Machine Learning
2a parte

Cesare Alippi, Professor

Università della Svizzera italiana, Lugano, Switzerland

COESI course

February 12th, 2020

The people you’’ll see

Cesare **Alippi**

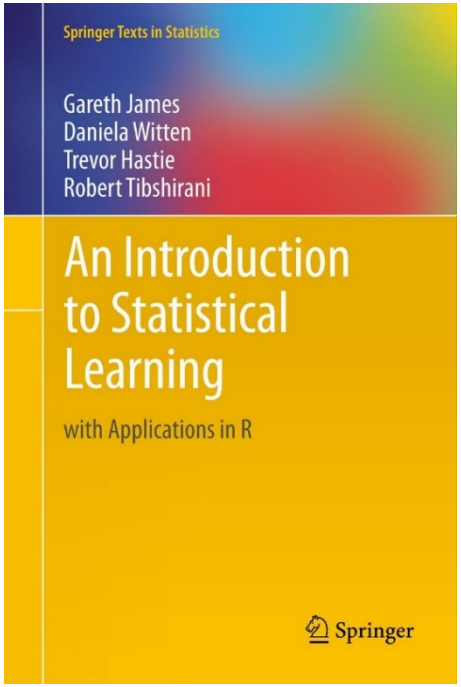
Andrea **Cini**

Mauro **Prevostini**

Reference books

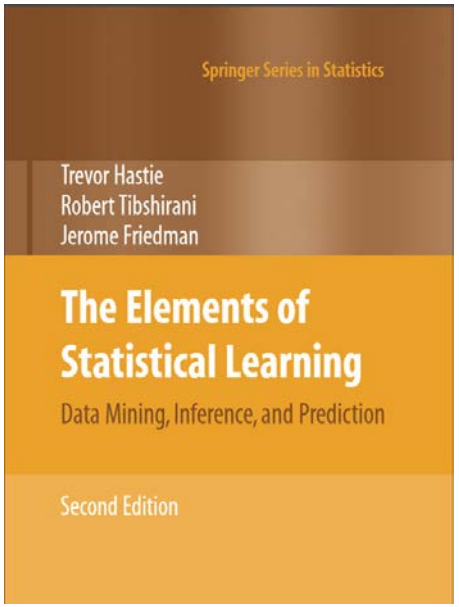
"An Introduction to statistical learning", James, Witten, Hastie, Tibshirani, Springer

https://www-bcf.usc.edu/~gareth/ISL/ISLR First Printing.pdf



- "The elements of statistical learning", Hastie, Tibshirani, Friedman, Springer

https://web.stanford.edu/~hastie/Papers/ESLII.pdf



Afternoon Lab

- Linear Regression
- Non-Linear Regression
- Classification
- Deep learning

What is machine learning

«Yes, it is a hot topic»

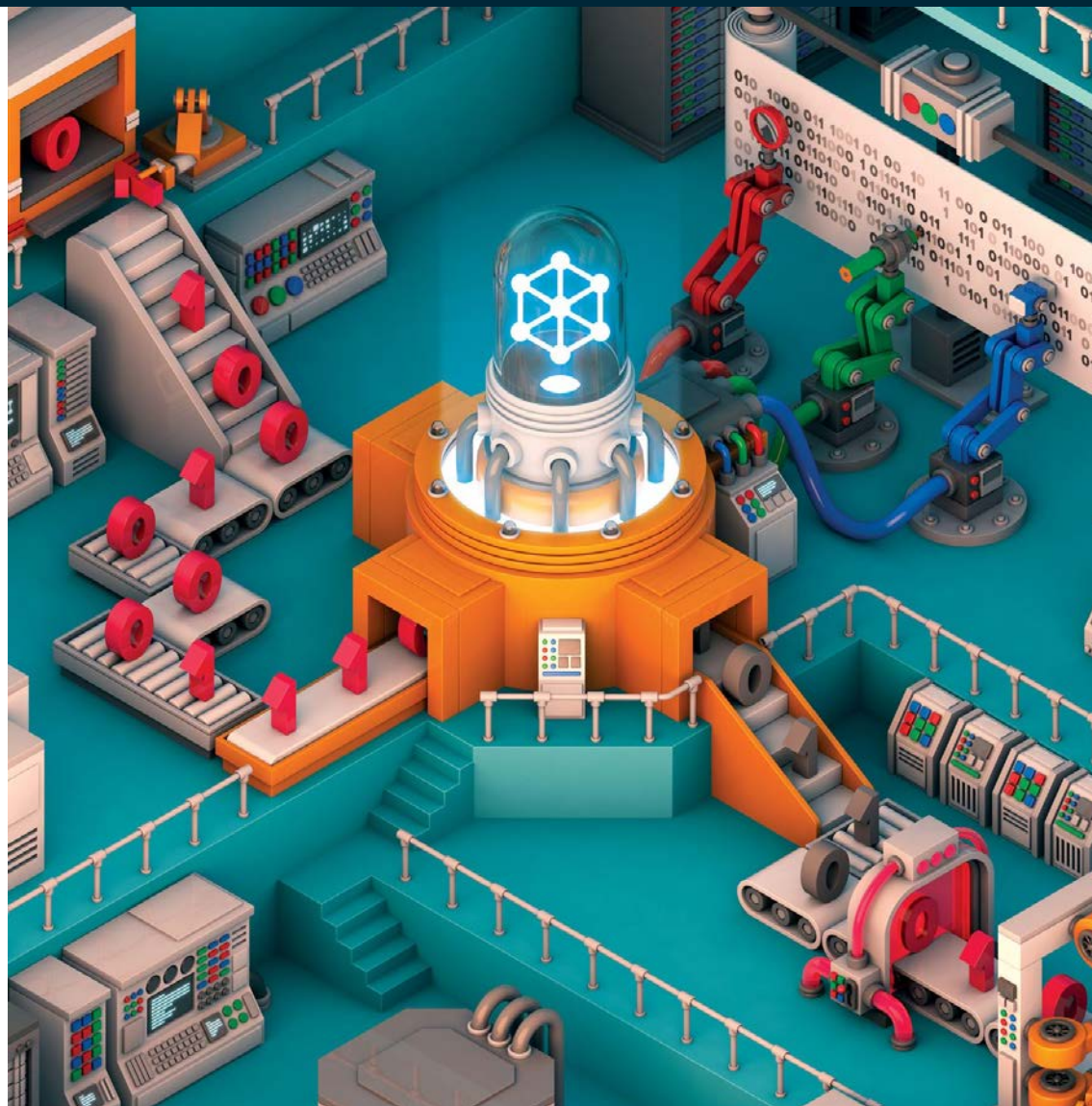
What does it mean “to learn”?

- Hastie, Tibshirani, Friedman:
  - “Vast amounts of data are being generated in many fields, and the statisticians’s job is to make sense of it all: to extract important patterns and trends, and to understand “what the data says”. We call this *learning from data.*”

- Mitchell:
  - “The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.”

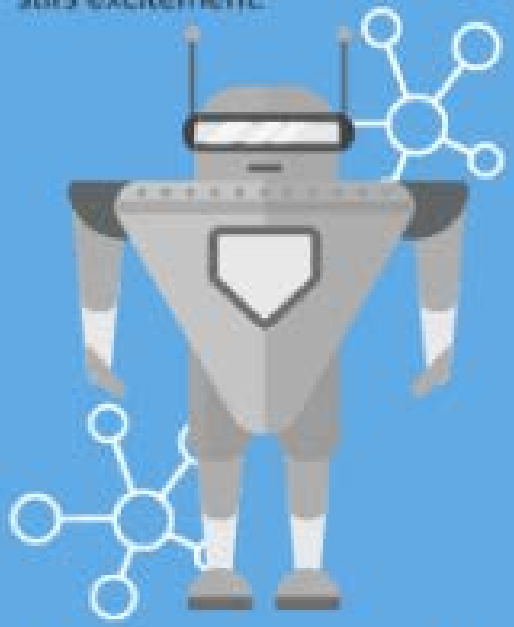
Mitchell formalization of the learning framework

A computer program is said to **learn** from **experience *E*** with respect to some class of **tasks *T*** and **performance measure *P***, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.



ARTIFICIAL INTELLIGENCE

Early artificial intelligence stirs excitement.

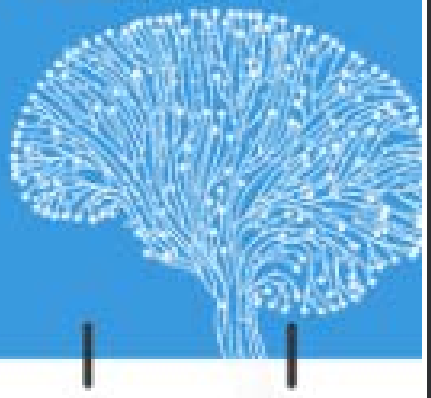


MACHINE LEARNING

Machine learning begins to flourish.

DEEP LEARNING

Deep learning breakthroughs drive AI boom.



1950's 1960's 1970's 1980's 1990's 2000's 2010's

Which classes of problems do we want to solve?

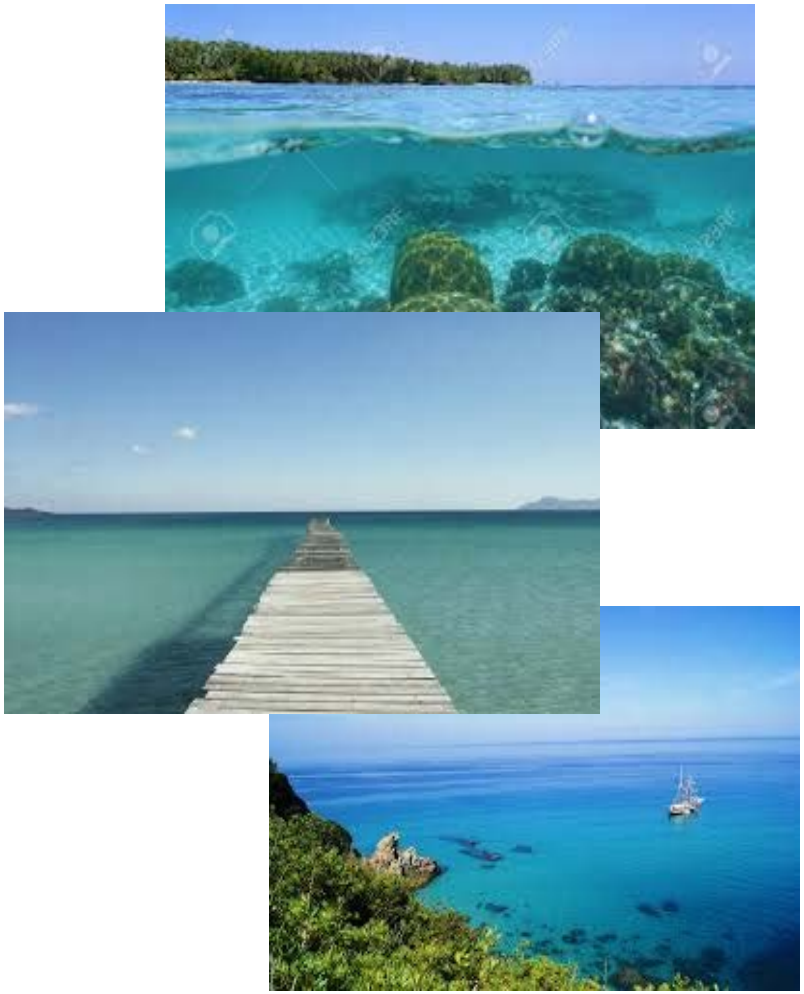
«Please show me some examples»

Classification

- **Associate a class to a given input**
  - **E.g., animal vs. human**

Classification

- **Sea vs. mountain**

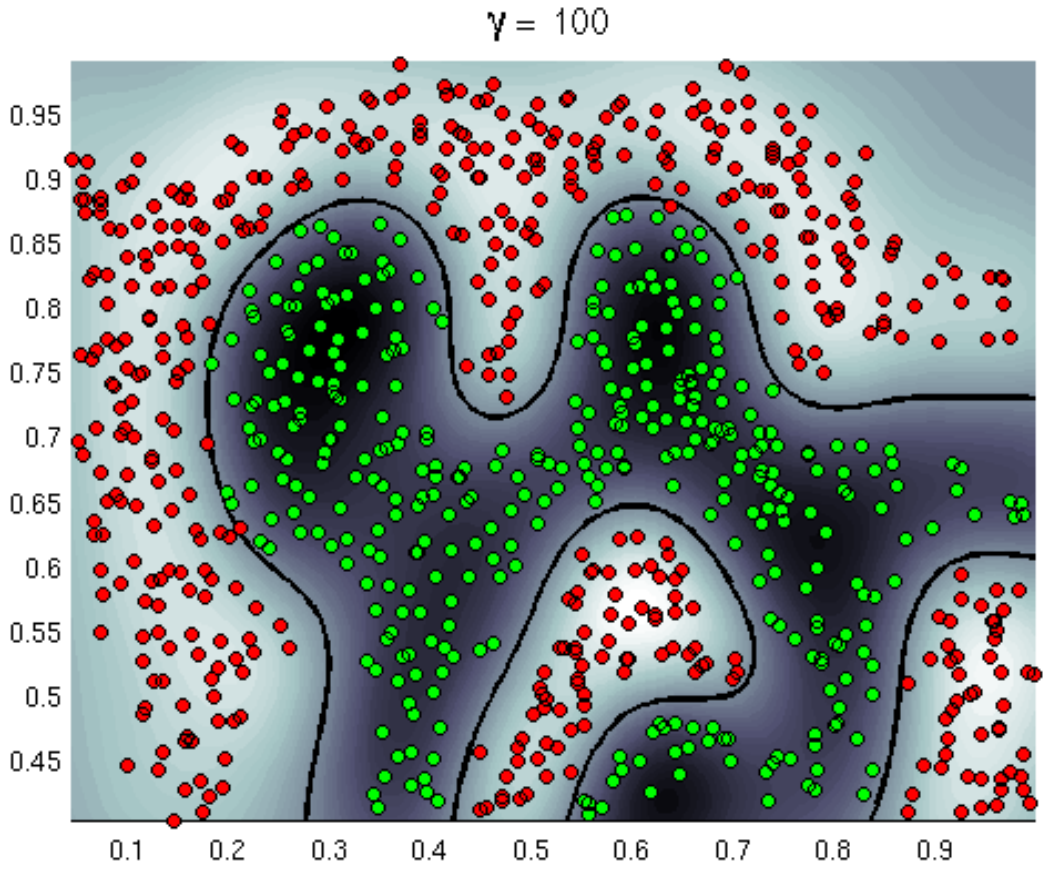


Classification

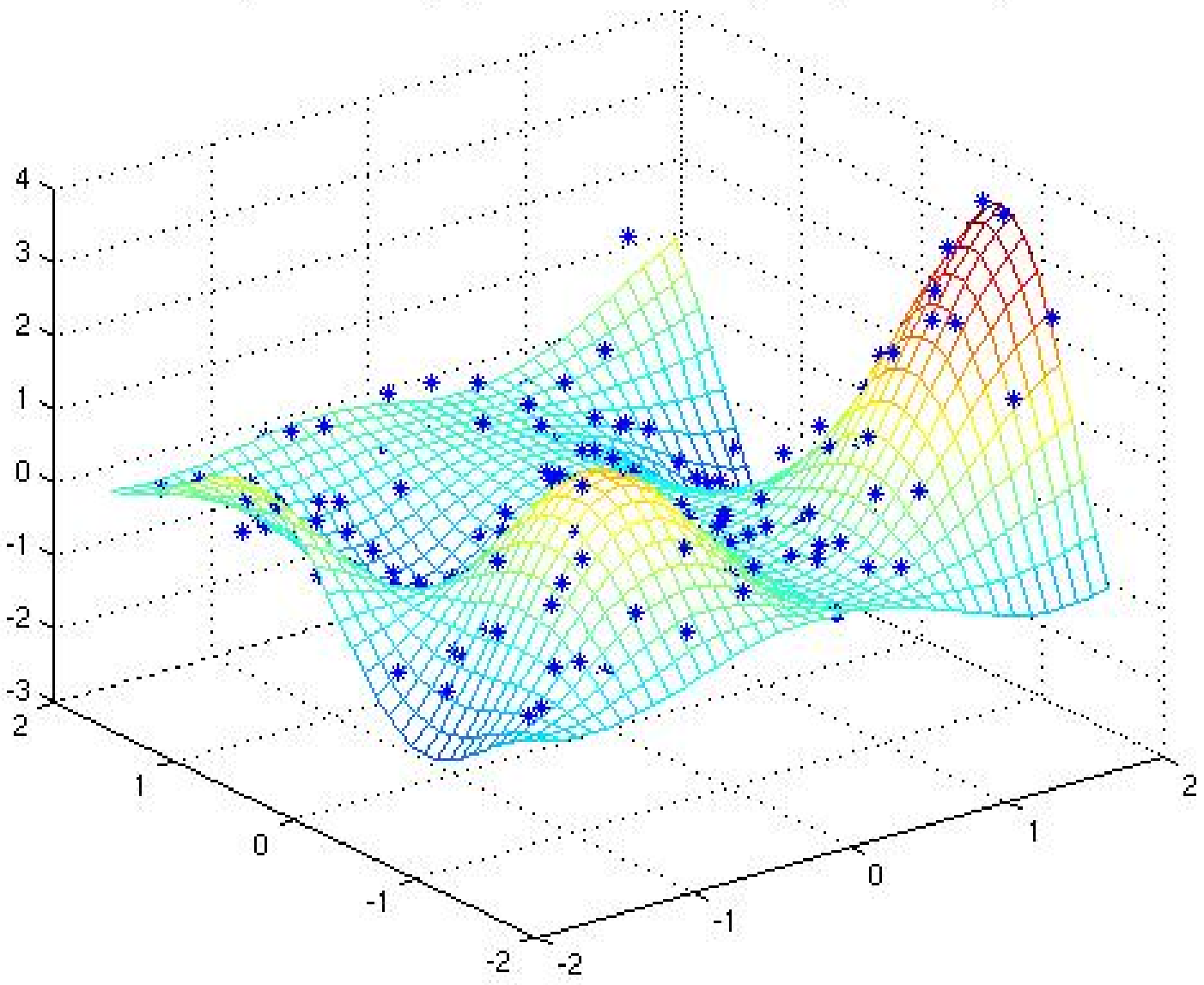
- **Multi-class classification**

Task: Classification

- **Final goal of classification: determine the function that partitions the inputs in classes**

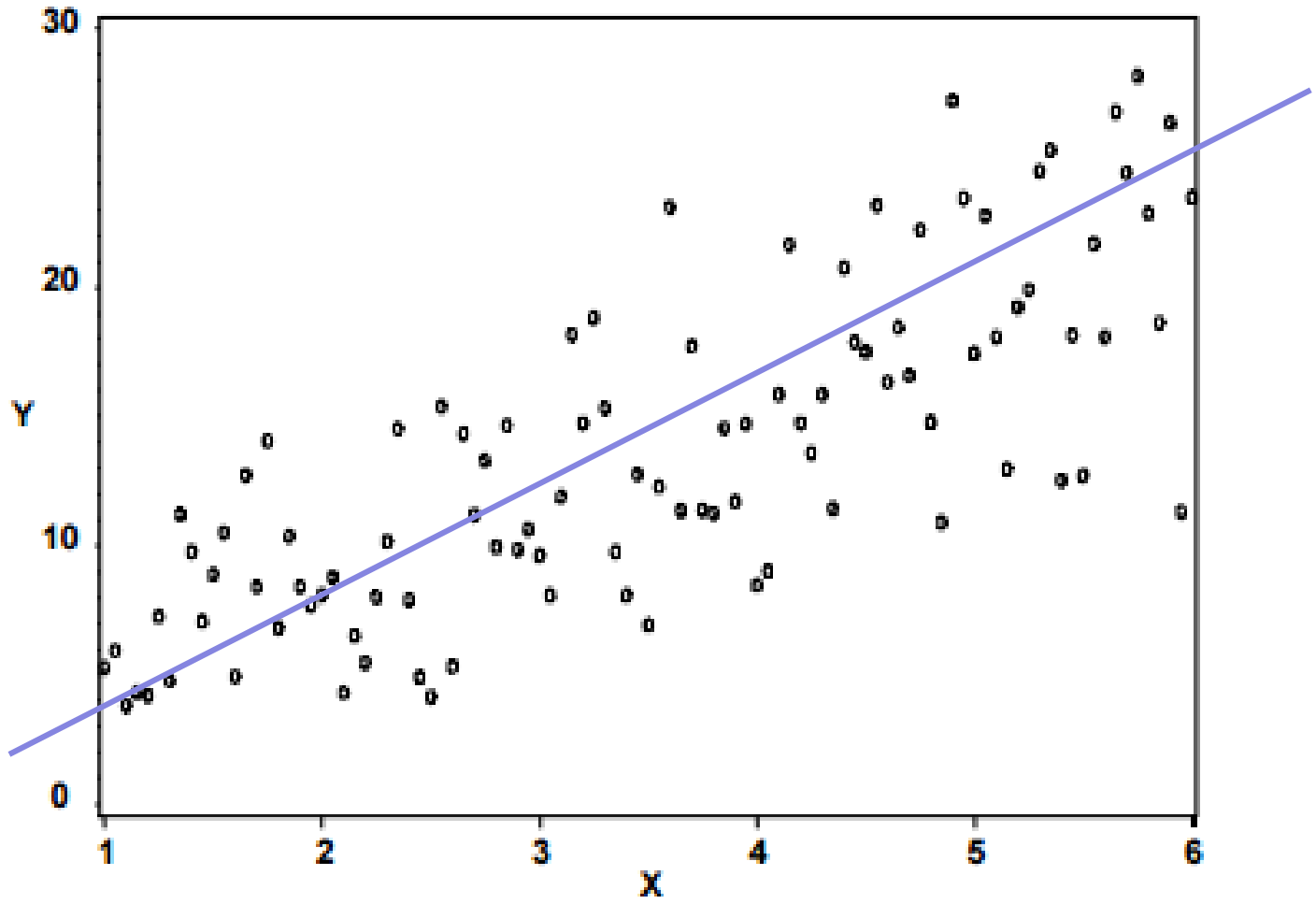


Regression



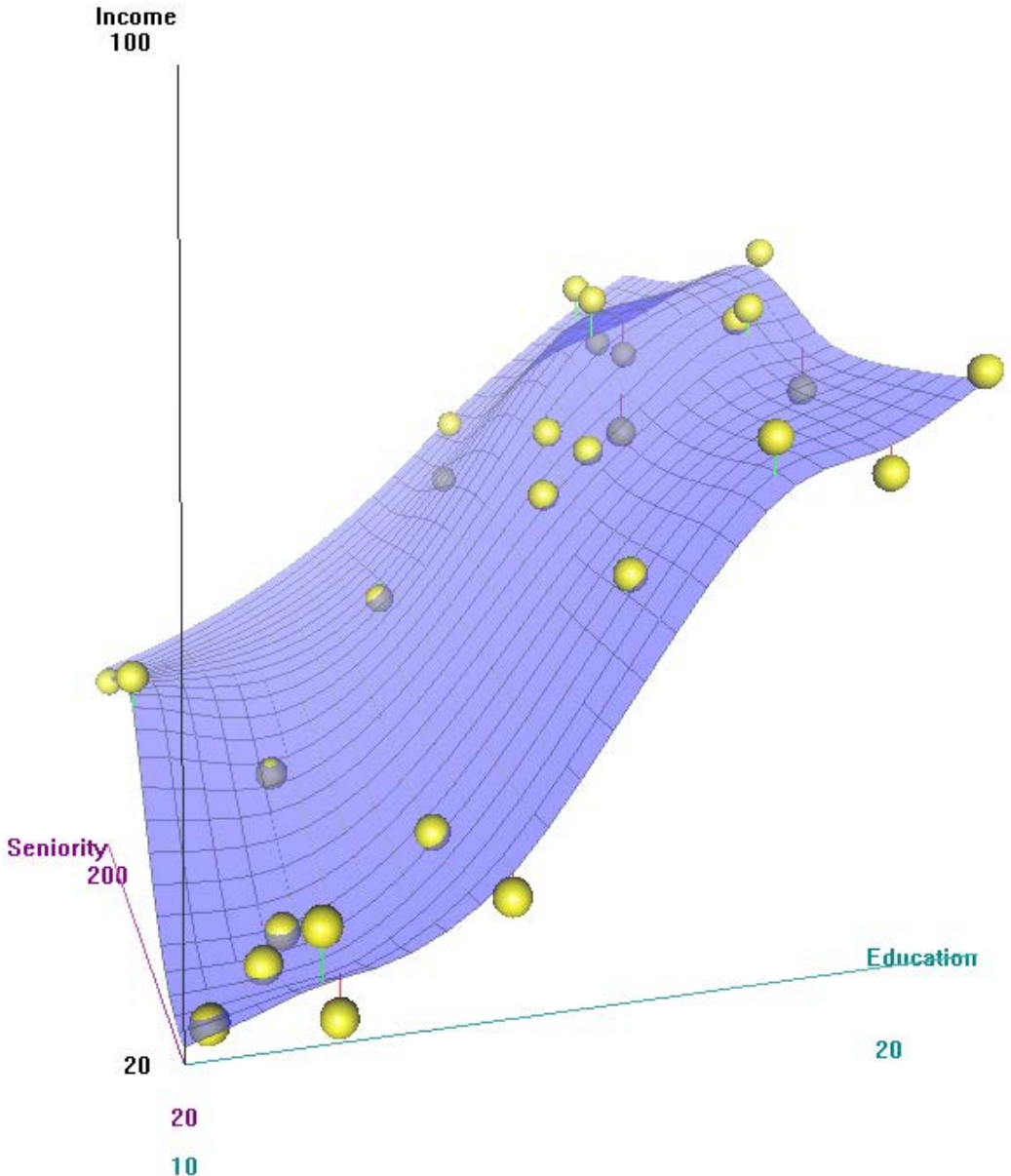
Some examples

- linear regression



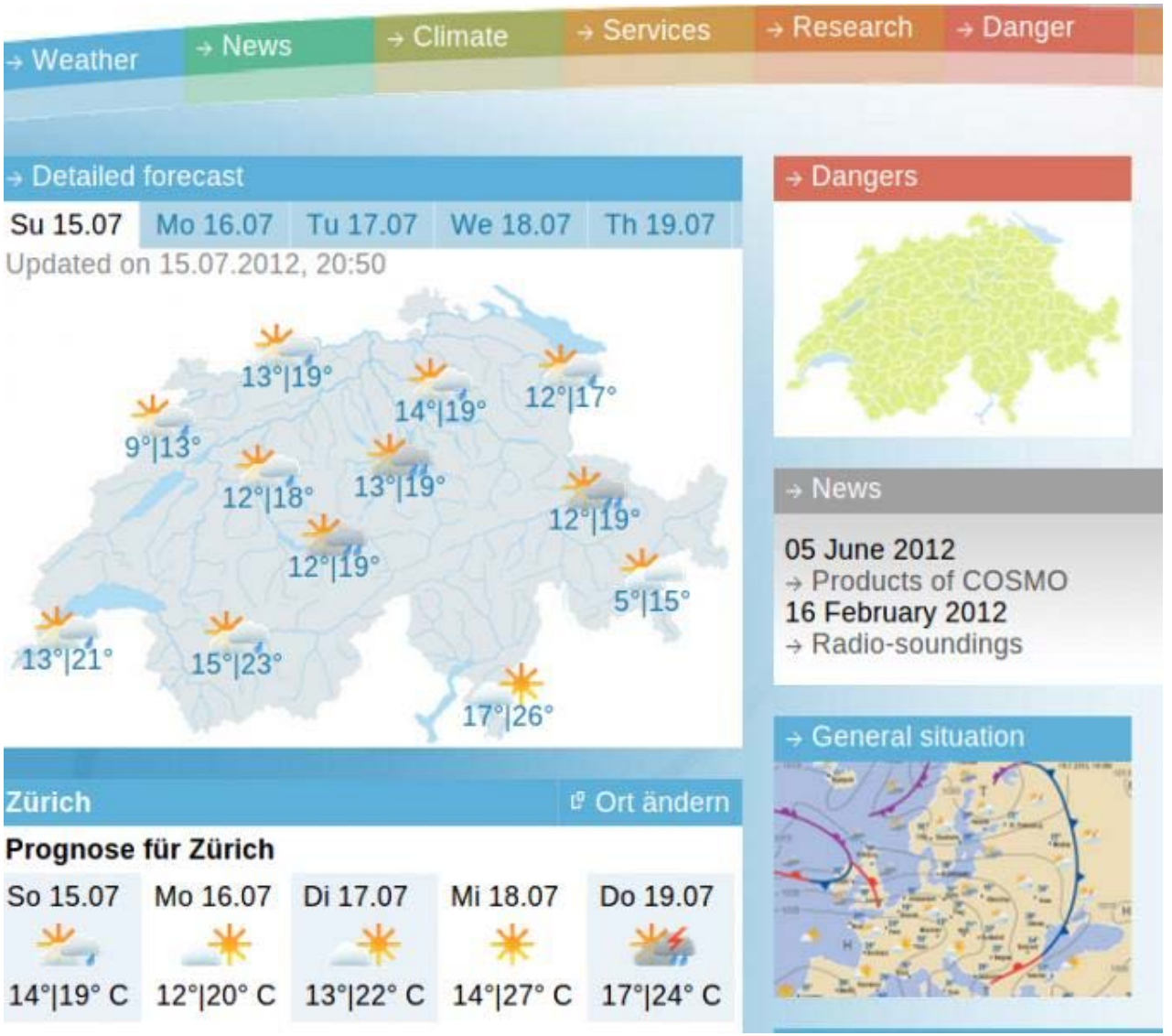
Task: Regression

- **Final goal of regression: determine the function that explains the given (input, output) instances**



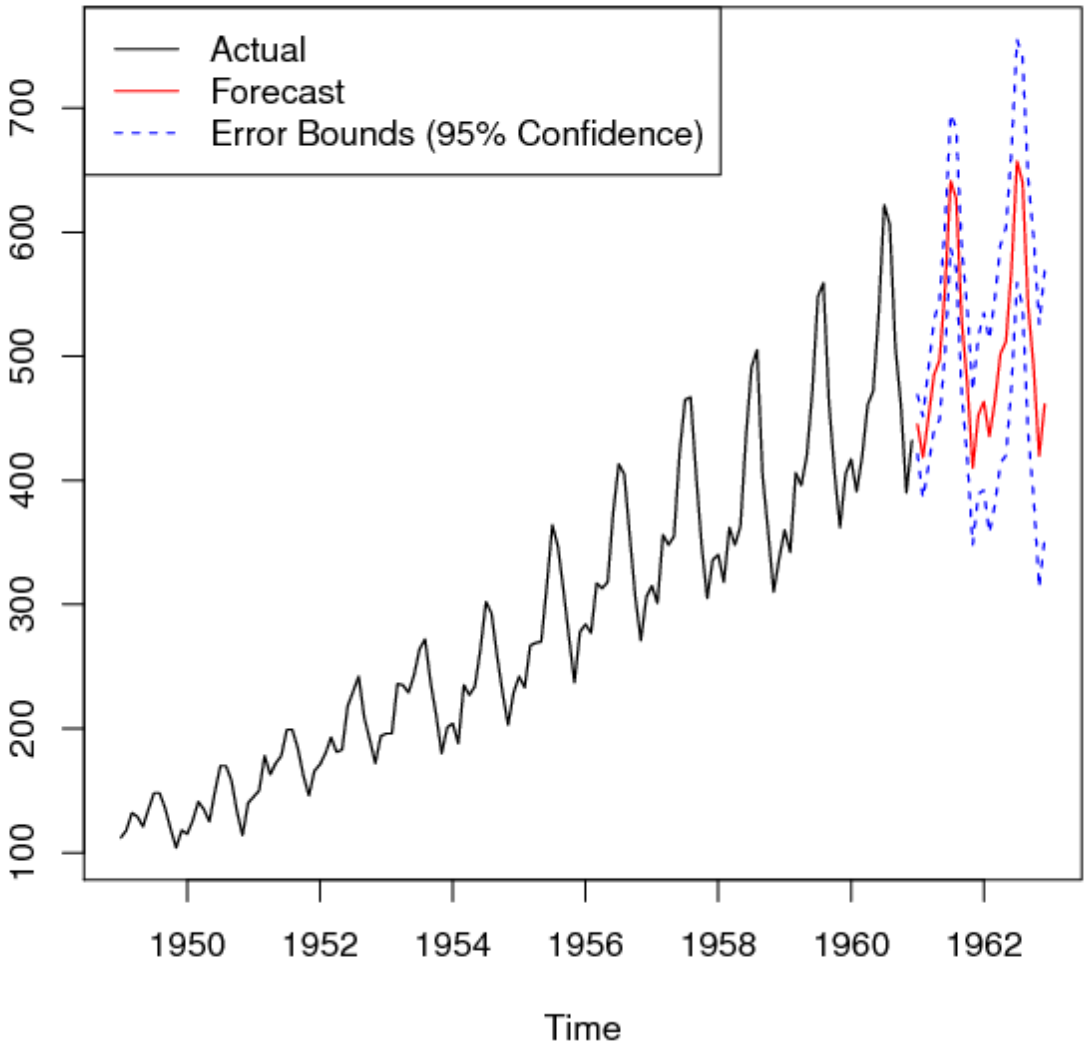
Prediction

- **What should we expect in the future?**
- **It complements/ completes the regression problem**



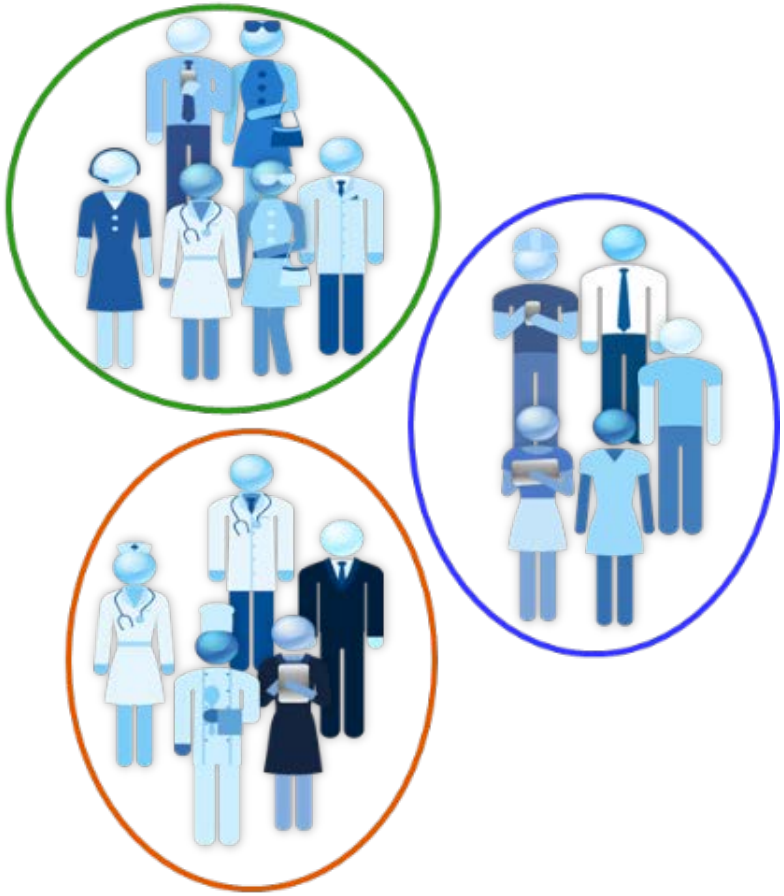
Task: Prediction

- **The final goal of prediction is to tell us which data will come next, possibly along with a confidence level**



Clustering

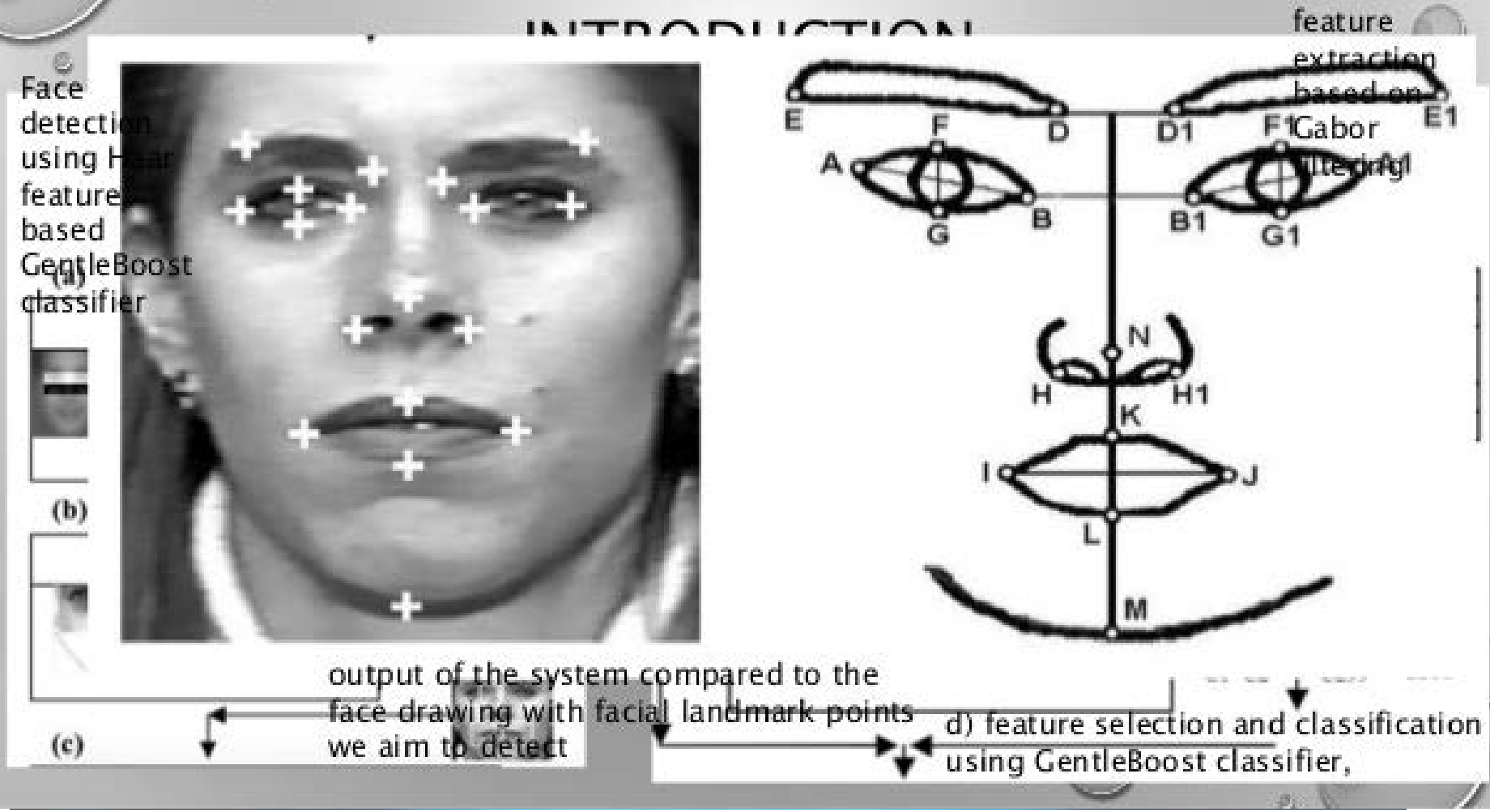
- **Its all about gathering similar instances**



From data to features

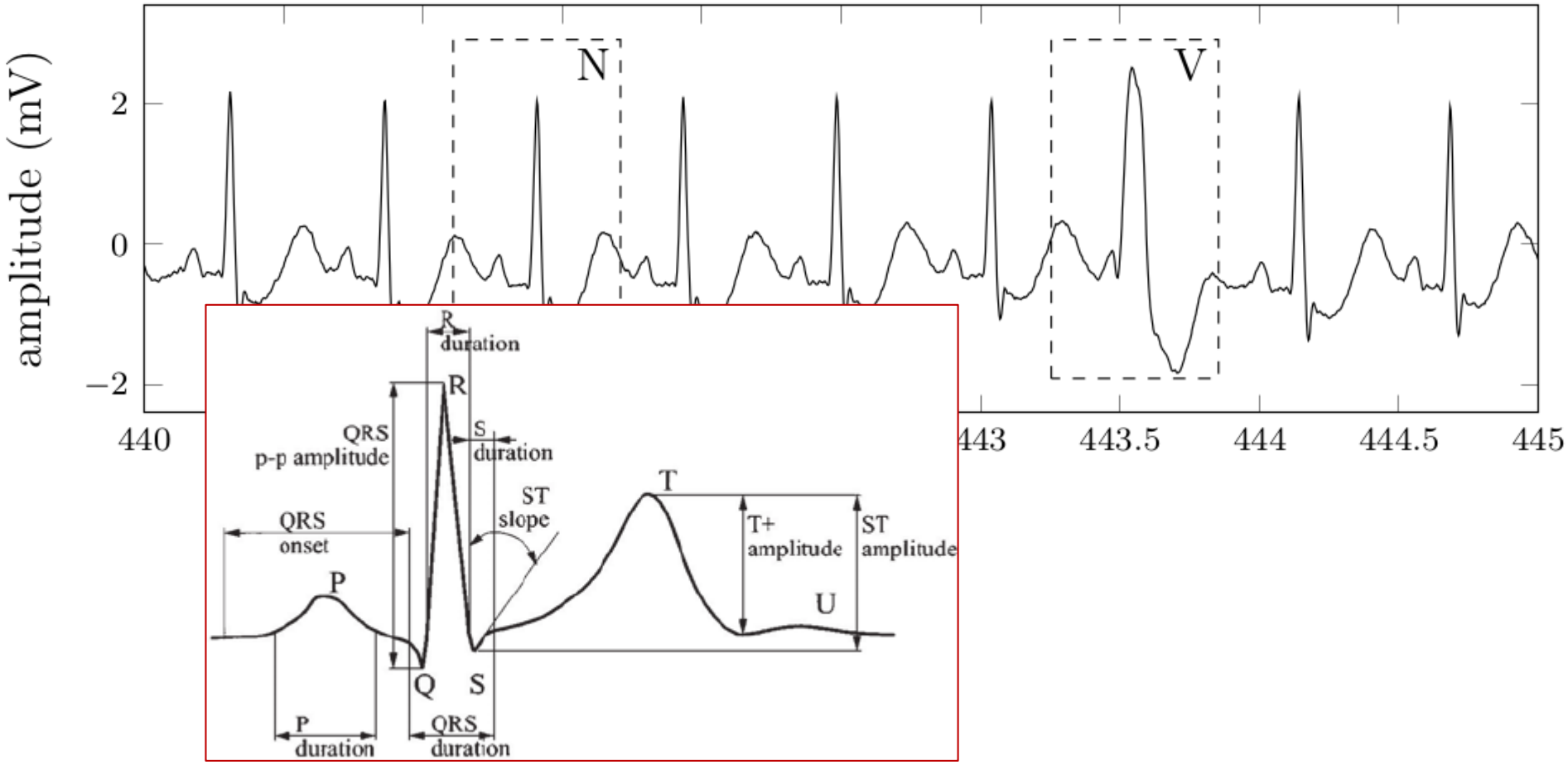
- In order to ease the learning task we might extract features from the data. Features
  - Provide a compact representation of inputs
  - Are particularly advantageous if we have prior information to take advantage of
  - Need to be reduced to a minimal set (feature selection) before feeding them into the inference engine

Features


Face detection using Haar feature based GentleBoost classifier
(a)
(b)
(c)
feature extraction based on Gabor filtering
E
F
D
D1
F1
E1
A
G
B
B1
G1
N
H
H1
K
I
J
L
M
output of the system compared to the face drawing with facial landmark points we aim to detect
d) feature selection and classification using GentleBoost classifier,

Features

- ElectroCardioGram

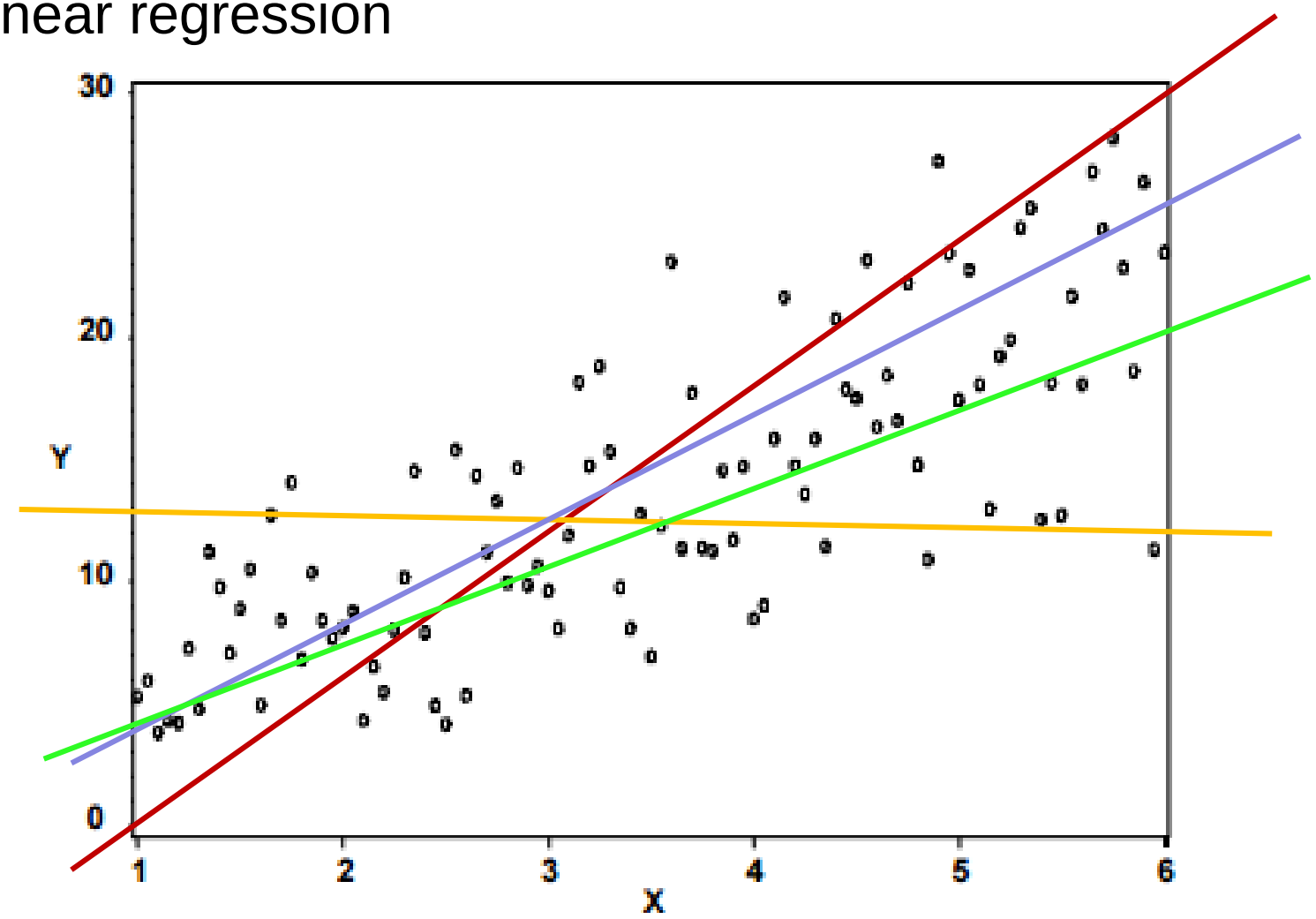


Linear regression

«Keep the model simple»

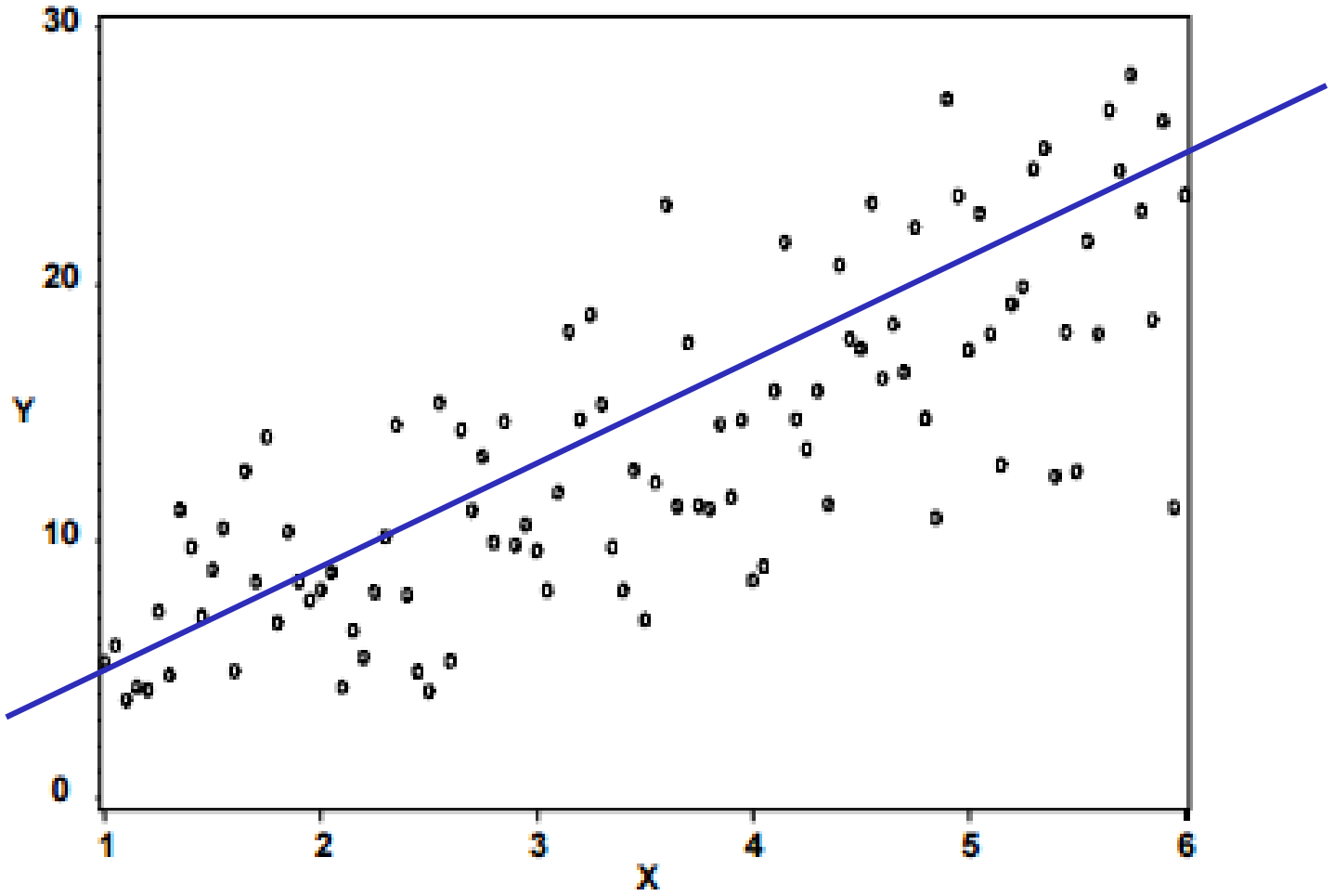
Some examples

- linear regression



Some examples

- linear regression: least mean square algorithm



Multiple Linear regression (as they taught you)

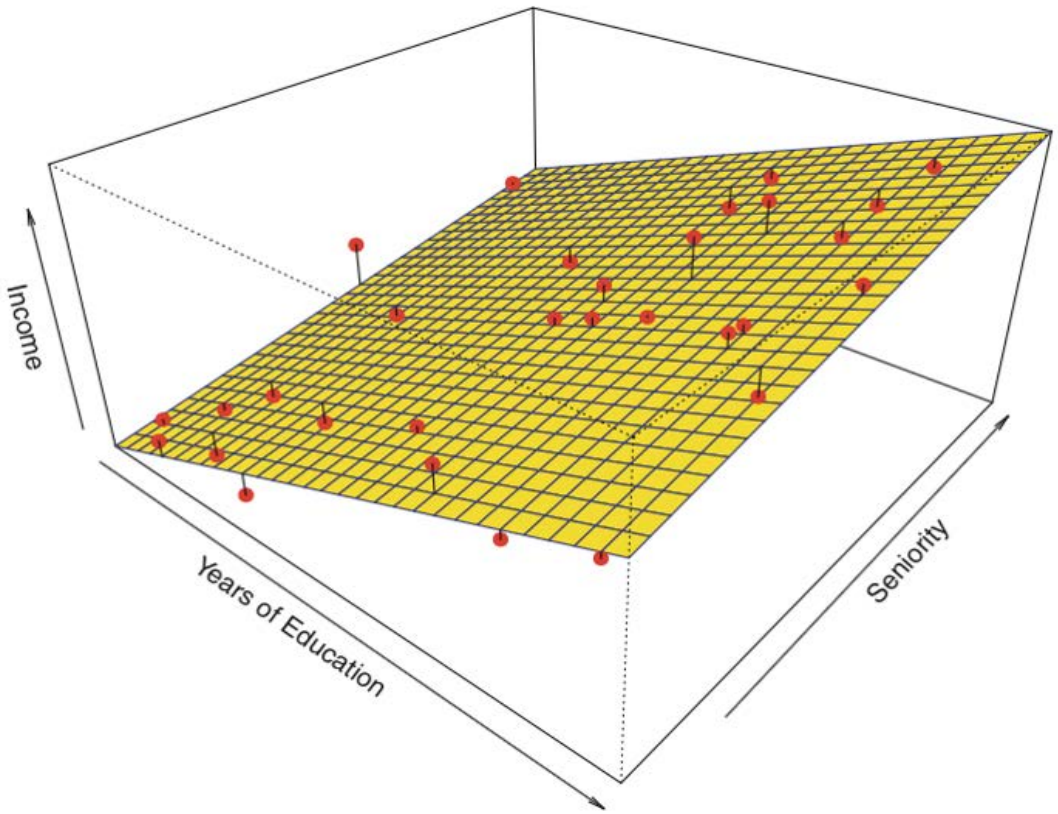
A set of n data couples (training set)

$$\{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$$

is given

$$x \in \mathbb{R}^d, y \in \mathbb{R}$$

x is column vector



Multiple linear regression

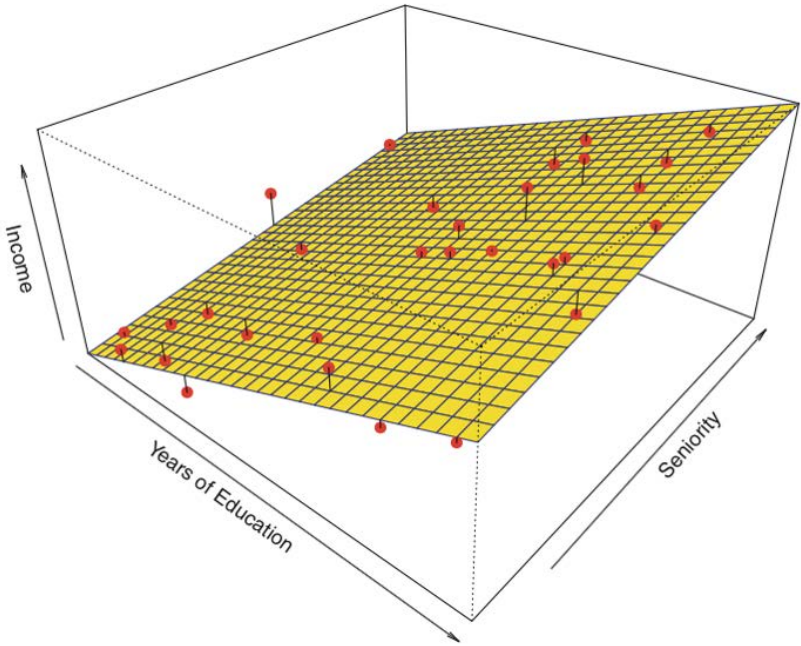
Assume that the unknown function that generates the data is linear and that there is a gaussian uncertainty affecting measurements in an additive way

$$y(x) = \theta_1^o + \theta_2^o x_2 + \cdots \theta_d^o x_d + \eta$$

Canonical form for system model

$$y(x) = x^T \theta^o + \eta$$

$$\theta^o \in \mathbb{R}^d \qquad \eta = N(0, \sigma_\eta^2)$$



Optimal parameters, grouped in a column vector, are unknown, as it is the variance of noise

Multiple linear regression

We know that the unknown system model is linear. Which family of models should we consider to best fit the available data?

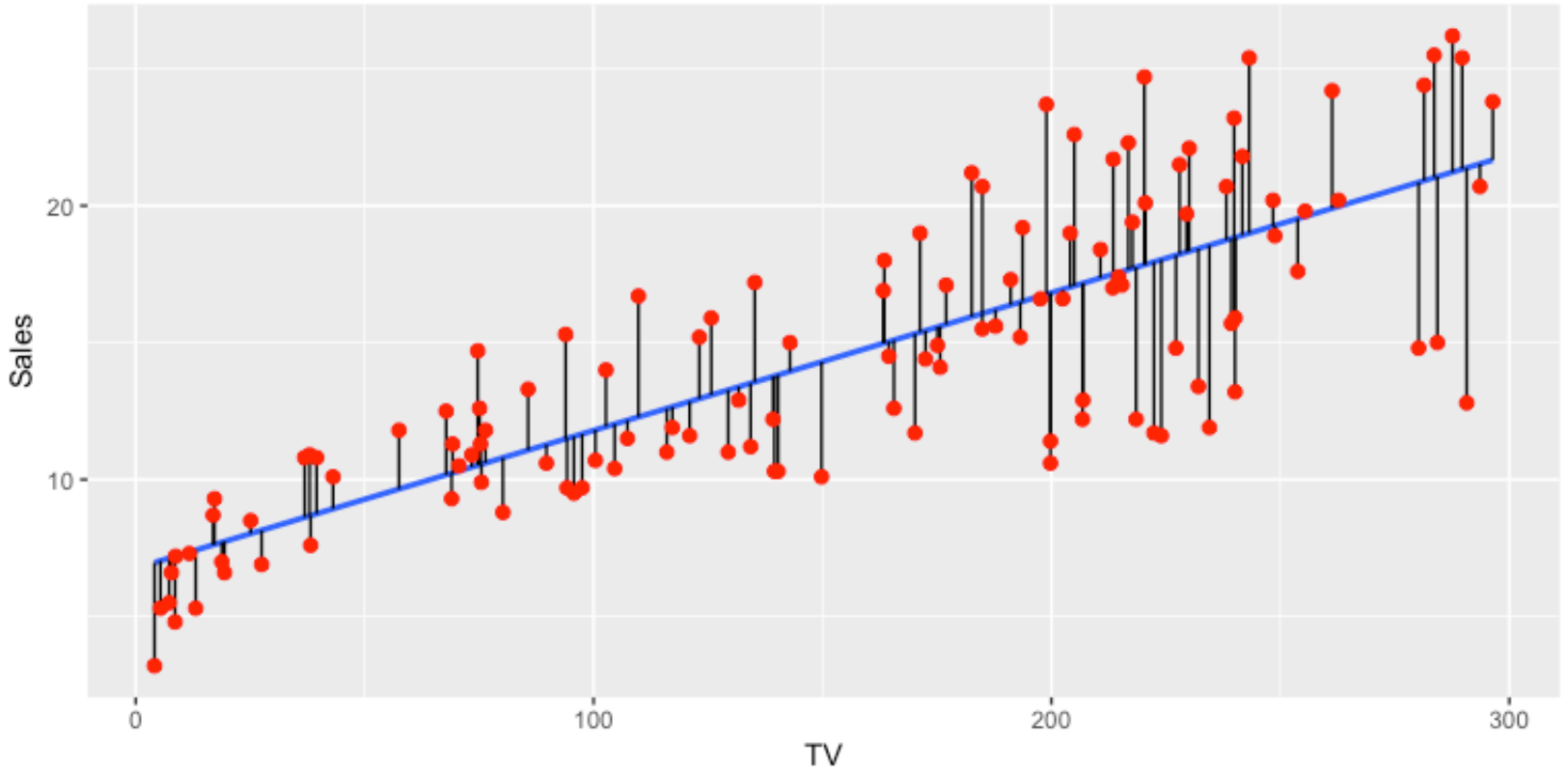
$$\hat{y}(x) = f(\theta, x) = x^T \theta$$

How to determine the best estimated parameter $\hat{\theta}$ so that we generate the model

$$f(\hat{\theta}, x) = x^T \hat{\theta}$$

approximating unknow function $x^T \theta^o$?

Multiple linear regression



Idea: select the linear function that minimizes the average distance between given points and the linear function: we obtain the **Least Mean Square –LMS-** procedure

Multiple linear regression

Performance function

$$V_n(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y(x_i) - f(\theta, x_i)\right)^2$$

The parameter vector to be chosen is

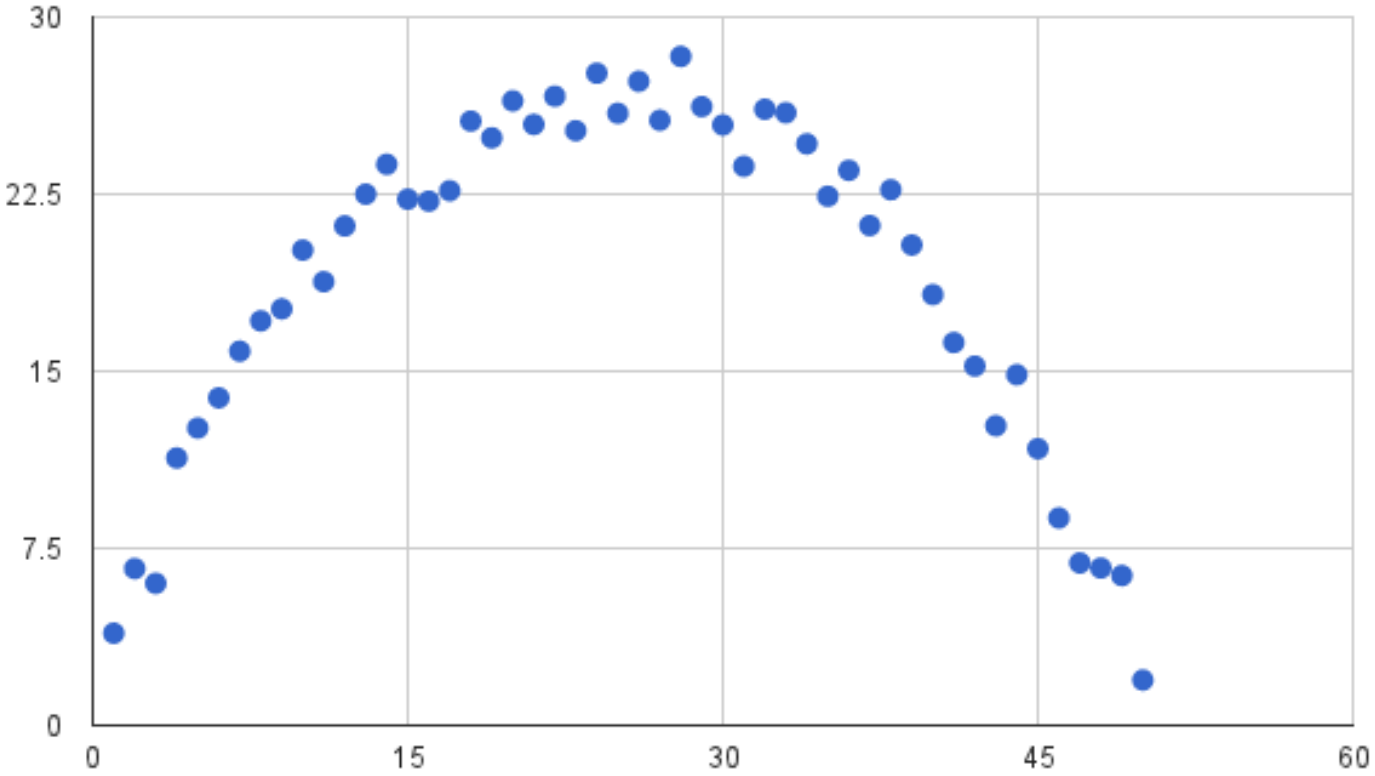
$$\hat{\theta} = \arg\min_{\theta \in \Theta} V_n(\theta)$$

Linearity must be intended according to the parameters!!!

- Linear regression

$$y = a + bz + cz^2 + dz^3$$

$$\theta = [a, b, c, d]; x = [1, z, z^2, z^3]$$



Multiple linear regression: how to estimate parameters

By grouping data as

$$Y=\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_n \end{bmatrix} \qquad X=\begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \\ \dots \\ x_n^T \end{bmatrix}$$

We can rewrite the loss function in a canonical form

$$V_n^*(\theta) = \sum_{i=1}^{n} \left(y(x_i) - x_i^T \theta\right)^2 = (Y - X\theta)^T (Y - X\theta)$$

Multiple linear regression

Stationary points are those for which

$$\frac{\partial V_n^*(\theta)}{\partial \theta} = -2X^T Y + 2X^T X\theta = 0$$

Therefore, the parameter vector minimizing the performance function is

$$\hat{\theta} = \left(X^T X\right)^{-1} X^T Y$$

and the best approximating model is

$$f(\hat{\theta}, x) = x^T \hat{\theta}$$

Multiple linear regression

- A computer program is said to **learn** from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.
- Let's list the different actors

- Task *T*: regression
- Experience *E*: $\{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$
- Performance measure *P*: Mean square error
- Performance at task: $V_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y(x_i) - f(\theta, x_i))^2$

Performance at task: mmh….

- In a typical classroom the teacher provides solution examples/instances related to a concept, e.g., what a cat is.



- However, at exam time, the problems the teacher provides you to test your understanding are not identical to the ones you saw during the course. E.g.,


Performance at target: mmh….

- Instead, we propose different instances



- Why?
- Performance at task assessed according to

$$V_n(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y(x_i) - f(\theta, x_i)\right)^2$$

is biased to the training set

How to measure performance at task?

- For now, based on our exam experience, we consider another data set, called test set,

$$\{(\bar{x}_1, \bar{y}_1), (\bar{x}_2, \bar{y}_2), \cdots, (\bar{x}_l, \bar{y}_l)\}$$

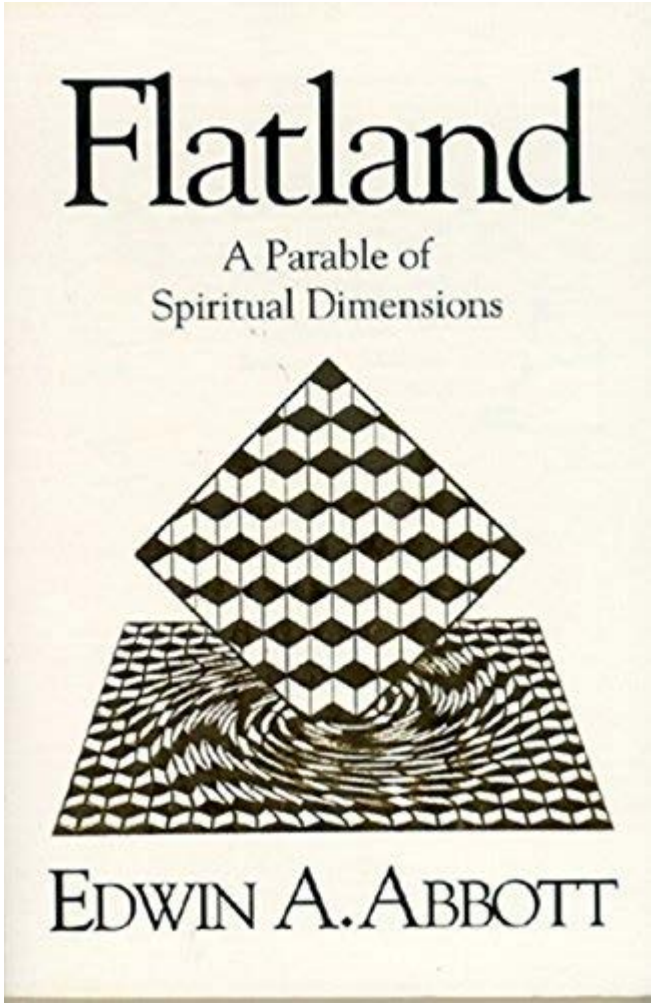
- and evaluate performance on it

$$V_l(\hat{\theta}) = \frac{1}{l} \sum_{i=1}^{l} (\bar{y}_i - \bar{x}^T \hat{\theta})^2$$

The procedure is named cross-validation

Nonlinear regression

«I like straight lines but sometimes curves are appreciable»



Non-linear regression: formalization

The *stationary* process generating the data

$$y = g(x) + \eta$$

provides, given input x_i output

$$y_i = g(x_i) + \eta_i$$

Collect a set of couples (training set)

$$Z_N = \{(x_1, y_1), \ldots, (x_N, y_N)\}$$

The goal of supervised learning is to design the simplest approximating model able to explain past Z_N data and future instances provided by the data generating process.

Non-linear regression: formalization

Model unknown function $g(x)$

with parameterized family of models $f(\theta, x)$

Consider loss function

$$L(y(x), f(\theta, x))$$

e.g.,

$$L(y(x), f(\theta, x)) = (y(x) - f(\theta, x))^2$$

The traditional statistical approach

The structural risk

$$\bar{V}(\theta) = \int L(y, f(\theta, x))\, p_{x,y} dxy$$

$$\theta^o = \arg\min_{\theta \in \Theta} \bar{V}(\theta)$$

The empirical risk

$$V_N(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(\theta, x_i))$$

$$\hat{\theta} = \arg\min_{\theta \in \Theta} V_N(\theta)$$

Non-linear regression: formalization

Determine the parameter estimate through a minimization problem

$$\hat{\theta} = \arg\min_{\theta \in \Theta} V_N(\theta)$$

carried out by a learning procedure

$$\theta_{i+1} = \theta_i - \varepsilon_L \frac{\partial V_N(\theta)}{\partial \theta}\Big|_{\theta_i}$$

and get model

$$f(\hat{\theta}, x)$$

Comments

The risk associated with the model (generalization ability, performance at task) can be decomposed in three terms

$$\bar{V}(\hat{\theta}) = \left(\bar{V}(\hat{\theta}) - \bar{V}(\theta^o)\right) + \left(\bar{V}(\theta^o) - V_I\right) + V_I$$

Comments

The inherent risk

$$V_I$$

depends only on the structure of the learning problem. This term can be improved only by improving the problem itself (e.g., by reducing the instrumentation noise)

Comments

The approximation risk

$$\bar{V}(\theta^o) - V_I$$

depends only on how close the approximating model family is to the process generating the data. As such, a more appropriate model family reduces the risk

Comments

The estimation risk

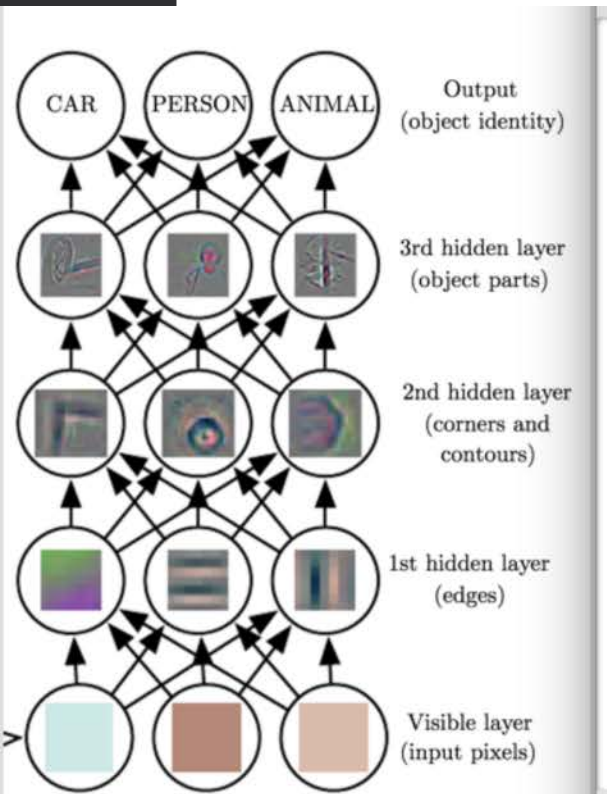
$$\bar{V}(\hat{\theta}) - \bar{V}(\theta^o)$$

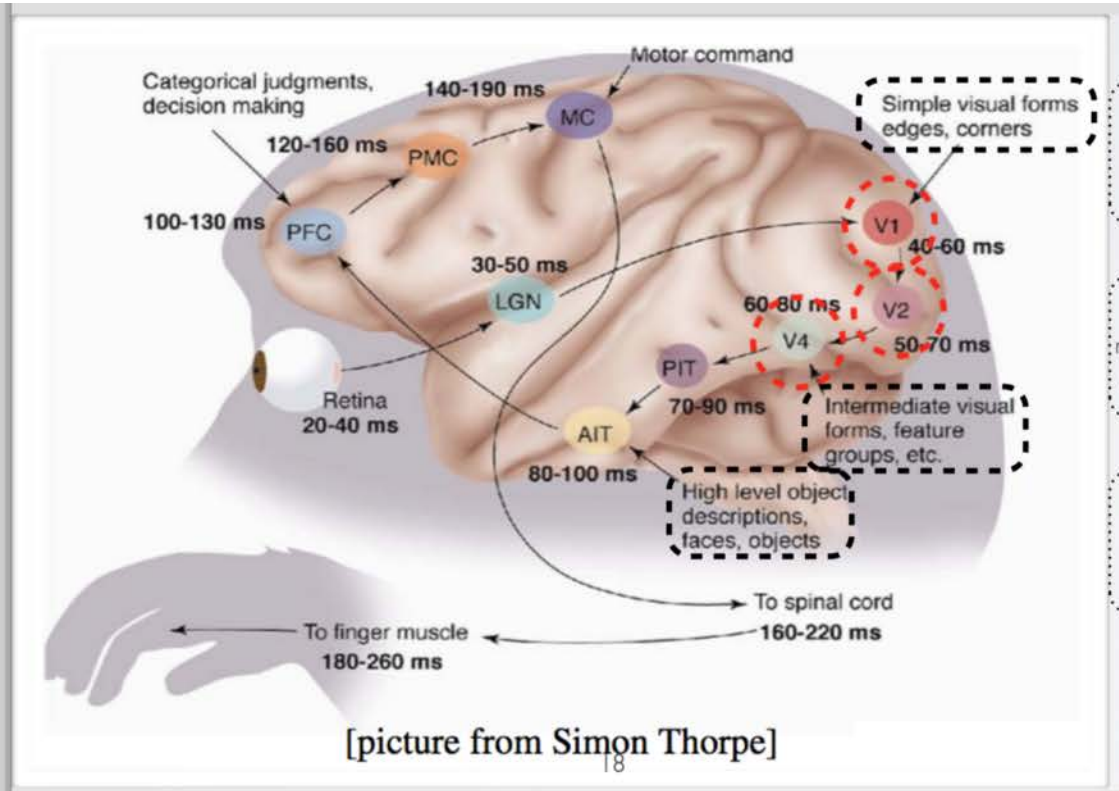
depends on the effectiveness of the learning procedure. As such, a more effective learning algorithm reduces the risk

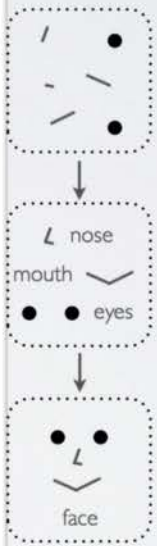
All consistent learning procedures are equally effective

Feed-forward neural networks

«An interesting computational paradigm»

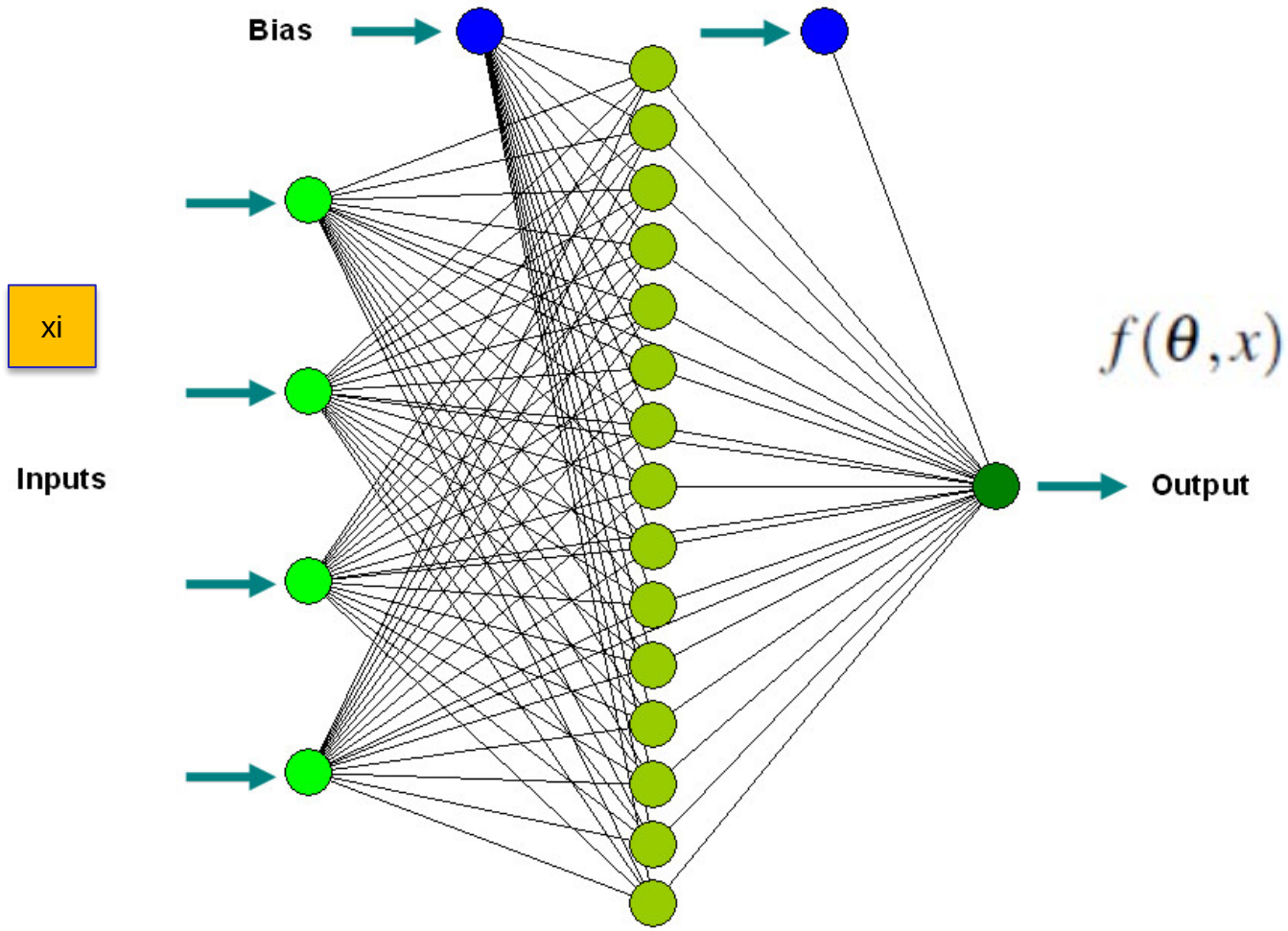




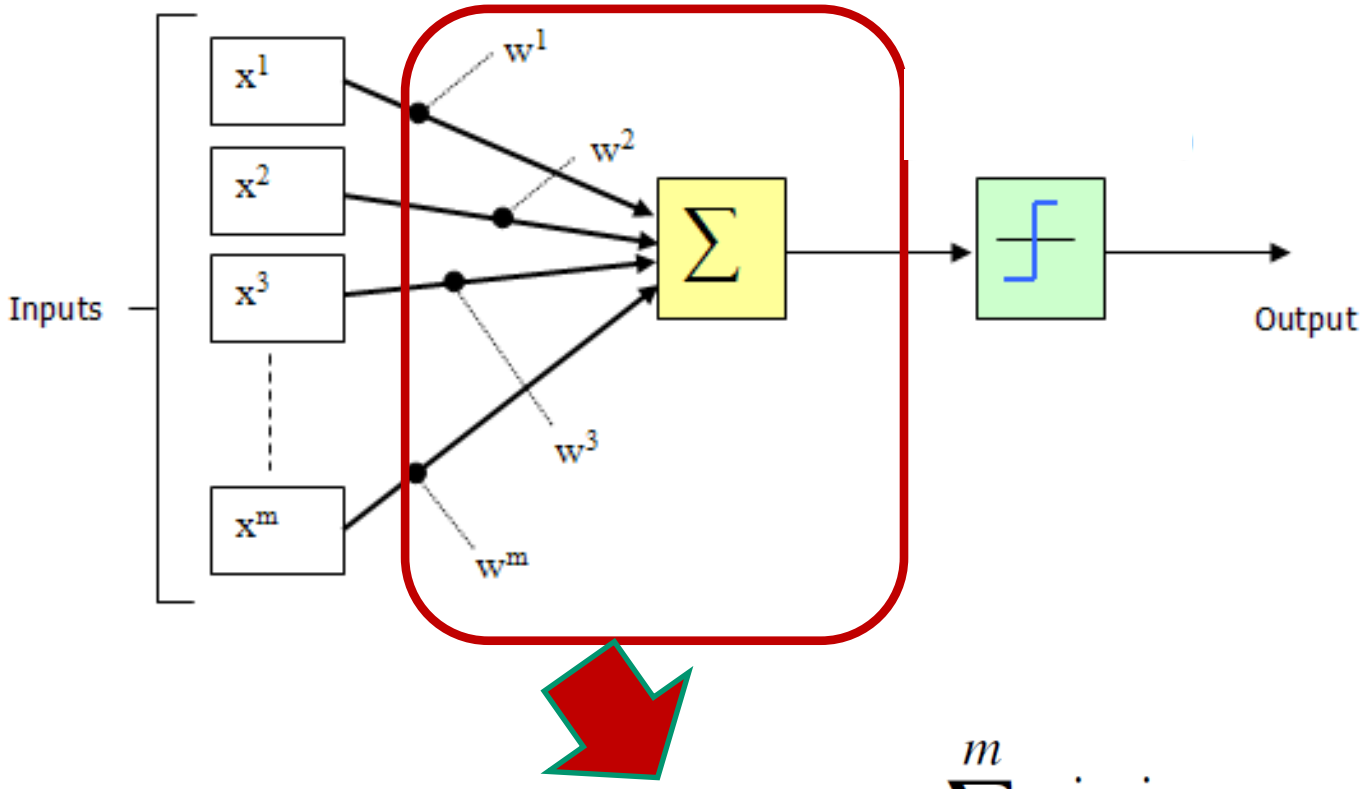


Feedforward Neural Networks

Not rarely $f(\theta, x)$ is chosen as

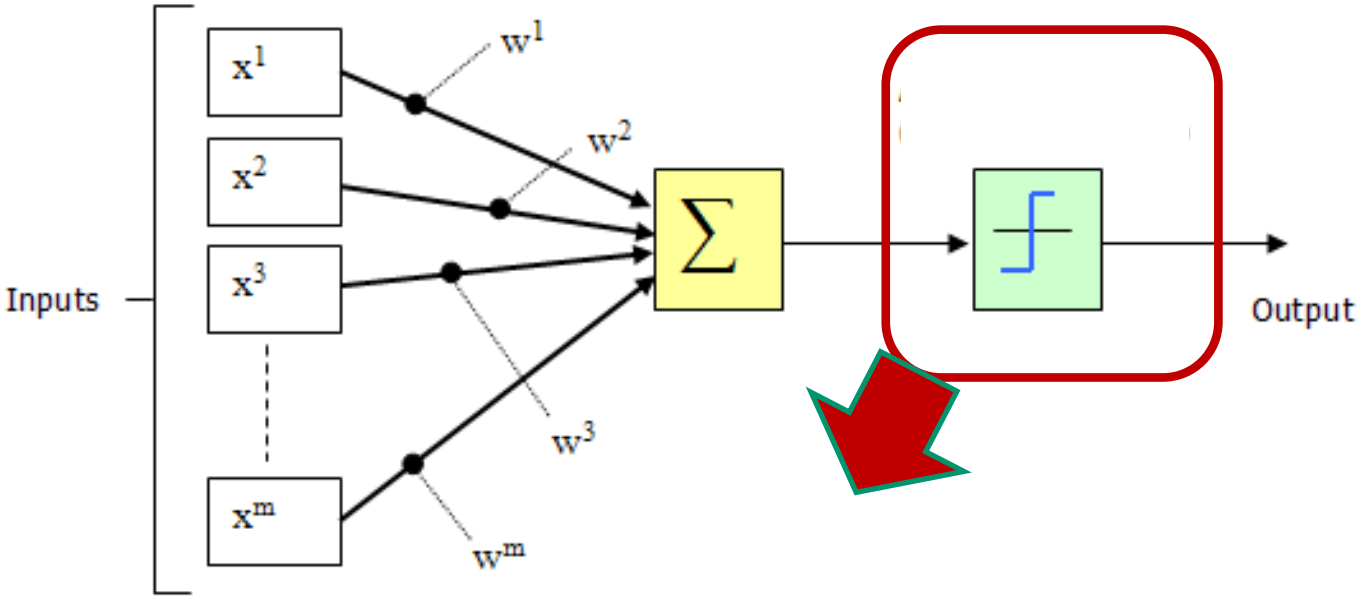


Neural computation



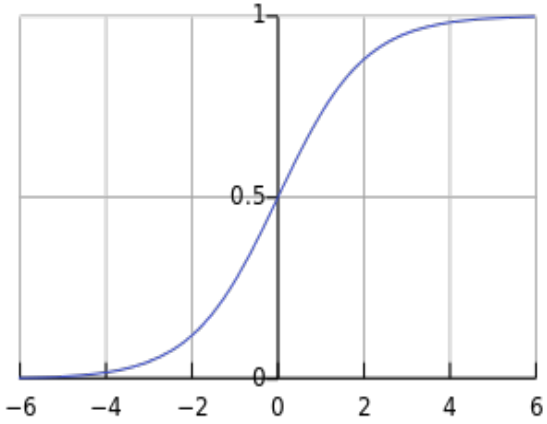
$$a_t = \sum_{i=1}^{m} x^i w^i$$

Neural computation



Some activation functions

- Sigmoidal (hidden)
- Linear (output)



$$Sig(x) = \frac{1}{1+e^{-x}}$$

$$TH(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

Universal approximation theorem

A feedforward network with a single hidden layer containing a finite number of neurons and a linear output neuron approximates any continuous function defined on compact subsets

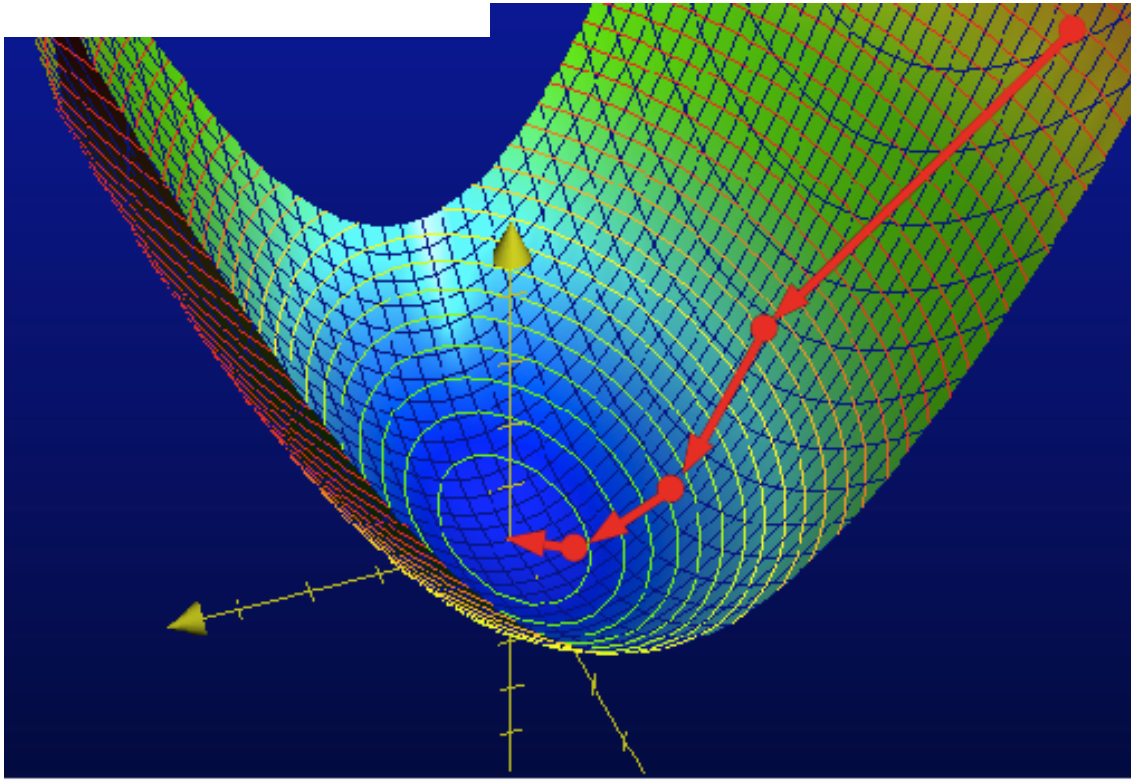
K. Hornik, "Approximation Capabilities of Multilayer Feedforward Networks", *Neural Networks*, No.4 Vol. 2, 251–257, 1991

Parameter configuration: Gradient-based optimization

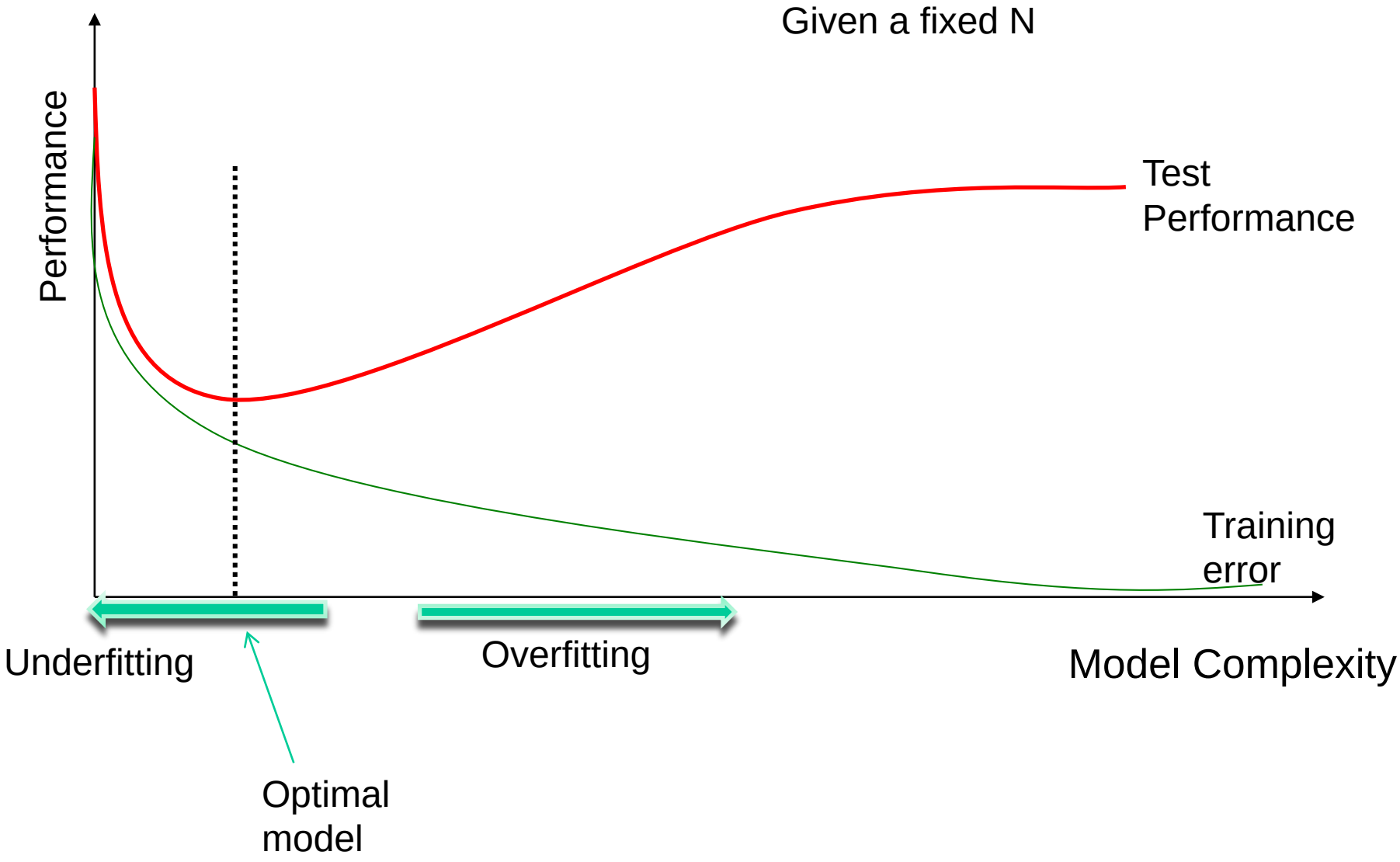
$$V_n(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y(x_i) - f(\theta, x_i)\right)^2$$

$$\theta_{i+1} = \theta_i - \varepsilon_L \frac{\partial V_n(\theta)}{\partial \theta}\Big|_{\theta_i}$$

- The minimization procedure applied to nonlinear functions is also known as «learning procedure»



Approximation performance vs. Model complexity



The classification problem

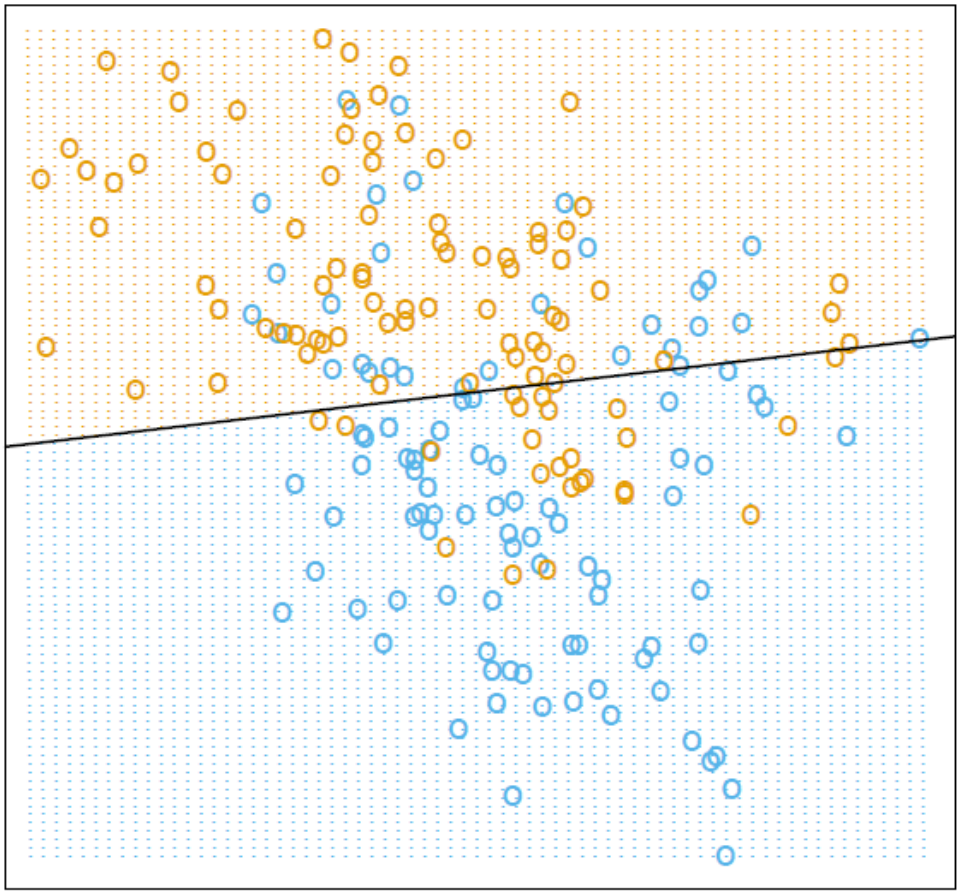
«Do not play around: say yes or no, please»

The problem

- Given an input vector *x* and a set of classes $C_1, \ldots C_k$ we wish to assign *x* to the most appropriate class
- Not rarely the classification is intended in probabilistic terms and we are interested in the probability that *x* belongs to each of the above classes.
  - The class with the maximum probability is the “winning” one
  - Mistakes can be made, given the probabilistic framework

An example

- Given an input vector *x* (two **features)** and two classes, what should we do?
- Do we know a tool for determining the discriminant function?



Linear regression

- Yes, we can use a linear regression after having coded
  - y=0, if blue
  - y=1, if orange
- Given a new x, evaluate the discriminant function

$$f(\hat{\theta}, x) = x^T \hat{\theta}$$

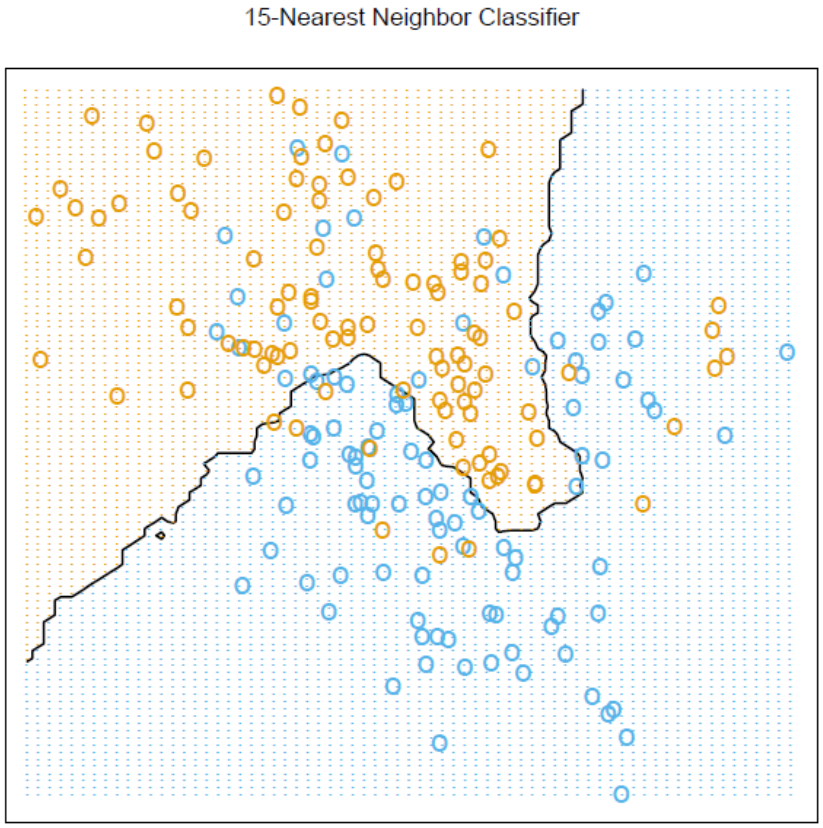
$$\hat{\theta} = (X^T X)^{-1} X^T Y$$

- If the value of the discriminant function is above 0.5 the classification output is orange.
- Extension to the K classes case (with some problems).
- However, we do not get “a probability” out of the discriminant function-> Logistic regression.

KNN: K-Nearest Neighbor classifier

- Look at the k-nearest neighbors to make a decision
- Use e.g., the Euclidean distance
- No training phase, large computational cost in evaluating the distances

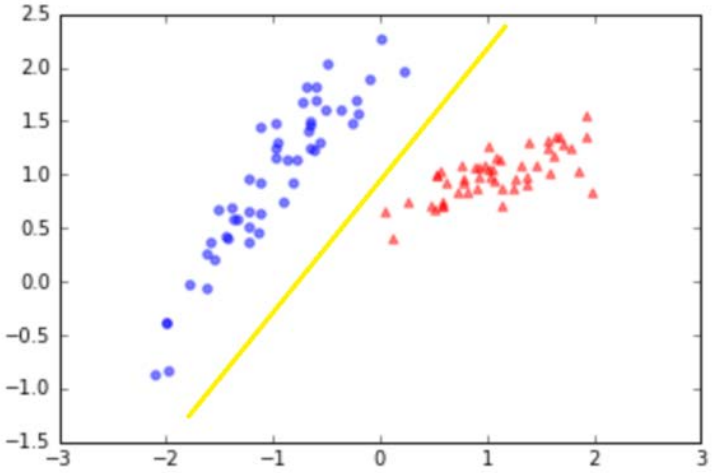
$$\begin{cases} n \to \infty, \implies k \to \infty \text{ and} \\ \lim_{n,k\to\infty} \frac{k}{n} = 0 \end{cases}$$

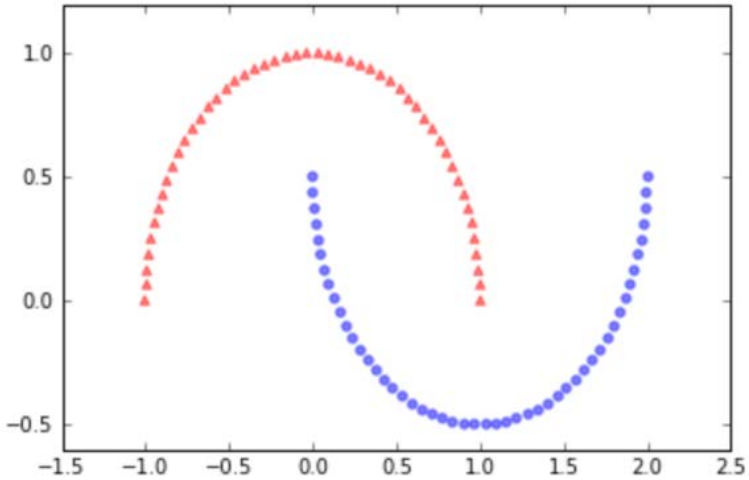


Perceptron

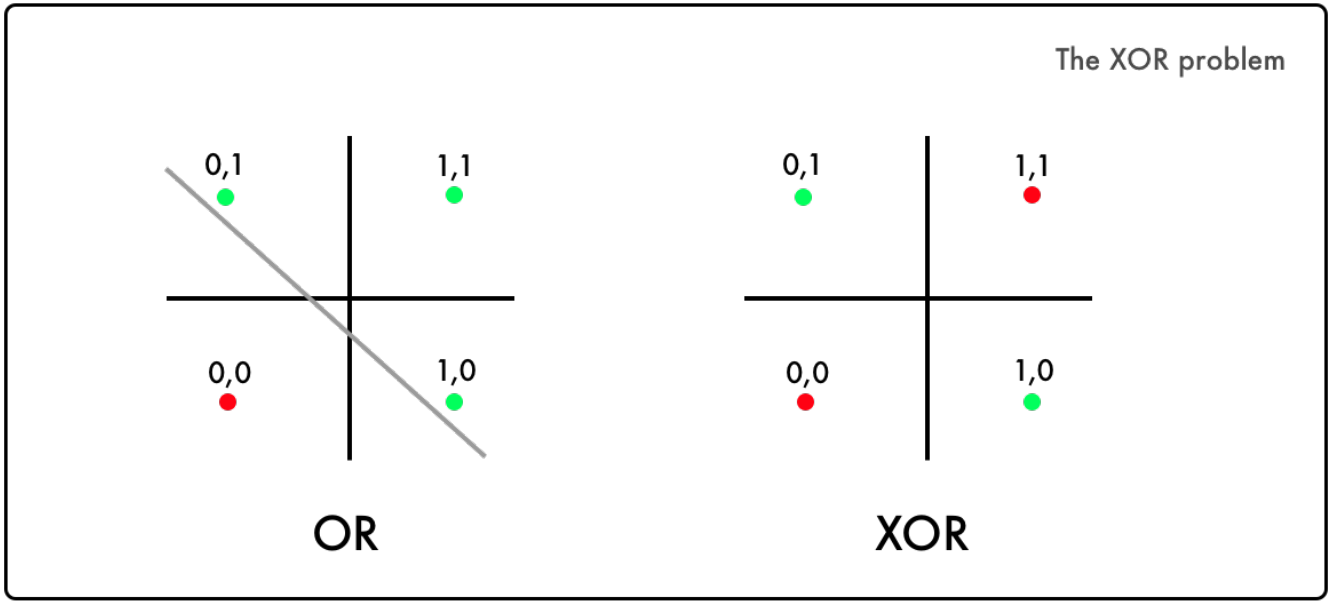
- The perceptron algorithm was invented in 1957 by F.Rosenblatt for pattern classification.
- It works well on linearly separable classes and is able to learn its parameters
- In 1969 Minsky and Papert showed that the perceptron is unable to solve the ex-or problem (and the interest in neural networks declined)
- Only in the ‘80 the field became popular thanks to the introduction of “deeper” layers, of feed-forward nature

Perceptron



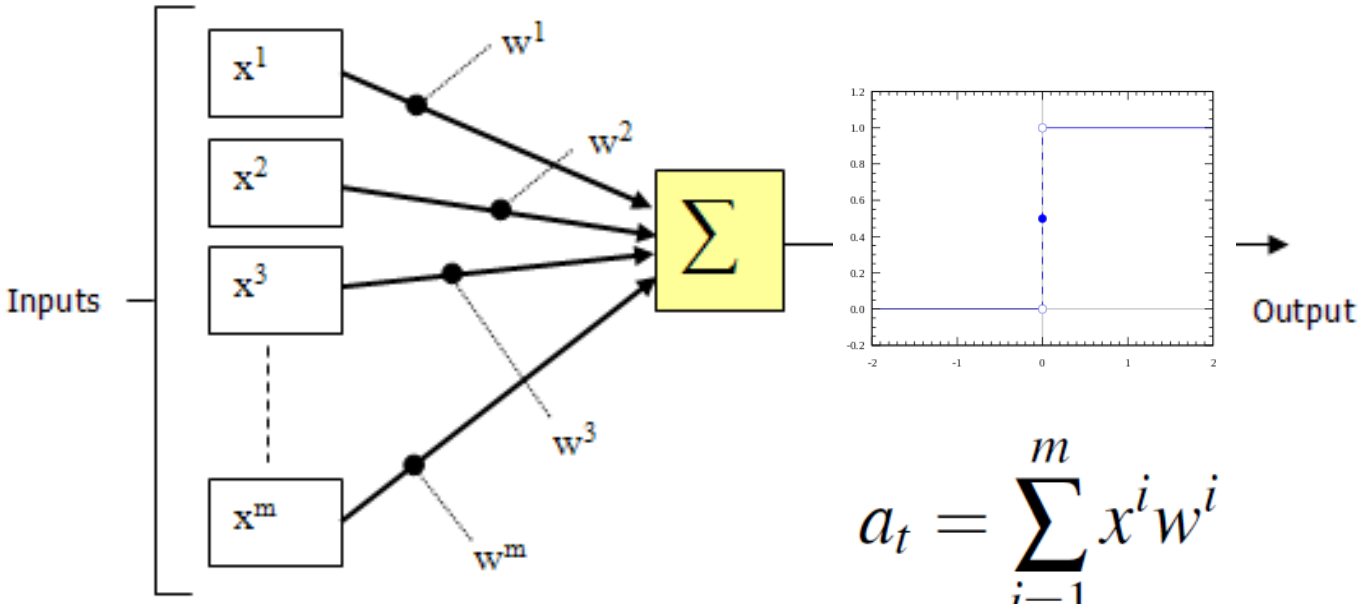


- Linear separability



Perceptron

- The architecture is that of a single neuron with a Heaviside activation function



$$a_t = \sum_{i=1}^{m} x^i w^i$$

Perceptron

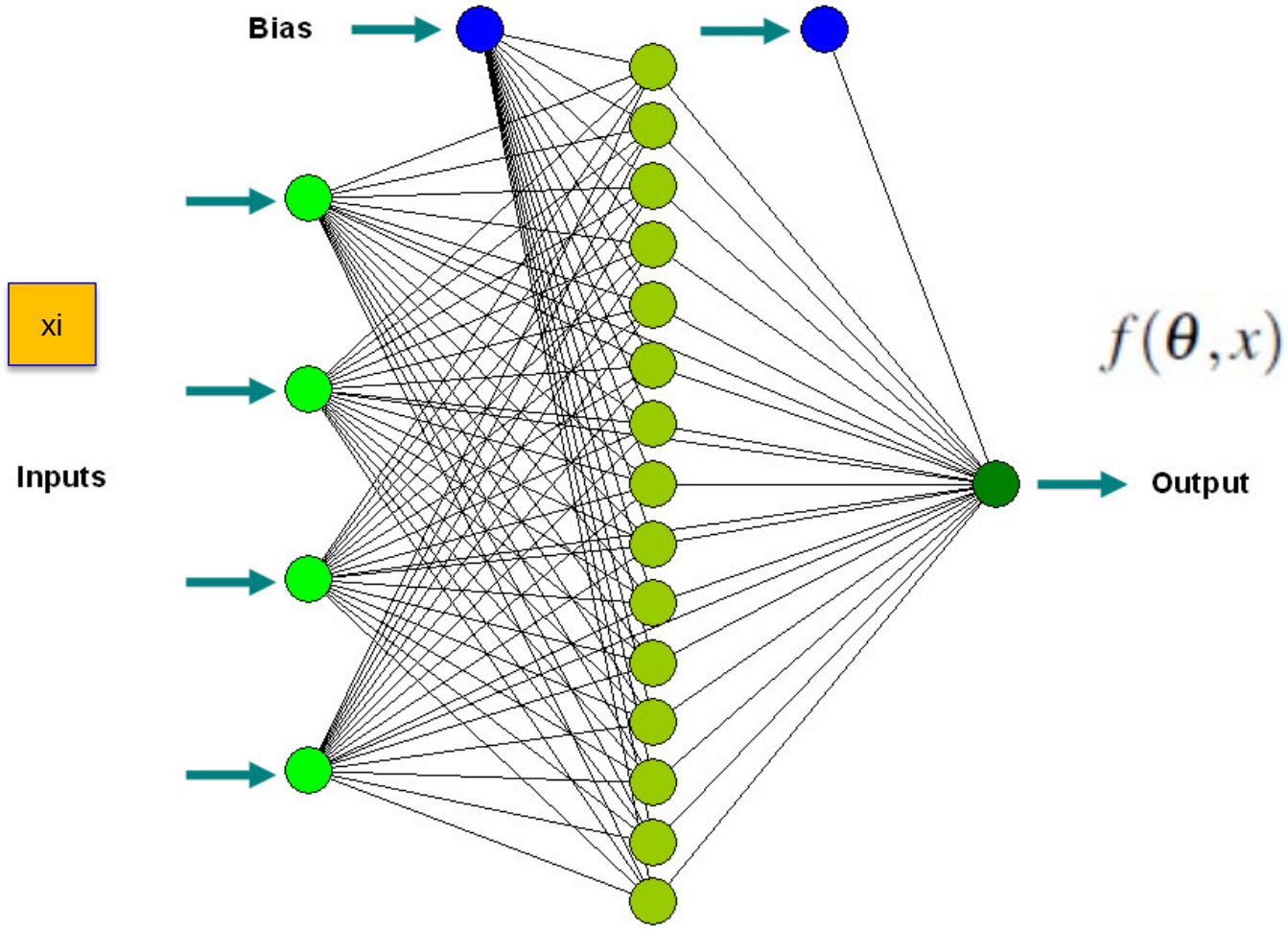
- After a random assignment for weights (small values)
- Training algorithm: weights are updated by
  - iteratively presenting the training set

$$\theta_{i+1} = \theta_i - \varepsilon_L(y(x_i) - f(\theta_i, x_i)x_i$$

  - or in batch
- Either way the algorithm converges to the optimal parameter configuration if the classes are linearly separable

Feedforward Neural Networks

For classification



F-NN classification: formalization

Define performance function

$$V_N(\theta) = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, f(\theta, x_i)\right)$$

Determine the parameter estimate through a minimization problem

$$\hat{\theta} = \arg\min_{\theta \in \Theta} V_N(\theta)$$

carried out by a learning procedure

$$\theta_{i+1} = \theta_i - \varepsilon_L \frac{\partial V_N(\theta)}{\partial \theta}\Big|_{\theta_i}$$

and get classifier $f(\hat{\theta}, x)$

Loss function for classifiers

- Square error

$$L(y(x), f(\theta, x)) = (y(x) - f(\theta, x))^2$$

- Cross entropy (binary classes)

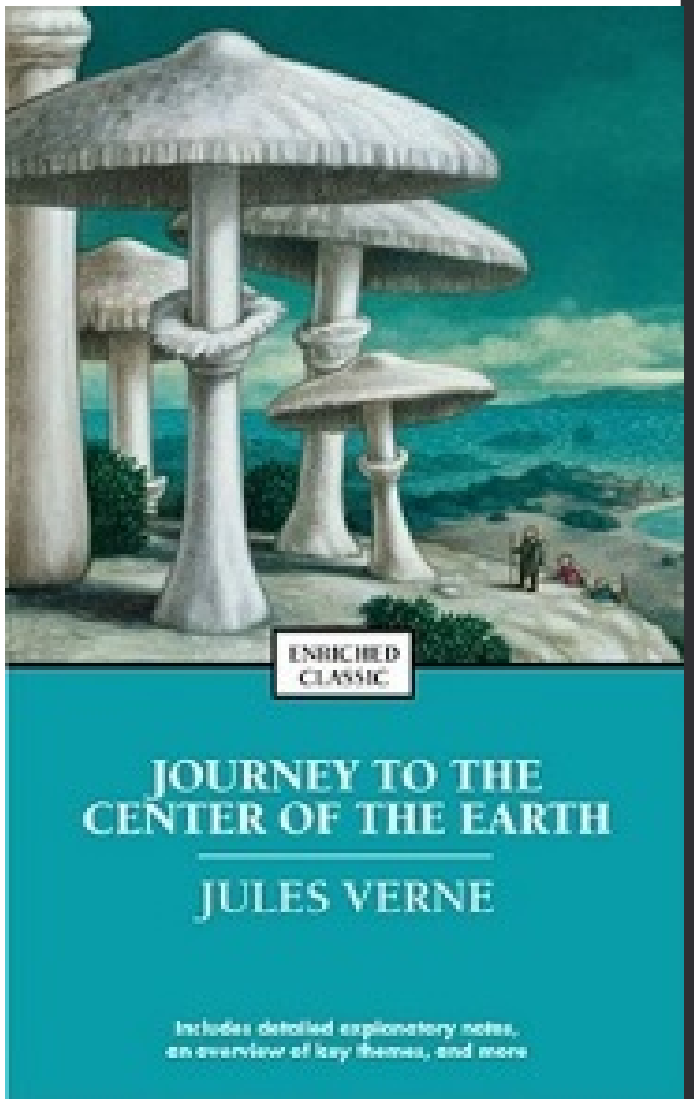
$$L(y(x), f(\theta, x)) = -[y(x) \log f(\theta, x) + (1 - y(x)) \log(1 - f(\theta, x))]$$

- Cross entropy (multi classes)

$$L(y(x), f(\theta, x)) = -\sum_{c=1}^{M} y_c(x) \log f_c(\theta, x)$$

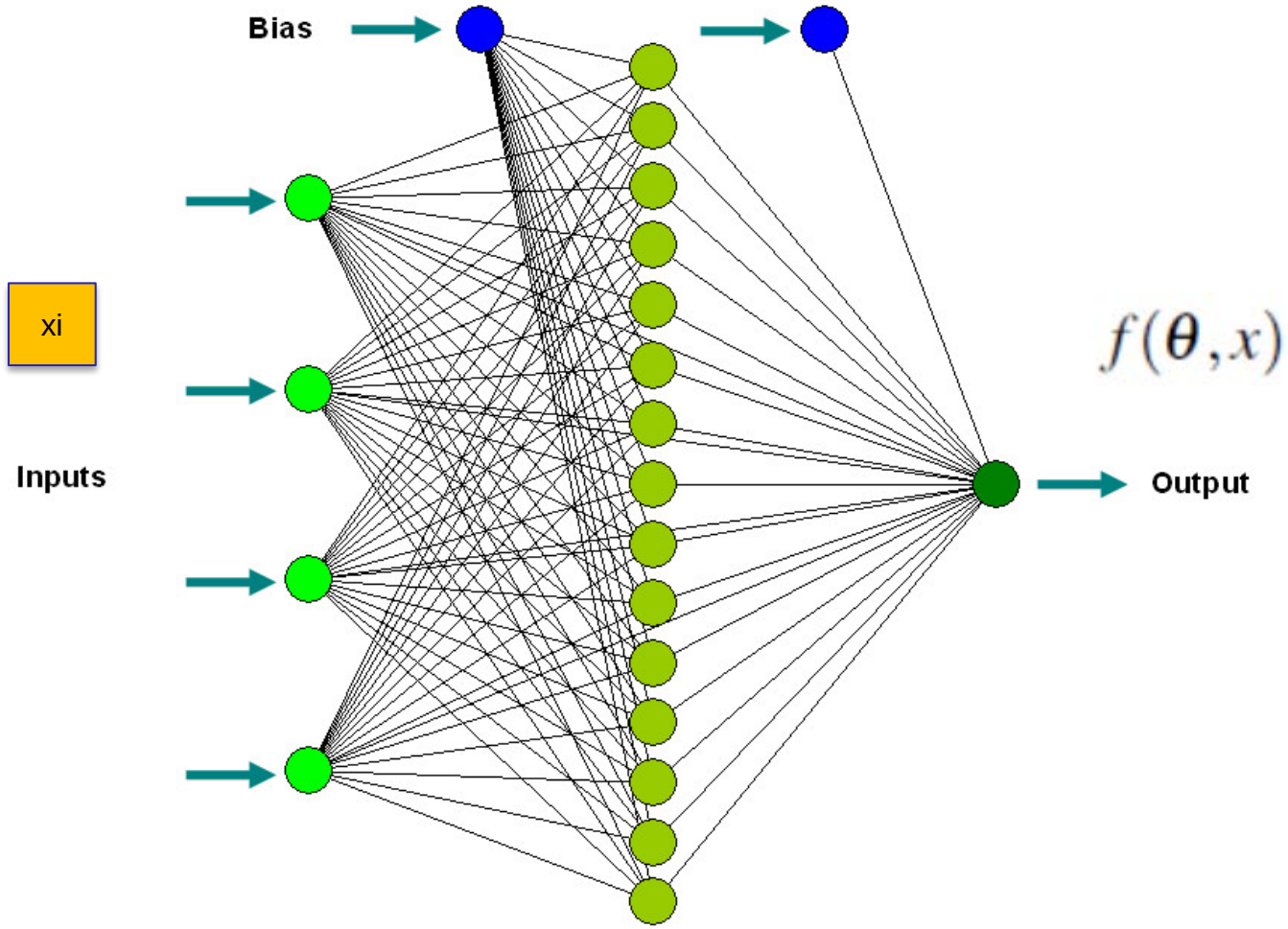
Deep learning

«Deep, deep towards the center of the earth»



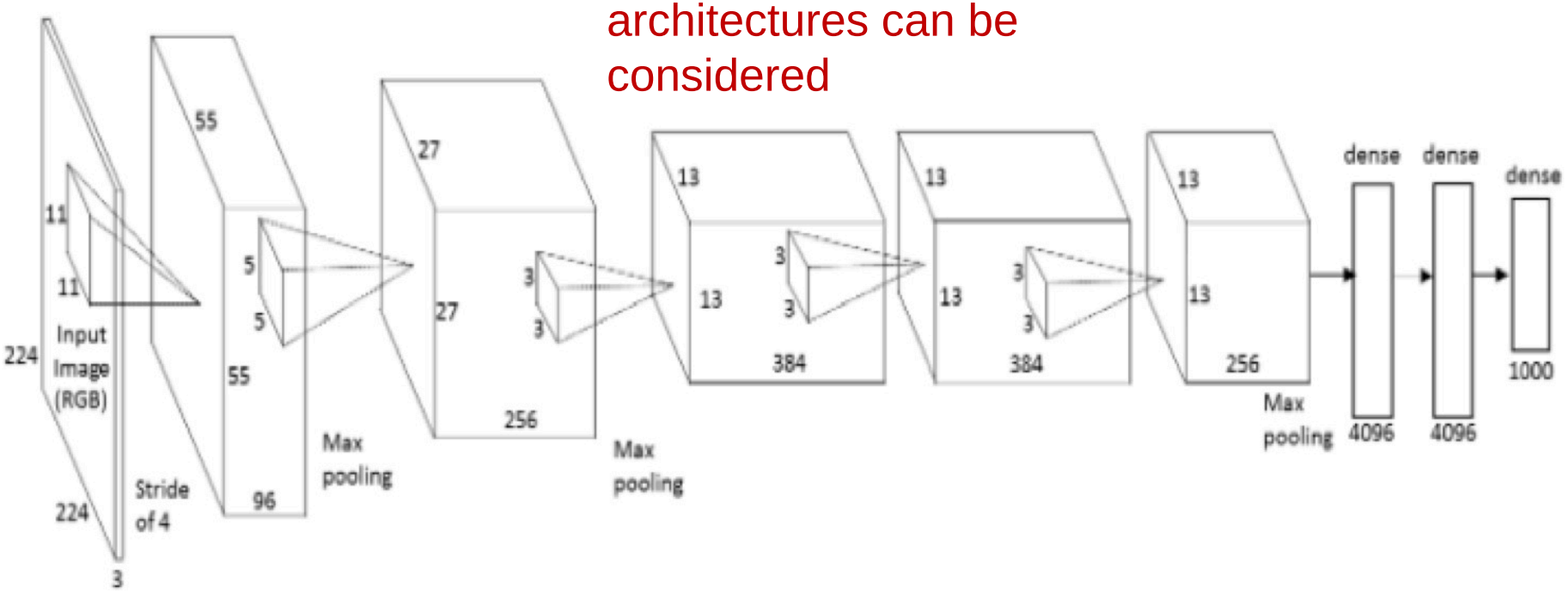
Shallow architectures

Not rarely $f(\theta, x)$ is chosen as



Deep architectures

Recurrent architectures can be considered



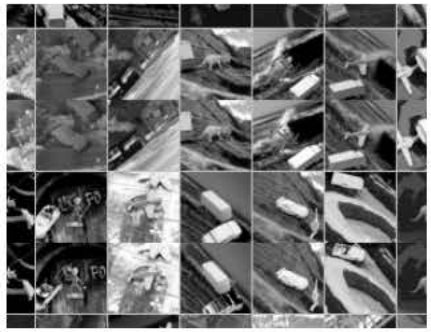
- Deep learning was enabled by
  - advances in computational power
  - Availability of big data for training

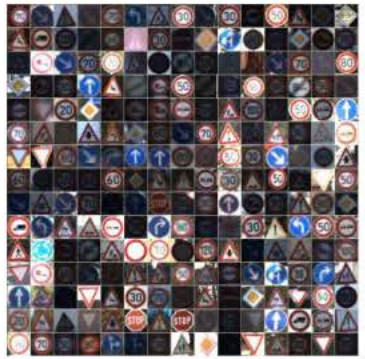
Some examples

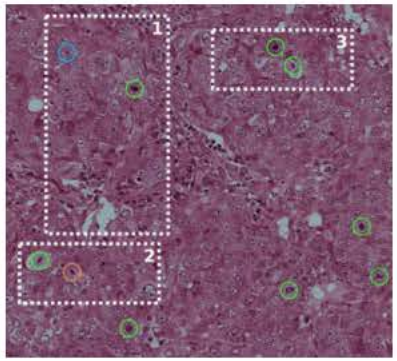
- **Detect –classify- different elements** within a scene (e.g., for self-driving cars).

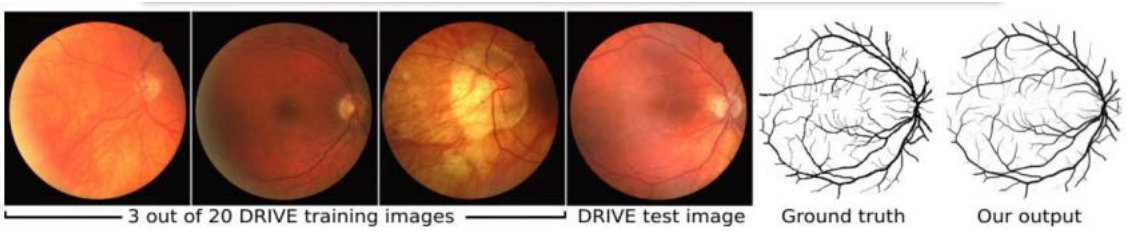
Some examples

- **Detect –classify- different elements** within a scene









Some examples

- Generate new images that look **extremely** realistic.

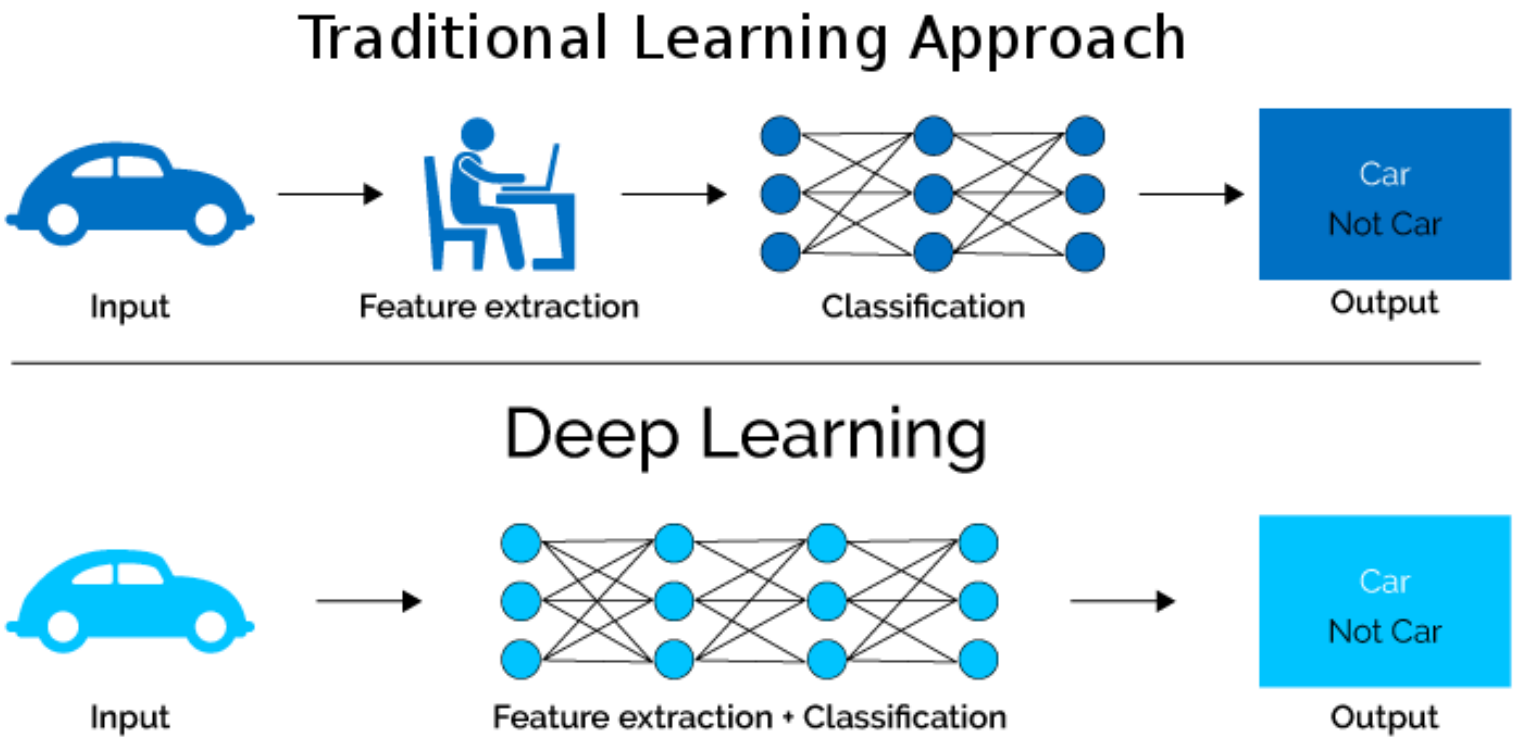
Some examples

- **Make decisions** with the same skills and “creativity” as humans.

World top 5 players of Go defeated by AlphaGo in 2016.

Representation learning

- **Traditional** approach: build on features hand-crafted by experts.
- **Deep** approach: features are part of the learning process



Representation learning

- **Hierarchical** abstraction of the features.

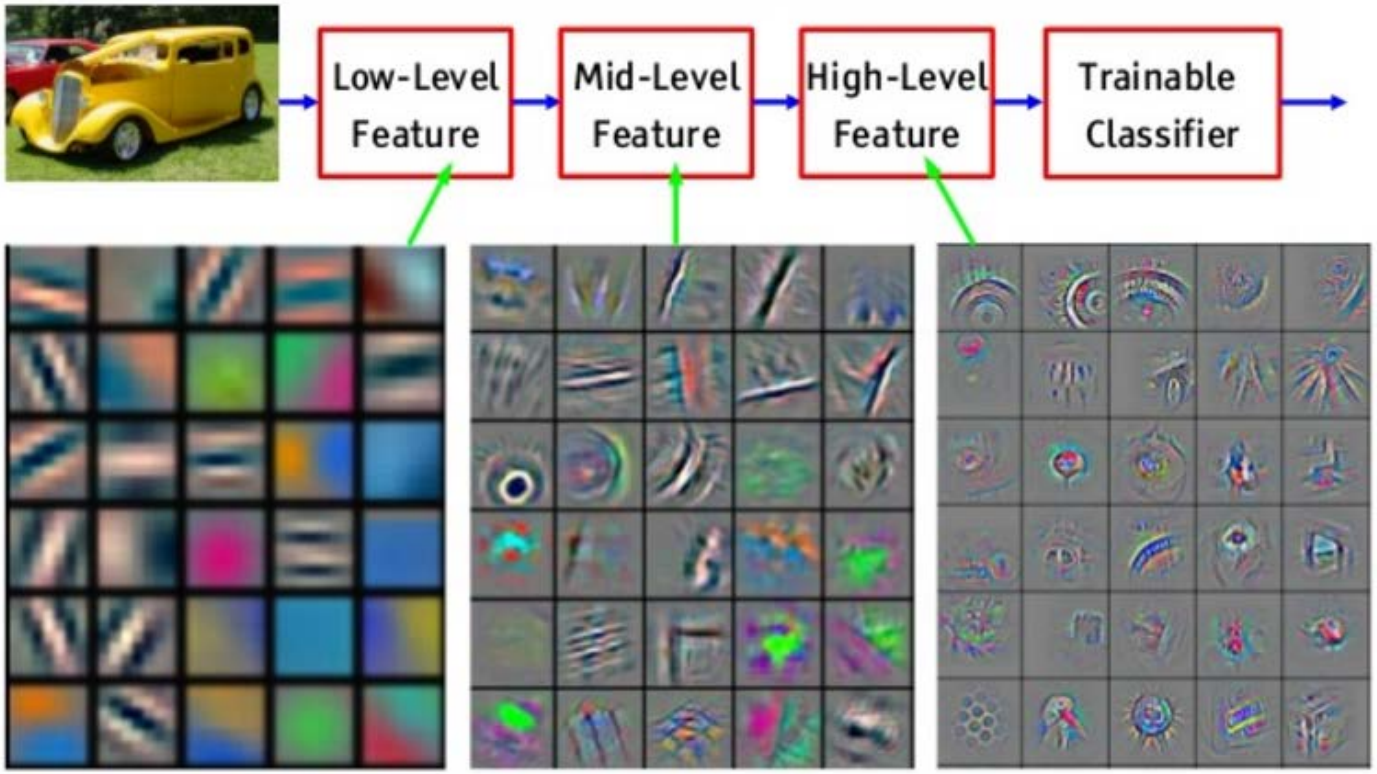
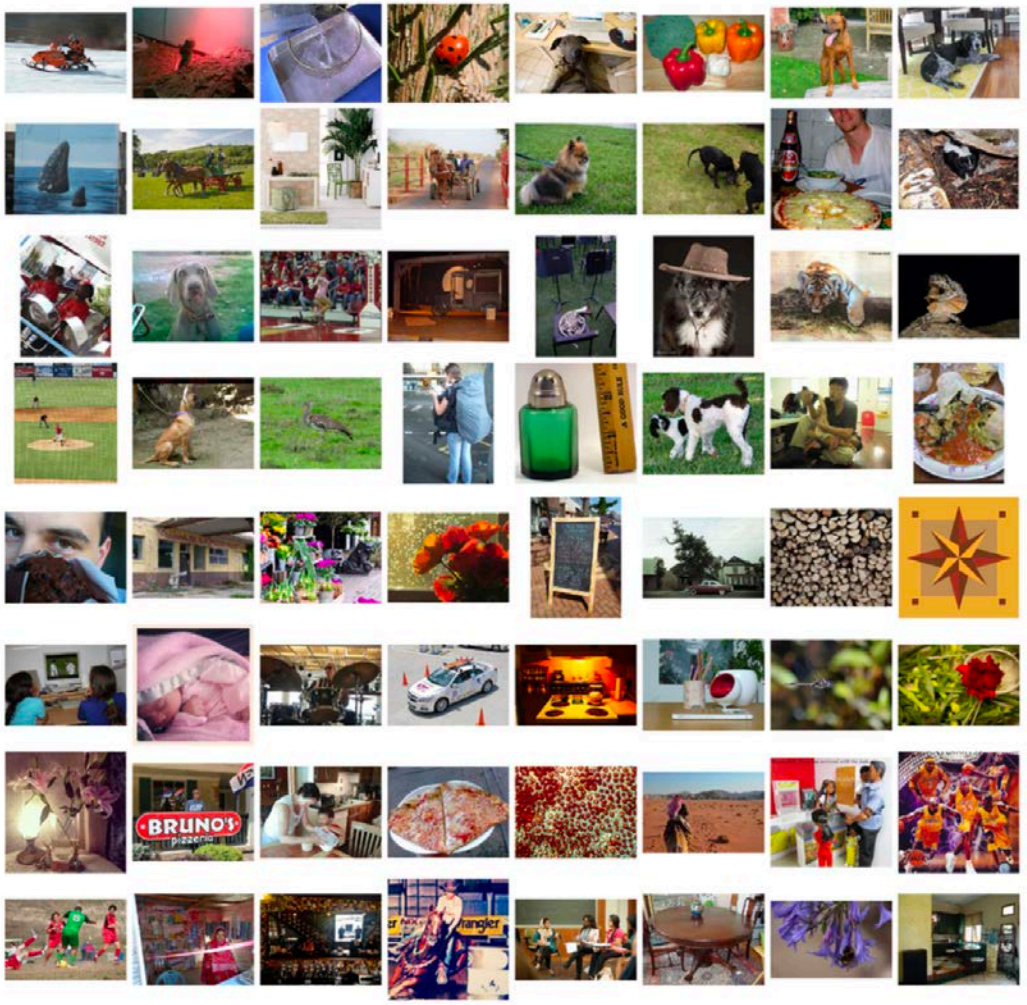
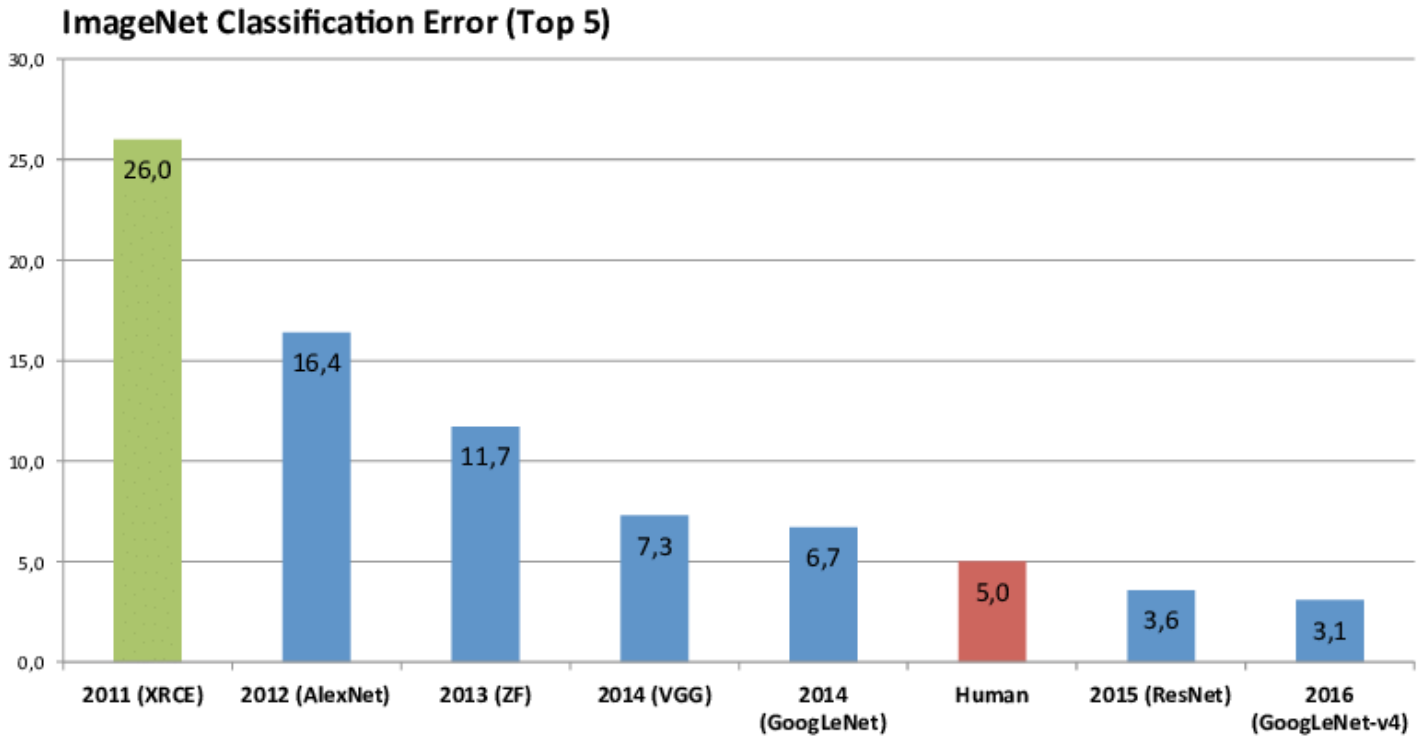


Image classification

- ImageNet dataset, used as benchmark task for computer vision.
- 1.3M training images.
- 1000 classes.
- Contest hold annually with the goal of improving classification accuracy.
- **Superhuman** performance reached by Microsoft in 2015.

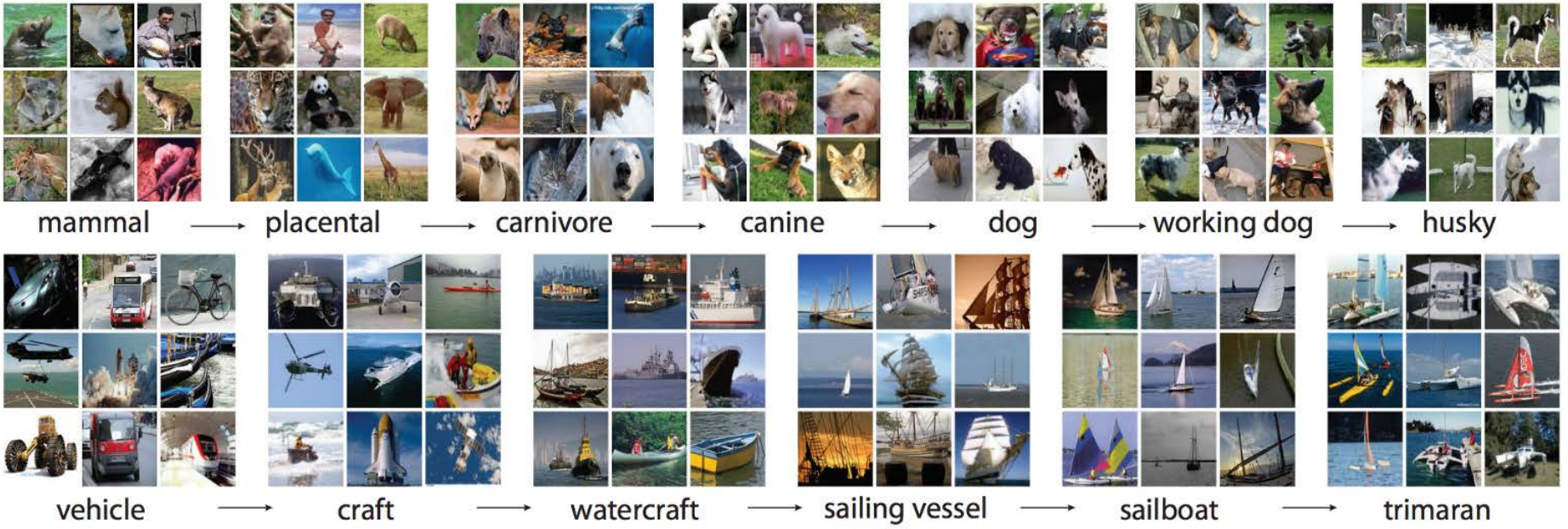


Representation learning



- **Exponential progress** has been made in very few years.

- **ImageNet:**
  hierarchical structure
  with classes and
  subclasses



Representation learning

- Powerful libraries are available to train deep learning models with minimal efforts.
- Can be easily parallelized on **GPUs.**

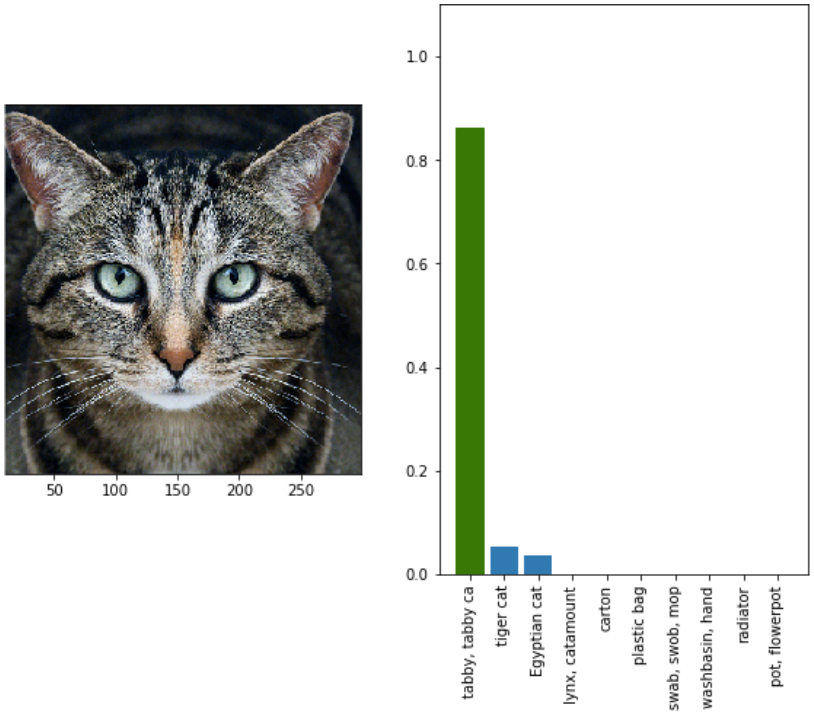
Representation learning

- A working script for image classification can take as little as 19 lines of code with **Keras**.

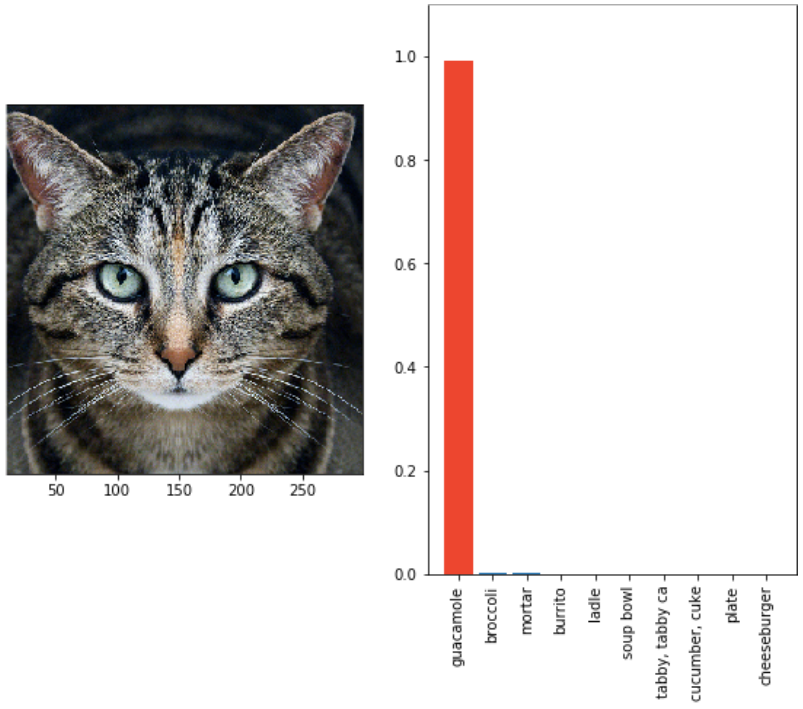
```
1  from keras.datasets import mnist
2  from keras.models import Sequential
3  from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
4
5  # Load and pre-process data
6  (x_train, y_train), (x_test, y_test) = mnist.load_data()
7
8  x_train = x_train.reshape(x_train.shape[0], 28, 28, 1) / 255.
9  x_test = x_test.reshape(x_test.shape[0], 28, 28, 1) / 255.
10 y_train = keras.utils.to_categorical(y_train, 10)
11 y_test = keras.utils.to_categorical(y_test, 10)
12
13 # Define model
14 model = Sequential()
15 model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
16 model.add(Conv2D(64, (3, 3), activation='relu'))
17 model.add(MaxPooling2D(pool_size=(2, 2)))
18 model.add(Flatten())
19 model.add(Dense(128, activation='relu'))
20 model.add(Dense(10, activation='softmax'))
21 model.compile(loss='categorical_crossentropy', optimizer='adam')
22
23 # Train and evaluate model
24 model.fit(x_train, y_train, epochs=20)
25 score = model.evaluate(x_test, y_test)
26 print('Test loss: {} - Accuracy: {}'.format(*score))
```

All that glitters is not gold

- The human brain is complex! **Do not be childish and claim to imitate it**, even with large networks.
- Neural networks can be fooled.



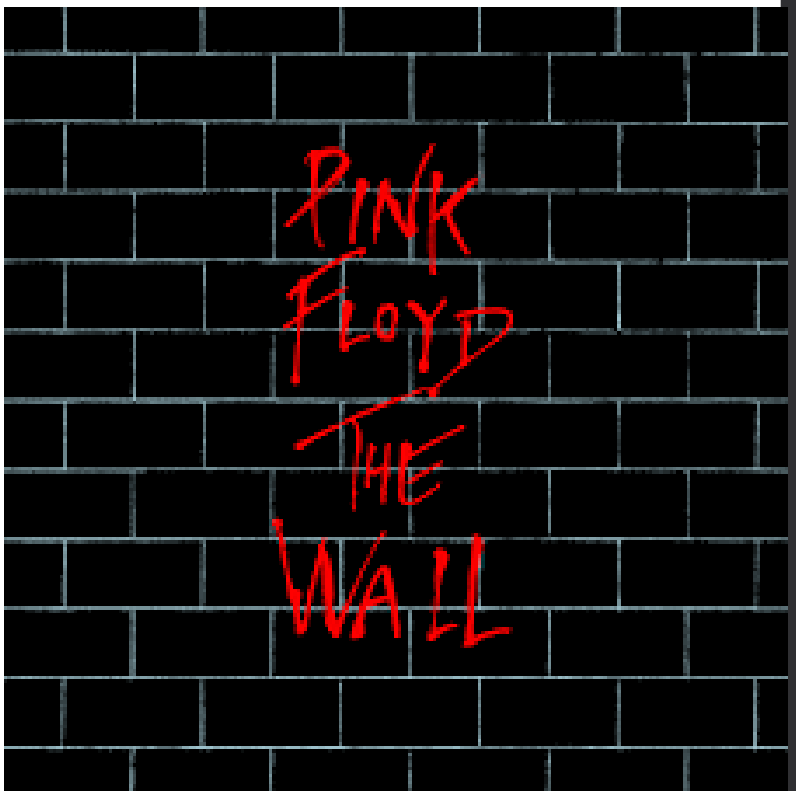
Classified as "tabby cat"
with 85% probability



Classified as "guacamole"
with 99% probability

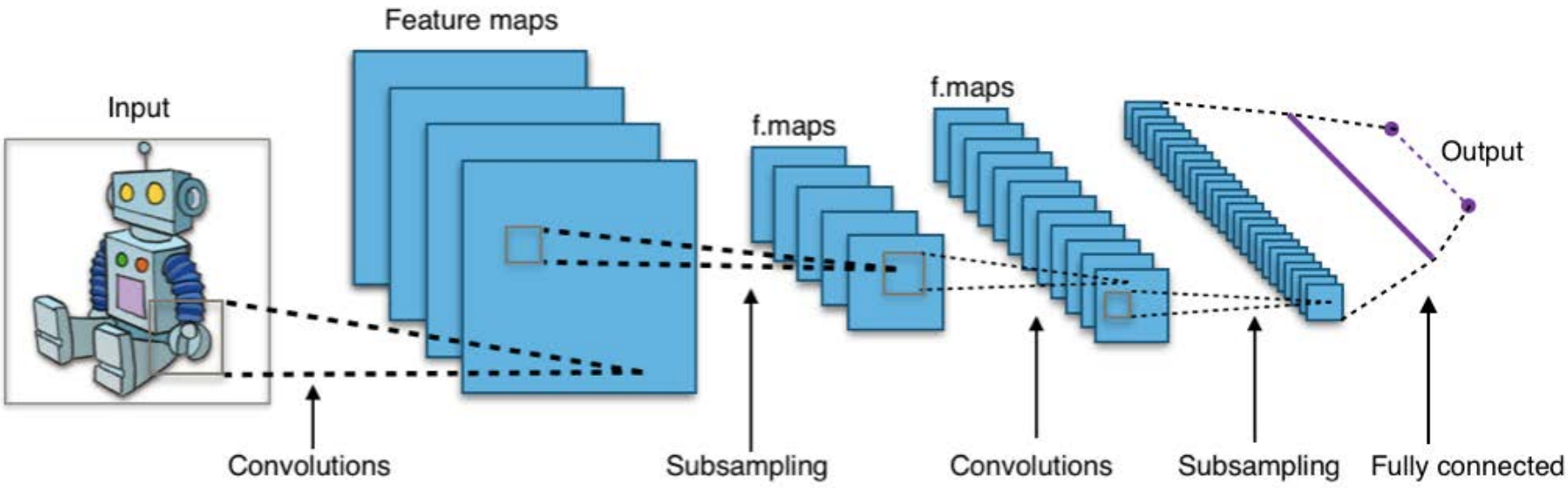
Building blocks

“…another brick on the wall”



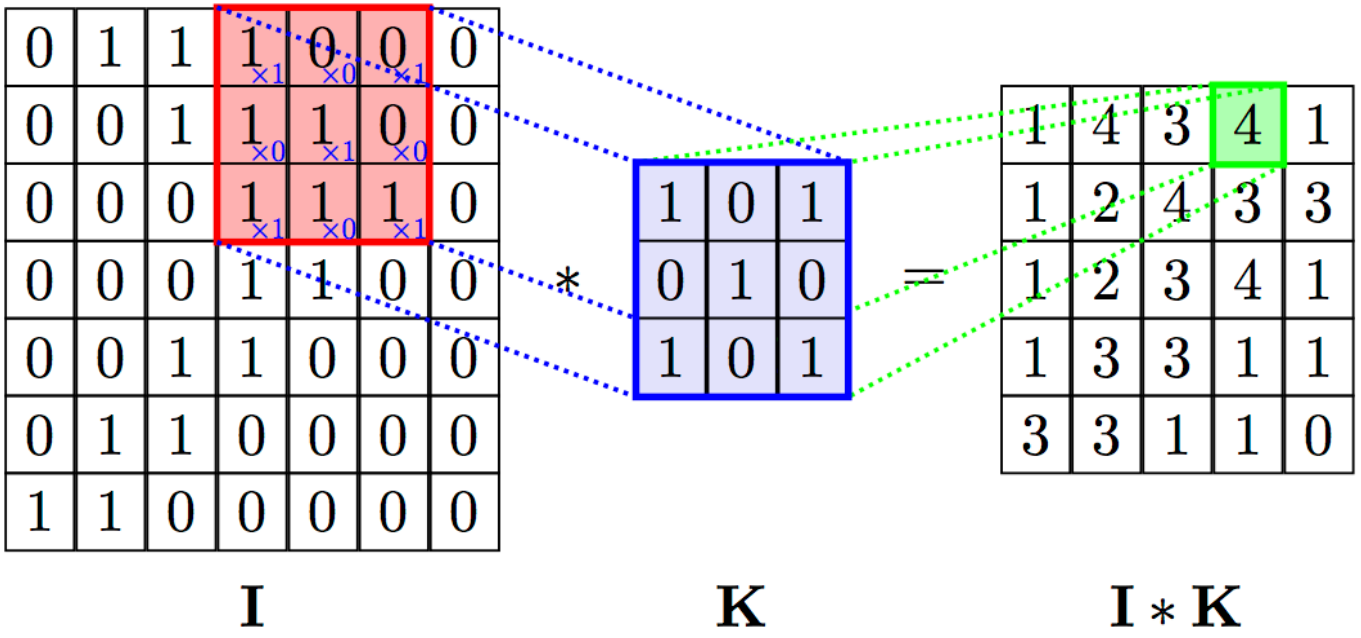
Deep Convolutional Networks

- Subsequent steps of **convolutional and pooling layers**.
- Each layer computes an higher abstract representation; the image size shrinks at each step.
- Feature maps are finally concatenated in a single vector and fed to a feedforward neural net.



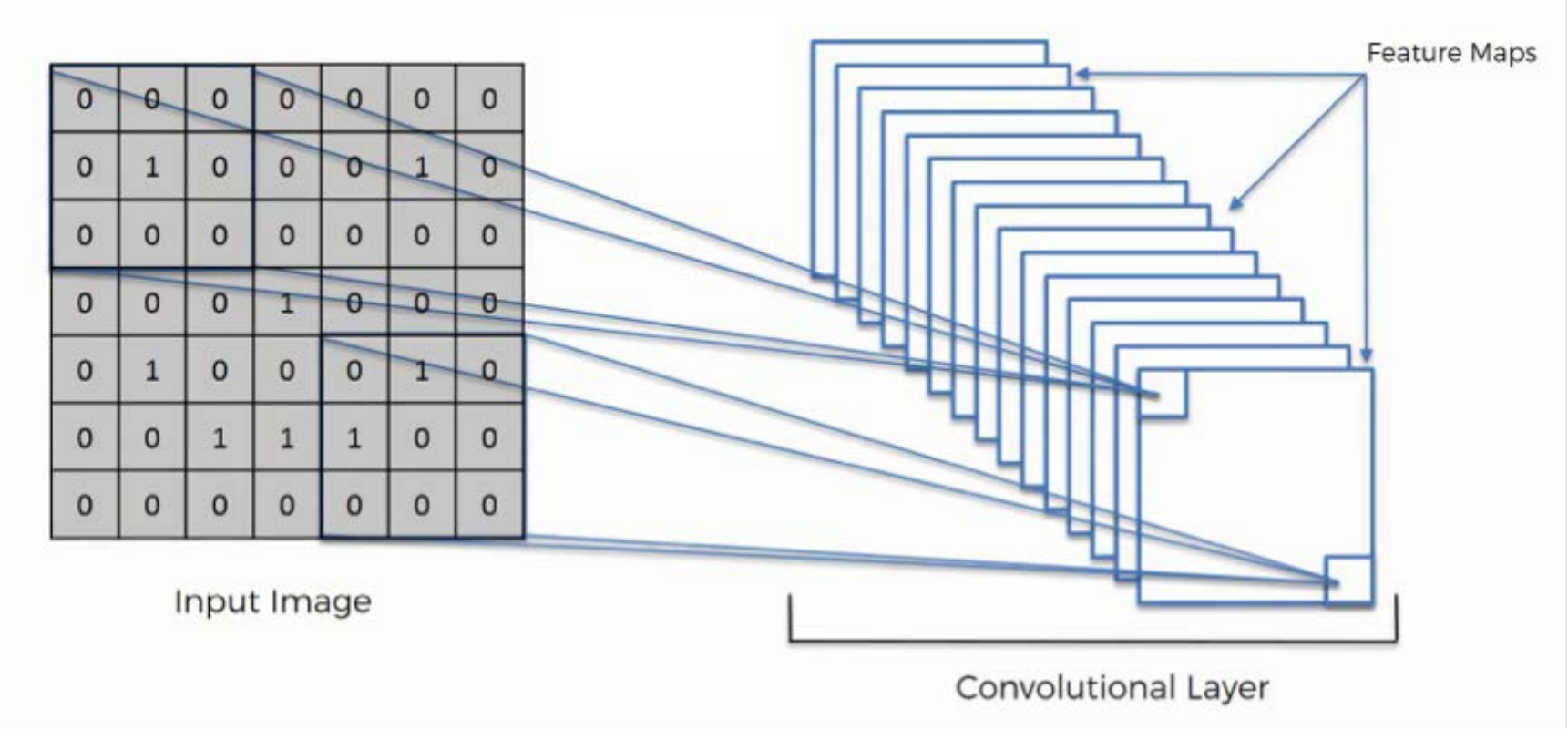
Convolution

- Convolutional layers: extract features based on the principle of locality.
- Receptive field applied to the image with a stride.



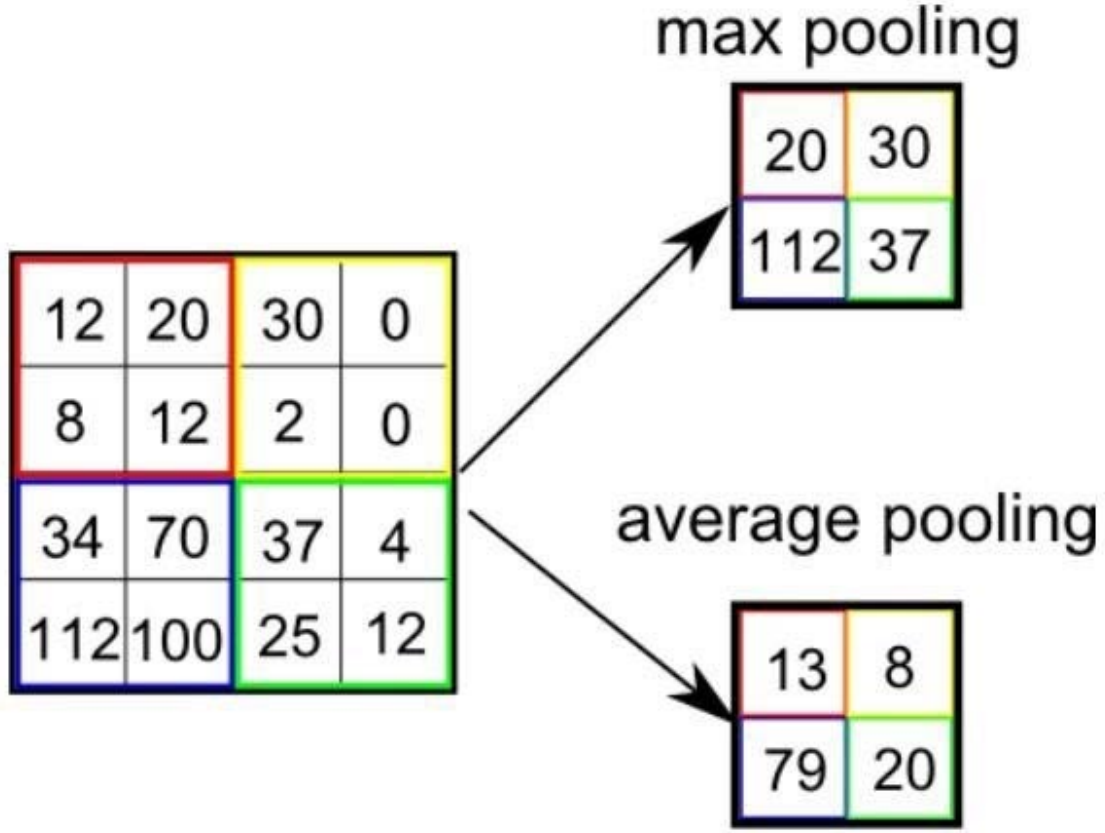
Convolution

- Many filters can be applied in parallel, each one computing a different **feature map**.



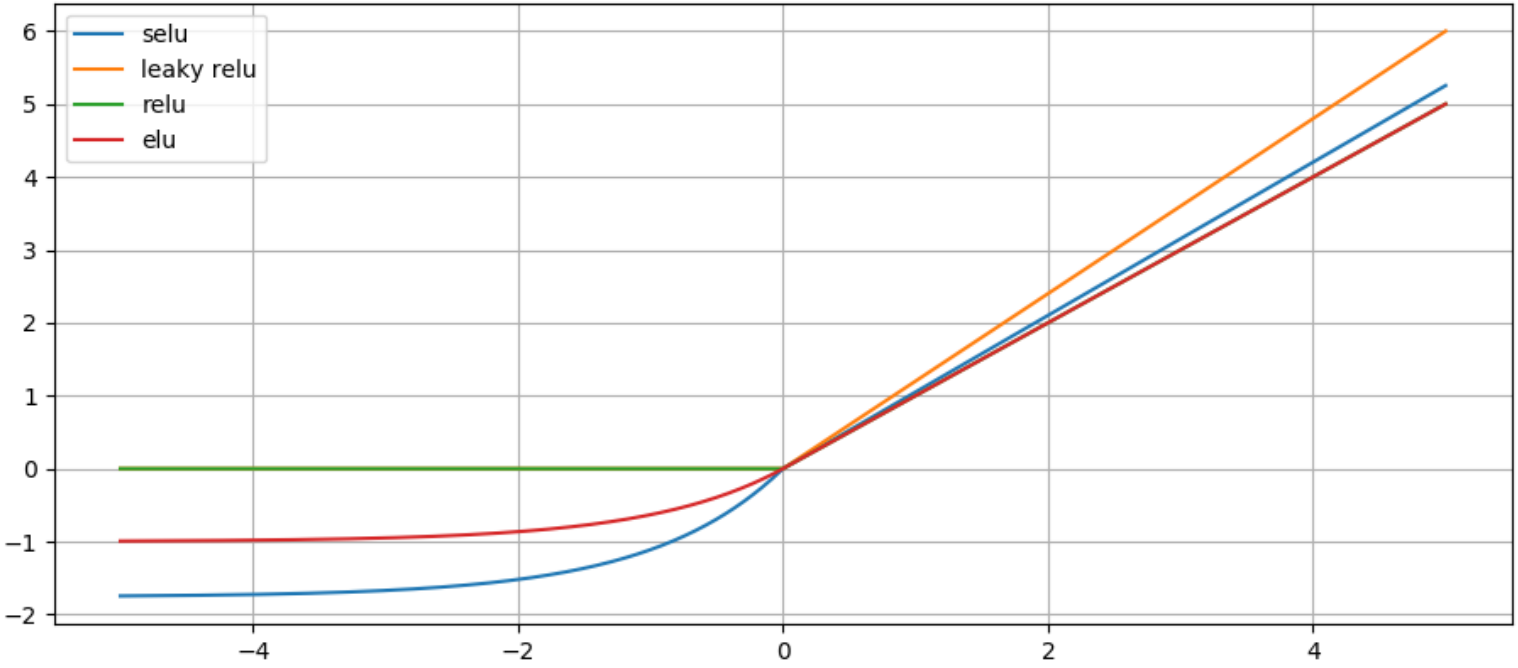
Pooling

- Pooling layers reduce the image size based on some rules.
- In doing so it also provides some local translation invariance property (i.e., it doesn't matter if the relevant parts of the image move around).



Special activation functions

- Stacking many layers may lead to problems when performing backpropagation (vanishing gradients problem).
- **Rectified Linear Unit (ReLU)** has been used as activation function to mitigate such problem.
- Several variant exist, all used in state-of-the-art models.



Figures of merit - Regression

- Mean Squared Error:

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Mean Absolute Error:

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- Mean Absolute Percentage Error:

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i}$$

Image classification: some details

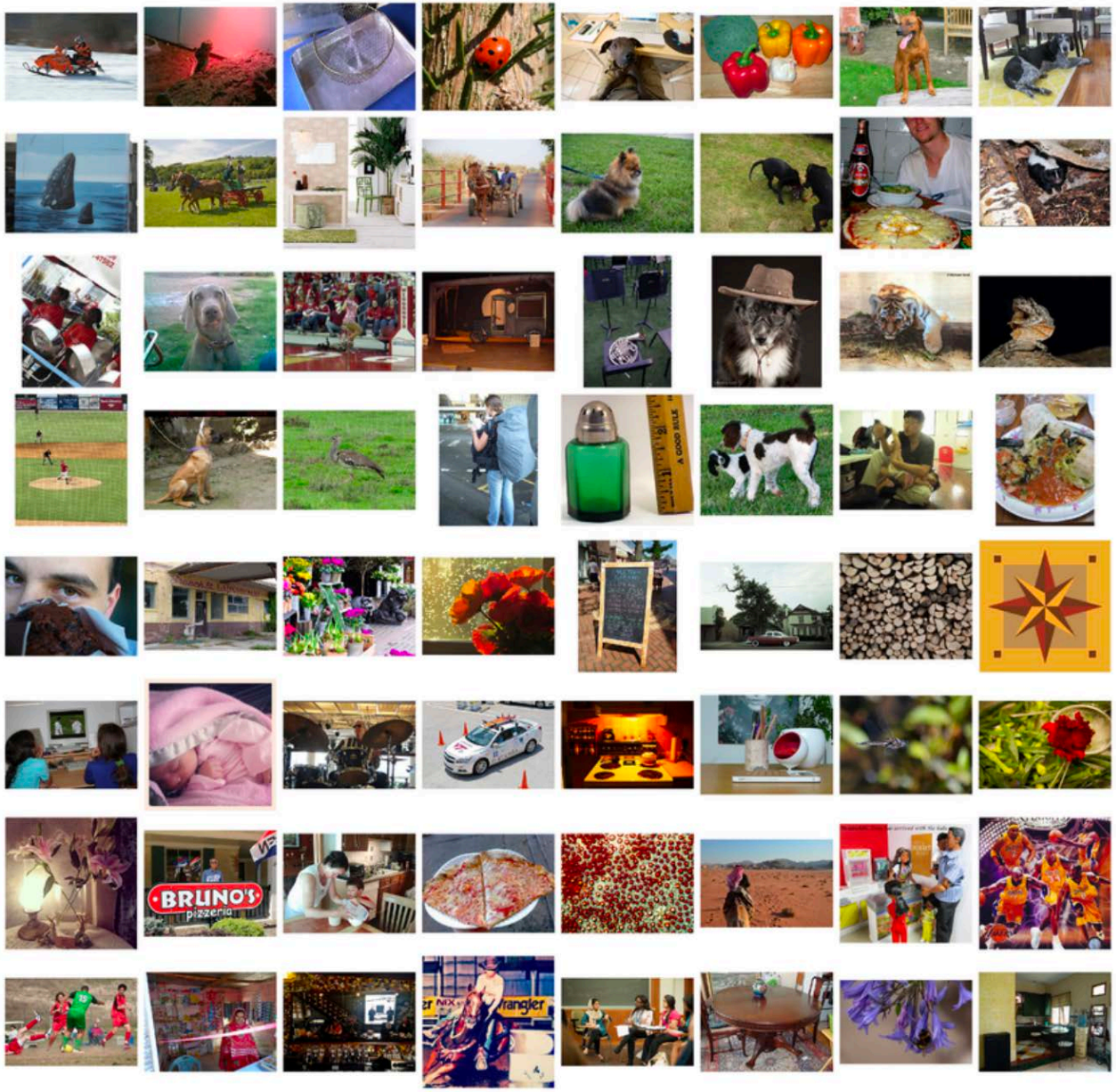
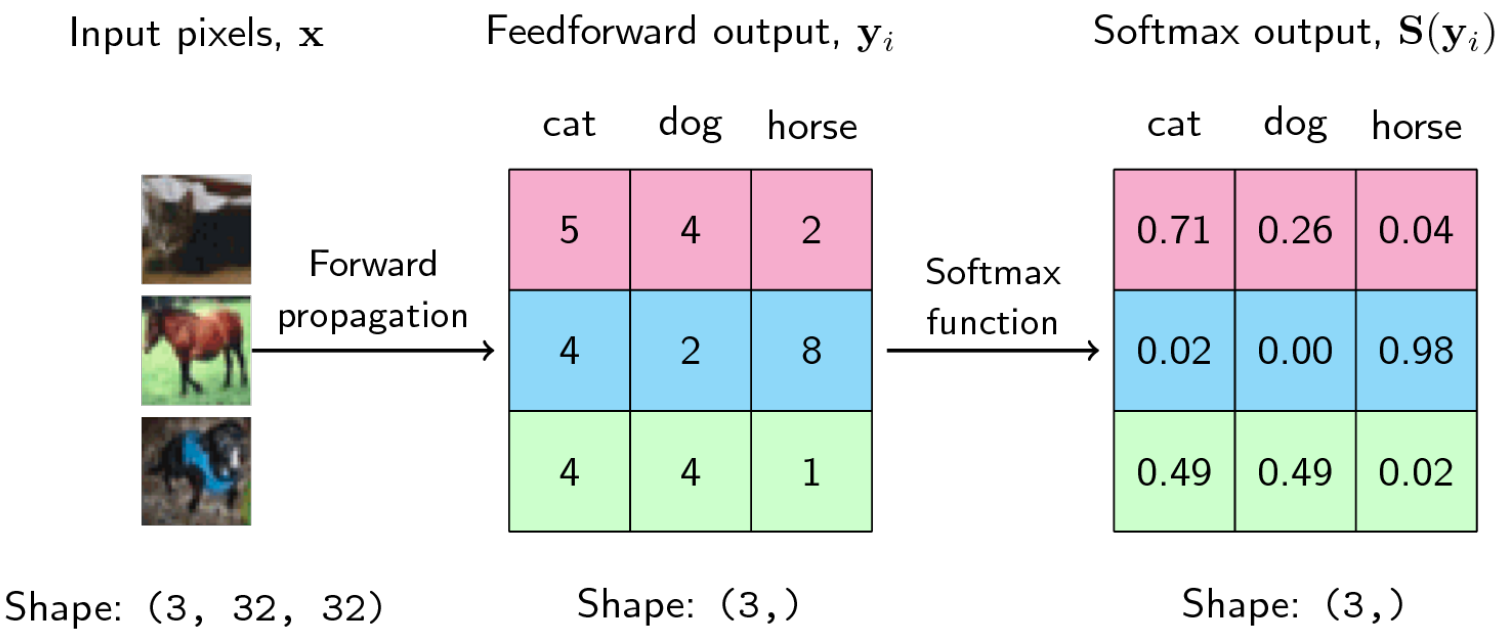


Image classification

- **Simple idea:** the network takes an image as input, and outputs a vector representing a probability distribution over the possible classes.



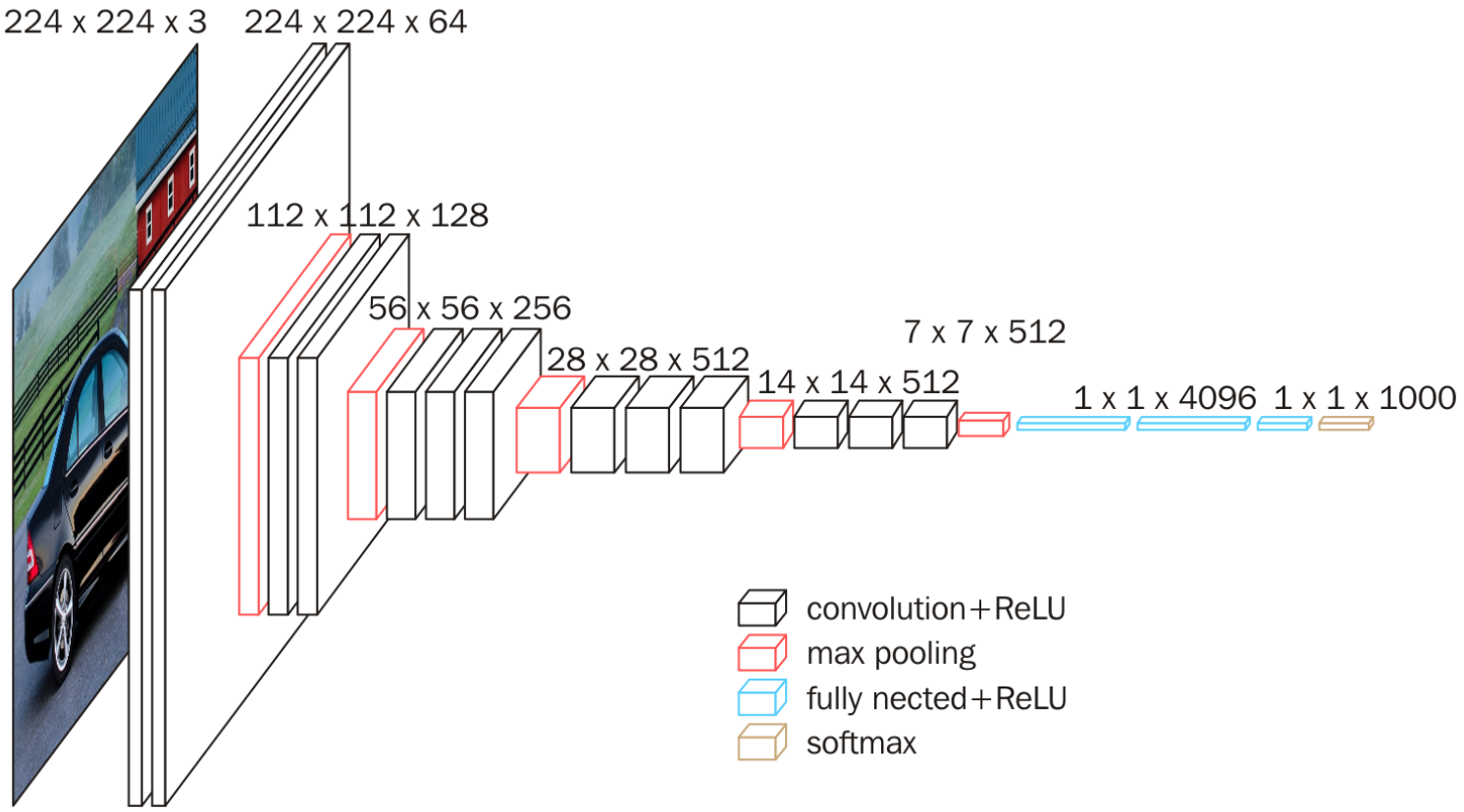
Softmax activation function: $$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Case study: VGG

- "Very Deep Convolutional Networks for Large-Scale Image Recognition.", K. Simonyan and A. Zisserman, ICLR 2015
- Proposed in two variants: 19 trainable layers (VGG-19) and 16 trainable layers (VGG-16).
- **VGG-16 is still used today** as it provides a good tradeoff between accuracy performance and processing efficiency (~500 images/second on Nvidia Titan V).

VGG-16 – Topological structure

- Layers: 13 convolutional, 5 max pooling, 3 fully connected.
- All convolutions use a 3x3 receptive field.
- Activations: all ReLU, with a final Softmax for classification.



VGG-16 - Training

- In the original paper they used SGD with momentum.
- The optimizer remembers the previous update and uses it to compute the new weights.

$$\Delta\theta = \alpha\Delta\theta - \eta \left. \frac{\partial V_n(\theta)}{\partial\theta} \right|_{Z_n, i}$$

$$\theta_{i+1} = \theta_i + \Delta\theta$$

VGG-16 - Training

- A more modern approach: **Adam** (Kingma and Ba, 2014).

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)}$$

$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2$$

Keep a running average of the previous gradients and second moments.

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-(\beta_1)^{t+1}}$$

$$\hat{v}_w = \frac{v_w^{(t+1)}}{1-(\beta_2)^{t+1}}$$

Estimate the current gradient and second moment using the running averages.

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon}$$

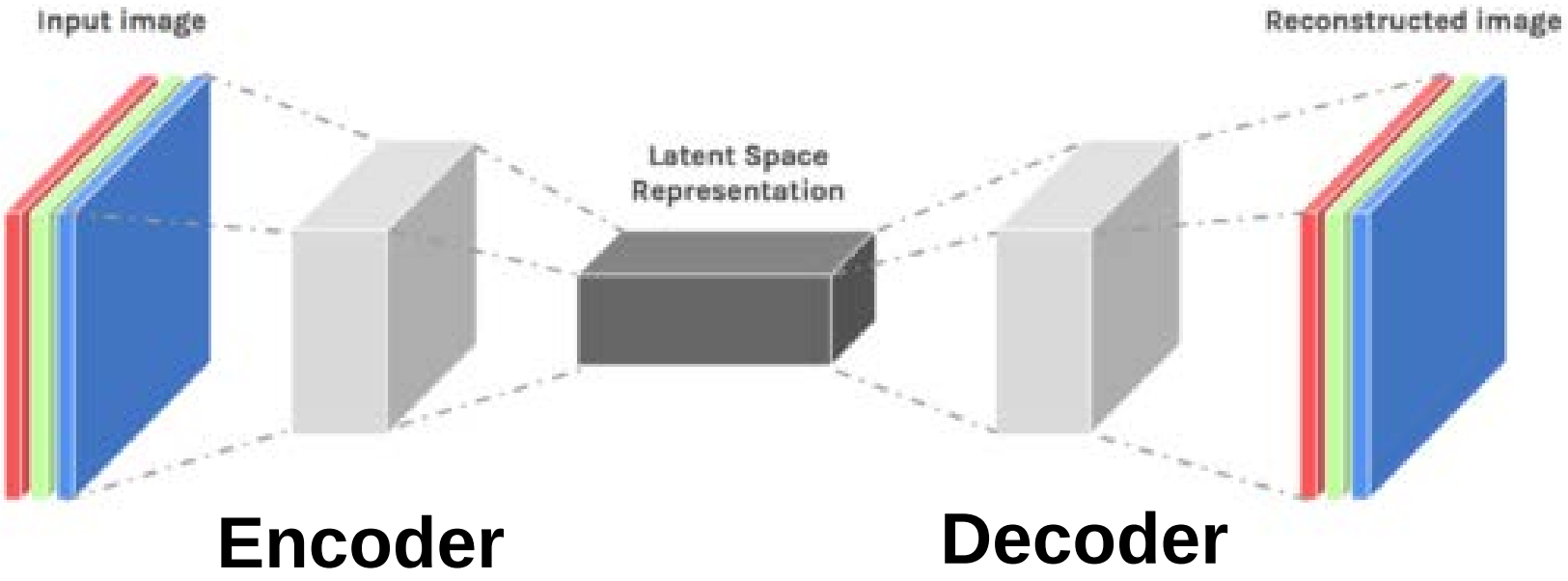
Update weights using the estimated values.

Generative networks

- Generate new images that look **extremely** realistic.

Autoencoders

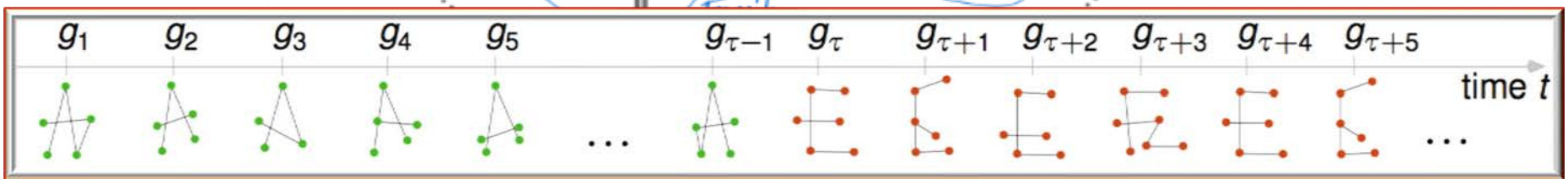
- The encoder is composed of convolutional layers and pooling ones

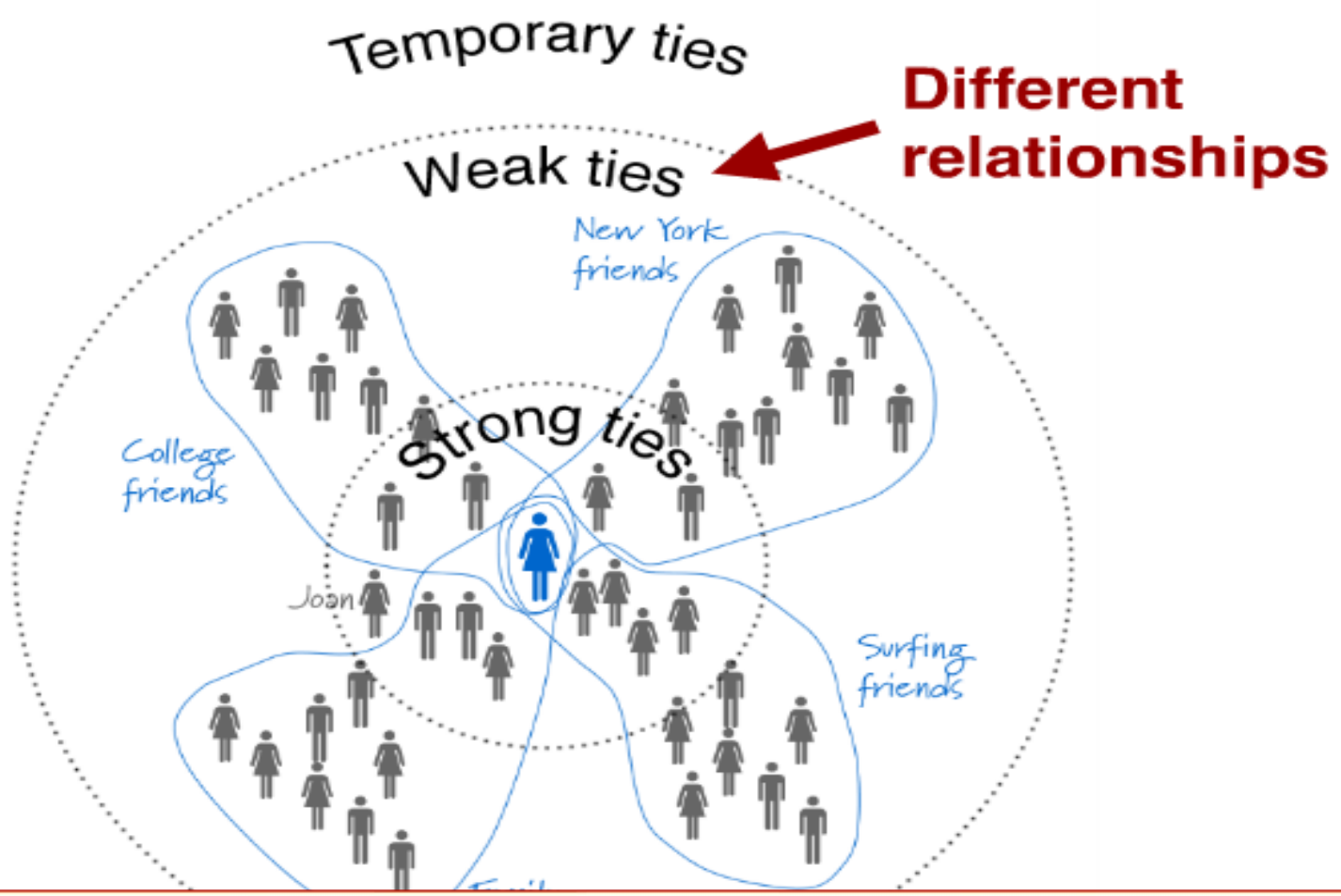


Graphs neural processing

«Take advantage of functional dependencies»

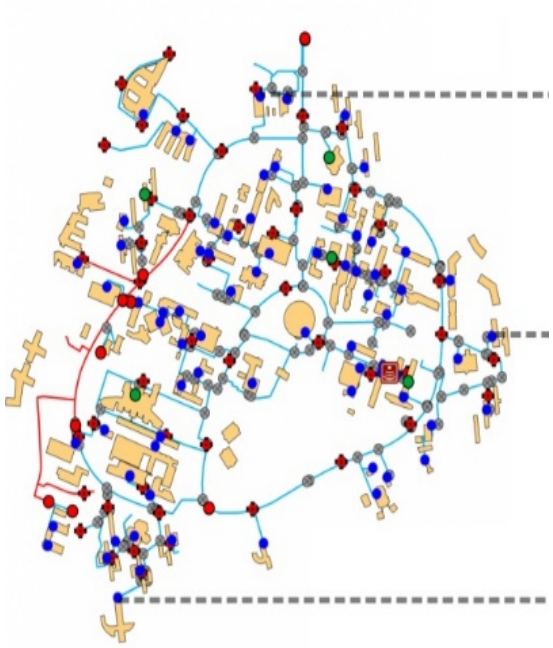
Why graphs

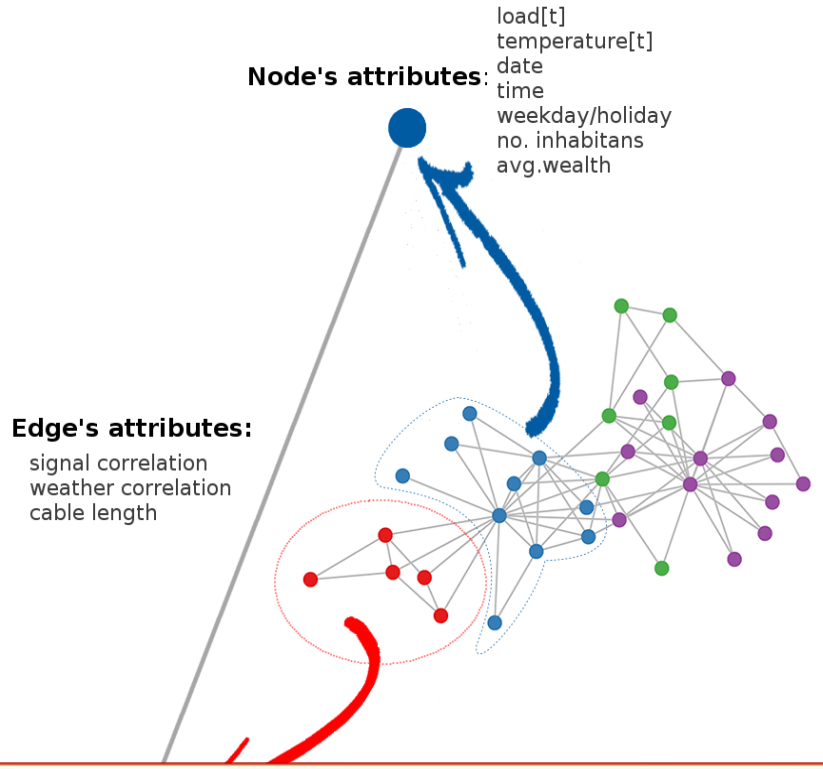

g_1 g_2 g_3 g_4 g_5 ... $g_{\tau-1}$ g_τ $g_{\tau+1}$ $g_{\tau+2}$ $g_{\tau+3}$ $g_{\tau+4}$ $g_{\tau+5}$ ...
time t

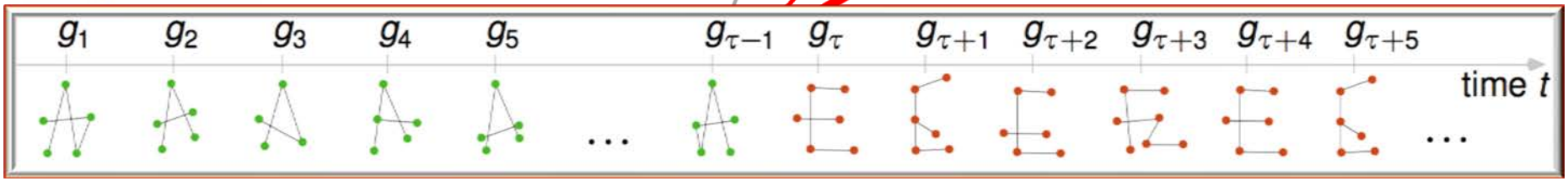

Temporary ties
Weak ties
Strong ties
Different relationships
New York friends
College friends
Joan
Surfing friends

Data streams and graph streams

- **In others, we derive graphs from signals**

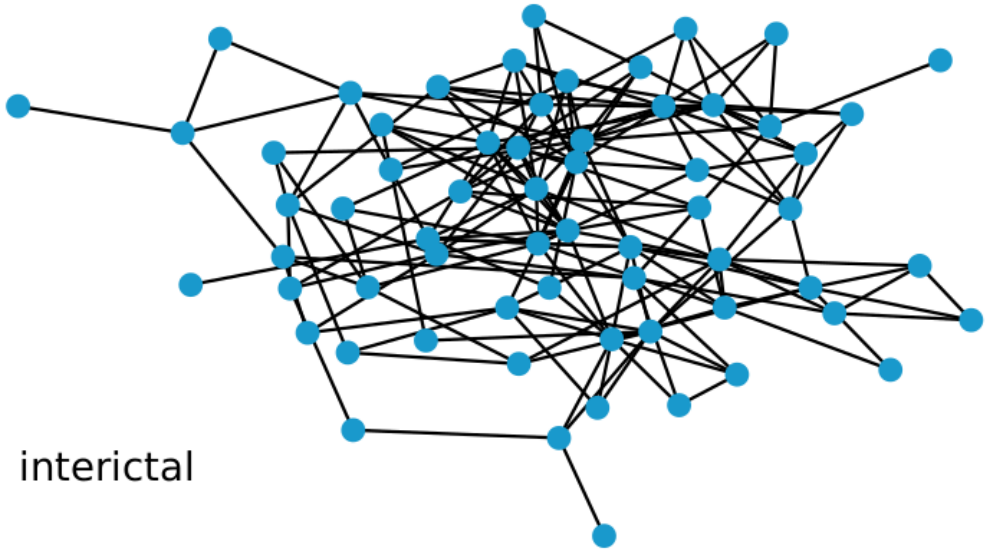




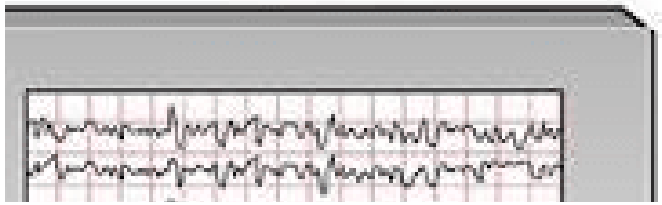


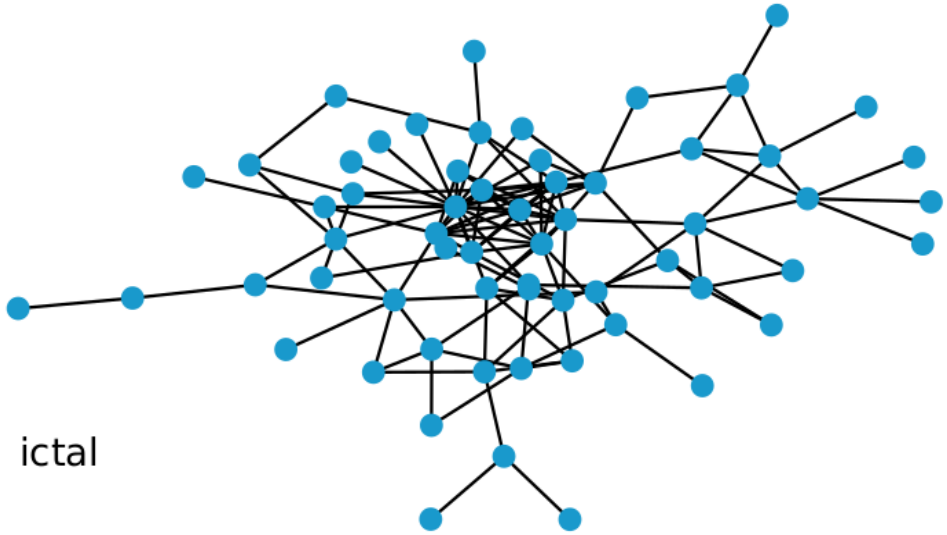
Data streams and graph streams

- **Functional connectivity networks for EEG (Epilepsy)**

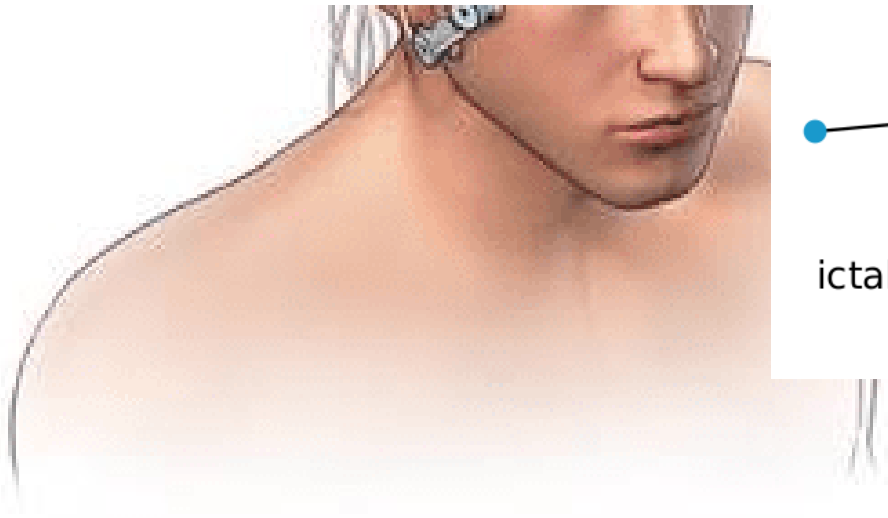


interictal





ictal



Representing a graph

Ad

0	0	1	1	0
0	0	1	0	1
1	1	0	1	1
1	0	1	0	1
0	1	1	1	0

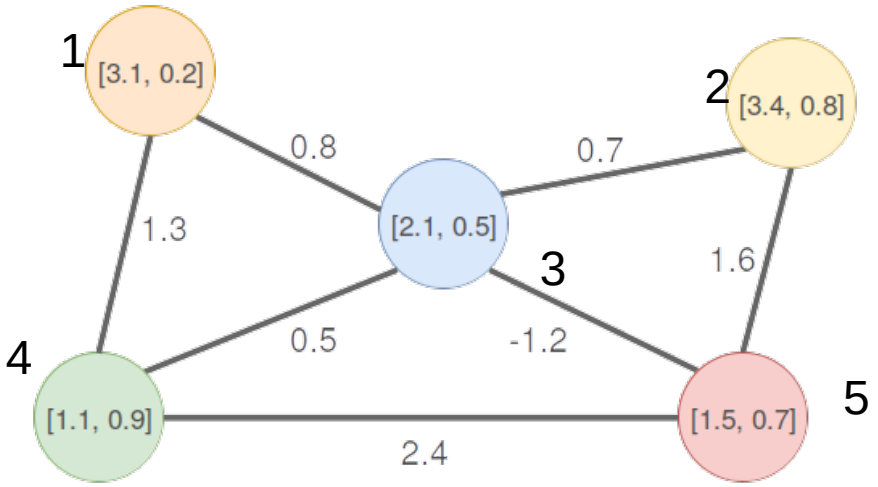
X

3.1	0.2
3.4	0.8
2.1	0.5
1.1	0.9
1.5	0.7

E

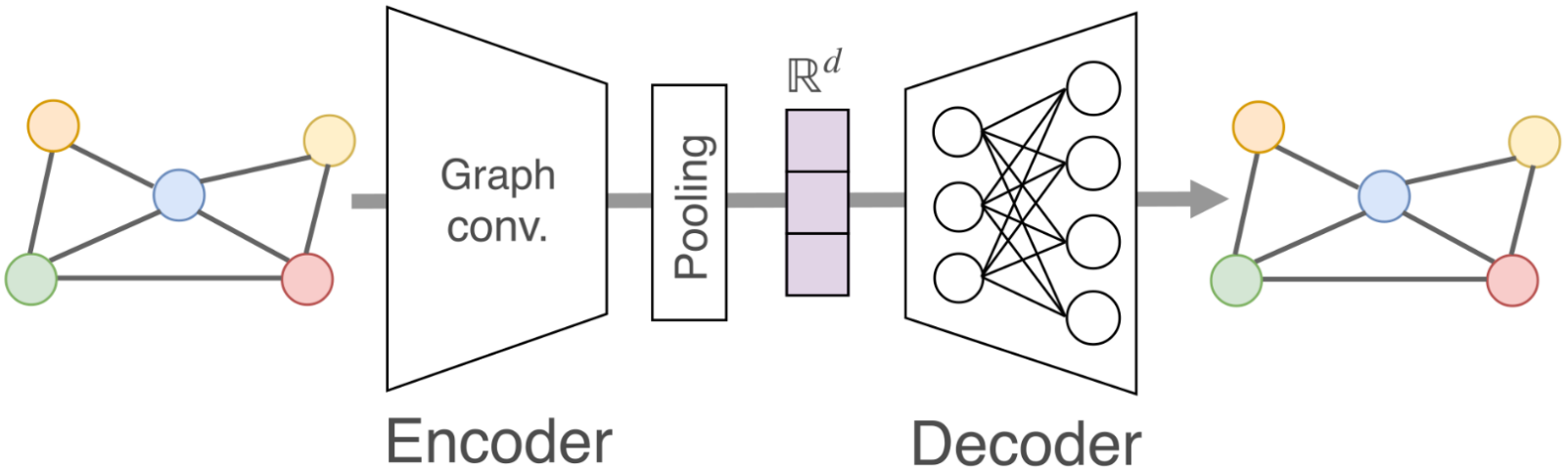
0.	0.	0.8	1.3	0.
0.	0.	0.7	0.	1.6
0.8	0.7	0.	0.5	-1.2
1.3	0.	0.5	0.	2.4
0.	1.6	-1.2	2.4	0.

- **Ad**: binary adjacency matrix
- **X**: node features
- **E**: edge features



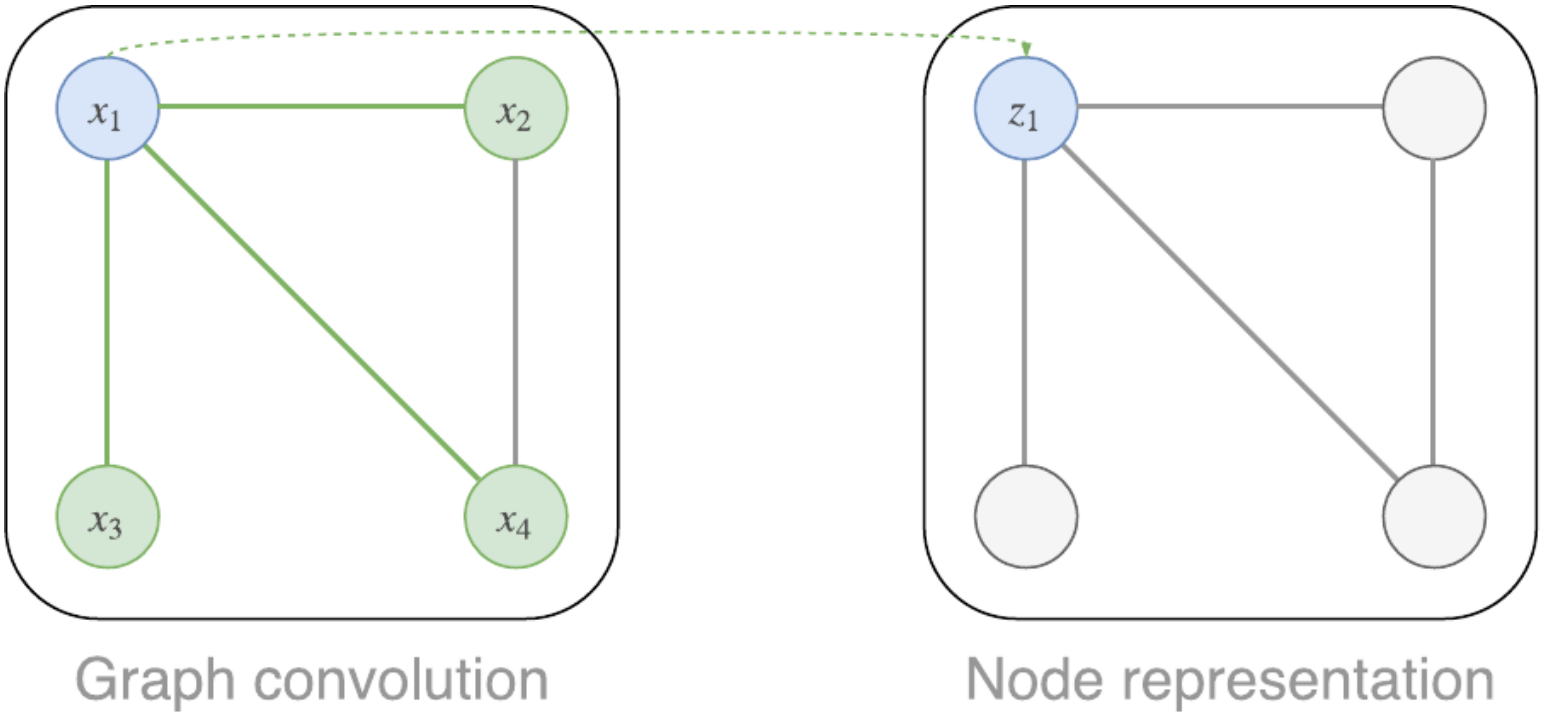
Graph autoencoders

- The encoder is composed of graph convolutional layers with the pooling one
- A dense decoder reconstructs the matrices describing the graph



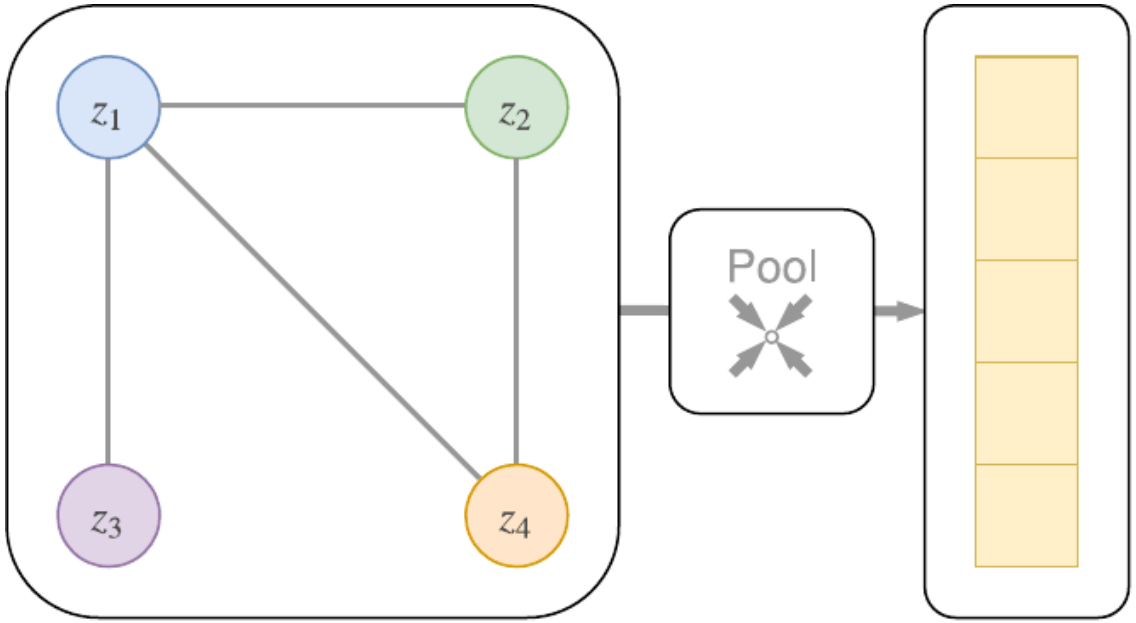
Graph Convolution

- Graph convolution works by exploiting the local neighborhood of each node to compute a node embedding



Global Graph Pooling

- To obtain a whole-graph embedding, we do global pooling. All node information is aggregated into a vector.



Conclusions

- Deep (machine) learning is a hype and everyone wants to go deep
- Deep learning is neither white nor black magic:
  - keep in mind application properties
  - basics coming from the theory
- Enjoy the existence of open source libraries