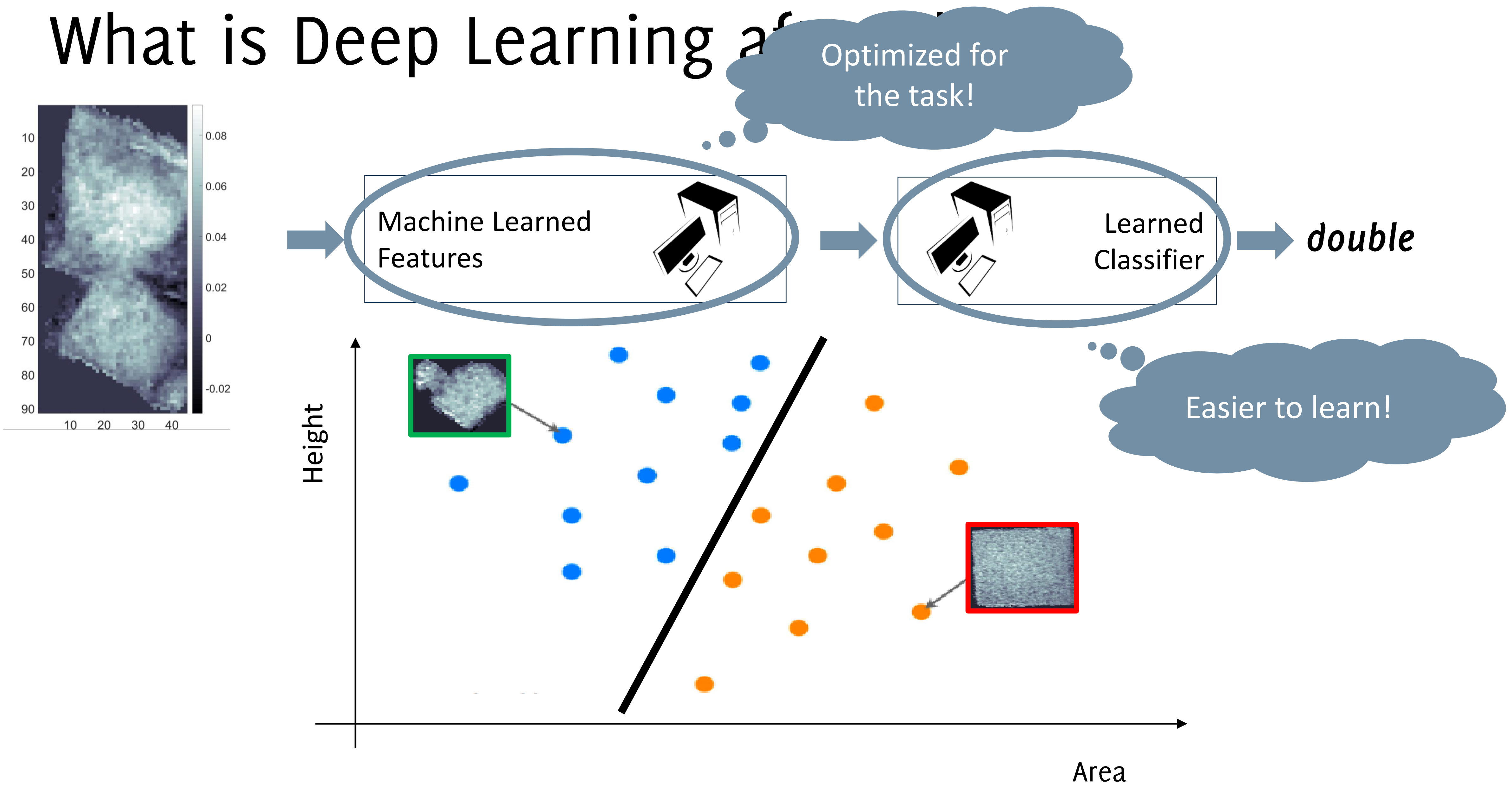
Data-Driven Features

Input image

$I_1 \in \mathbb{R}^{r_1 \times c_1}$

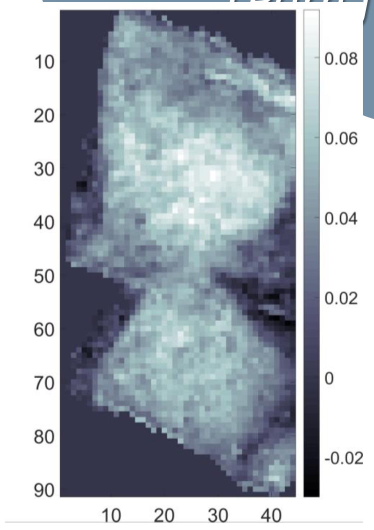


What is Deep Learning af

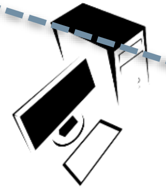


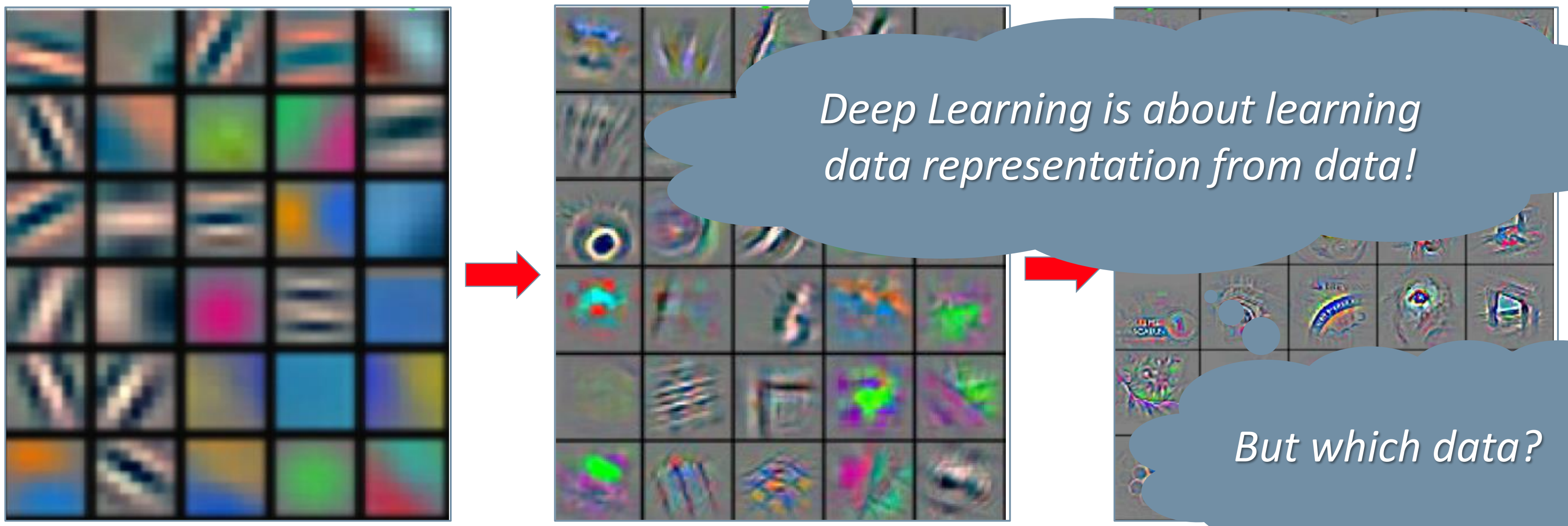
Wh… Learning aft…

Learn from data!



Hierarchical representation optimized for the task!

→ Learned features | Learned features | Learned features →  Learned Classifier → ***double***



Deep Learning is about learning data representation from data!

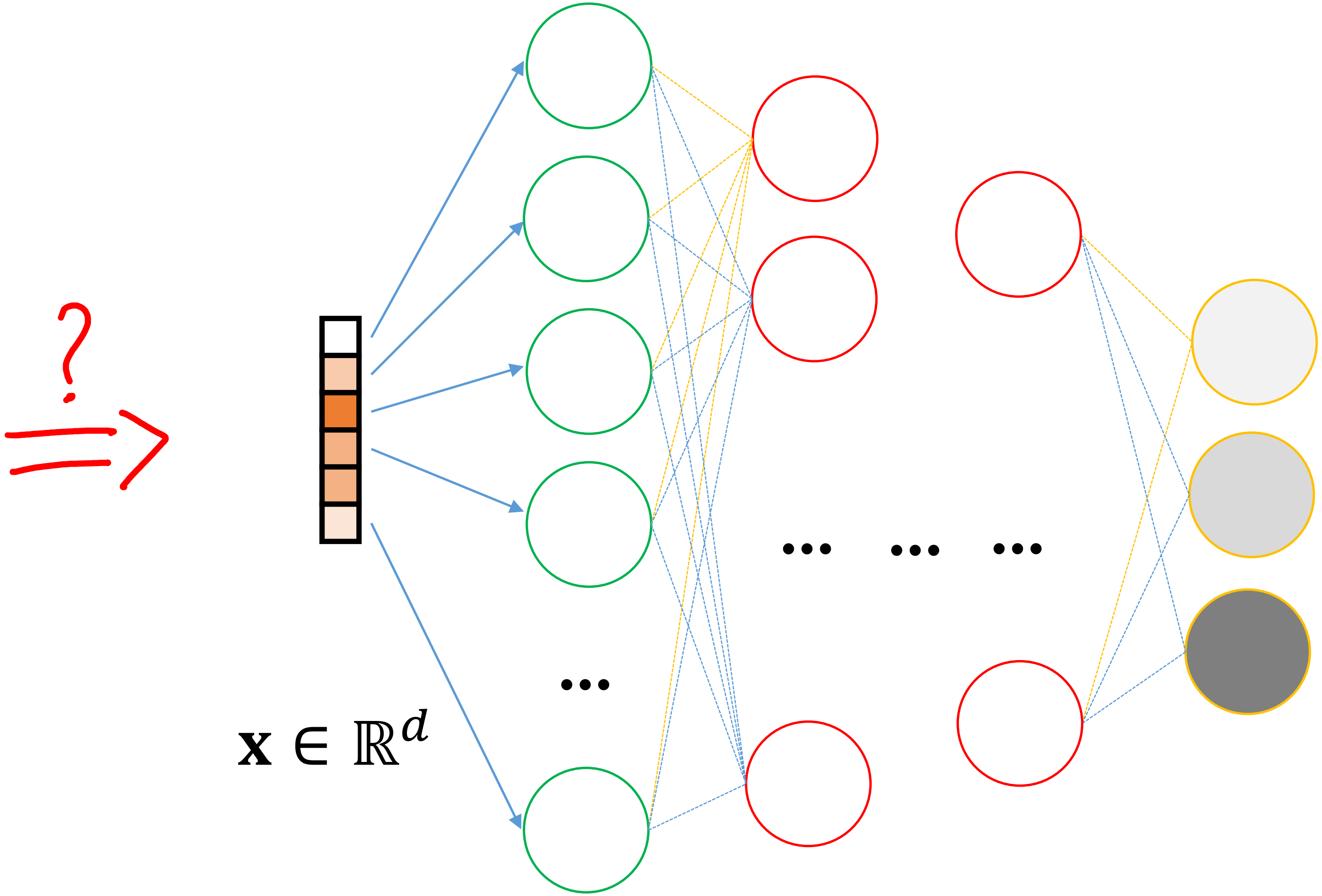
But which data?

Linear Classifier

the basic building block for deep architectures

How to feed images to NN?

$I \in \mathbb{R}^{R \times C \times 3}$



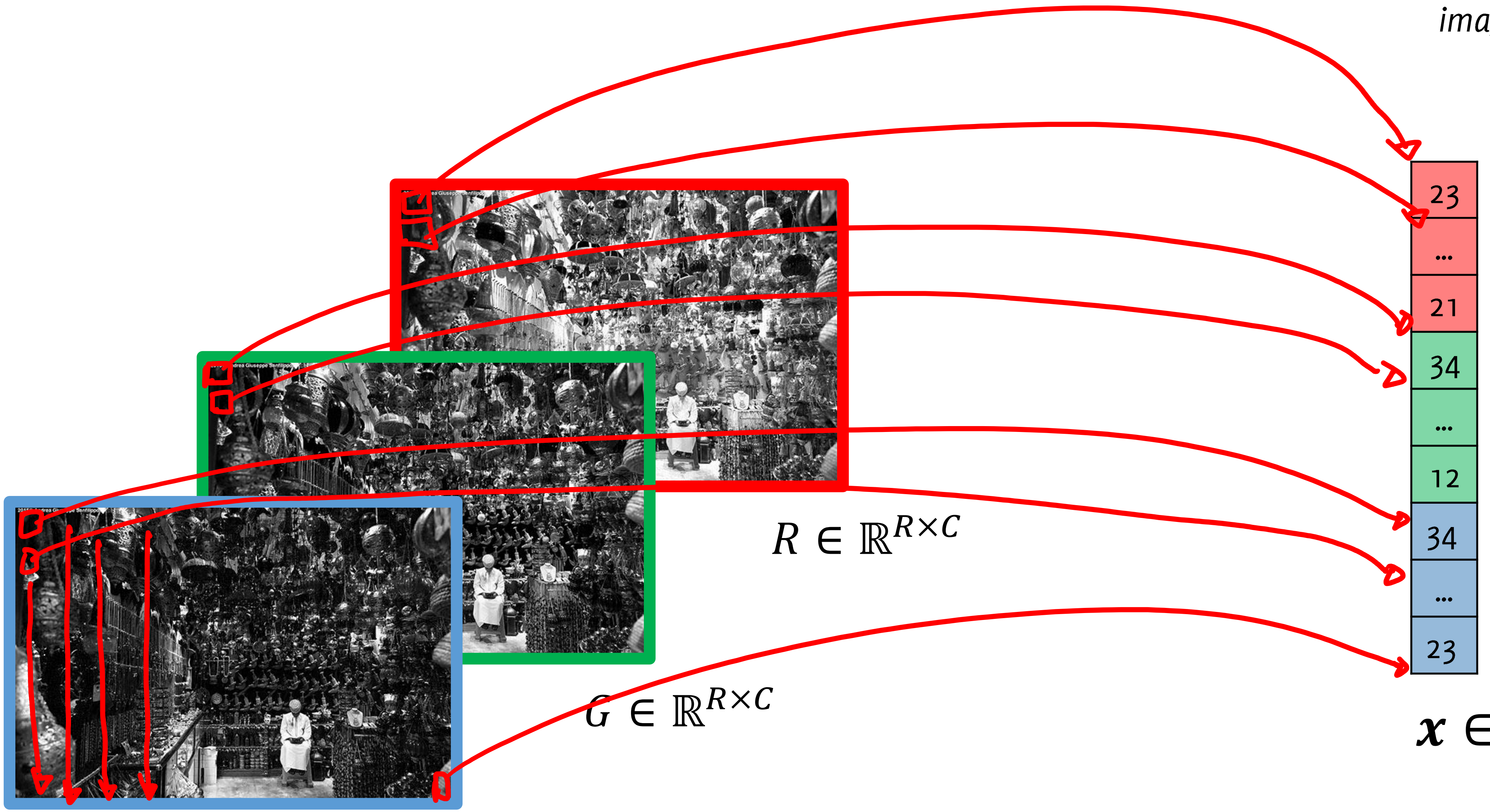
$\mathbf{x} \in \mathbb{R}^d$

input layer

Hidden layer(s)

Column-wise unfolding

Colors recall the color plane where images are from



$R \in \mathbb{R}^{R \times C}$

$G \in \mathbb{R}^{R \times C}$

$B \in \mathbb{R}^{R \times C}$

$\boldsymbol{x} \in \mathbb{R}^d$

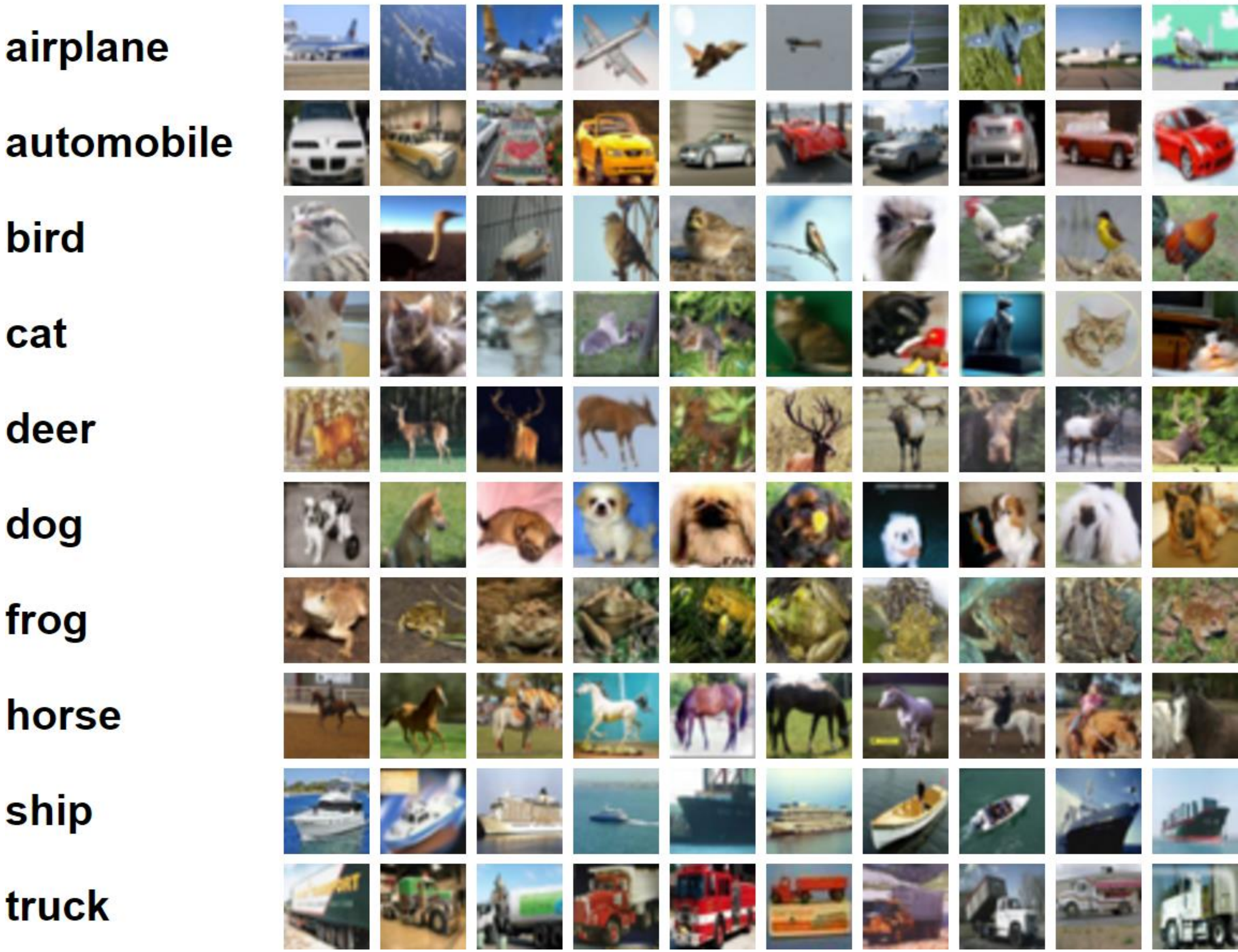
$d = R \times C \times 3$

Classification over the CIFAR-10 dataset

The CIFAR-10 dataset contains 60000 images:

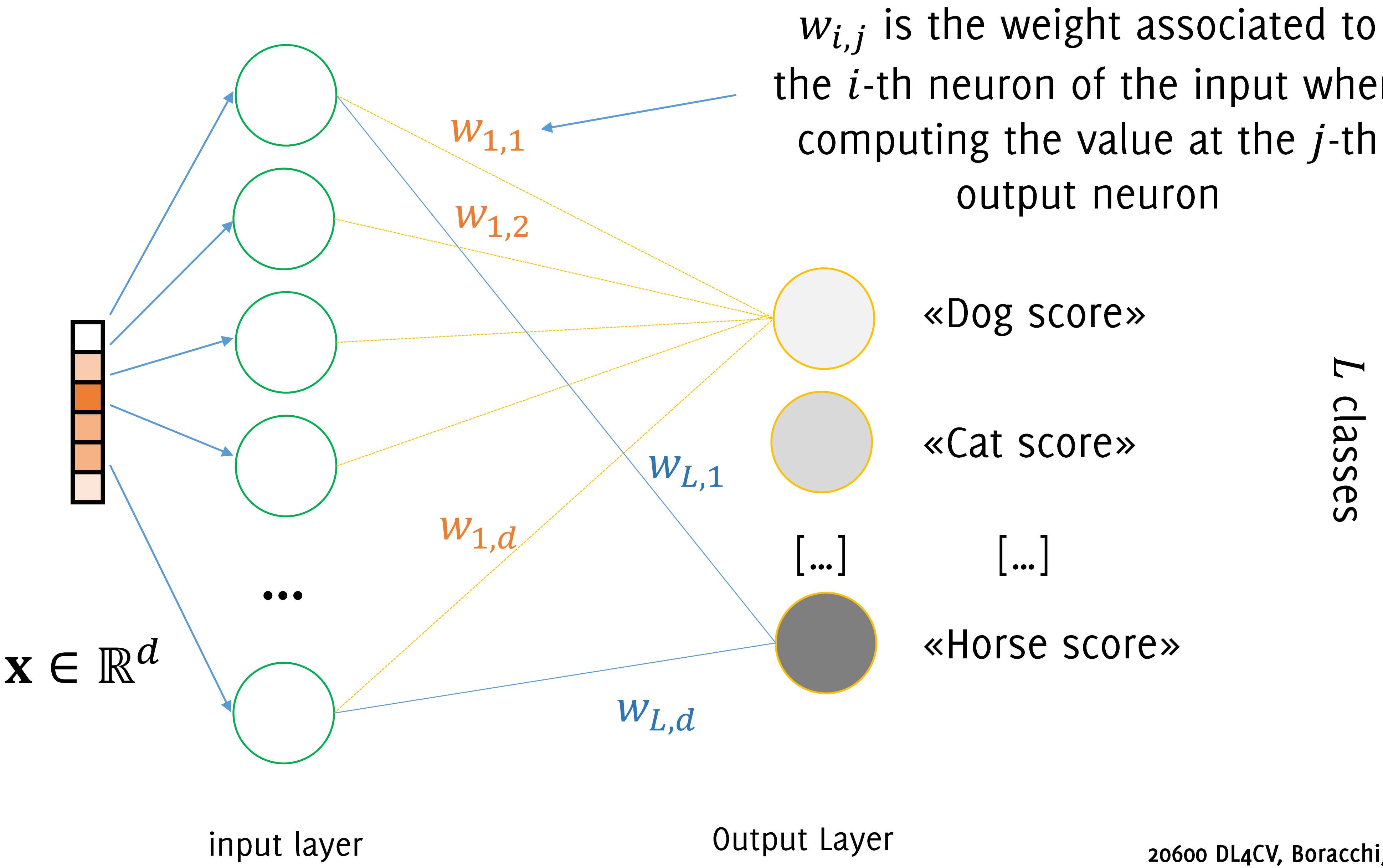
- Each image is 32x32 RGB
- Images are in 10 classes
- 6000 images per class

$$\boldsymbol{x} \in \mathbb{R}^d, d = 3072$$

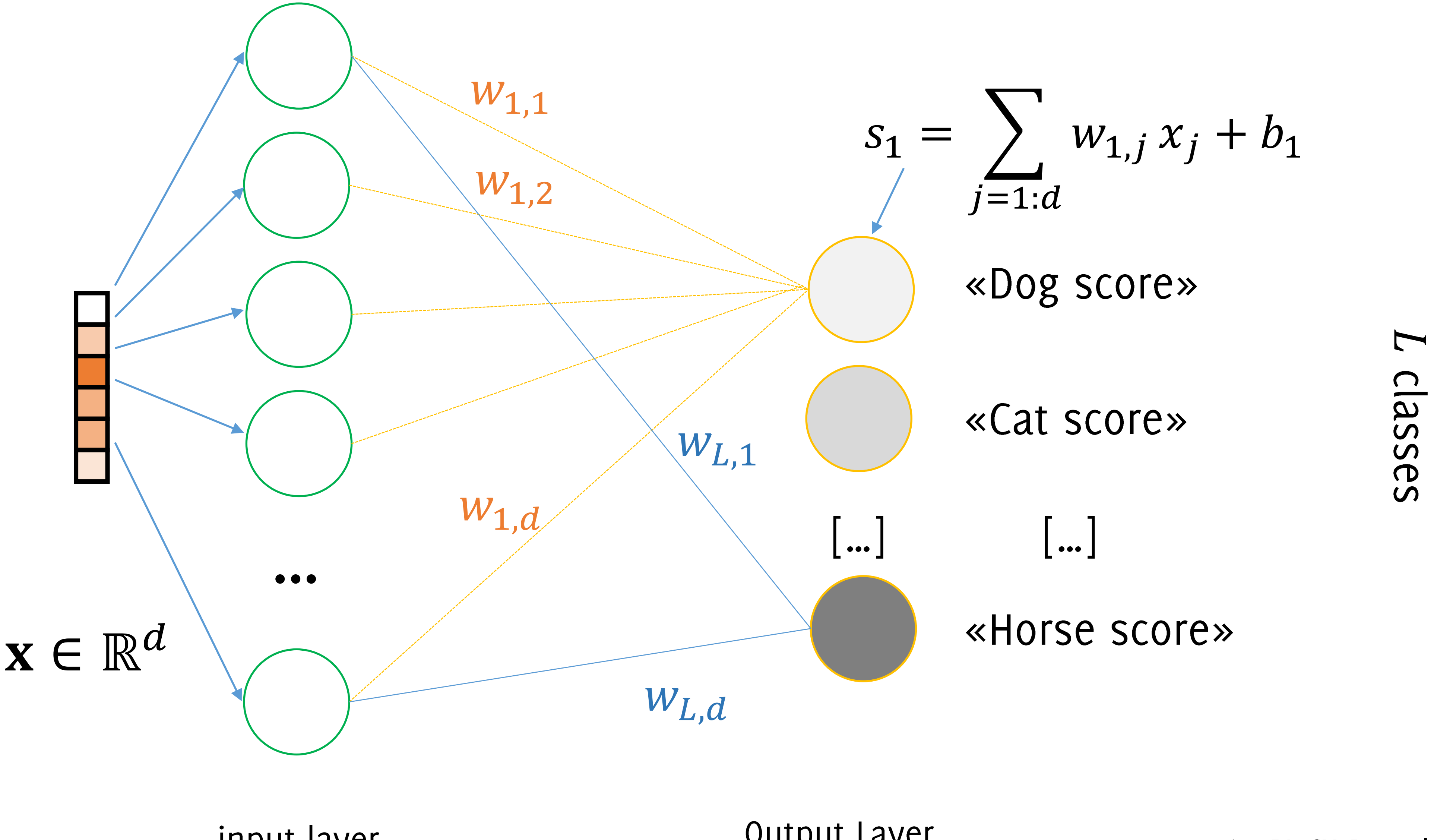


Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

A 1-layer NN to Classify Images



A 1-layer NN to Classify Images



model.summary();

```
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           [(None, 32, 32, 3)]       0
_________________________________________________________________
Flatten (Flatten)            (None, 3072)              0
_________________________________________________________________
Output (Dense)               (None, 10)                30730
=================================================================
Total params: 30,730
Trainable params: 30,730
Non-trainable params: 0
```

Why don't we take a larger network?

Dimensionality prevents us from using in a straightforward manner deep NN as those seen so far.

Let's take a network with **an hidden layer** having half of the neurons of the input layer.

On CIFAR10 images, the number of neurons would be:

- 3072 first layer
- 1536 second layer
- 10 output layer

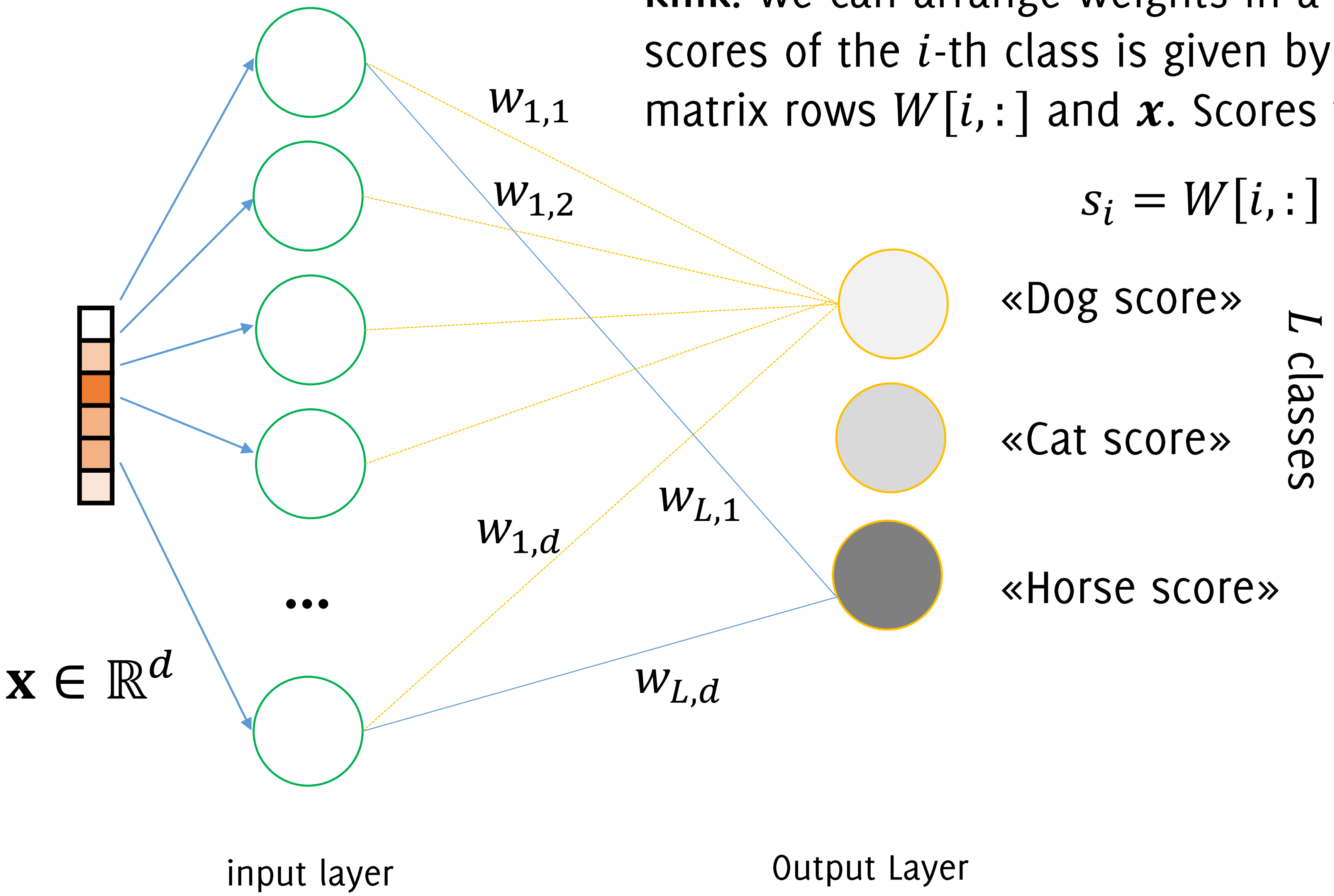
1,536 * 3,072 + 1,536 = **4,720,128 parameters (!)**

10* 1,536 + 10 = 15,370 parameters

A 1-layer NN to Classify Images

Rmk: we can arrange weights in a matrix $W \in \mathbb{R}^{L \times d}$, then the scores of the i-th class is given by inner product between the matrix rows $W[i,:]$ and $\boldsymbol{x}$. Scores then becomes:

$$s_i = W[i,:] * \boldsymbol{x} + b_i$$



Rmk: nonlinearity is not needed here since there are no layers following

Rmk: we can also ignore the softmax in the output since this would not change the order of the scores (would just normalize them)

A 1-layer NN to Classify Images

$W \in \mathbb{R}^{L \times d}$

-8.1	…	2.7	9.5	…	-9.0	-5.4	…	4.8
9.0	…	5.4	4.8	…	1.2	9.5	…	-8.0
1.2	…	9.5	-8.0	…	8.1	-2.7	…	9.5

$*$

$\boldsymbol{x}$ = [23, …, 21, 34, …, 12, 34, …, 23]

$+$

$\boldsymbol{b}$ = [-2, 32, -1]

$=$

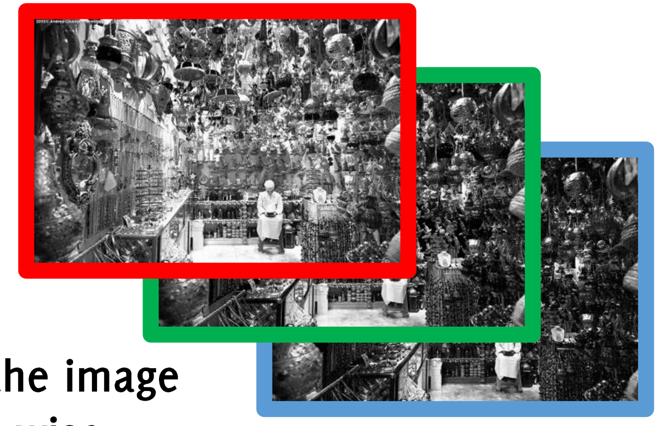
$\mathcal{K}(\boldsymbol{x}; W, \boldsymbol{b})$ = [-4, 22, 33]

s_1 dog score

s_2 cat score

s_3 horse score

Rmk: colors indicate to which color plane in the image these weights refer to



Unroll the image column-wise

A 1-layer NN to Classify Images

$W \in \mathbb{R}^{L \times d}$

W[1,:]	-8.1	…	2.7	9.5	…	-9.0	-5.4	…	4.8
W[2,:]	9.0	…	5.4	4.8	…	1.2	9.5	…	-8.0
W[3,:]	1.2	…	9.5	-8.0	…	8.1	-2.7	…	9.5

$*$

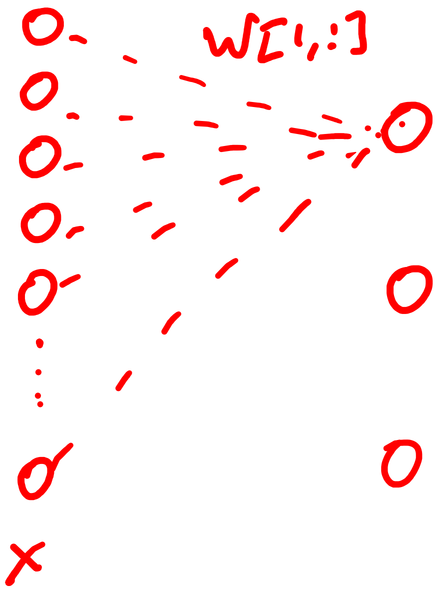
$\boldsymbol{x}$ = [23, …, 21, 34, …, 12, 34, …, 23]

$+$ $\boldsymbol{b}$ = [-2, 32, -1]

$=$ $\mathcal{K}(\boldsymbol{x}; W, \boldsymbol{b})$ = [-4, 22, 33]

s_1 dog score
s_2 cat score
s_3 horse score

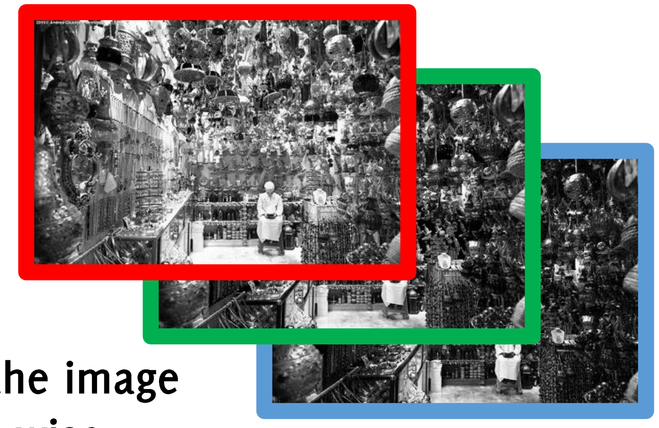
Rmk: colors indicate to which color plane in the image these weights refer to



$s_1 = W[1,:] \cdot x + b_1$

$s_2 = W[2,:] \cdot x + b_2$

$s_3 = W[3,:] x + b_3$



Unroll the image column-wise

This simple layer is a linear classifier

In linear classification $\mathcal{K}$ is a linear function:

$$\mathcal{K}(\boldsymbol{x}) = W\boldsymbol{x} + b$$

where $W \in \mathbb{R}^{L\times d}$, $b \in \mathbb{R}^{L}$ are the parameters of the classifier $\mathcal{K}$.

W are referred to as the weights, b the bias vector.

L

-8.1	...	2.7	9.5	...	-9.0	-5.4	...	4.8
9.0	...	5.4	4.8	...	1.2	9.5	...	-8.0
1.2	...	9.5	-8.0	...	8.1	-2.7	...	9.5

W

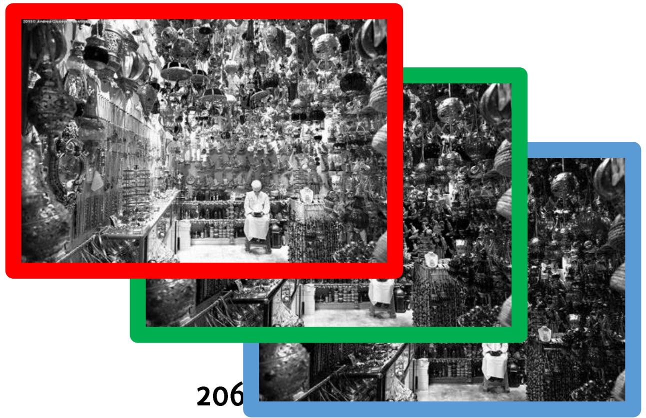
$$* \begin{bmatrix} 23 \\ \dots \\ 21 \\ 34 \\ \dots \\ 12 \\ 34 \\ \dots \\ 23 \end{bmatrix} + \begin{bmatrix} -2 \\ 32 \\ -1 \end{bmatrix} = \begin{bmatrix} -4 \\ 22 \\ 33 \end{bmatrix}$$

$\boldsymbol{x}$ $\boldsymbol{b}$ $\mathcal{K}(\boldsymbol{x}; W, \boldsymbol{b})$

s_1 dog score

s_2 cat score

s_3 rabbit score



Unroll the image column-wise

This simple layer is a linear classifier

The classifier assign to an input image the class corresponding to the largest score

$$\hat{y}_j = \underset{i=1,..,L}{\text{argmax}} \left[s_j\right]_{\boldsymbol{i}}$$

being $\left[\boldsymbol{s}_j\right]_{\boldsymbol{i}}$ the i –th component of the vector

$$\mathcal{K}(\boldsymbol{x}) = \mathbf{s} = W\boldsymbol{x} + \boldsymbol{b}$$

-8.1	...	2.7	9.5	...	-9.0	-5.4	...	4.8
9.0	...	5.4	4.8	...	1.2	9.5	...	-8.0
1.2	...	9.5	-8.0	...	8.1	-2.7	...	9.5

W

$*$

23
...
21
34
...
12
34
...

$\boldsymbol{x}$

$+$

-2
32
-1

$\boldsymbol{b}$

$=$

-4	s_1 dog score
22	s_2 cat score
33	s_3 horse score

$\mathcal{K}(\boldsymbol{x}; W, \boldsymbol{b})$

Rmk: softmax is not needed as long as we take as output the largest score: this would be the one yielding the largest posterior

The Parameters of a Linear Classifier

The **score of a class is the weighted sum** of all the image pixels.
Weights are actually the classifier parameters.

The weights are:

-8.1	...	2.7	9.5	...	-9.0	-5.4	...	4.8
9.0	...	5.4	4.8	...	1.2	9.5	...	-8.0
1.2	...	9.5	-8.0	...	8.1	-2.7	...	9.5

W

-2
32
-1

$\boldsymbol{b}$

and indicate which are the most important pixels / colours

Why nonlinear layers?

Each layer in a NN can be seen as matrix multiplication (+ bias).

$$\mathbf{s} = W\boldsymbol{x} + \boldsymbol{b}$$

If we stack 3 layers without activations:

$$\mathbf{s} = \big((W_1\boldsymbol{x} + \boldsymbol{b}_1)\mathrm{W}_2 + \boldsymbol{b}_2\big)W_3 + b_3$$

This becomes equivalent to

$$\mathbf{s} = W\boldsymbol{x} + \boldsymbol{b}$$

This is a further confirmation why it becomes pointless to stack many layers without including a nonlinear activations…

Training the Linear Classifier

Training a Classifier

Given a training set TR and a loss function, define the parameters that minimize the loss function over the whole TR

In case of linear classifier

$$[W, b] = \underset{W \in \mathbb{R}^{L \times d},\ b \in \mathbb{R}^{L}}{\text{argmin}} \sum_{(\boldsymbol{x}_i, y_i) \in TR} \mathcal{L}_{W,b}(\boldsymbol{x}, y_i)$$

Solving this minimization problem provides the weights of our classifier

Loss Function

Loss function: a function $\mathcal{L}$ that measures our unhappiness with the score assigned to training images

The loss $\mathcal{L}$ will be high on a training image that is not correctly classifier, low otherwise.

Loss Function Minimization

Loss function can be minimized by gradient descent and all its variants (see Prof. Matteucci classes)

The loss function has to be typically regularized to achieve a unique solution satisfying some desired property

$$[W, b] = \underset{W \in \mathbb{R}^{L \times d},\ b \in \mathbb{R}^{L}}{\operatorname{argmin}} \sum_{(\boldsymbol{x}_i, y_i) \in TR} \mathcal{L}_{W,b}(\boldsymbol{x}, y_i) + \lambda\, \mathcal{R}(W, b)$$

being $\lambda > 0$ a parameter balancing the two terms

… Once Trained

The training data is used to learn the parameters $W, \boldsymbol{b}$

The classifier is expected to assign to the correct class a score that is larger than that assigned to the incorrect classes.

Once the training is completed, it is possible to discard the whole training set and keep only the learned parameters.

-8.1	…	2.7	9.5	…	-9.0	-5.4	…	4.8
9.0	…	5.4	4.8	…	1.2	9.5	…	-8.0
1.2	…	9.5	-8.0	…	8.1	-2.7	…	9.5

W

-2
32
-1

$\boldsymbol{b}$

Geometric Interpretation of a Linear Classifier

$\mathbf{W}[\mathbf{i},:]$ is a d −dimensional vector **containing the weights** of the score function for **the i-th class**.

Computing the score function for the i −th class corresponds to computing **the inner product** (and summing the bias)

$$W[i,:] * \boldsymbol{x} + b_i$$

Thus, the NN computes the inner products against L different weights vectors, and selects the one yielding the largest score (up to bias correction)

Rmk: these "inner product classifiers" operate independently, and the output of the j-th row is not influenced weights at a different row

Rmk: this would not be the case if the network had hidden layer that would mix the outputs of intermediate layers

Geometric Interpretation of a Linear Classifier

In Python notation:

In Python $*$ denotes the element-wise product, here I mean the inner product of vectors:

$$\mathrm{np.inner}(W[i,:], \boldsymbol{x}) + b_i$$

Geometric Interpretation

Interpret each image as a point in $\mathbb{R}^d$.

Each classifier is a weighted sum of pixels, which corresponds to a linear function in $\mathbb{R}^d$

In $\mathbb{R}^2$ these would be

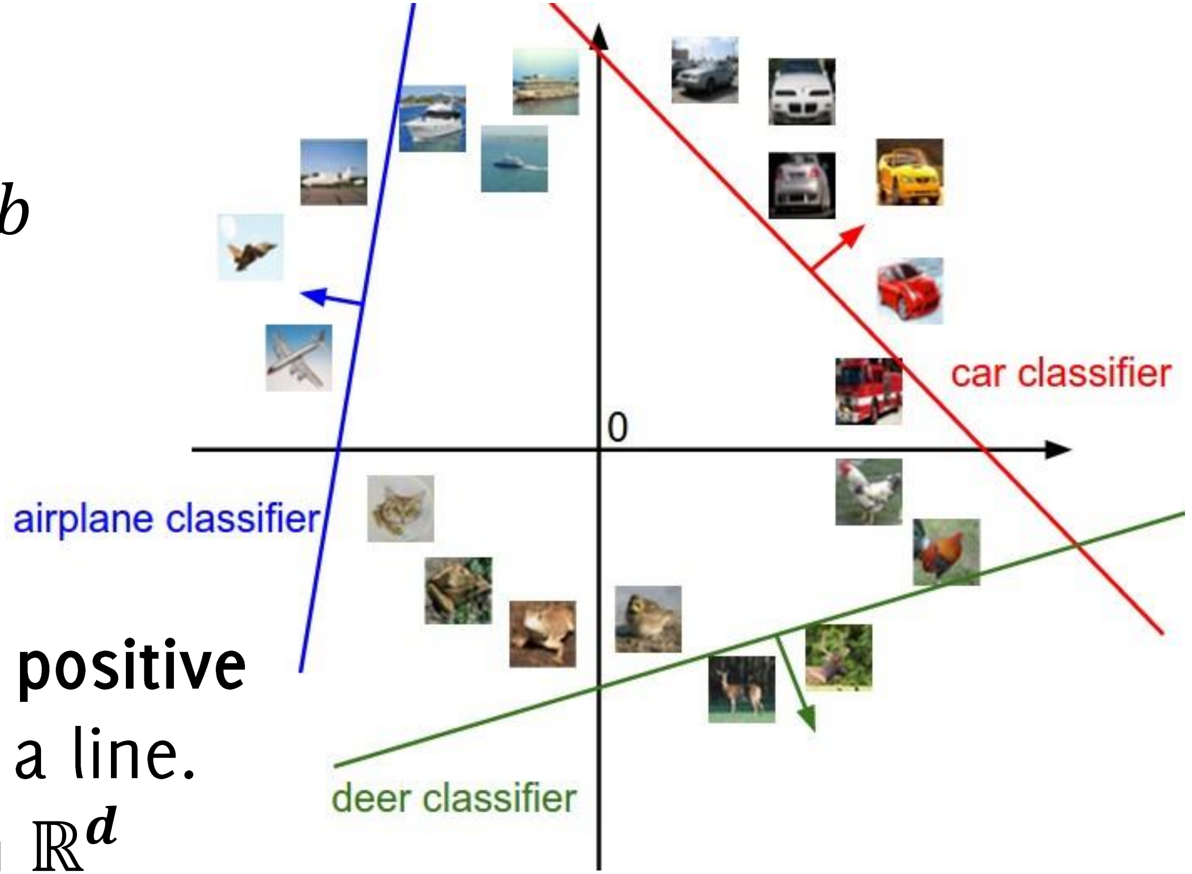
$$f([x_1, x_2]) = w_1 x_1 + w_2 x_2 + b$$

Then, points $[x_1, x_2]$ yielding

$$f([x_1, x_2]) = 0$$

would be lines.

Thus, in $\mathbb{R}^2$ the **region that separates positive from negative scores for each class is** a line. This region becomes **an hyperplane in** $\mathbb{R}^d$



Template Matching Interpretation

In Python notation:

- $W[i,:]$ is a d −dimensional vector containing the weights of the score function for the i −th class
- Computing the score function for the i −th class corresponds to computing the inner product

$$W[i,:] * \boldsymbol{x} + b_i$$

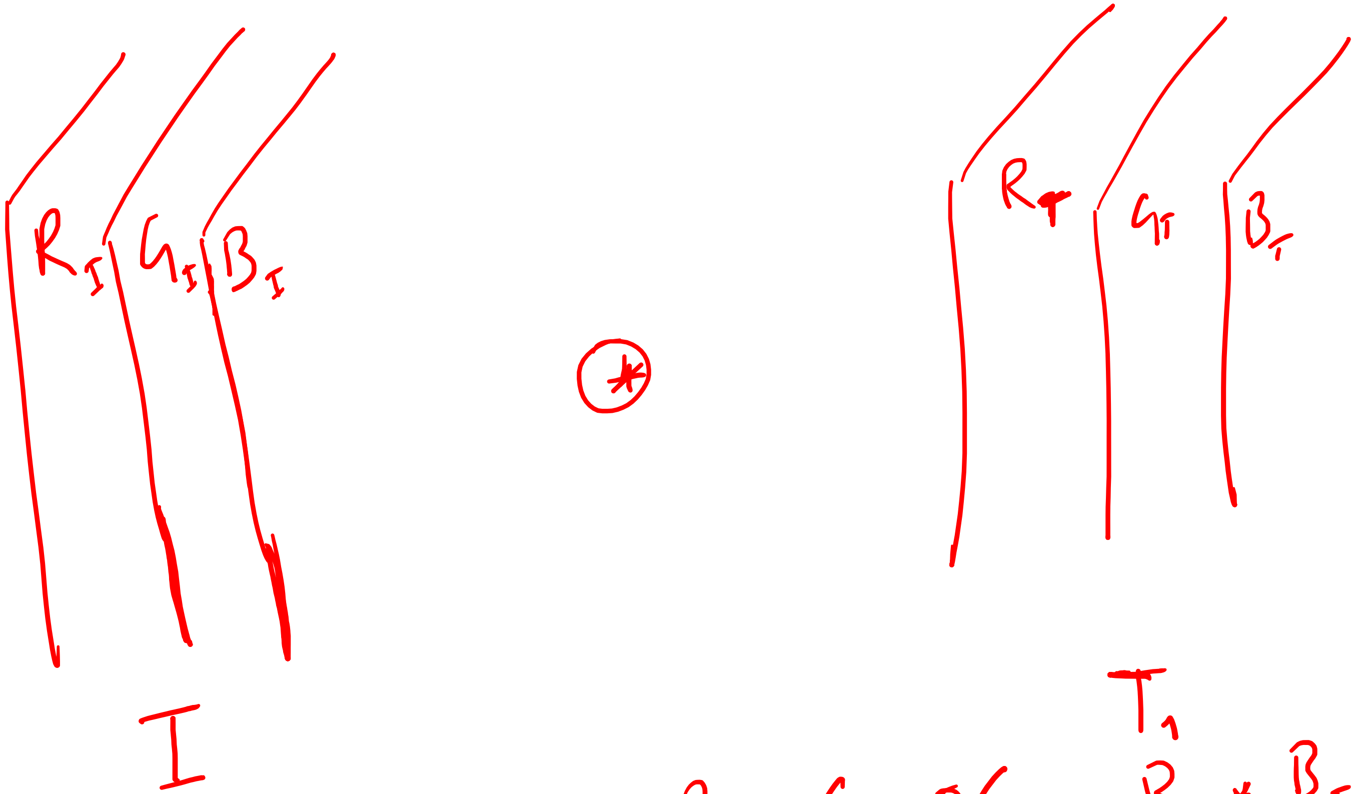
Then, $W[i,:]$ **can be seen as a template used in matching (the output of correlation in the central pixel)**

The template $W(i,:)$ is learned to match at best images belonging to the i −th class

Let's have a look at these templates

Correlation between two RGB images

The image and the filter have the same size



$(R_I * R_T)(0,0) +$

$(G_I * G_T)(0,0) +$

$(R_I * G_T)(0,0)$

$W_1 \cdot x$

$R_I \otimes R_T + G_I \otimes G_T + B_I * B_T =$

$\sum_{x, y} R_I(x,y) \cdot R_T(x,y) + \dots$

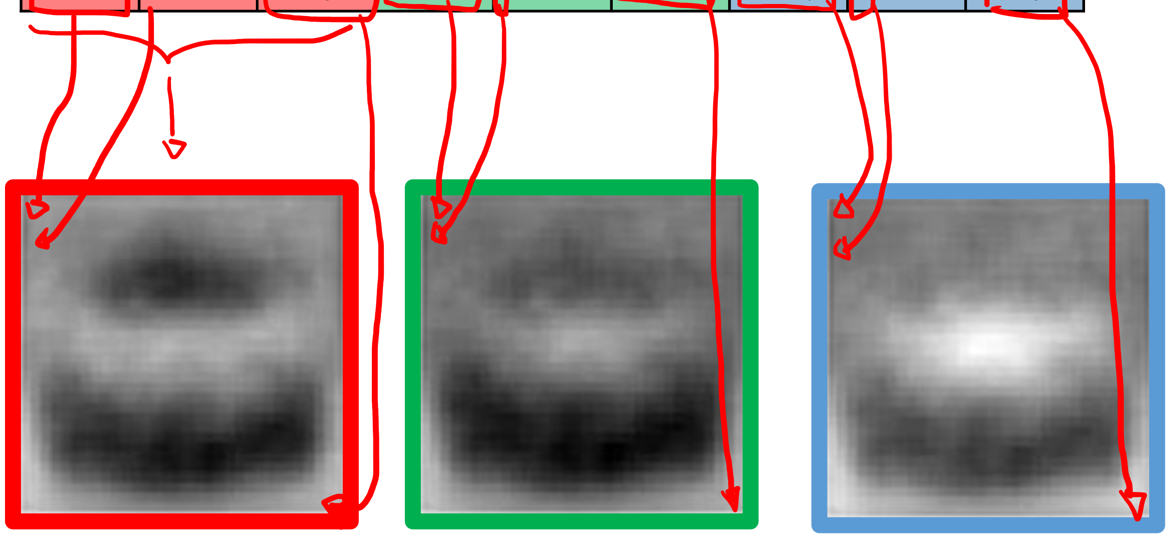
Bring the classifier weights back to images

W

-8.1	…	2.7	9.5	…	-9.0	-5.4	…	4.8
9.0	…	5.4	4.8	…	1.2	9.5	…	-8.0
1.2	…	9.5	-8.0	…	8.1	-2.7	…	9.5

$W[i,:] \in \mathbb{R}^d$, car classifier

$d = R \times C \times 3$



$R \in \mathbb{R}^{R\times C}$ $G \in \mathbb{R}^{R\times C}$ $B \in \mathbb{R}^{R\times C}$

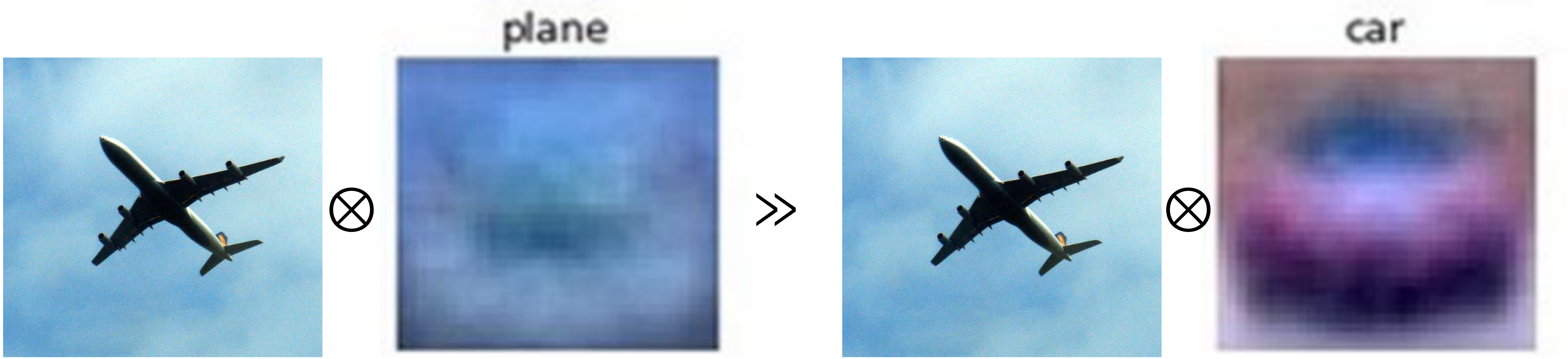
car template in $\mathbb{R}^{R\times C\times 3}$

Templates Learned on the CIFAR-10 dataset

CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

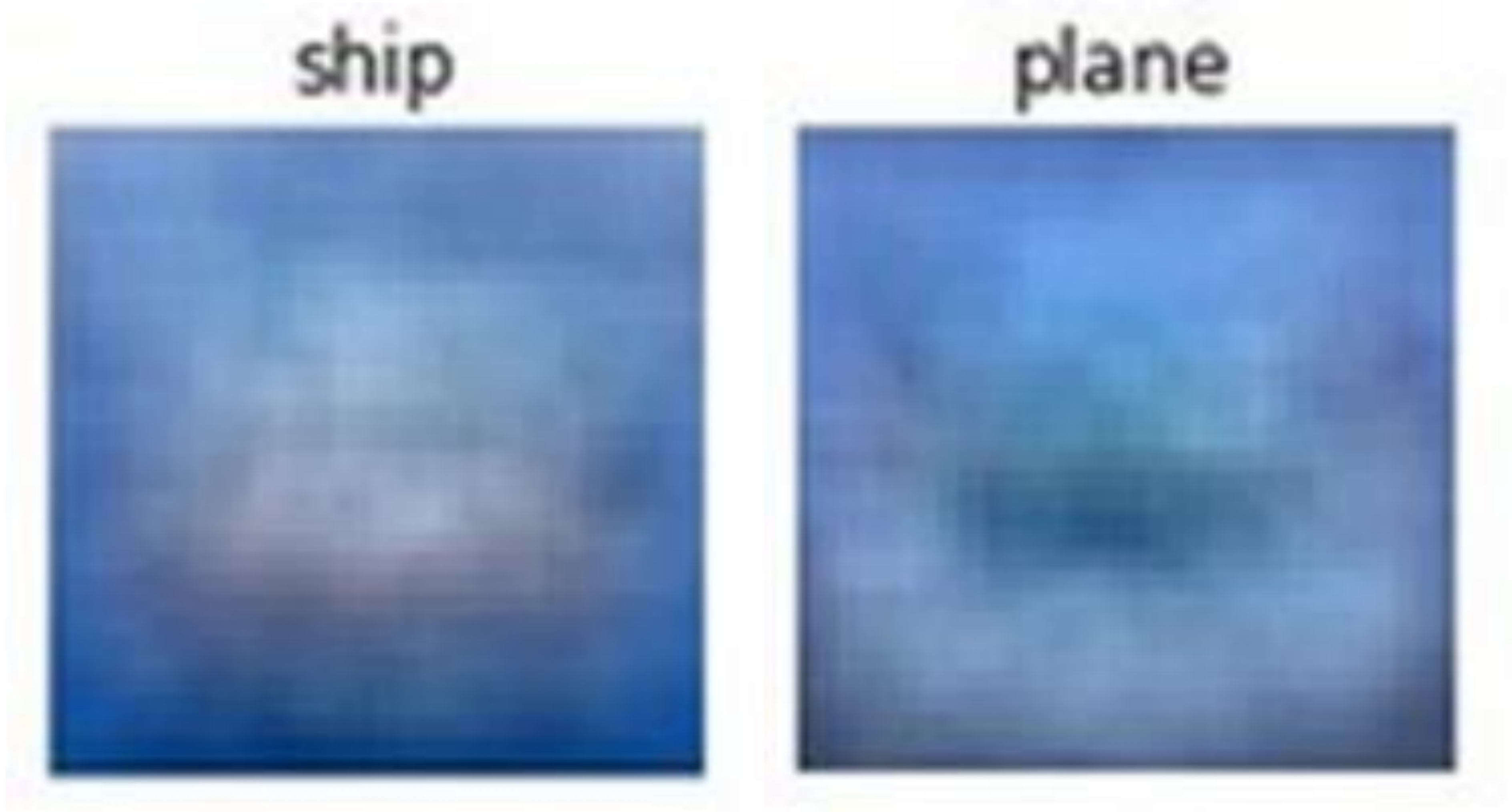
The Class Score

The classification score is then computed as the correlation between each input image and the «template» of the coresponding class

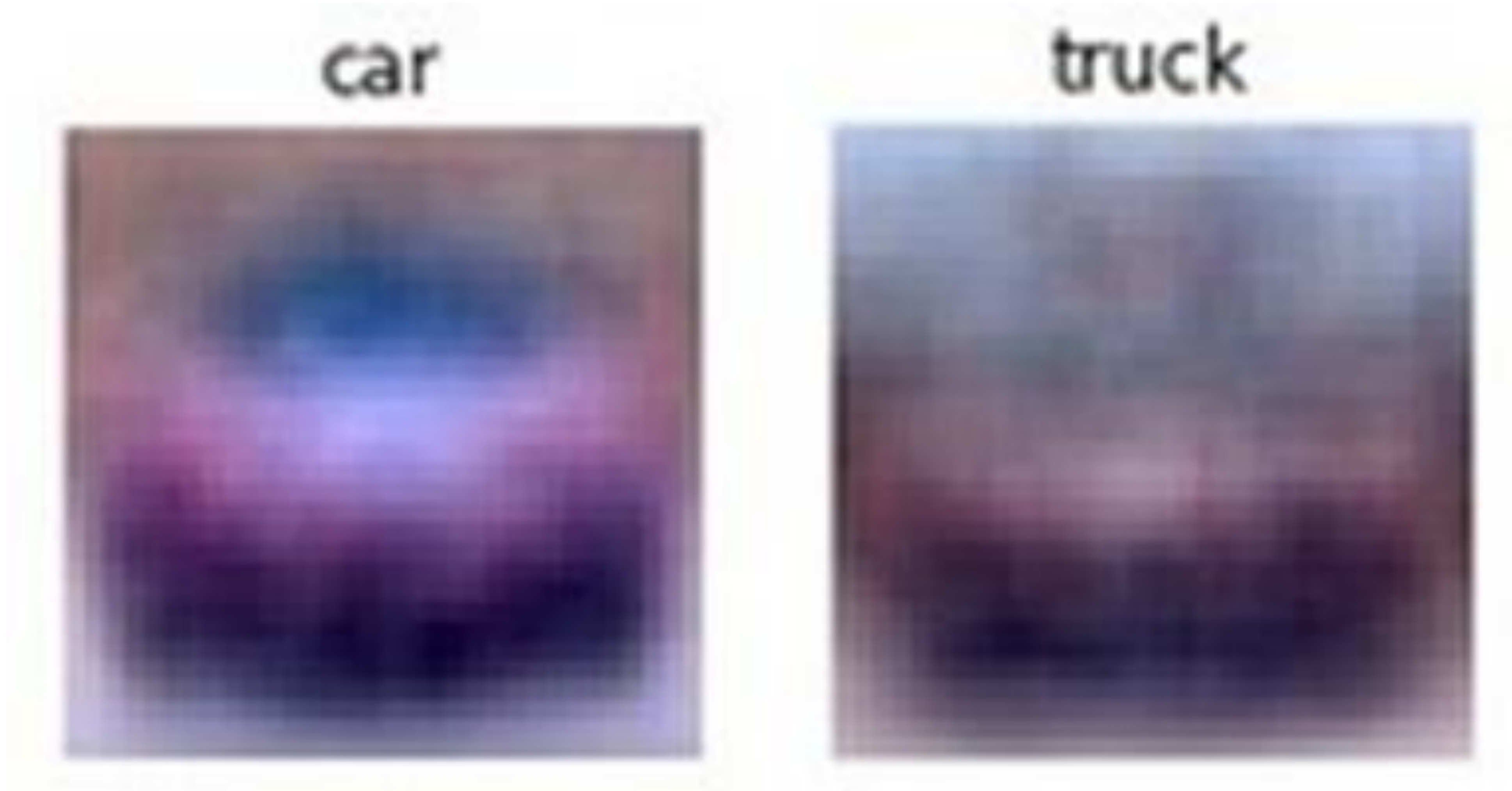


$$(I \otimes T_1)(0,0) = \sum_{(x,y) \in U} T_1(x,y) * I(x,y)$$

Templates Learned on the CIFAR-10 dataset



Templates Learned on the CIFAR-10 dataset



[CS231n: Convolutional Neural Networks for Visual Recognition](http://cs231n.github.io/) http://cs231n.github.io/

Templates Learned on the CIFAR-10 dataset

Templates Learned on the CIFAR-10 dataset

CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

Linear Classifier as a Template Matching

What has the classifier learned?

- That the background of bird and frog is green, (plane and boat is blue)
- Cars are typically red
- Horses have two heads! ☺

The model was definitively too simple / data were not enough for achieving higher performance and better templates

However:

- Linear Classifiers are among the most important layer of NN
- Such a simple model can be interpreted (with more sophisticated models you typically can't)

Linear Classifier as a Template Matching

What has the classifier learned?

- That the background of bird and frog is gr[illegible] and boat is blue)
- Cars are typically red
- Horses have two heads! ☺

The model was definitively [illegible] data were not enough for achieving higher performance and [illegible]lates

However:

- Linear Classifiers are [illegible]ng the most important layer of NN
- Such a simple model can be interpreted (with more sophisticated models you typically can't)

There should be a better way for handling images

Do it yourself!

https://colab.research.google.com/drive/1kflPH3CDgnvk1JptUoCbp2LK-owoh6R3?usp=sharing

Credits Eugenio Lomurno! (visualization with clipped colors)