Politecnico di Torino

Machine Learning & Deep Learning

Soluzioni di grafica 3D in applicazioni biometriche

Leonardo Tanzi
Ph.D. Student
leonardo.tanzi@polito.it

Table of Contents

Machine Learning

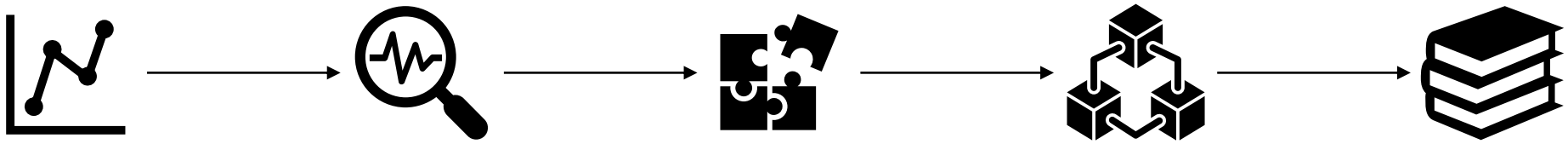
Definition

Application of Artificial Intelligence that provides a system with the ability to learn automatically from experience without being explicitly programmed for the given task.

Arthur Samuel - 1959

A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

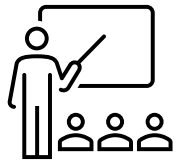
Tom Mitchell - 1998

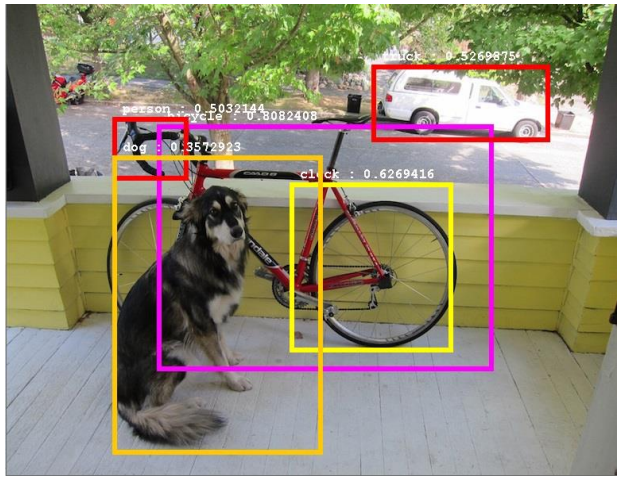


Input Data → **Analyze Data** → **Find Patterns** → **Predict** → **Learn from Feedback**

Types of Learning

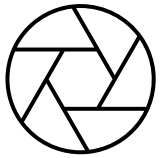
Supervised Learning

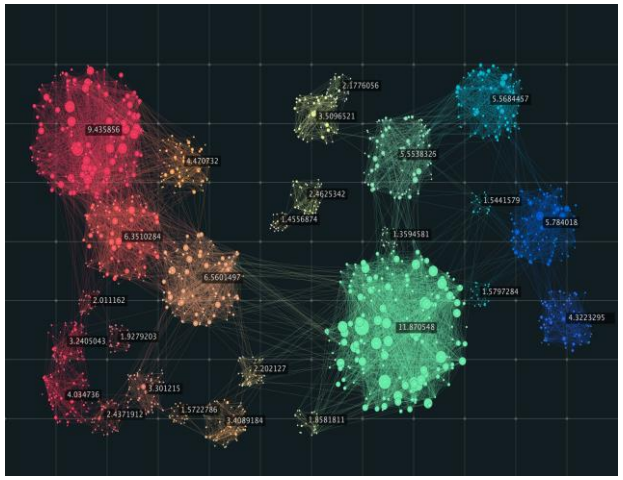




The machine learns from labeled data

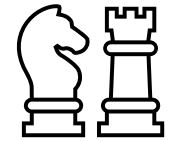
Unsupervised Learning

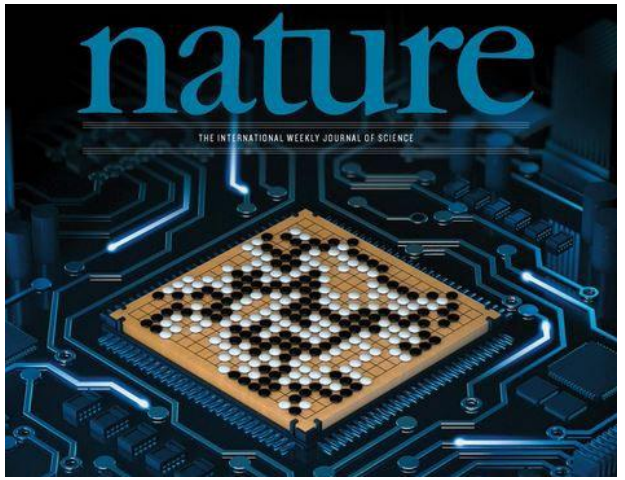




The machine learns without labeled data

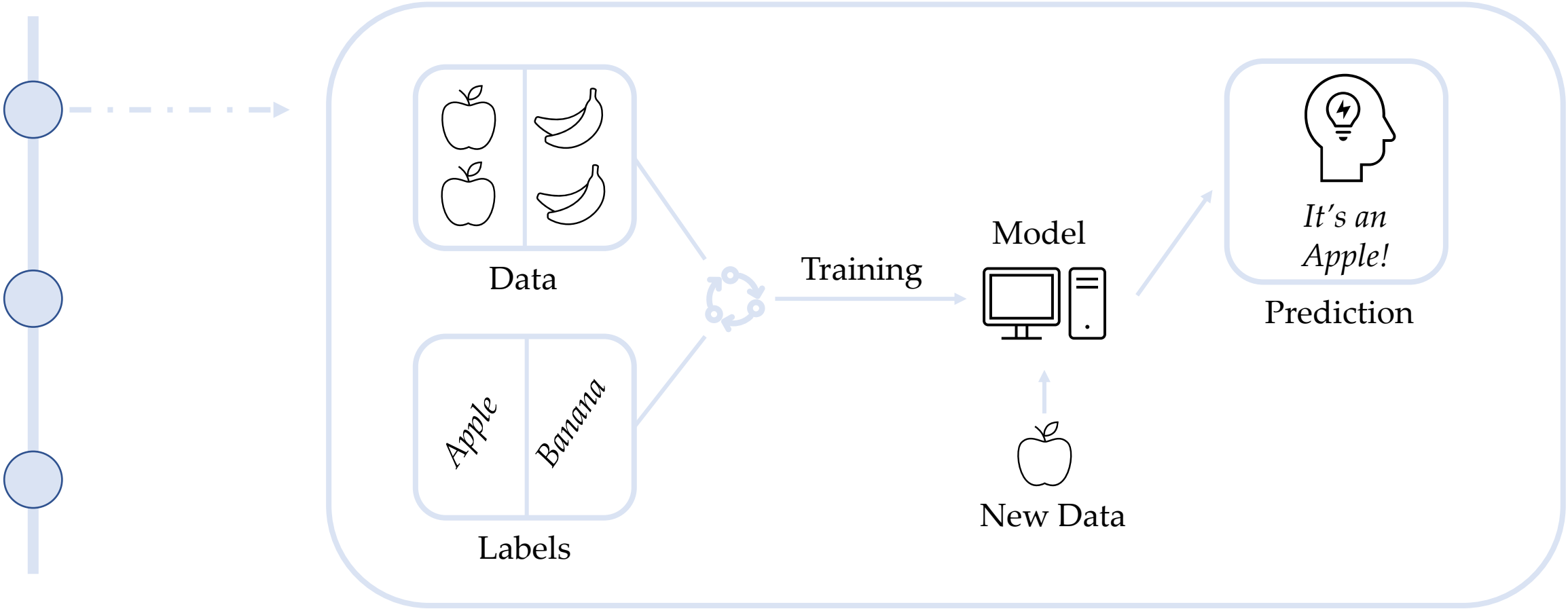
Reinforcement Learning



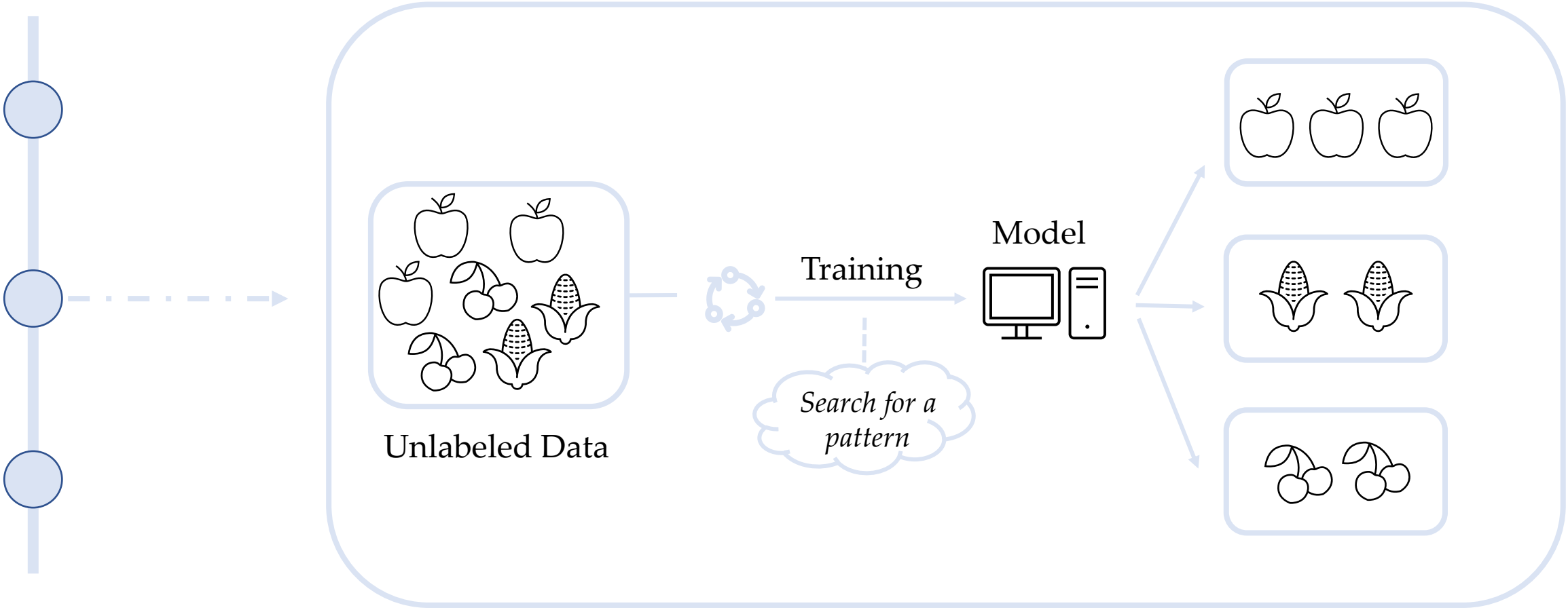


The machine learns on its own

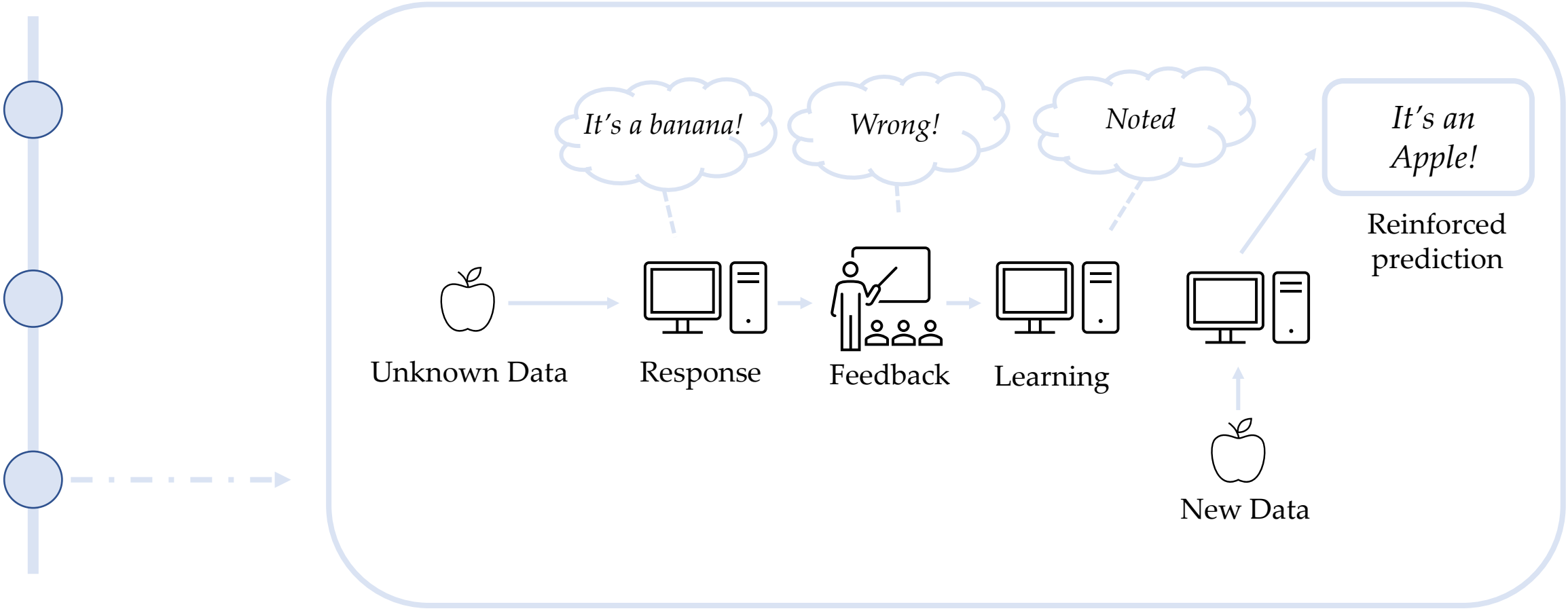
Supervised Learning



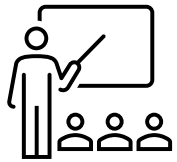
Unsupervised Learning



Reinforcement Learning



Approaches

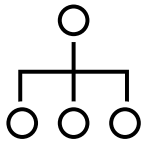


Supervised Learning

Regression

Used when the output is continuous (ex. price of a house)

Linear Regression

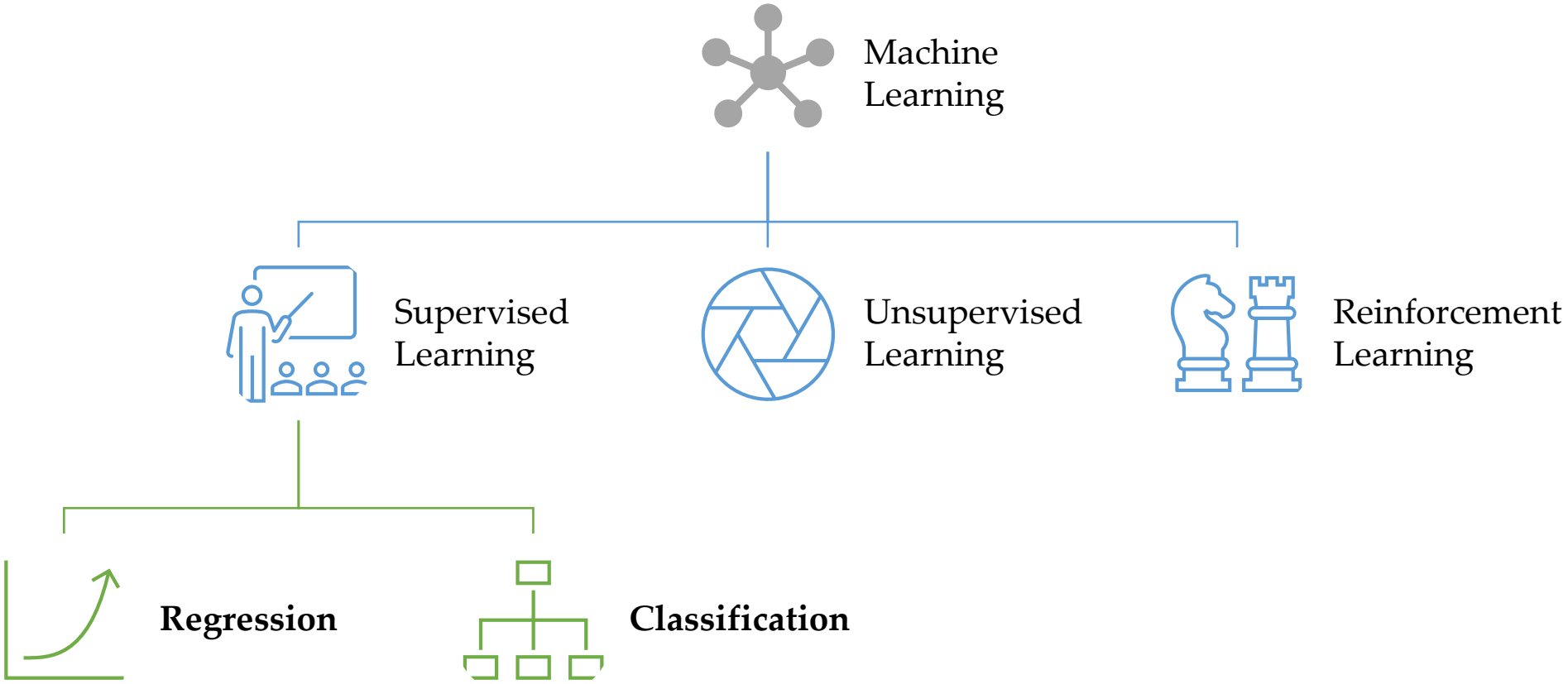


Classification

Used when the output is categorical (ex. *Yes* or *No*)

Logistic Regression
KNN
Decision Trees
SVM

Summary

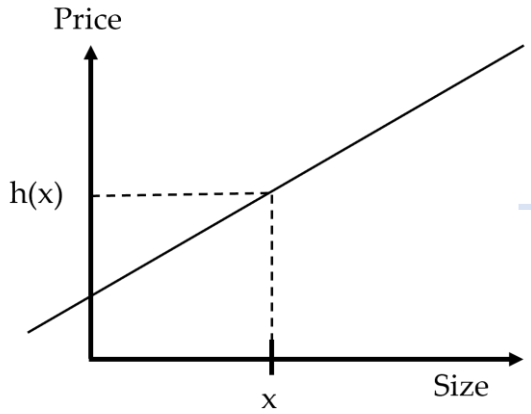

Machine Learning
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Regression
Classification

Regression: Definition

Size (m^2)	Price (€)
100	120.000
53	80.000
220	340.000
78	110.000

Input variables

Regression Model



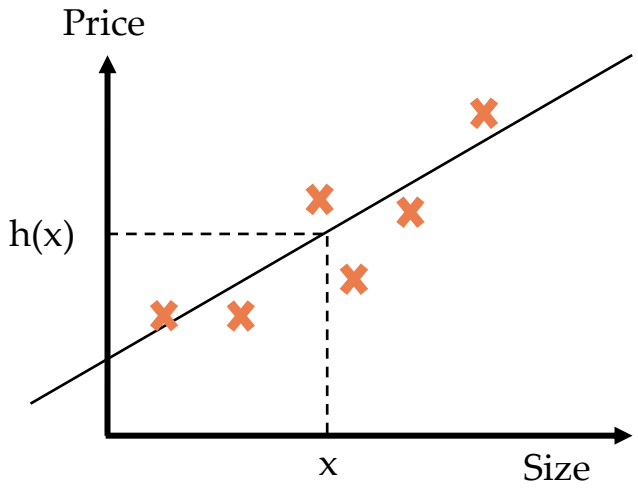
New Size
131m^2

145.000€

Continuos
Output Variable

Regression: Linear Regression

House Price Prediction: estimate the price of a house given its size



We want to fit a line that best approximates the behavior of the training data.

We call this function hypothesis **h(x)** and it is a function of $\boldsymbol{w_1}$ (slope) and $\boldsymbol{w_0}$ (y-intercept)

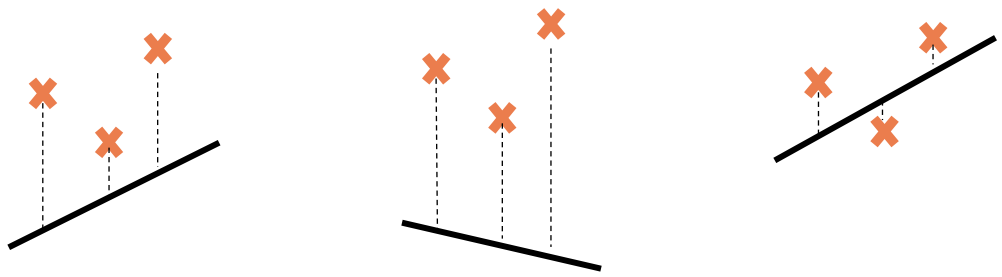
$$h(x) = w_0 + w_1 x$$

→ **How can we fit this line?**

Find the values of w_0 and w_1 which minimizes the difference between the prediction and the actual values

Regression: Loss and Gradient Descent

Loss



$$h_w(x) = w_0 + w_1 x$$

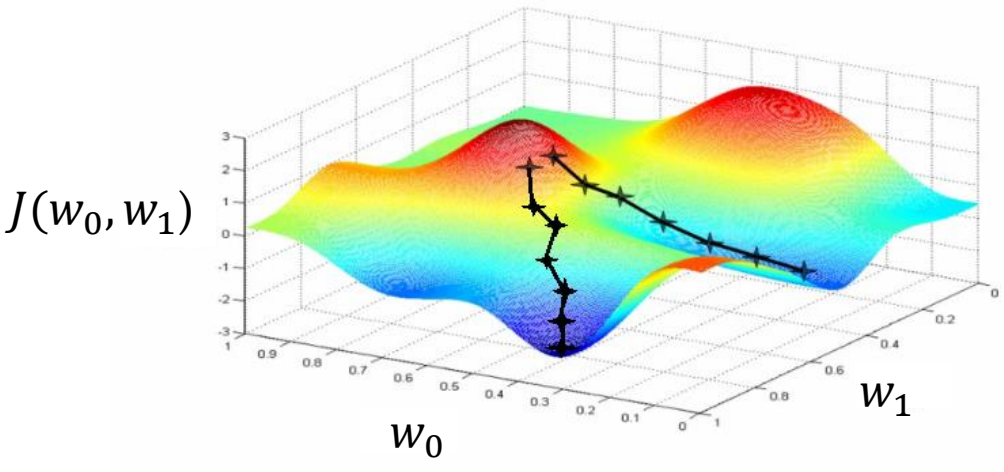
Changing w_0 → move the line up and down
Changing w_1 → change the slope

Squared Error: $J(w_0, w_1) = \frac{1}{2} \sum (h_w(x) - y)^2$

↓

Should be minimized!

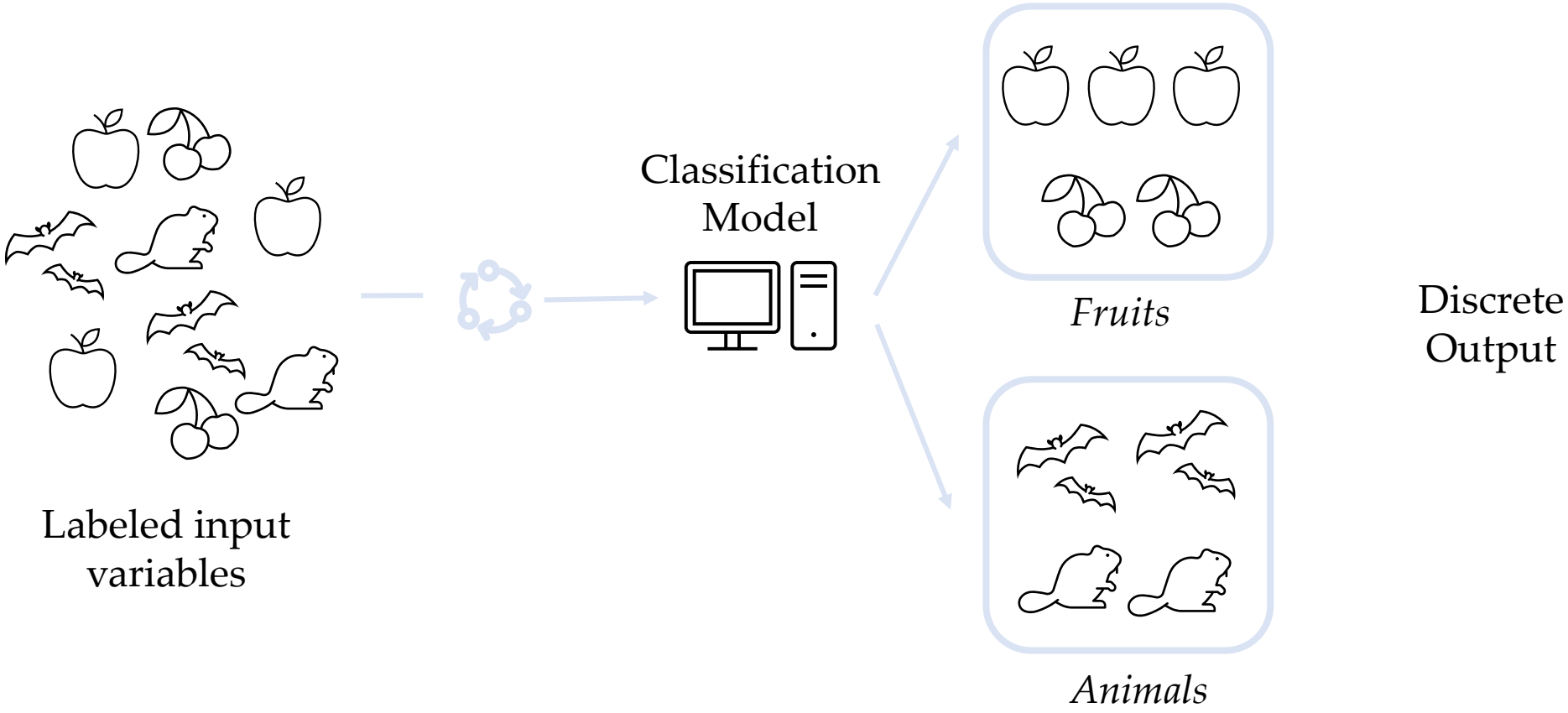
Gradient Descent



In the beginning, the line has random w_0 and w_1. **Gradient descent** moves the point (by updating the values of w_0 and w_1) in the direction where the **Loss** function decrease.

This process is repeated until a minimum is reached.

Classification: Definition



Classification: Important Terminologies

Keyword	Definition
Classifier	An algorithm used to map the input data to a specific category
Classification model	The model that predicts a class given an input data
Feature	An individual measurable property of the observed data
Train	The process where the model learn to distinguish between classes
Predict	The process where the model make a prediction on unseen data
Binary classification	Condition with two outcomes which are either true or false
Multi-class classification	Condition with more than two classes where each sample is assigned to one and only label
Multi-label classification	Condition where each sample is assigned to a set of labels or targets

Classification: Metrics

Training a classifier to predict whether a person is diabetic or healthy (binary output). Four possible outcomes:

These are good

True Positive **TP** Pred: diabetic Correct: diabetic	True Negative **TN** Pred: healthy Correct: healthy
False Positive **FP** Pred: diabetic Correct: healthy	False Negative **FN** Pred: healthy Correct: diabetic

These are bad (especially FN)

How many people were correctly labeled among all the people? → $Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$

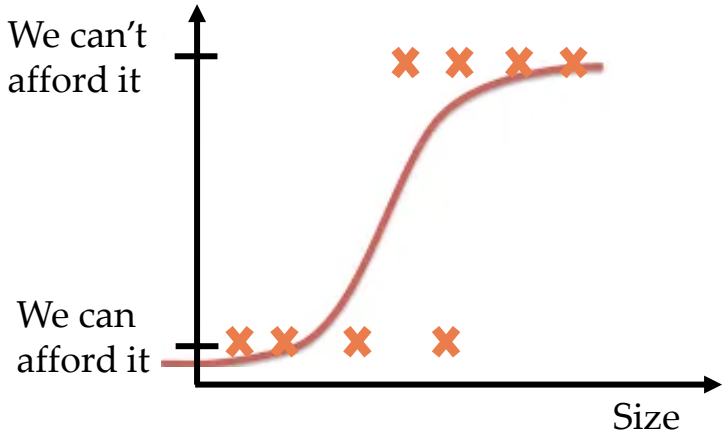
How many of those labeled as diabetic are actually diabetic? → $Precision = \frac{TP}{TP + FP}$

Of all the diabetic people, how many were correctly predicted? → $Recall/Sensitivity = \frac{TP}{TP + FN}$

Of all the healthy people, how many were correctly predicted? → $Specificity = \frac{TN}{FP + TN}$

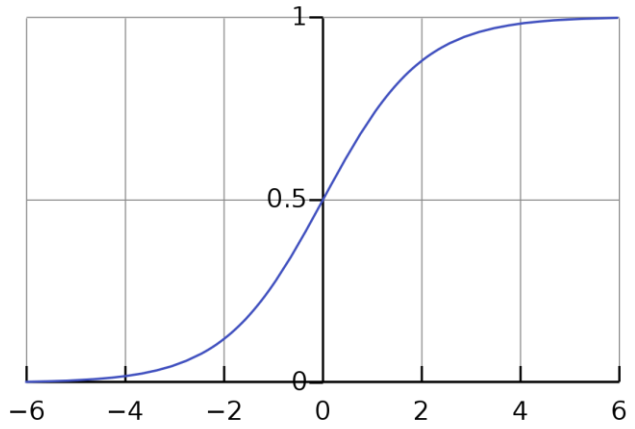
Classification: Logistic Regression

In linear regression, we computed a continuous value (the price of a house given its size).
In logistic regression, we want to output a **discrete** value $\hat{y}$. For example, in a binary problem, given the size of a house, we want to build an algorithm that returns 0 if we can afford it and 1 if we can't.



We need an S-shaped function to fit this data

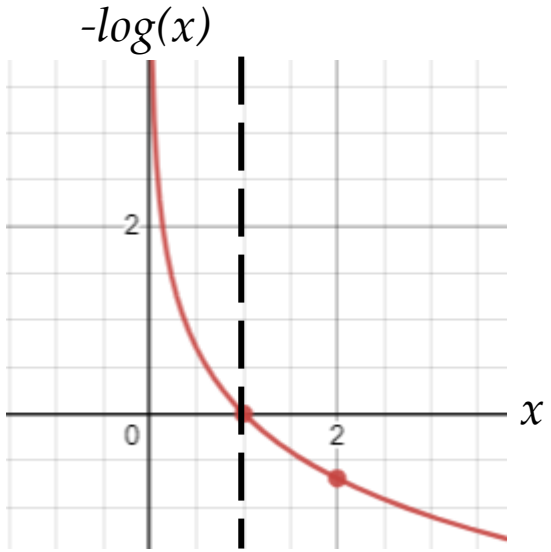
Sigmoid



$$y = \frac{1}{1 + e^{-x}}$$

Classification: Logistic Regression

Starting from the linear regression function $h_w(x) = w_0 + w_1 x$, we add a Sigmoid function that squeezes the output between 0 and 1, in this way, the system will predict values between 0 and 1 (values > 0.5 will be assigned to one class while values < 0.5 to the other): $\boldsymbol{h_w(x) = \sigma(w_0 + w_1 x)}$



NB: in our case, x has a value between 0 and 1, for this reason, the minimum value is obtained for $x = 1$ and not for $x > 1$

Loss function → **has to be minimized**!

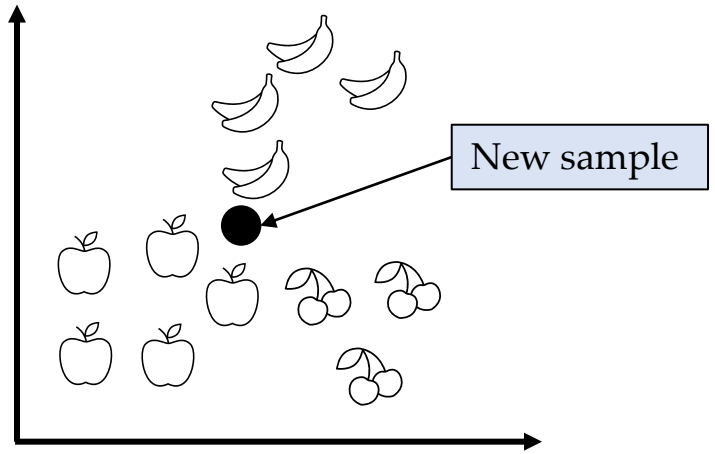
$$J(\hat{y}, y) = -(y\log(\hat{y}) + (1 - y)\log(1 - \hat{y}))$$

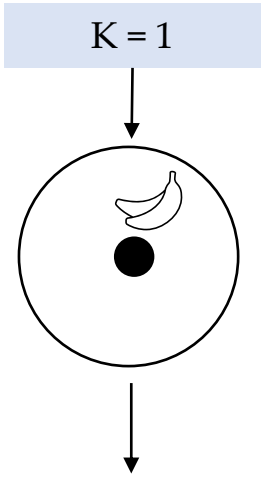
If y = 0 → $J(\hat{y}, y) = -\log(1 - \hat{y})$ → When the input y is 0 we want $\hat{y}$ close as 0 as possible. The minimum value of $-\log(1 - \hat{y})$ is obtained when $\hat{y}$ =0, because $-\log(1 - 0) = -\log(1) = 0$

If y = 1 → $J(\hat{y}, y) = -\log(\hat{y})$ → When the input y is 1, we want $\hat{y}$ close as 1 as possible. The minimum value of $-\log(\hat{y})$ is obtained when $\hat{y}$ =1, because $-\log(1) = 0$

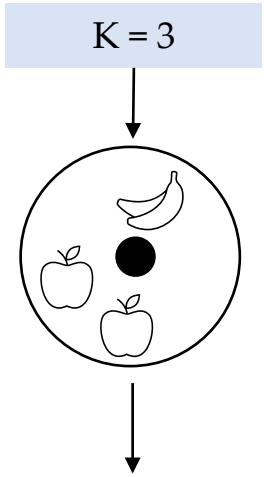
Classification: KNN Intuition

K Nearest Neighbors
We have a distribution, when a new sample arrives, we assign it to a class looking at the classes of the *k* nearest samples



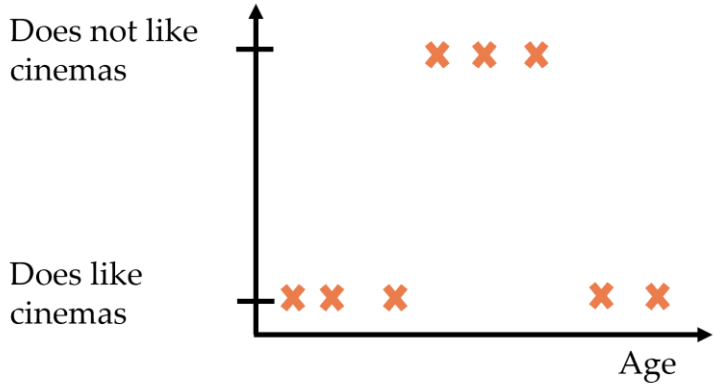


We assign it to the *banana* class



We assign it to the *apple* class

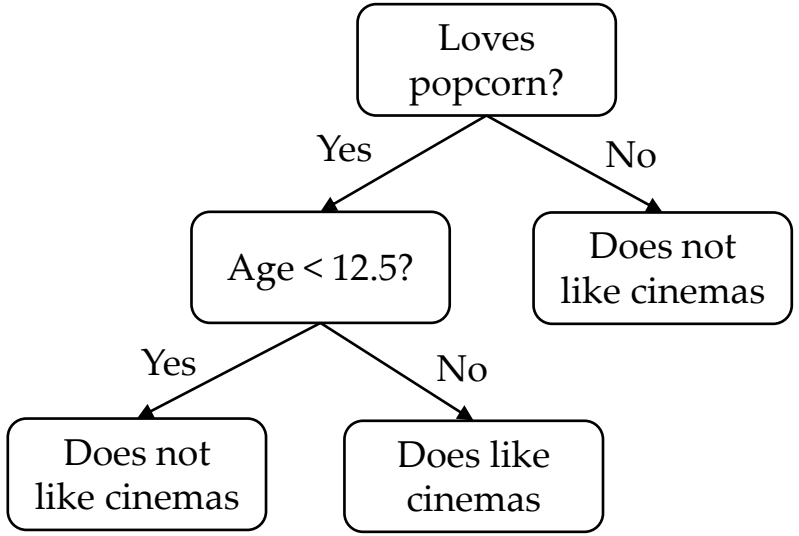
Classification: Decision Trees Intuition



If we have a distribution like this, we can not fit a S-shaped function as we showed for logistic regression

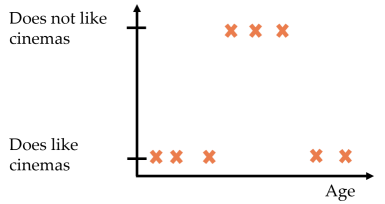
Loves popcorn	Age	Likes cinemas
Yes	7	No
Yes	12	No
No	18	Yes
No	35	Yes
Yes	38	Yes
Yes	50	No
No	83	No

Based on this data, we can build a Decision Tree that is able to return the prediction for a new person



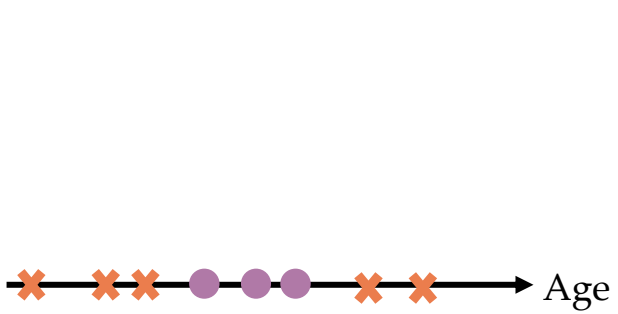
Classification: SVM Intuition

We have just seen that we cannot fit an S-shaped function to this specific distribution

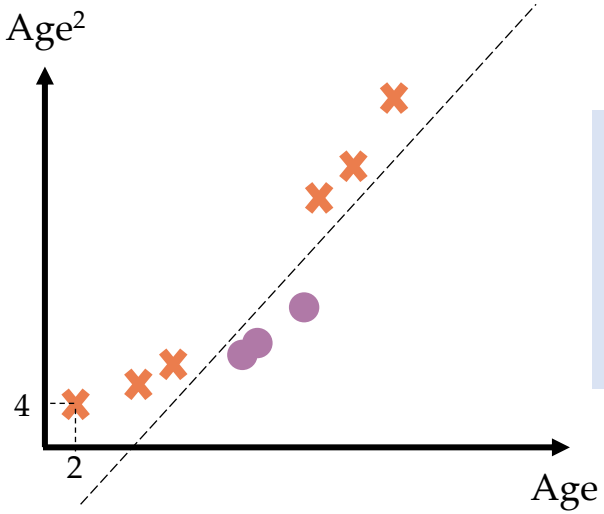


Support Vector Machine

We represent data in the lowest dimension possible, and with SVM we can add a new axis to the data and move the points in a way that makes it relatively easy to draw a **straight line** that correctly classifies people



Add y-axis



Support Vector Machines use something called **Kernel Functions** that we can be used to *systematically* find higher dimensions' axis

Deep Learning

Introduction

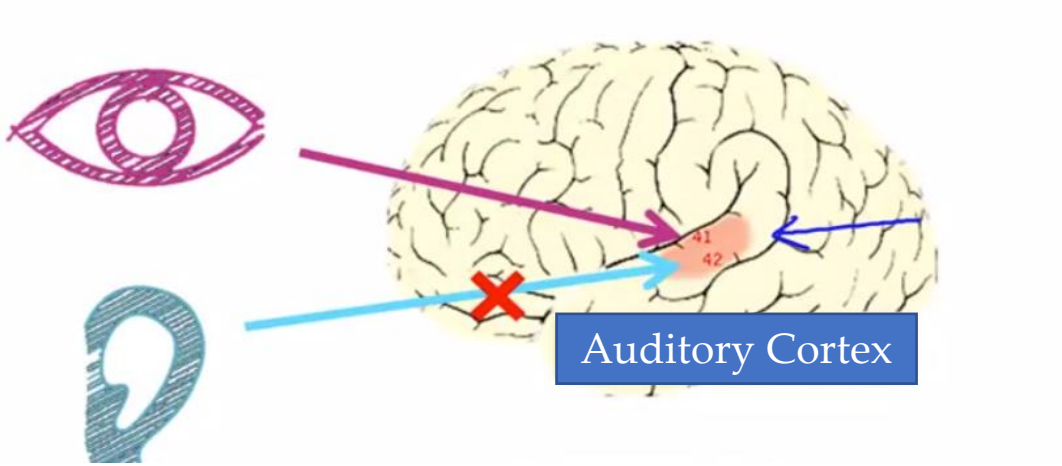
Many features / Big dataset → Classic ML algorithm become inefficient→ Alternative: **Deep Learning**

Deep Learning is a branch of Machine Learning which utilizes neural network for computation

Main idea

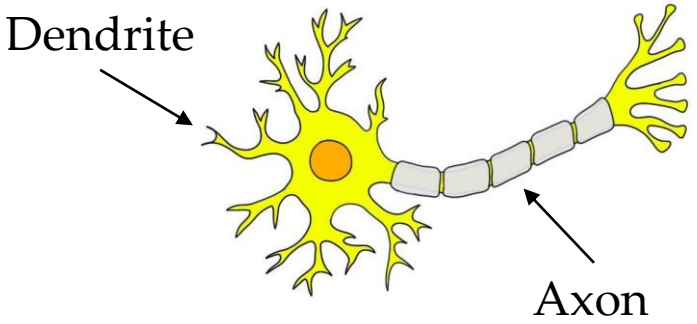
Neural networks are algorithms inspired by the human brain.

The brain uses just a single learning algorithm that can do anything, can we find an approximation of what the brain does and try to implement it?



Neuron

A biological neuron in the brain receives inputs from the *dendrites*, performs some computation, and outputs the result through the *axons*

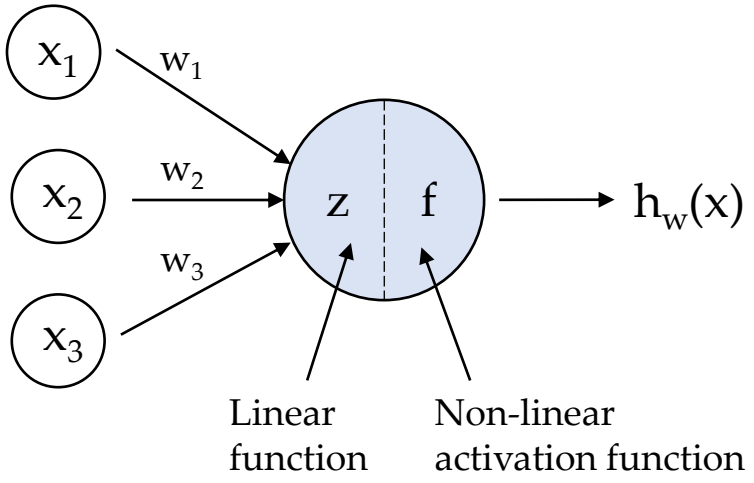


An artificial neuron tries to mimic its behavior:
it receives some inputs (x_1, x_2, x_3), computes a linear function operation using **weights** (w_1, w_2, w_3)

$$z = w_1 x_1 + w_2 x_2 + w_3 x_3$$

and applies a **non-linear** activation function to formulate the hypothesis

$$h_w(x) = g(z)$$



Neural Network

A neural network is a group of these neurons.

Each neuron has a linear combination of inputs and weights and then applies non-linear activation, for example, for the first layer:

$$a_1^{(1)} = g(w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 + w_{31}^{(1)}x_3)$$

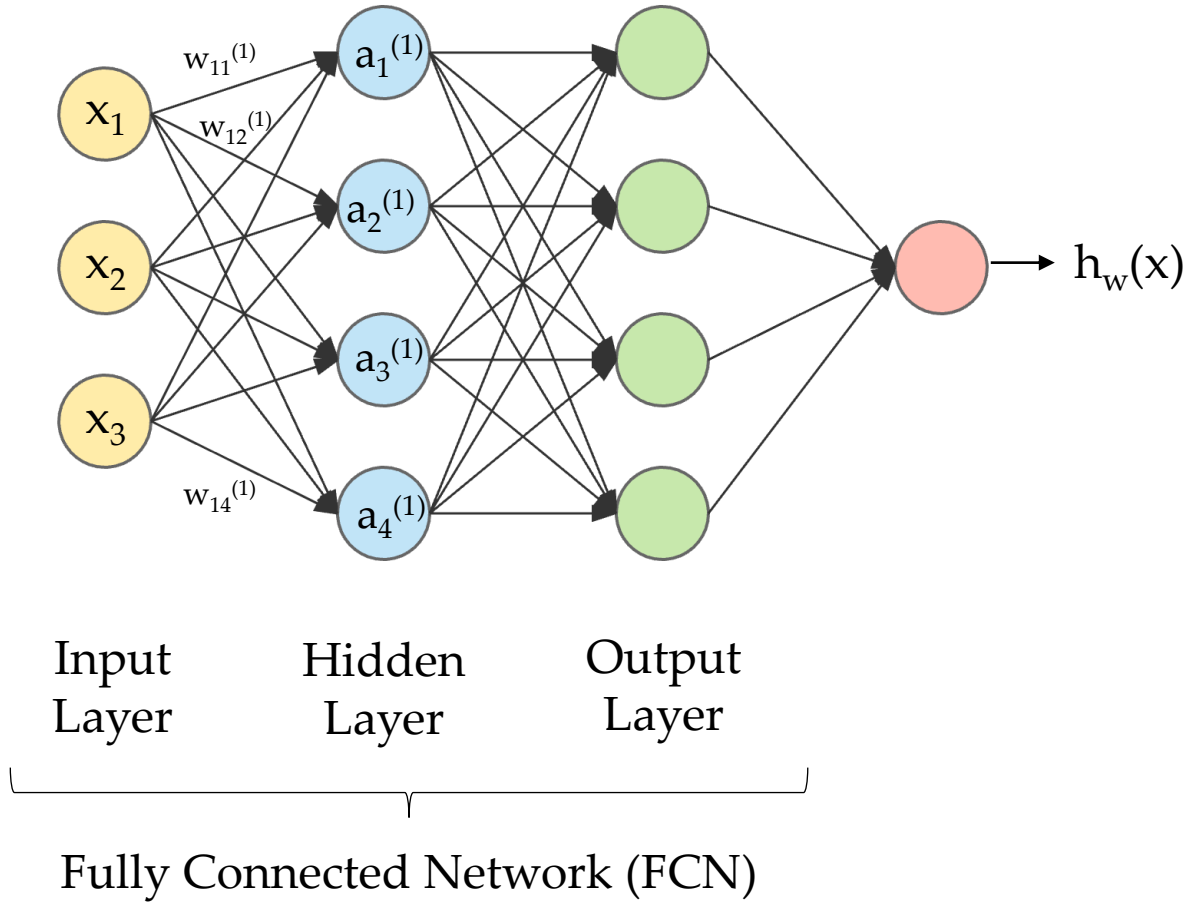
$$a_2^{(1)} = g(w_{12}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{32}^{(1)}x_3)$$

$$\vdots$$

Notation

$a_n^{(k)}$ → n indicates the neuron and k the layer

$w_{ij}^{(k)}$ → i is the index of the neuron of the input layer, j the index of the hidden layer, and k is the layer



Neural Network

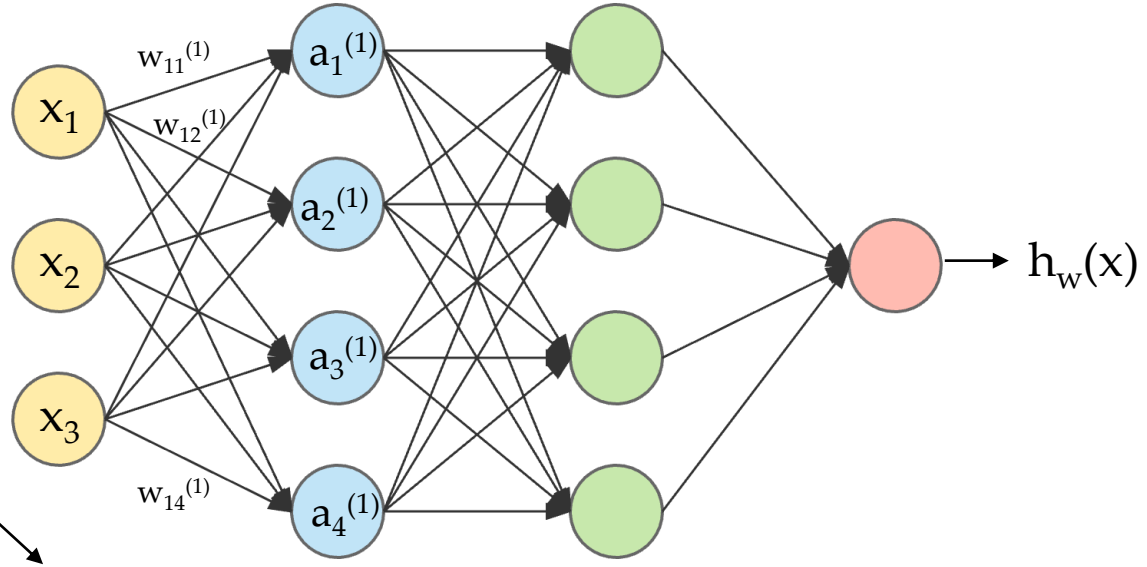
A neural network is a group of these neurons.

Each neuron has a linear combination of inputs and weights and then applies non-linear activation, for example, for the first layer:

$$a_1^{(1)} = g(w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 + w_{31}^{(1)}x_3)$$

$$a_2^{(1)} = g(w_{12}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{32}^{(1)}x_3)$$

$$\vdots$$

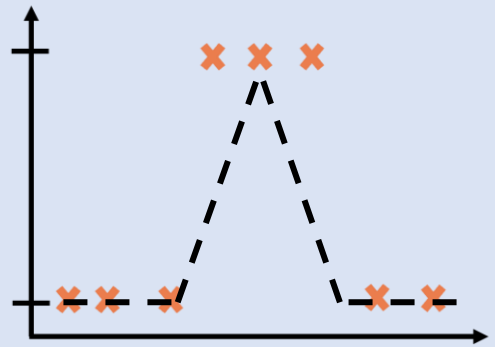


Notation

$a_n^{(k)}$ → n indicates the neuron and k the layer

$w_{ij}^{(k)}$ → i is the index of the neuron of the input layer, j the index of the hidden layer, and k is the layer

Non-linear activation functions make neural networks flexible and able to fit just about any data



Neural Network

For the second layer:

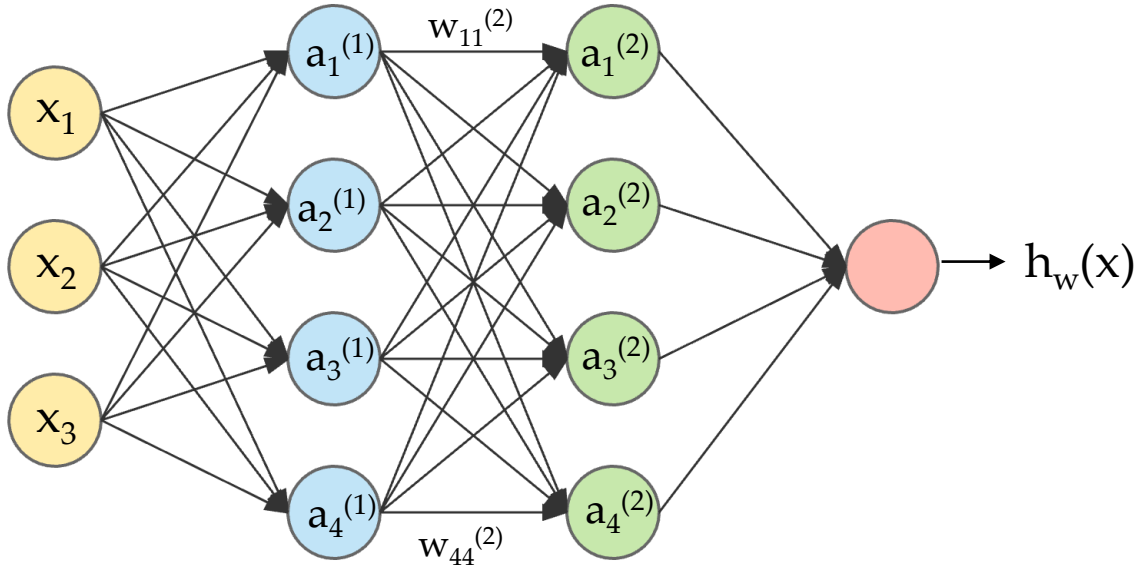
$$a_1^{(2)} = g(w_{11}^{(2)} a_1^{(1)} + w_{21}^{(2)} a_2^{(1)} + w_{31}^{(2)} a_3^{(1)} + w_{41}^{(2)} a_4^{(1)})$$

$$\vdots$$

And finally:

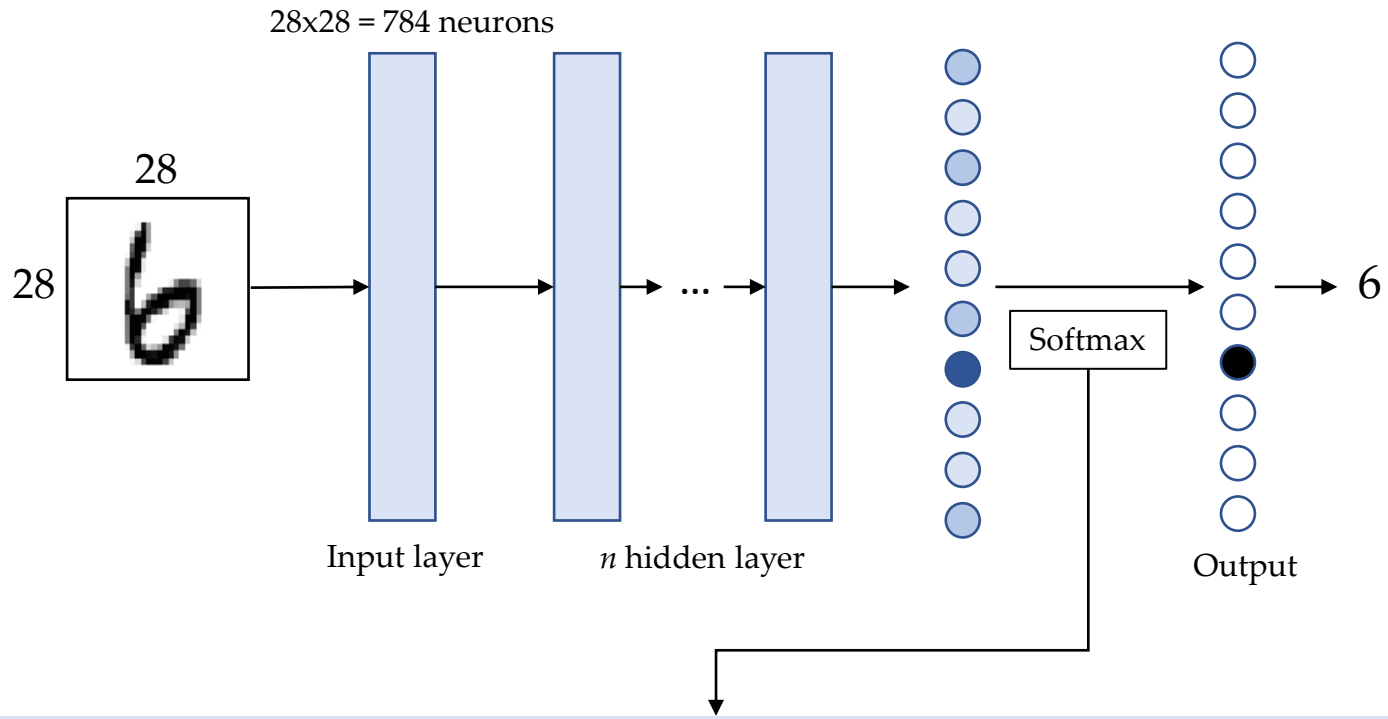
$$h = g(w_{11}^{(3)} a_1^{(2)} + w_{21}^{(3)} a_2^{(2)} + w_{31}^{(3)} a_3^{(2)} + w_{41}^{(3)} a_4^{(2)})$$

Each $w^{(k)}$ is a matrix of weights that controls the mapping from layer k to layer $k+1$



Neural Network Classification

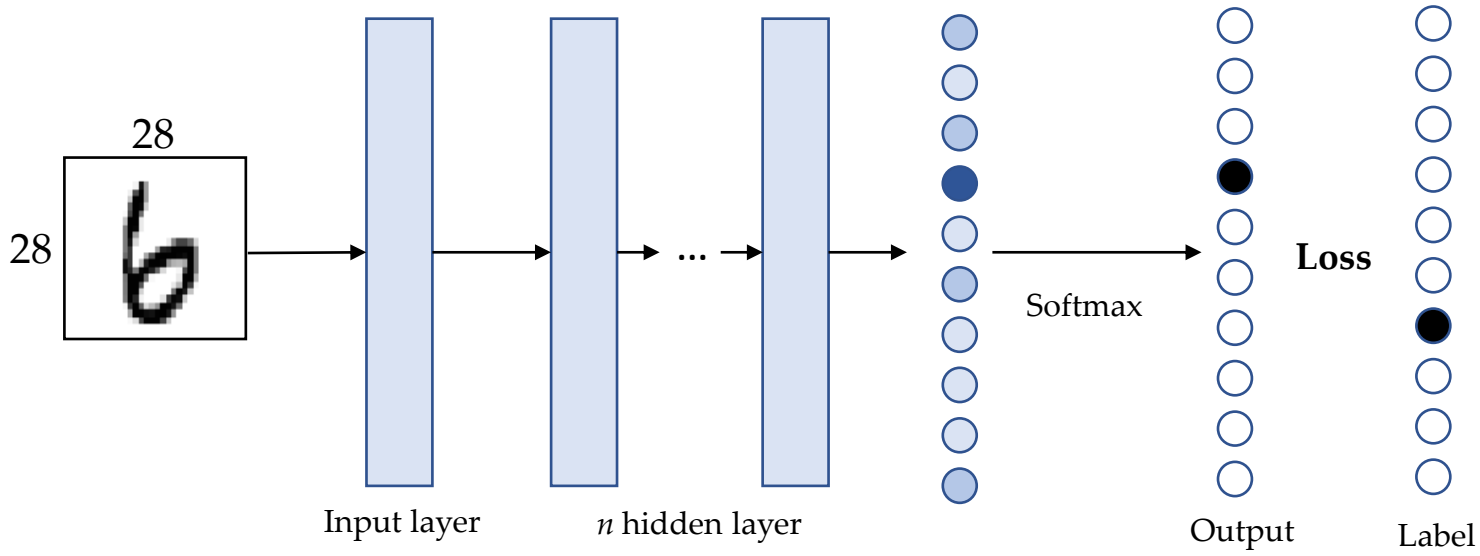
MNIST
10 handwritten digits



Softmax is a function used for multiclass classification that turns a vector of K real values into a vector of K real values that sum to 1

[2.33, -1.46, 0.56] $\xrightarrow{\text{Softmax}}$ [0.83, 0.01, 0.14]

Neural Network Training



One of the most used functions for multi-class classification is **cross-entropy loss**, defined as:

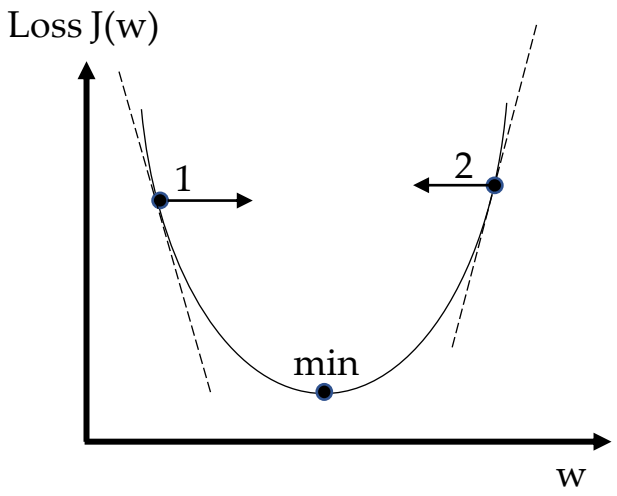
$$CE = -\sum_i y_i \log(\hat{y}_i)$$

y_i is the label vector with one-hot encoding, meaning that the number "6" is coded as [0 0 0 0 0 0 **1** 0 0 0].
$\hat{y}_i$ is the (wrong) prediction after the Softmax, let's say [0.01, 0.02, 0.11, 0.70, 0.01, 0.08, **0.03**, 0.01, 0.02, 0.01], which is actually a "3".

Thus, CE will be equal to -log(0.03) as all the other values are multiplied by 0. Given the minus sign, in minimizing this function, we are trying to maximize the 6th value in the output vector.

Gradient Descent & Backpropagation

As we already said, the loss function represents how much the prediction is similar to the actual values. For this reason, the more we minimize the loss function, the more the prediction will be accurate. To do this, we need two powerful tools: gradient descent and backpropagation.



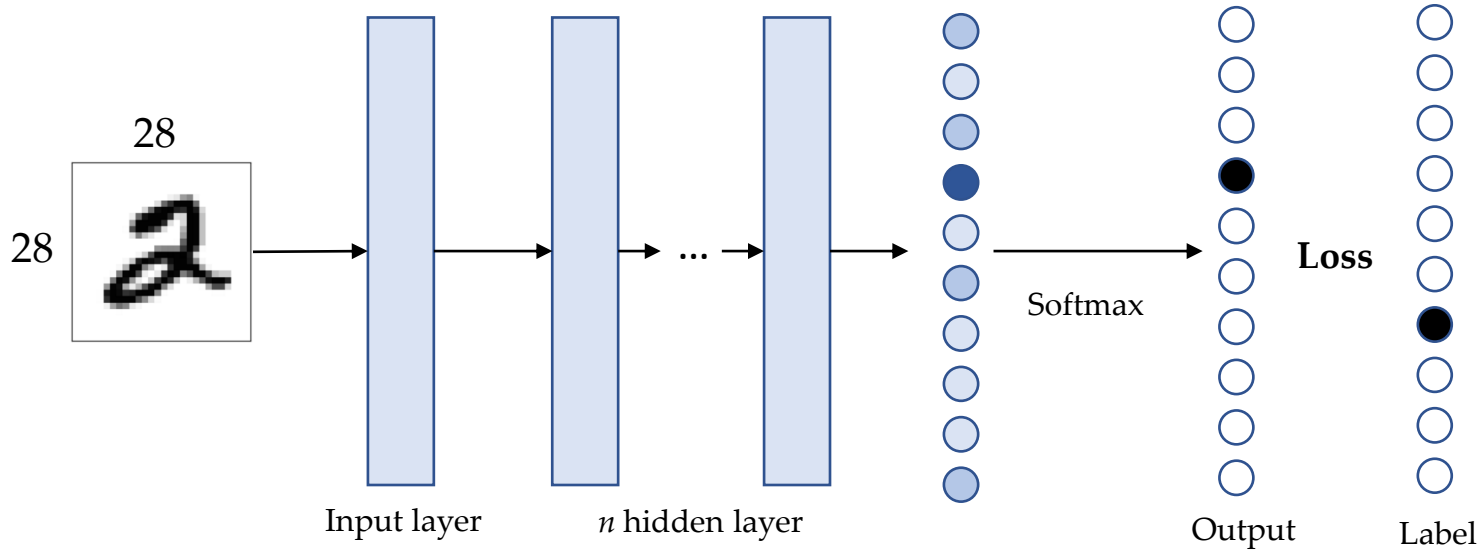
Backpropagation is the technique that allows to compute the derivative of the loss with respect to the parameters.

Optimization: we want to find the value of w which minimizes $J(w)$. To do this, we will update w at each step by the quantity:

$$w = w - \alpha \frac{dJ(w)}{dw}$$

α is the learning rate, which is a parameter that determines how big the steps are in the direction of the minimum.
$\frac{dJ(w)}{dw}$ is the derivative of $J(w)$ with respect to w, i.e., the slope. Thus, if we are in **1**, the slope is negative, so $-\alpha \frac{dJ(w)}{dw}$ will be a positive value, meaning that we are increasing w in the direction of the minimum. On the contrary, if we are in **2**, the slope is positive, so $-\alpha \frac{dJ(w)}{dw}$ will be a negative value, meaning that we are decreasing w in the direction of the minimum.

Neural Network Overview



In summary, during training, we take an image (or more than one, i.e., a *batch*), pass it through the network, and make a prediction. We use the prediction and the actual label to compute the loss. We then use backpropagation to compute the derivative of the loss with respect to all the weights. Finally, we apply gradient descent to adjust the value of the weights in the direction of the minimum. This process is repeated until the minimum (or a local minimum) of the loss is reached, meaning that the network has (hopefully) learned to distinguish the different classes.

Reminder: all this works with $28 \times 28 \times 1$ images, which can be flattened to just 784 neurons. But if we use high-res images, such as $1920 \times 1080 \times 3$, the input layer should have around 6 million neurons, becoming computationally infeasible

Important Terminologies

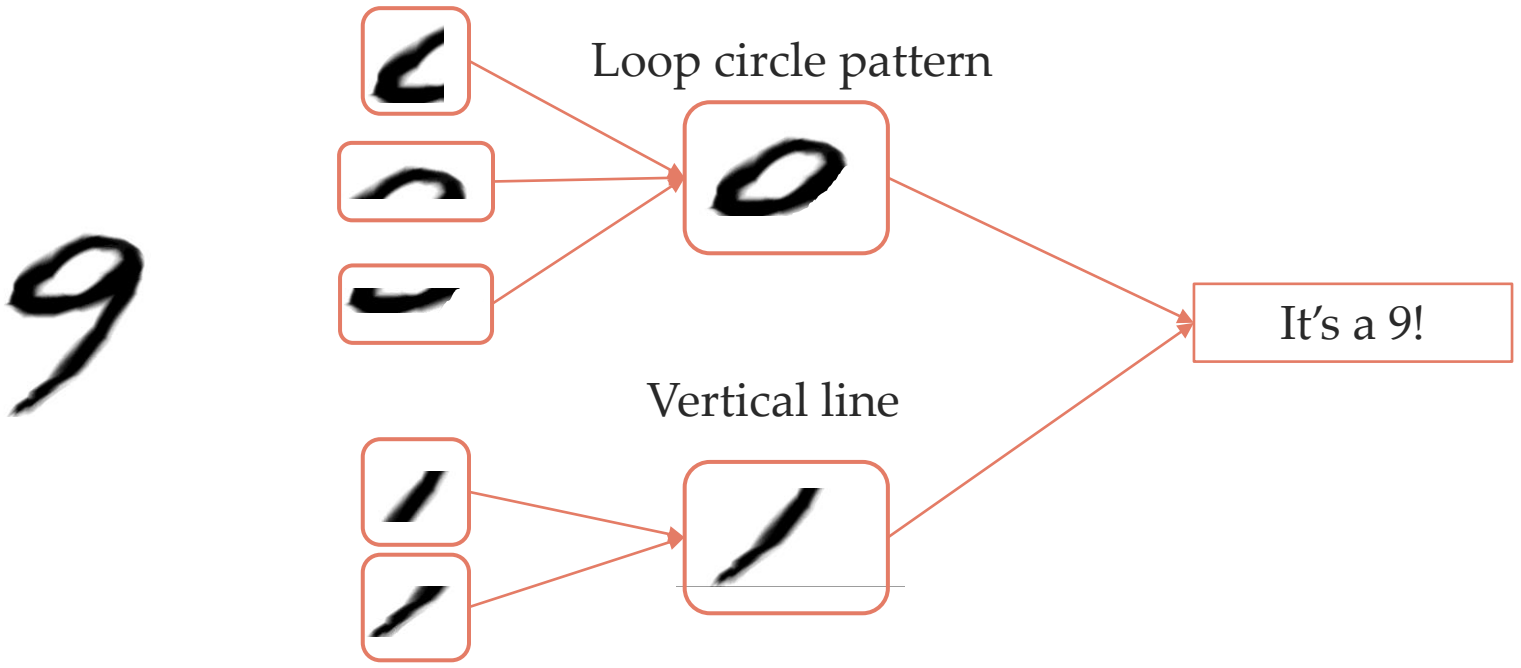
Keyword	Definition
Batch	The set of samples used in one iteration (that is, one gradient update) of training
Epoch	A full training pass over the entire dataset such that each example has been seen once
Learning rate	The size of the steps of the gradient descent algorithm
Overfitting	The model matches the training data so closely that it fails to make correct predictions on new data
Underfitting	The model has poor predictive abilities because it hasn't captured the complexity of the training data
Test/Train/Validation	The three subsets in which the main dataset is usually divided
Regularization	Techniques which penalize the model complexity and prevent overfitting
Transfer Learning	Transfer information from one model trained on a task into another task
Hyper-parameters	Not learnable parameters, such as learning rate, number of epochs, etc.

Convolutional Neural Network

Using neural networks with images has two main disadvantages:

1. Too much computation
2. They are sensitive to the location of an object in an image (if we move the object, the NN may be not able to recognize him)

How does the human brain recognize images?



This approach is replicated with filters in Convolutional Neural Networks

1	1	1
1	-1	1
1	1	1

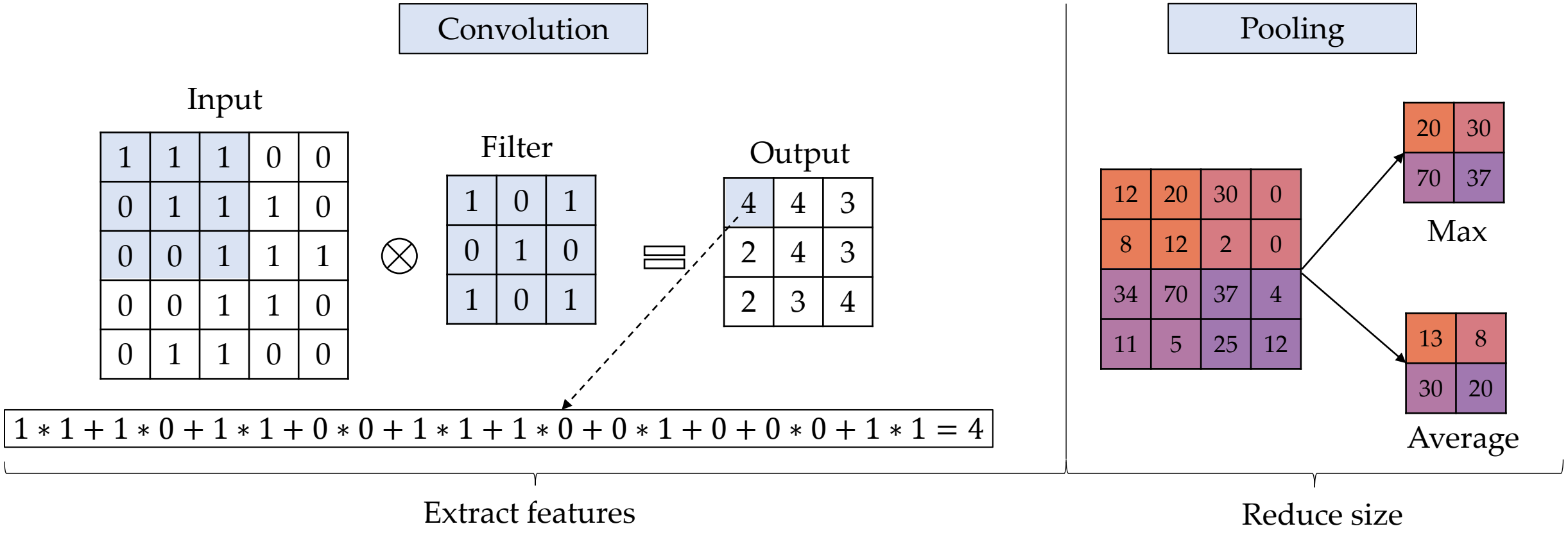
-1	1	-1
-1	1	-1
-1	1	-1

Filters to recognize loops and vertical lines

Convolutional Neural Network

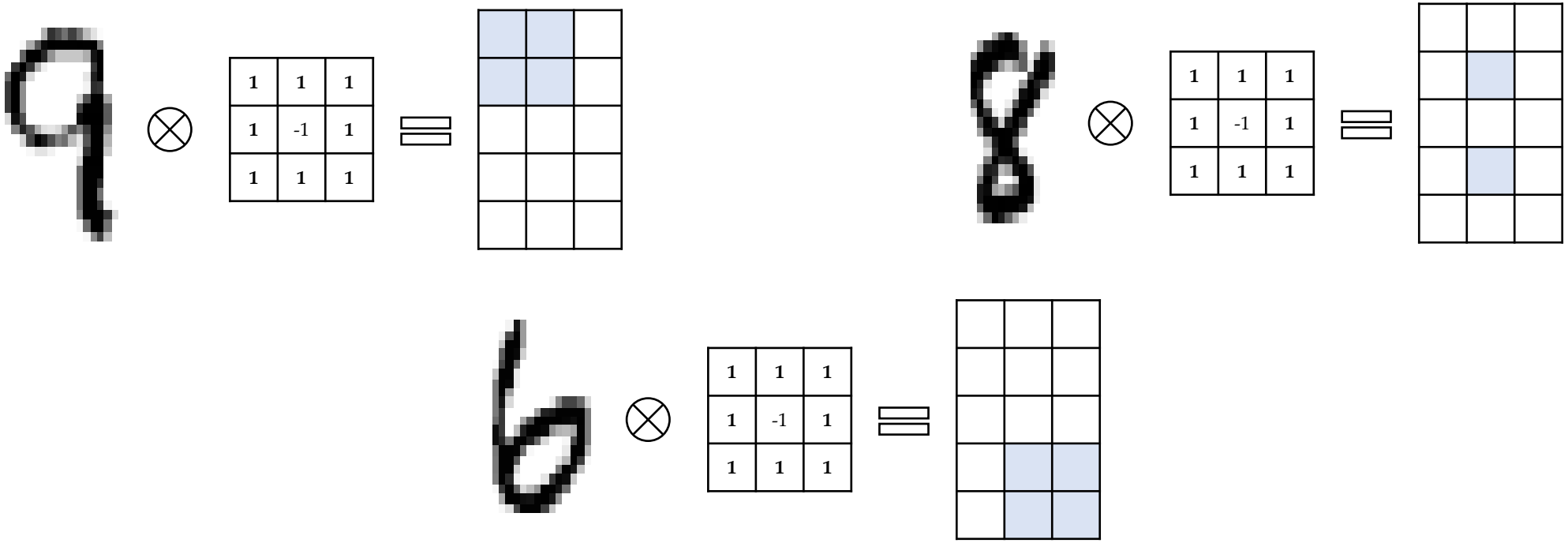
*A **CNN** is an algorithm that can take in an input image and apply filters that, during training, learn to distinguish various aspects/objects in the image*

Compared to training a network with flattened images as before, with CNNs is possible to capture the **Spatial dependencies** in an image. The architecture better fits the image dataset due to the reduction in the number of parameters involved and shared weights.



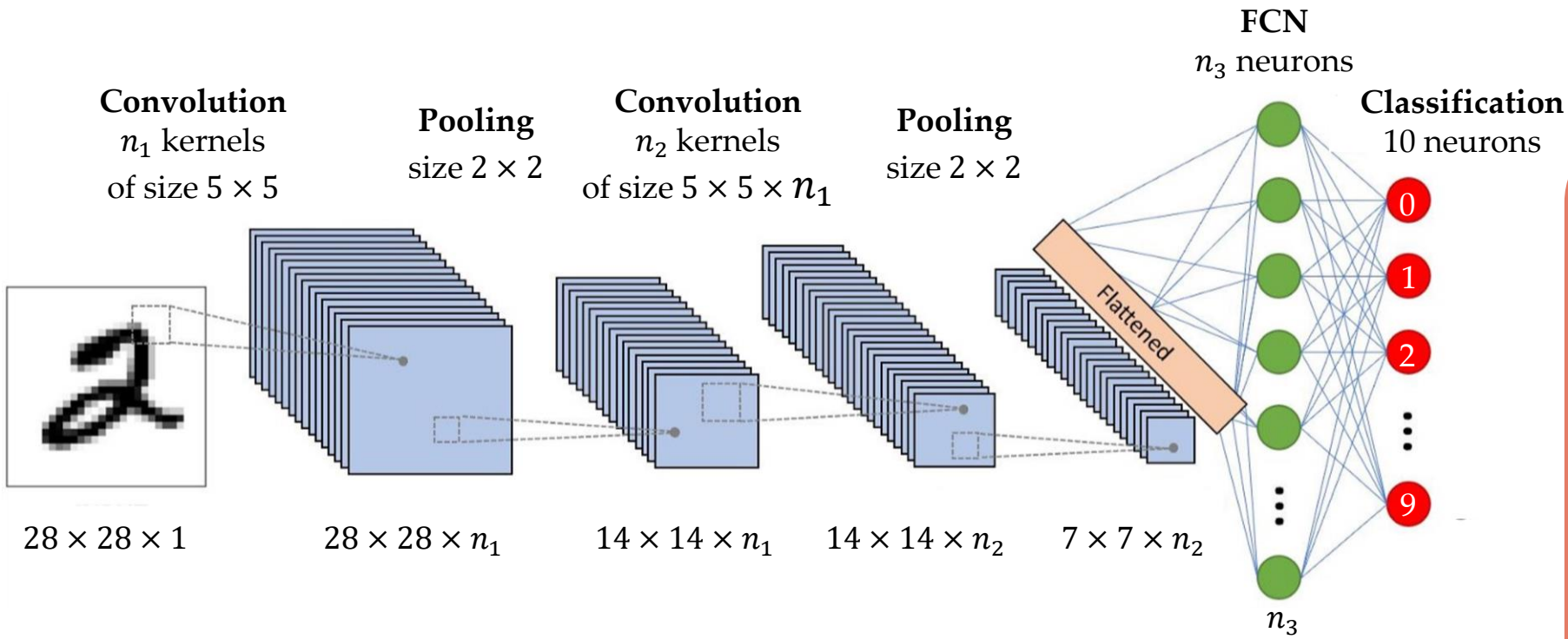
Convolutional Neural Network

Filters are location invariant feature detectors (the loopy circle is detected wherever it is in the image). In the illustrations below, the highlighted part in the output are the neurons activated by a loop circle in the input image

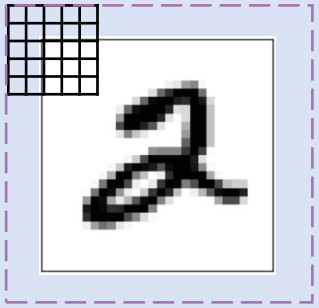


In CNN, these filters are not hardcoded as in this example but learned during training

Convolutional Neural Network



The role of the Convolution and Pooling layers is to reduce the images and extract features without losing information which is critical for getting a good prediction. The final FCN learns to handle the variety (in position, shape, etc.) of the high-level features extracted with the first part of the network.



In the first convolution layer, you slide each of the n_1 filters (with a step size named *stride*) over the $28 \times 28 \times 1$ input and compute the convolution operation. This results in an output of size $28 \times 28 \times n_1$. The height and width are the same because we used *padding* (dashed external square) and n_1 is the depth dimension, as we are using n_1 filter and each filter produces a 28×28 output. The pooling operation simply halves both the height and the width.

Case Study: Coding a CNN

Coding a CNN

We are now going to implement and train the network below.
A Google Colab file is available here to reproduce the results.

NB: teaching Python and PyTorch is outside the scope of this lesson. For this reason, the code will be discussed in the form of pseudocode, to understand the general behavior without focusing too much on the implementation

Five main steps:

1. Load the dataset
2. Define the CNN
3. Create the CNN and the optimizer
4. Train the model
5. Test the model