QML-Mod2-Classical Machine Learning

Riccardo Marega

March 2025

Indice

1 Introduction to classical machine learning -22/03/2025

Despite maybe we were not aware but we've already trained models, in our life, with machine learning algorithms: an example is given by the tools asking the users to spot cars, traffic signals, etc. in some website before entering them. An other example is given by all the filters that one can apply before taking a picture while using social media.

Artificial Intelligence vs Machine Learning vs Deep Learning

- artificial intelligence: any technique that enables computers to mimic human intelligence. It includes machine learning
- machine learning: a subset of AI that includes techniques that enable machines to improve at tasks with experience. It includes deep learning
- deep learning: a subset of machine learning based on neural networks that permit a machine to train itself to perform tasks

Machine learning includes two kinds of modalities to train the algorithm: supervised and unsupervised learning. In the latter we don't give any label to the algorithm and we let it to find itself the correct answer to the given problem.
An example of machine learning algorithm is to be found in surgical data science, where AI is at service of surgeons.
The choice of the best algorithm to apply has to be done considering the dimension and the type of data one has to manage.

$$\text{machine learnign}\begin{cases}\text{Unsupervised learning}\begin{cases}\text{Clustering}\end{cases}\\ \text{Supervised learning}\begin{cases}\text{Classification}\\ \text{Regression}\end{cases}\end{cases}$$

The goal of supervised training is mapping inputs in outputs. The difference between classification and regression is that the first one does a prediction of discrete outputs while the second one does a prediction of continuous outputs.
The main problem with unsupervised learning is that evaluating its performance is not always immediate.
Note that one can combine supervised and unsupervised learning.

1.1 Input data

Standard algorithms of ML usually take as inputs features and characteristics extracted from the data set. Those features are handcrafted or manually extracted.

The data is divided in a training set and a validation set. An algorithm is subject to underfitting of the data set if it has a poor performance on the training set. Otherwise it is said being subject to overfitting is it works well with the data training but has poor results with data of the validation set.
After having tested the algorithm with data from training and validation sets, we go to another phase of the analysis which implies the use of a third data set called test set.

Cross validation strategy : one can repeat the whole training of the model swapping every time the data from one set to another (what was first, for example, in the training set now will go in the validation set).

Generalization In Machine Learning, generalization refers to a model's ability to perform well on unseen data, meaning data that was not used during training. A well-generalized model captures the underlying patterns in the data rather than memorizing specific examples, allowing it to make accurate predictions on new inputs. Poor generalization can lead to overfitting (where the model performs well on training data but poorly on new data) or underfitting (where the model fails to learn meaningful patterns from the training data).
A **bias** is a distortion of the training data which is propagated in the algorithm. The goals related to the supervised training are a low error during training, validation and testing phases. The selection of the algorithm is based on:

- velocity of the training
- storage capacity
- accuracy on new data prediction
- transparency and interpretability

1.2 Practical examples of ML

- **Logistic regression**: the model is trained to predict the probability of a binary choice.
- **k nearest neighbor (kNN)**: is a pattern recognition algorithm based on distance of the data. If a subject A has, for example, characteristic close to the one of a subject B, then probably they are associated to the same category.

The ML learning model, in the training phase, learns a set of rules which depends both on the data set and a fixed combination of hyperparameters. The automatic learning of a model is not a single process, in necessary to experiment different models fixing different values of the hyperparameters.

```python
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_estimators=?, max_depth=?)
```

Imports the RandomForestClassifier model from the scikit-learn library. This model is a supervised learning algorithm based on decision trees. Creates an instance of the Random Forest classifier with two key parameters:

- `n_estimators=?`: defines the number of trees in the forest (typically between 10 and 1000).
- `max_depth=?`: sets the maximum depth of the trees (can be `None` to allow them to grow until all leaves are pure).

1.3 Cross validation and hyperparameter tuning

We divide the training and the validation set in multiple sets so that the model can be trained with all the available data. **Cross validation** is statistical method used to estimate the ability of different models of performing automatic learning. This procedure is done by defining a parameter k which represents the number of sets in which every set is subdivided. We can perform cross validation for hyperparameter tuning in either inner loops or outer loops.

Accuracy: $\frac{TP+TN}{TP+FP+FN+TN}$

Recall (Sensitivity/True positive rate): $\frac{TP}{TP+FN}$

Precision: $\frac{TP}{TP+FP}$

Specificity: $\frac{TN}{TN+FP}$

F1 score: $2 \times \frac{Precision \times Recall}{Precision+Recall}$

ROC Curve & AUC (Area under the curve)

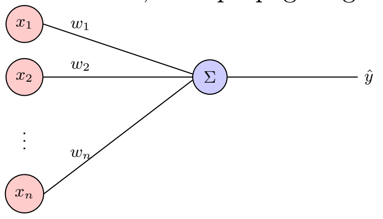
- **ROC curve:** Plot true positive rate (recall) vs. False Positive rate [...]
- **AUC** Probability [...]

2 Introduction to deep learning -28/03/2025

2.1 Training an artificial neural network

Modeling problems the first example is defining the position of a car whose positions is given exactly by $d(t) = d_0 + vt$. This problem is fully solvable. That is not always the case; indeed, a problem could be characterized by a large number of variables. The problem with working with large number of variables is that not always every variable has the same importance as the other. The problem, can be reduced to: $y = \alpha x_1 + \beta x_2 + \gamma x_3$.
Machine learning allow to solve complex problems in which we can easily tell what variables are involved in it.
Another problem that could be modeled is image recognition.
In ML we have a phase that does not appear in deep learning which is feature education: in traditional ML, model require pre-computed features that are manually designed based on domain knowledge. This feature engineering requires human expertise to identify the most relevant attributes for a given task.
Deep learning eliminates the need for manual feature extraction by learning hierarchical representations directly from raw data. Using AI networks, DL can automatically detect patterns at multiple levels of abstraction.

Artificial Neural Networks Biologically, neurons are unique cells that can communicate with one another, thus propagating information.



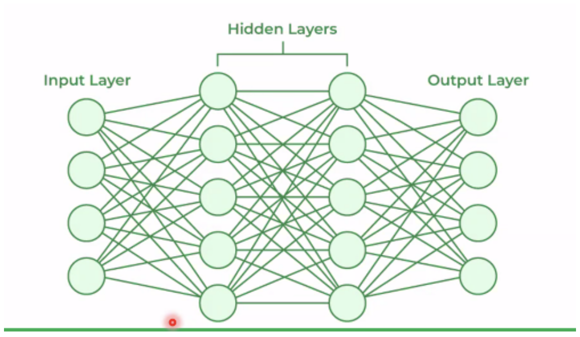
It receives n inputs, each scaled by a factor (weights), ad it sums them all scaled by a bias factor:

$$y = \sum_{i=1}^{n} w_i \times x_i + b.$$

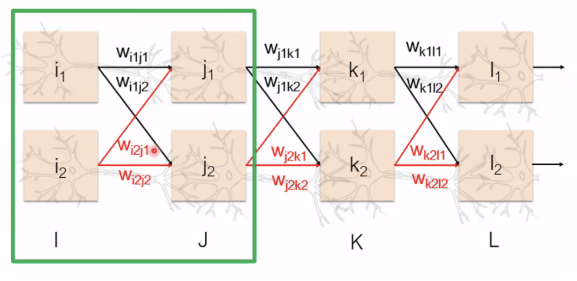
Weights and bias are the parameters of the neural network. Note that very few models are correctly modeled using linear combinations of the variables; indeed, we introduce an activate factor such that:

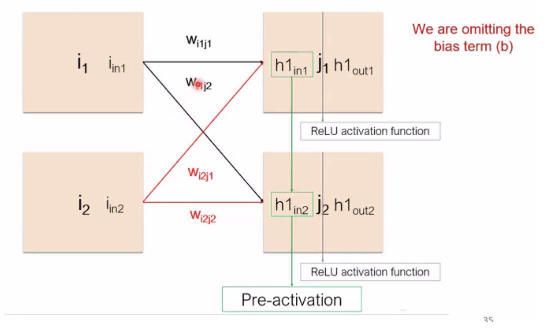
$$y = \sum_{i=1}^{n} g(w_i \times x_i + b).$$

Typical activation functions are: sigmoid, tanh, ReLU ($\max(0, x)$), Leaky ReLU ($\max(0.1x, x)$). The sigmoid and tanh functions limit the output, indeed, the output is always confined between 0 and 1.



This is what a simple neural network looks like: an input layer, a bunch of hidden layers, and an output layer. Each layer extracts information from the input and transmits it to the next one for further data processing.





$$[i_{in1} i_{in2}] \times \begin{bmatrix} w_{i1j1} w_{i1j2} \\ w_{i2j1} w_{i2j2} \end{bmatrix} = [h1_{out1} h2_{out2}]$$

What we are building is a neural architecture. All these parameters will have to be adjusted. This process is exactly what is called train of the neural network. The training process continues with making mistakes.
Let's consider the problem of predicting the cost of house: y is the real cost and $\hat{y}$ is the predicted one. y is also called ground-truth value and is used to supervise the training. The idea under the training process lays in computing the error, and what we want is indeed try to minimize this error. Error minimization is an optimization problem: I need a parameter configuration that explains the problem in the best possible way (i.e. i get as close as possible to the ground-truth value)

$$error = |y - \hat{y}|.$$

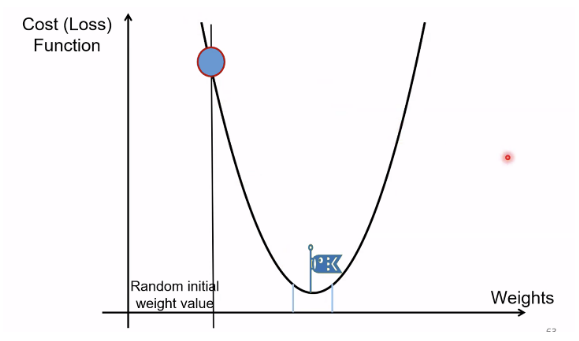
To optimize the parameters of a neural network we use the gradient method. What we always know when computing an algorithm is whether the error is increasing or decreasing.

- step 0: the error is very big
- step 1: i change the parametes
- step 2: ...

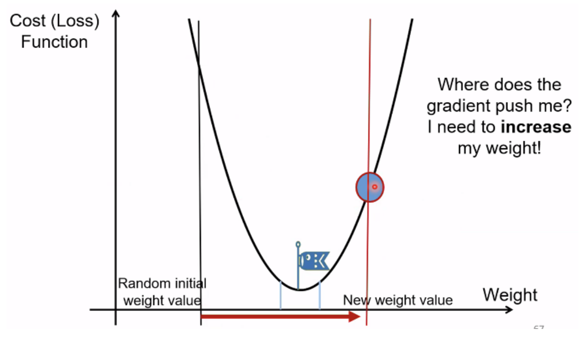
Gradient descent is the most used optimization technique to minimize the error. It means that, for each step, I compute the error and change the parameters in order to minimize the function.

$$error = f(w, b)$$

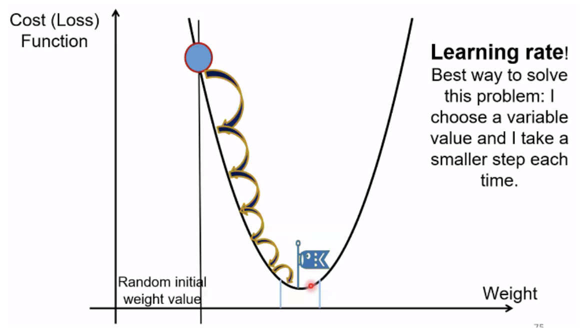
this function is called **cost function**. First order derivative (gradient) ...



Initializing the parameters randomly gives better results.


Cost (Loss) Function
Where does the gradient push me? I need to **increase** my weight!
Random initial weight value
New weight value
Weight

and so on till we reach the minimum.
In general the first step to reach the minimum is the bigger among all the steps that will be done. In this case we talk about learning rate. While "learning" we have to decrease the learning rate (i should take smaller steps) in order to reach the target.


Cost (Loss) Function
Learning rate!
Best way to solve this problem: I choose a variable value and I take a smaller step each time.
Random initial weight value
Weight

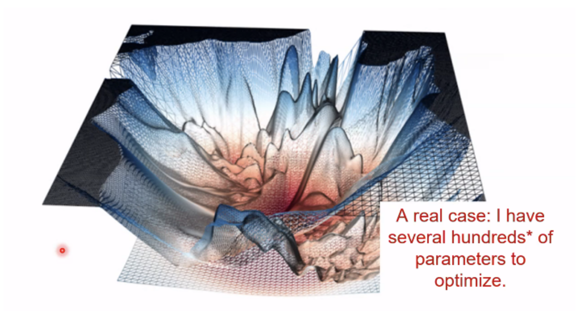
Mathematically, training a neural network means to change its parameters N times in order to minimize the cost function (error). Each update can be expressed as

$$w_{t+1} = w_t - \eta \times \nabla error$$

where η is the learning rate and $\nabla error$ is the gradient of the cost function with respect to w. A typical loss function is the MSE. The only constraint that we impose to the loss functions is to be differentiable everywhere. The error can be computed only from the output of the layers.

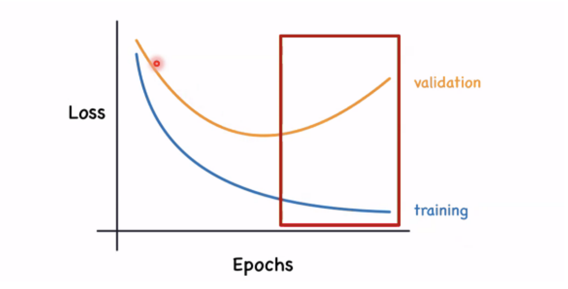
$$\frac{dE_1}{dw_{k1l1}} = \frac{dE_1}{dO_{out1}} \times \frac{dO_{out1}}{dO_{in1}} \times \frac{dO_{in1}}{dw_{ikl1}}$$

For a forward pass of information we have an error back-propagation. This kind of neural network is called fully connected.


A real case: I have several hundreds* of parameters to optimize.

Usually, what we need is a dataset, i.e. a set of (x,y) samples where x is the input and y is the output. The dataset is divided in three sub-sets: a training set, a validation set and a test set. The training process involves parameters optimization on the same training samples several times, or epochs.
AI does not really "learns": it just finds patterns. It's like a student who only studies math by doing every exercise on the book but never reads a page of theory. How can we make sure that the student is learning and not just memorizing? During the test, the student has no way to learn new knowledge because it has no supervision. [...]. Therefore, our neural network needs to perform well both on the training set and on the two test sets. What happens if it doesn't? The loss function on the training set has a very nice trend: as the network is iteratively optimized on the training sample, its error on these samples decreases. However, the loss function on the validation test doesn't seem as good: while the student improves with those "training" exercises, they make a lots of mistakes during the exam.
Two things are never to be expected:

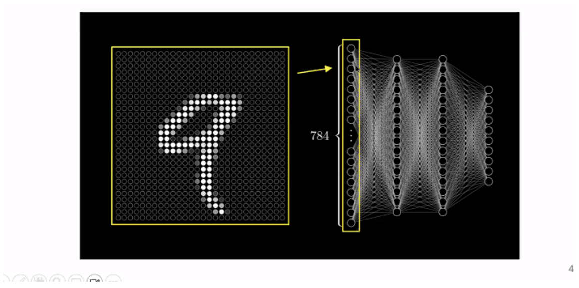
- no error or 100% accuracy
- better results in validation



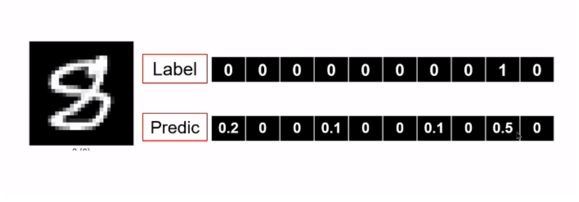
However, we should minimize the distance between training and validation performance: **overfitting**.

2.2 Artificial neural networks: tasks

Regression: predict a number from a continuous set of numbers. An example is given by the prediction of the cost of a house.
Classification: predict a number from a discrete set of numbers.
The number of neurons in the input and output layers will depend on the task. In regression, we need to predict one number: one neuron is needed. The optimal number of neurons in the hidden layer (as well as the number of hidden layers) cannot be assessed a priori. The more the neurons and the layers, the more abstract information we can extract from the input data.
Check out the following site: playground tensorflow
Binary classification Classification is the most studied problem in DL. The MNIST dataset is on of the most famous one: our network needs to classify the input image as one of the 10 possible labels (the digits 0-9).



Check out: The stilwell brain
...
For classification, a perfect loss function is (binary) cross-entropy.



Label	0	0	0	0	0	0	0	0	1	0
Predic	0.2	0	0	0.1	0	0	0.1	0	0.5	0

Figura 1: Note: 0.6 instead of 0.5

I want my prediction to be a probability distribution. The sum of all my elements needs to be 1. In the case of a odd vs even prediction the binary classification will give more precise results.

Loss function: Binary cross enrtopy

$$-\sum_{j=1}^{M} y_j \log(p(y_j))$$

$$-\sum_{i=1}^{N} y_i \log(p(y_i)) + (1 - y_i) \log(1 - p(y_i))$$

Final activation: softmax

In order to obtain a probability distribution, I need to use the softmax activation function in the outer layer.

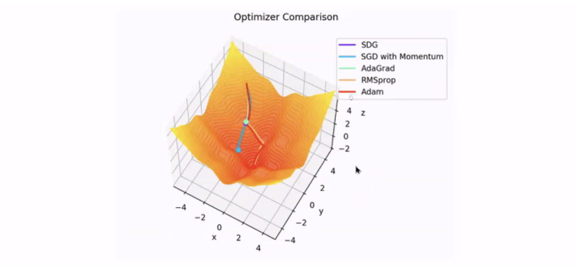
$$\begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \\ 0.7 \\ 1.1 \end{bmatrix} = \left[\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \right] = \begin{bmatrix} 0.02 \\ 0.9 \\ 0.05 \\ 0.01 \\ 0.02 \end{bmatrix}$$

Metric: accuracy

$$\begin{bmatrix} \text{True positive (TP)} & \text{False Positive (FP)} \\ \text{False Negative (FN)} & \text{True Negative (TN)} \end{bmatrix}$$

where

- Recall $= \frac{\sum TP}{\sum TP+FN}$
- Precision $= \frac{\sum TP}{\sum TP+FP}$
- Accuracy $= \frac{\sum TP+TN}{\sum TP+FP+FN+TN}$


Optimizer Comparison
SDG
SGD with Momentum
AdaGrad
RMSprop
Adam
x
y
z