Advanced ROMs in SciML

Introduction to Scientific Machine Learning

Federico Pichi

18 June 2025

Introduction

Scientific Machine Learning (SciML) recently emerged as a way of combining *physics–based* and *data–driven* models for the numerical approximation of differential problems.

- **Physics–based** models rely on the physical understanding of the problem at hand, subsequent mathematical formulation, and numerical approximation.
- **Data–driven** models instead aim to extract relations between input and output data without arguing any causality principle underlining the available data distribution.

What is the connection with more standard Reduced Order Models (ROMs)?

(i) huge availability of data, (ii) cheap computing power, (iii) powerful ML algorithms.

Disclaimer : Teaching a course on state-of-the-art ML approaches is a very stupid idea.
The whole scientific community is trying to develop and exploit novel strategies.
Even focusing on our scientific sector, more than 200 papers are uploaded everyday on arXiv to cs.ML.

Why and when

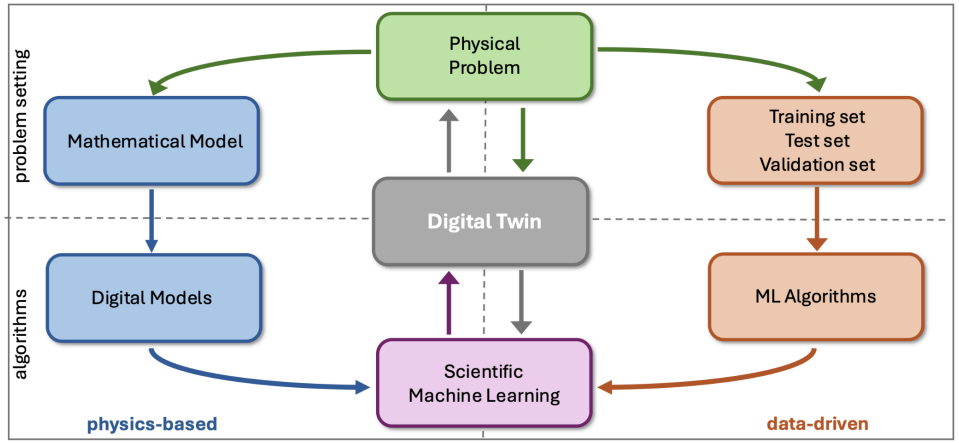
SciML leverages the physical awareness of physics–based models and, the efficiency and versatility of data–driven algorithms.

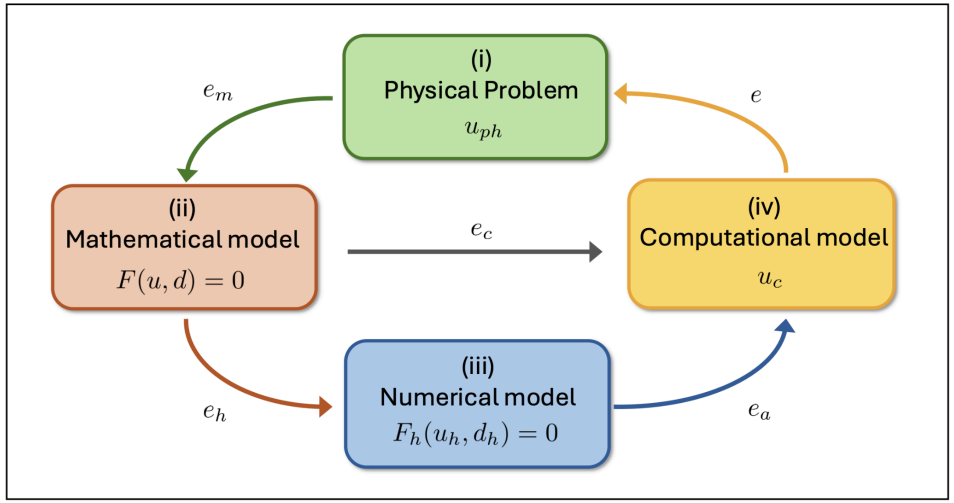
- Inject physics and mathematical knowledge into machine learning algorithms.
- Capability to discover complex and non–linear patterns from data.

From fundamental principles to mathematical models for solving problems from basic and applied sciences.

These are studied via numerical approximations or computational models, resulting in robust but complex approaches.

What about the well-posedness of the problem itself?



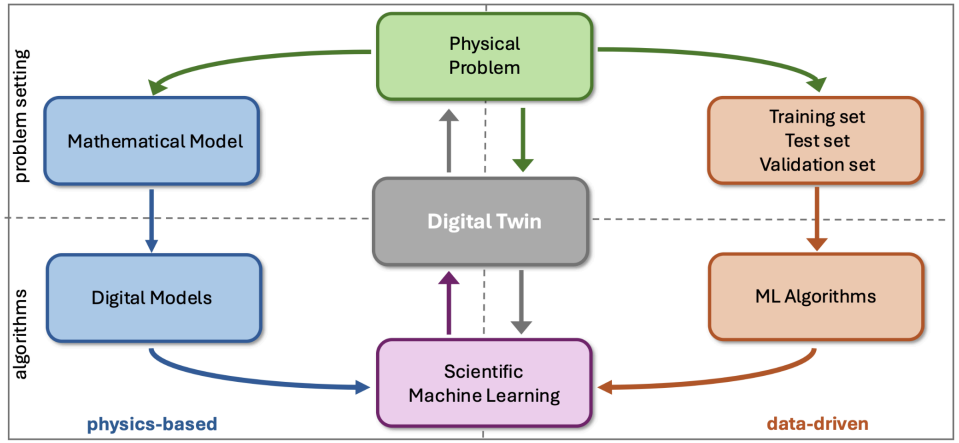


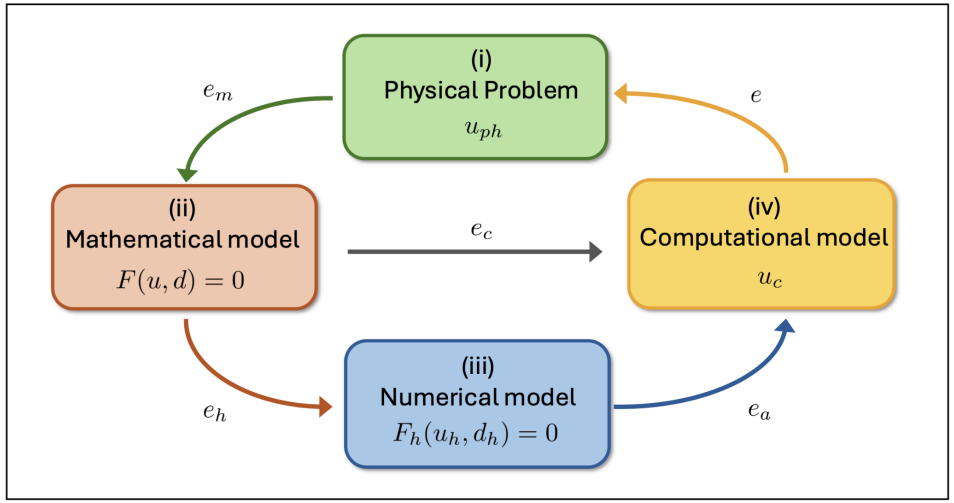
Interplay between numerical methods and machine learning

- Numerical methods can be exploited to make machine learning algorithms more accurate (regularize loss, augment data, learn operators).
- Machine learning algorithms can enhance the computational efficiency of numerical methods (infer laws, sensitivity analysis, inverse problems).

Goal: Find new paths to combine the consistency of physics–based models and the efficiency of data–science algorithms.

SciML paradigm is fundamental in the realization of *digital twins*, when a real–time dialogue between the physical asset and its digital model is essential.





Computational setting for PDEs

Mathematical models translate the first principles into mathematical equations describing some fundamental characteristics, e.g. conservation laws (mass, momentum, energy)

$$F(u, d) = 0.$$

Let us denote by $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$ an open and bounded set and $(t_0, T) \subset \mathbb{R}$ a bounded interval, we look for the function $u = u(\mathbf{x}, t) : \Omega \times (t_0, T) \to \mathbb{R}^m$ satisfying

$$\begin{cases} \frac{\partial u}{\partial t} + \mathcal{L}u + \mathcal{N}(u) = f, & \text{in } \Omega \times (t_0, T) \\ \text{boundary conditions}, & \text{on } \partial\Omega \times (t_0, T) \\ u = u_0, & \text{in } \Omega \times \{t_0\} \end{cases} \tag{1}$$

where $\mathcal{L}$ and $\mathcal{N}$ are, respectively, linear and non–linear differential operators involving derivatives with respect to space variables, while other terms can be treated as data.

Model error: express how well we are describing the real physics as $e_m = \|u_{ph} - u\|$.

Numerical approximation

We need to approximate the mathematical model in (1) with a numerical model:

- Finite Element, Finite Volume, Finite Difference, Spectral Element, etc.

As an example, in the Galerkin Finite Element (FE) method, the first step is to construct a mesh for the domain Ω with characteristic size h, and the finite–dimensional space V_h, with dimension N_h, by piecewise polynomials defined on such partitions.

Denoting by $\{\varphi_j\}_{j=1}^{N_h}$ a suitable basis of the space V_h, for any $t \in (t_0, T)$ we look for a function $u_h(\mathbf{x}, t) \in V_h$ that can be expressed as

$$u_h(\mathbf{x}, t) = \sum_{j=1}^{N_h} u_{h,j}(t)\varphi_j(\mathbf{x}).$$

Numerical approximation - space discretization

For any $t \in (t_0, T)$, let $u_h(t) = [u_{h,1}(t), \dots, u_{h,N_h}(t)]^\top$ be an array in $\mathbb{R}^{N_h}$, the semidiscrete problem reads as

$$\begin{cases} \mathbf{M}\frac{d\mathbf{u}_h}{dt}(t) + \mathbf{L}\mathbf{u}_h(t) + \mathbf{N}(\mathbf{u}_h(t)) = \mathbf{f}, \quad \text{in } (t_0, T) \\ \mathbf{u}_h(t_0) = \mathbf{u}_{0,h}, \end{cases} \tag{2}$$

where $\mathbf{L} \in \mathbb{R}^{N_h \times N_h}$ and $\mathbf{N} : \mathbb{R}^{N_h} \mapsto \mathbb{R}^{N_h}$ are suitable linear and non–linear Galerkin projection operators, while $\mathbf{M} \in \mathbb{R}^{N_h \times N_h}$ such that $\mathbf{M}_{ij} = \int_\Omega \varphi_i \varphi_j \, d\Omega$ is the mass matrix, and $u_{0,h}$ is the array containing the values of u_0 at the mesh nodes.

Numerical approximation - time discretization

We introduce a partition of the time interval $[t_0, T]$ into N_t sub–intervals of size Δt corresponding to a set of nodes $t_n = t_0 + n\Delta t$, with $n = 0, \dots, N_t$, and $t_{N_t} = T$.

By approximating with a finite difference scheme the time derivative, we can write $\mathbf{u} = [\mathbf{u}_{h,1}^n, \dots, \mathbf{u}_{h,N_h}^n]^\top$, where $\mathbf{u}_{h,j}^n$ is the approximation of $u_{h,j}(t_n)$ for any $n = 0, \dots, N_t$.

For $n = 0, \dots, N_t - 1$, we look for $\mathbf{u}^{n+1}$ satisfying the fully discrete system

$$\begin{cases} \dfrac{1}{\Delta t}\mathbf{M}\mathbf{u}^{n+1} + \mathbf{L}\mathbf{u}^{n+1} + \mathbf{N}(\mathbf{u}^{n+1}) = \dfrac{1}{\Delta t}M\mathbf{u}^n + \mathbf{f}^{n+1}, \\ \mathbf{u}^0 = \mathbf{u}_{0,h}. \end{cases} \tag{3}$$

These are non-linear algebraic systems that can be solved e.g. by Newton method, where for each iteration, a linear system of size N_h has to be solved (with possibly large condition number causing the ill-posedness of the problem at the physical and numerical level).

Convergence, consistency and stability

Convergence: a key property for the well-posedness is that $\lim_{h\to 0} u_h = u$.
Consistency: a necessary condition for convergence $\lim_{h\to 0} F_h(u, d) = 0$.
Stability: the system is well--posed if it admits a unique solution, and if small perturbations in the data result in controllable variations in the solution.

Numerical error: express how well we are approximating the terms as $e_h = \|u - u_h\|$.

Goal: analyse the error e_h of the numerical model showing that it tends to zero as the discretisation parameters h and Δt tend to zero, e.g. for steady problems one seeks estimates

$$\|u - u_h\| \leq C(u, d)h^p,$$

where the exponent p is the order of convergence of the numerical model with respect to h.

The *a-priori analysis* allows for predicting the error decay trend for vanishing h, while providing an accurate numerical quantification of the error for a fixed h is the task of the *a-posteriori analysis* exploiting the residual quantity $\|r_h\| = \|F(u_h, d)\|$.

Reduced Order Models

When dealing with parametrized problems, the computational cost is excessive for real-time and many-quary simulations, and it is fundamental to develop Reduced Order Models (ROMs).

Let us denote by $\mu \in P \subset \mathbb{R}^p$ a set of parameters characterizing the solution of the PDE, then: given $\mu \in P$, for any $t \in (t_0, T)$ we look for the solution $\mathbf{u}_h(t;\mu)$ such that

$$\begin{cases} M(\mu)\dfrac{d\mathbf{u}_h}{dt}(t;\mu) + L(\mu)\mathbf{u}_h(t;\mu) + \mathbf{N}(\mathbf{u}_h(t,\mu);\mu) = \mathbf{f}(t;\mu) \quad \text{in } (t_0,T) \\ \mathbf{u}_h(t_0;\mu) = \mathbf{u}_{0,h}(\mu), \end{cases} \tag{4}$$

where the solutions belong to a manifold $M_h = \{\mathbf{u}_h(t;\mu) : \ t \in (t_0,T), \mu \in P\}$.

ROM strategy:

1. Construct a database of discrete solutions in M_h, the so-called *snapshots*, for μ in P.
2. Extract a set of $N \ll N_h$ linearly independent basis that span a linear subspace V_N of M_h
3. Given a new $\overline{\mu}$, perform a Galerkin projection of the equation (4) onto the subspace V_N.

Reduced Order Models

If we denote by V_N the matrix whose columns contain the degrees of freedom of the FE basis function and set $M_N(\overline{\mu}) = V_N^T M(\overline{\mu}) V_N$, $L_N(\overline{\mu}) = V_N^T L(\overline{\mu}) V_N$, and $\mathbf{f}_N(\overline{\mu}) = V_N^T \mathbf{f}(t, \overline{\mu})$, then: for any $t \in (t_0, T)$ find the solution $\mathbf{u}_N(t; \overline{\mu}) \in V_N$ of

$$\begin{cases} M_N(\overline{\mu}) \dfrac{d\mathbf{u}_N}{dt}(t; \overline{\mu}) + L_N(\overline{\mu}) \mathbf{u}_N(t; \overline{\mu}) + V_N^T \mathbf{N}(V_N \mathbf{u}_N(t, \overline{\mu}); \overline{\mu}) = \mathbf{f}_N(t; \overline{\mu}) \quad \text{in } (t_0, T) \\ \mathbf{u}_N(t_0; \overline{\mu}) = \mathbf{u}_0(\overline{\mu}). \end{cases} \tag{5}$$

The most widely used strategies are Greedy approaches and Proper Orthogonal Decomposition (POD) methods.

Ad-hoc strategies are needed for:

- time-dependent problems (POD-Greedy, nested-POD)
- nonlinear and non-affine problems (tensors, hyper-reduction, a-posteriori estimate)
- slow Kolmogorov n-width decay (e.g. advection dominated problems)

Computational aspects

The numerical model is implemented on computers using appropriate algorithms. Thus, instead of the numerical solution u_h, we end up with the computational solution u_c.

Computational error: quantify the finite arithmetic limitations as $e_c = \|u_h - u_c\|$

Overall error: corresponds to the real approximation error, measuring the difference between the physical solution u_{ph} and the computational solutions u_c as $e = \|u_{ph} - u_c\|$

$$e = e_m + e_h + e_c = \underbrace{\|u_{ph} - u\|}_{\text{model error}} + \underbrace{\|u - u_h\|}_{\text{numerical error}} + \underbrace{\|u_h - u_c\|}_{\text{computational error}}$$

Numerical analysis guarantees that the errors e_h and e_c are small and controllable (*reliability*) with the least possible computational cost (*efficiency*).

- Reliability: control the error $e_h + e_c$ as a function of the problem's data and the parameters
- Efficiency: amount of resources (computation time, memory usage) needed to compute u_c.

Data-driven models by Machine Learning

Data-driven models exploit measurements, imaging, and experiments to obtain a solution to the problems. Modern data-driven models consists of employing Machine Learning (ML) tools, instead of analytical approaches, to draw conclusions.

Four pillars in **Scientific Computing**:

1. *The knowledge of the theory (mathematical models)*
2. *The availability of experiments needed to validate the models*
3. *The numerical simulations*
4. *Artificial Intelligence (AI)*

AI: a set of techniques enabling computers to mimic human intelligence acquiring knowledge
Why: powerful computers, huge amounts of data, *ML algorithms*
Goals: *Regression and classification tasks*, generative models, and rewards–maximization
Learning: *Supervised*, *Unsupervised*, Reinforcement Learning

Machine Learning

"ML is the field that gives computers the ability to learn without being explicitly programmed"

A large subset of ML is represented by artificial (*deep*) neural networks (NNs), which are computing systems made of (*many*) layers of artificial neurons.
The essential elements to train a ML algorithm are the availability of large volumes of data, and some knowledge/insights on the modeling assumptions.

Common issues:

- datasets are sparse and partial, e.g. if data acquisition is expensive,
- training datasets are not available,
- need for interpretable models or outputs.

In these cases, the training of the algorithms might be limited, and what they have learned is not enough to allow them to apply successfully to more general situations.

Example (classification)

A computer program learns from experience E w.r.t. some task T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

Task T: (*classification*) given a picture $\mathbf{x}$, output $y = 1$ if it portrays a dog, $y = 0$ otherwise.

1. experience training data $(\hat{\mathbf{x}}_i, \hat{y}_i)$ for $i = 1, \dots, N$, i.e., pictures $\hat{\mathbf{x}}_i$ with labels $\hat{y}_i \in \{0, 1\}$
2. select a set of candidate models $y = f(\mathbf{x}; \mathbf{w})$ in terms of input $\mathbf{x}$ and parameters $\mathbf{w} \in \mathbb{R}^M$
3. define the loss function $\mathcal{L}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} [d(\hat{y}_i, f(\hat{\mathbf{x}}_i; \mathbf{w}))]^2 \; + \; \text{regularization}$
4. choose a metric to monitor and measure the performance of the model
5. train the model by optimizing the loss function $\mathcal{L}$, i.e., find $\mathbf{w}^* = \operatorname{argmin} \mathbf{w} \in \mathbb{R}^M \mathcal{L}(\mathbf{w})$
6. measure the performance of $f(\cdot; \mathbf{w}^*)$ by evaluating the metric on test data.

Issues: $\mathcal{L}$ has to be differentiable, regularization needed since $\mathcal{L}$ is not convex

Example (regression)

investigate relations between input and output variables to:

1. predict outputs for new inputs
2. learn the effect of inputs on output

Dataset: $\mathcal{S} = \{\mathbf{x}^i, y^i\}_{i=1}^N$ of N input-output pairs, where $\mathbf{x}^i \in \mathbb{R}^D, y^i \in \mathbb{R}$.

Model: A function $f : \mathbb{R}^D \to \mathbb{R}$ approximating the outputs for given inputs $y^i \approx f(\mathbf{x}^i)$.

Example: Linear input-output relationship

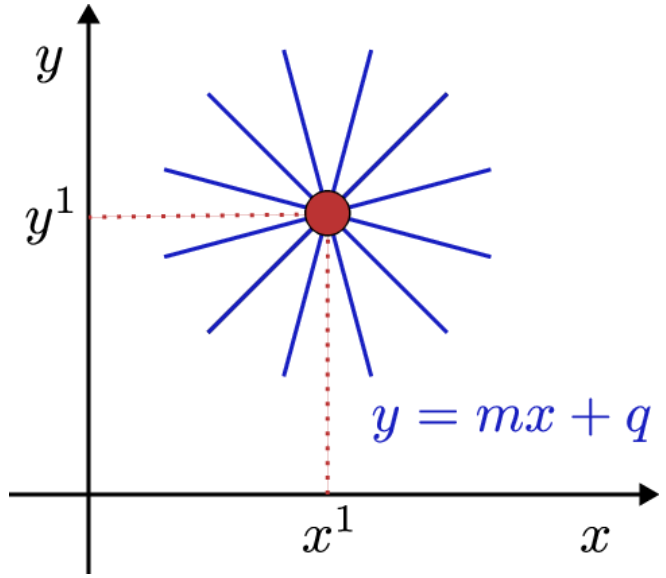
$$y^i \approx f_w(\mathbf{x}^i) = (\mathbf{x}^i)^T \mathbf{w} = \sum_{j=1}^{D} x_j^i w_j,$$

where $\mathbf{w} = [w_1, \cdots, w_D]$ are the parameters of the model and describe f.

Goal: Given $\mathcal{S}$ learn/estimate the weights $\mathbf{w}$ fitting the model.

Remarks

- **Dimension of the model**: For many machine learning models, the number of **weight** parameters, i.e. the dimension of $\mathbf{w}$, can be very different from D, i.e. the number of **features**.
- **Overparametrization**: ($D > N$) The number of parameters exceeds the amount of available data, and we have more variables than information.



$\rightsquigarrow$ The task is *under-determined* and the model is *over-parameterized*.

Example: Consider a unique point ($N = 1$), and fit an affine model $y = mx + q$, that is: starting from $\mathcal{S} = \{x^1, y^1\}$ find $\mathbf{w} = [m, q]$.

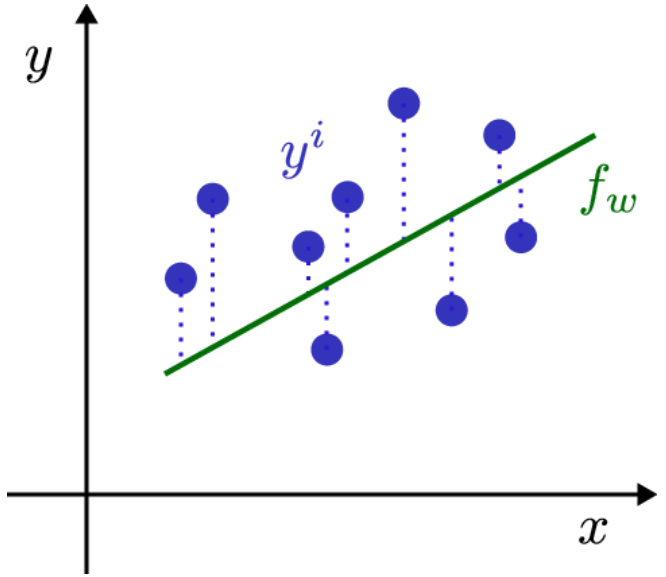
Loss functions

Task: Estimate values of $\mathbf{w}$ given $\mathcal{S}$.

Tool: The **loss** function (or energy, cost) is used to learn such parameters, quantifying how well our model explains the data, or how costly our mistakes are.

Example: Mean Squared Error (MSE) defined as

$$MSE(w) = \frac{1}{N}\sum_{i=1}^{N}(y^i - f_w(\mathbf{x}^i))^2$$



Pros: simple and efficient, convex, symmetric, smooth

Cons: sensitive to outliers, bias towards mean, prone to underfitting

Loss Functions

- **Mean Absolute Error**:

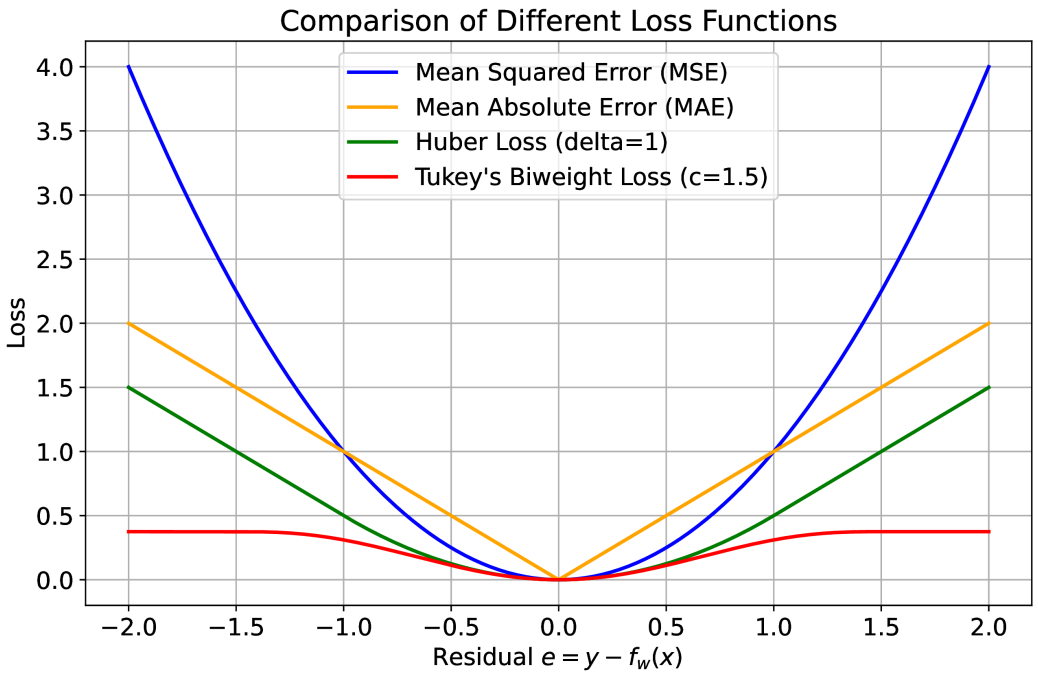
$$\mathrm{MAE}(w) = \frac{1}{N}\sum_{i=1}^{N} |y^i - f_w(\mathbf{x}^i)|$$

- **Huber loss**:

$$H(e) = \begin{cases} \frac{1}{2}e^2, & \text{if } |e| \le \delta \\ \delta|e| - \frac{1}{2}\delta^2, & \text{if } |e| > \delta \end{cases}$$

- **Tukey loss**:

$$\frac{\partial L}{\partial e} = \begin{cases} e(1 - \frac{e^2}{c^2})^2, & \text{if } |e| \le c \\ 0, & \text{if } |e| > c \end{cases}$$



Loss landscapes

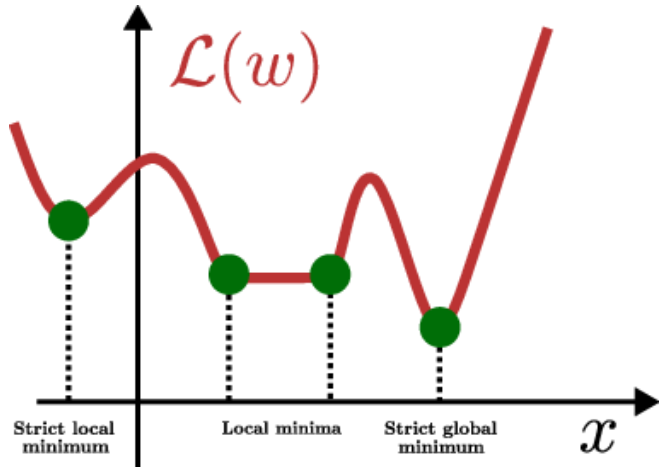
- A vector $\mathbf{w}^*$ is a *local* minimum of $\mathcal{L}$ if there exists an $\epsilon > 0$ s.t.

$$\mathcal{L}(\mathbf{w}^*) \leq \mathcal{L}(\mathbf{w}) \quad \forall\, \mathbf{w} \text{ with } ||\mathbf{w} - \mathbf{w}^*|| < \epsilon.$$

- A vector $\mathbf{w}^*$ is a *global* minimum of $\mathcal{L}$ if

$$\mathcal{L}(\mathbf{w}^*) \leq \mathcal{L}(\mathbf{w}) \quad \forall\, \mathbf{w} \in \mathbb{R}^D.$$

- A local/global minimum is said to be *strict* if the inequality is strict for $\mathbf{w} \neq \mathbf{w}^*$.



Optimization problem

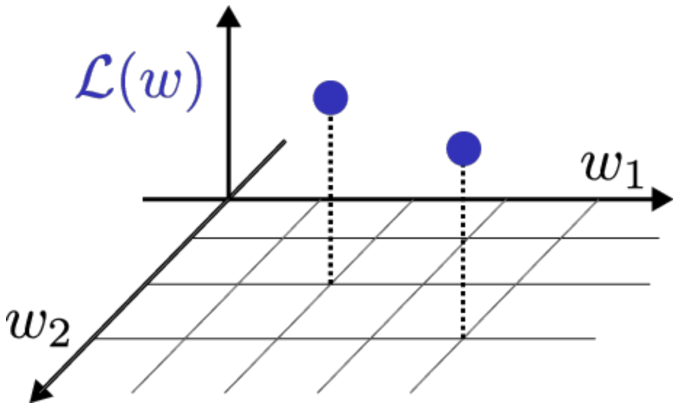
Given a cost function $\mathcal{L}$, we aim at finding $\mathbf{w}^*$ which minimizes the cost

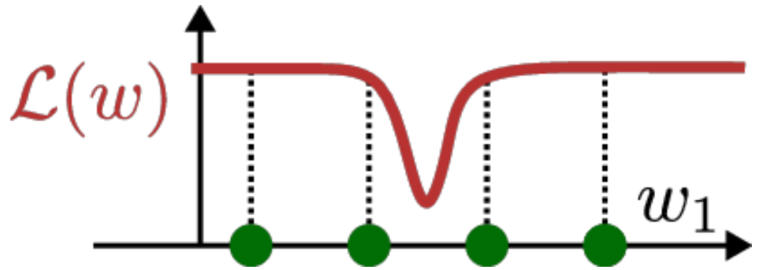
$$\min_{\mathbf{w}\in\mathbb{R}^D} \mathcal{L}(\mathbf{w}) = \min_{\mathbf{w}\in\mathbb{R}^D} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i(\mathbf{w})$$

Possible strategy:

- Grid search - compute the cost for "all" values $\mathbf{w}$ and pick the argmin.
- Issue: exponential cost (3^D if granularity is 3, and $D \approx 10^6$)

$\rightsquigarrow$ No guarantee to end up near the optimum



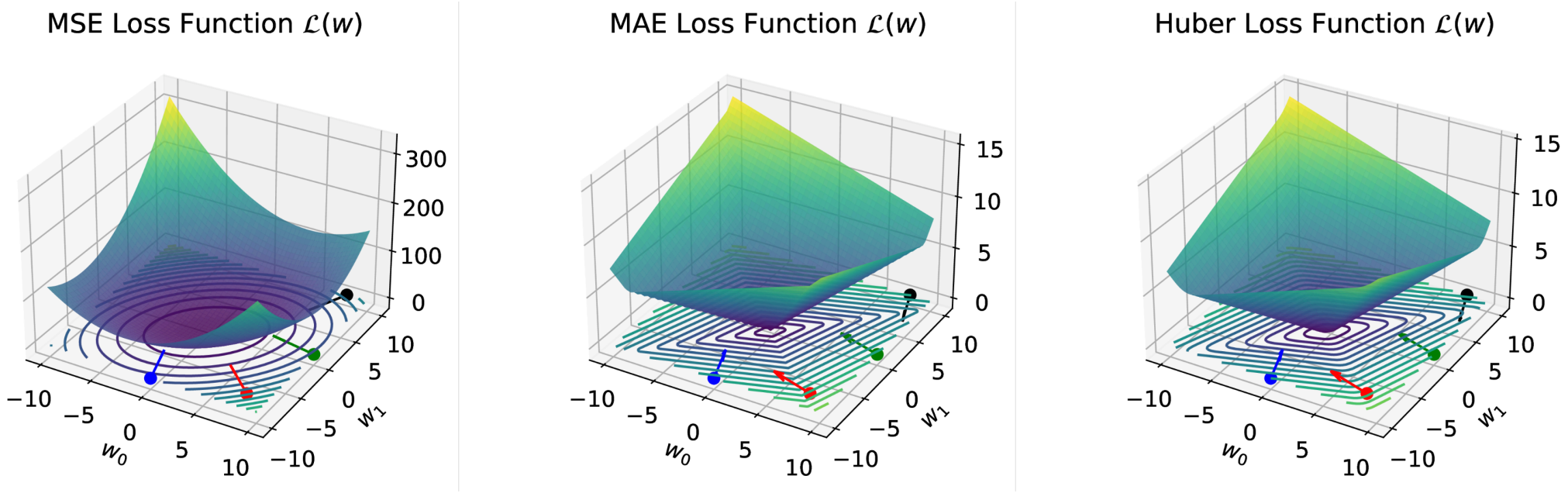


Optimizer

- **Gradient-based approaches**: The gradient is the slope of the tangent to the function, pointing to the direction of the largest increase of the function:

$$\nabla \mathcal{L}(\mathbf{w}) = \left[\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_D} \right]^T \in \mathbb{R}^D$$

Example: $y^i = w_1 x^i + w_0$ with $y^T = [2, -1, 1.5]$ and $x^T = [1, -1, -1]$



Gradient Descent (GD)

Starting from an initial configuration $\mathbf{w}_0$, to minimize the loss function $\mathcal{L}(\mathbf{w})$ we take a step in the *opposite* direction of the gradient, i.e.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \nabla \mathcal{L}(\mathbf{w}_t),$$

where γ is the so-called *learning rate*.

Example (Linear MSE): Given $\mathbf{y} = \left[y^1, \dots, y^N\right]^T, \mathbf{X} = \left[\mathbf{x}^1, \dots, \mathbf{x}^N\right]^T \in \mathbb{R}^{N \times D}$ with $\mathbf{x}^i \in \mathbb{R}^D$, we define $\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{w} \in \mathbb{R}^N$ and $\mathcal{L}(\mathbf{w}) = \frac{1}{2N} \sum_{i=1}^N (y^i - (\mathbf{x}^i)^T \mathbf{w})^2$.

Then, the gradient can be computed as

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_j} = -\frac{1}{2N} \sum_{i=1}^N 2x_j^i (y^i - (\mathbf{x}^i)^T \mathbf{w}) = -\frac{1}{N} (\mathbf{X}_j)^T \mathbf{e},$$

and can be written as $\nabla \mathcal{L}(\mathbf{w}) = -\frac{1}{N} \mathbf{X}^T \mathbf{e}$.

Cost: Computing the gradients takes $\mathcal{O}(N \cdot D)$.

Stochastic Gradient Descent (SGD)

Cost functions are usually defined as a sum over the training samples, i.e.

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i(\mathbf{w}),$$

$\rightsquigarrow$ instead of updating w.r.t. the full loss, sample a point in the training set $k \in \{1, \dots, N\}$ and update with the corresponding term

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \nabla \mathcal{L}_k(\mathbf{w}_t),$$

Why: Cheap and unbiased estimate of the gradient

$$\mathbb{E}_i \left[\nabla \mathcal{L}_i(\mathbf{w}) \right] = \frac{1}{N} \sum_{i=1}^{N} \nabla \mathcal{L}_i(\mathbf{w}) = \nabla \left(\frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i(\mathbf{w}) \right) = \nabla \mathcal{L}(\mathbf{w}).$$

Mini-batch SGD

Generalizing the gradient descent approaches from **GD** to **SGD**.
Sample a batch $B \subset \{1, \ldots, N\}$ of size $|B|$ and approximate the gradient as:

$$\mathbf{g} = \frac{1}{|B|} \sum_{i \in B} \nabla \mathcal{L}_i(\mathbf{w}_t),$$

and then update the weights as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \mathbf{g}.$$

Trade-off: (*pure SGD*) $|B| = 1$, (mini-batch SGD) $|B| = b$, (*full GD*) $|B| = N$.

Cost: SGD $\mathcal{O}(D)$ is N time cheaper than GD $\mathcal{O}(N \cdot D)$,

Remark: When the function is non-smooth (e.g. MAE) the subgradient can be used instead of the gradient for convex loss functions.

SGD Variants

Overcome local minima and accelerates convergence by adding a fraction of the previous weight update.

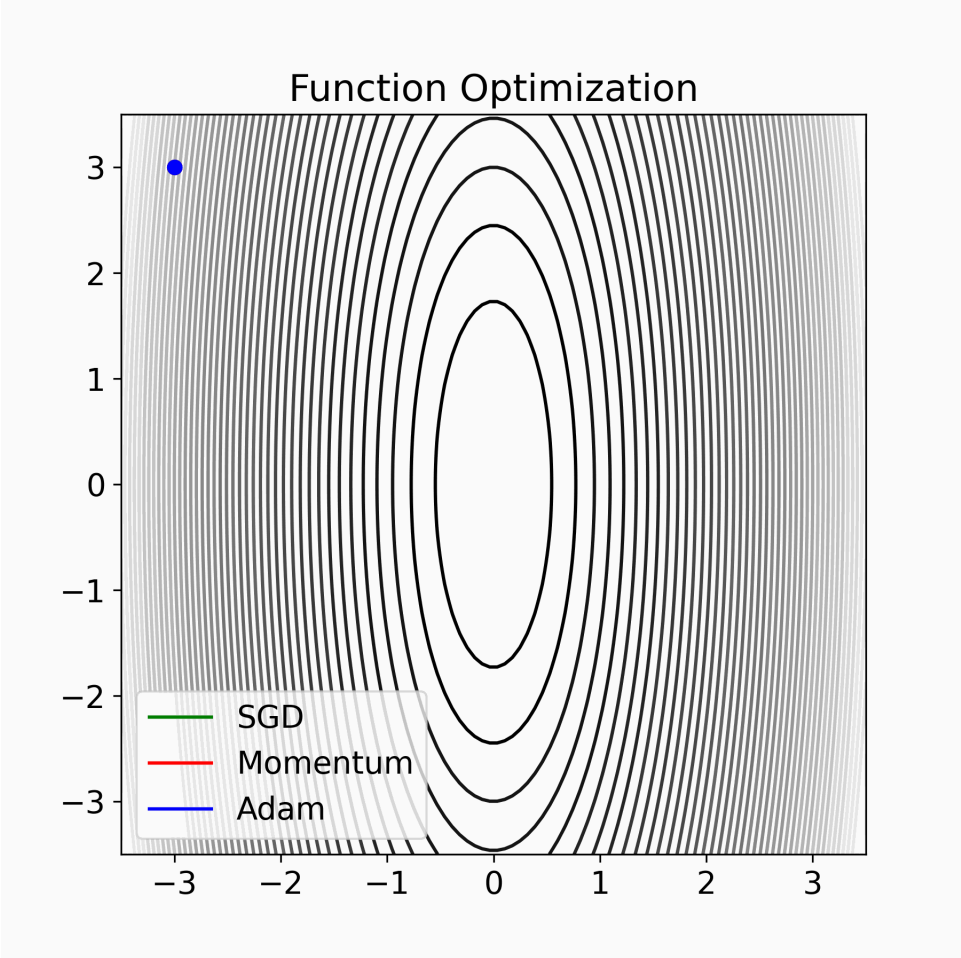
- **Momentum**: Heavy ball method, promoting the same direction and preventing oscillations.
  Chosen a stochastic gradient $\mathbf{g}$ the updating formula is given by:

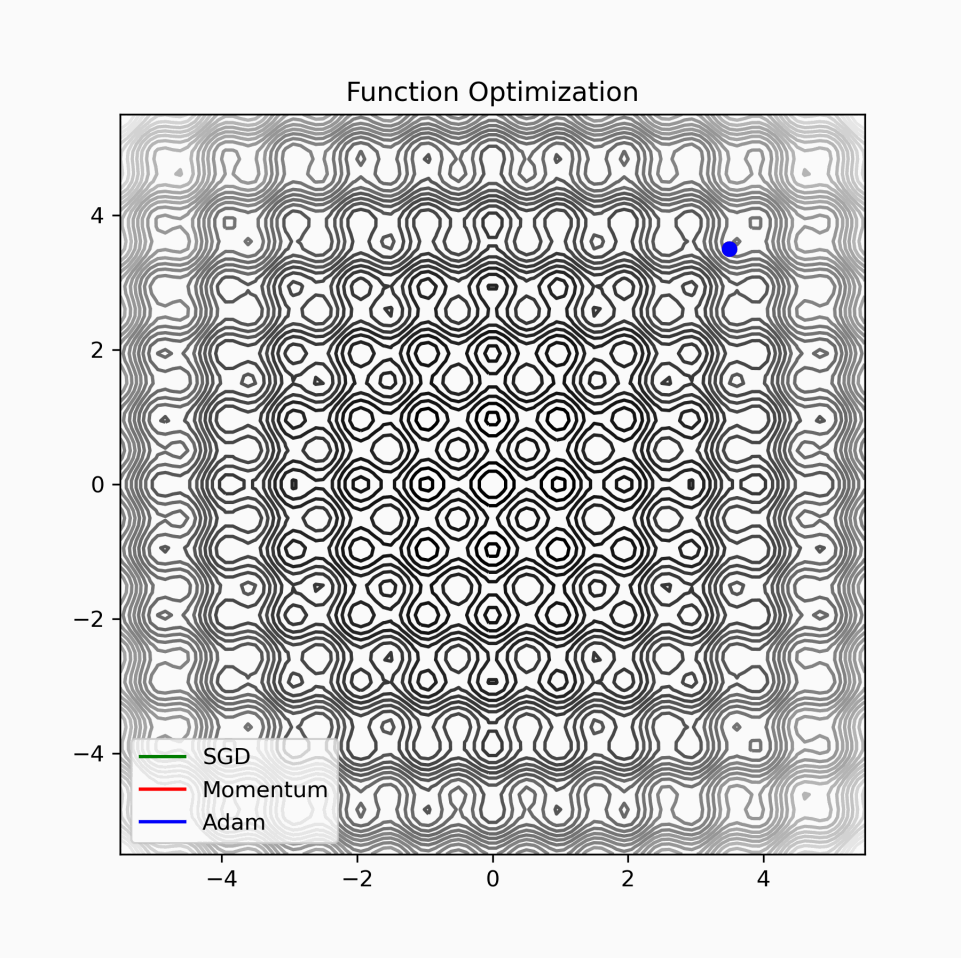
$$\mathbf{m}_{t+1} = \beta \mathbf{m}_t + (1-\beta)\mathbf{g},$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \mathbf{m}_{t+1},$$

- **ADAM**: Adaptive Moment Estimation, combining momentum and adaptive learning rates, and updating the weights as:

$$\mathbf{m}_{t+1} = \beta_1 \mathbf{m}_t + (1-\beta_1)\mathbf{g},$$
$$\mathbf{v}_{j,t+1} = \beta_2 \mathbf{v}_{j,t} + (1-\beta_2)(\mathbf{g}_j)^2, \quad \forall j$$
$$\mathbf{w}_{j,t+1} = \mathbf{w}_{j,t} - \frac{\gamma}{\sqrt{\mathbf{v}_{j,t+1}}}\mathbf{m}_{j,t+1}, \quad \forall j$$

Comparison between optimizers





Common issues

- Non-convex loss functions and $||\nabla \mathcal{L}(\mathbf{w})||$ close to zero $\rightsquigarrow$ possibly far from the global minima.
- If the learning rate γ is too big the method may diverge, if it is too small the convergence is slow.
- GD is sensitive to ill-conditioning, it is usually common to normalize features to preconditioning.
- Learning rate scheduler adapts the learning rate during training based on conditions.
- L2 (*Ridge*) and L1 (*Lasso*) regularization add penalty terms to the loss function preventing overfitting.
- Weights initialization to help the optimization process starting from a better initial guess.

Feedforward Neural Networks

Neural networks: functions $f(\mathbf{x}, \mathbf{w})$ consisting in layers (*depth*) of interconnected nodes (*width*), and are designed to identify and extract features from data.

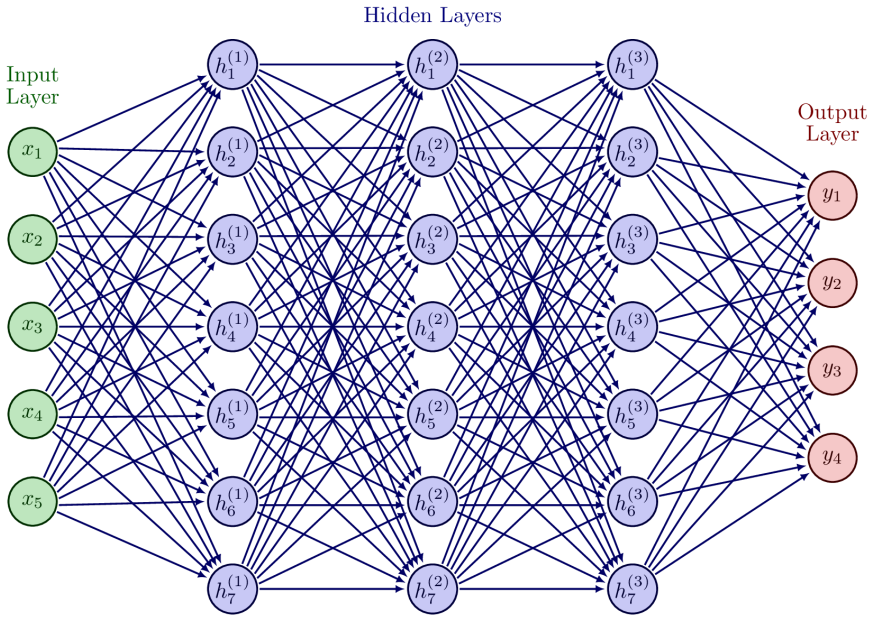
Goal: Learn an approximation to the function $F : \mathbb{R}^D \to \mathbb{R}^Q$ such that $\mathbf{y} = F(\mathbf{x})$.

Model: Given a dataset $\mathcal{S} = \{\mathbf{x}^i, \mathbf{y}^i\}_{i=1}^N$, find the weights $\mathbf{w}$ corresponding to the best approximation of the function F, i.e. $f(\mathbf{x}, \mathbf{w})$ is close to $F(\mathbf{x})$.

$$f(\mathbf{x}, \mathbf{w}) = (g^{(L+1)} \circ g^{(L)} \circ \cdots \circ g^{(1)})(\mathbf{x})$$

$$\mathbf{h}^{(l)} = g^{(l)}(\mathbf{h}^{(l-1)})$$

$$= \phi(\underbrace{\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}}_{\mathbf{z}^{(l)}})$$



FNNs definition

Features: composition of linear layers (weights and biases), and component-wise nonlinear activations

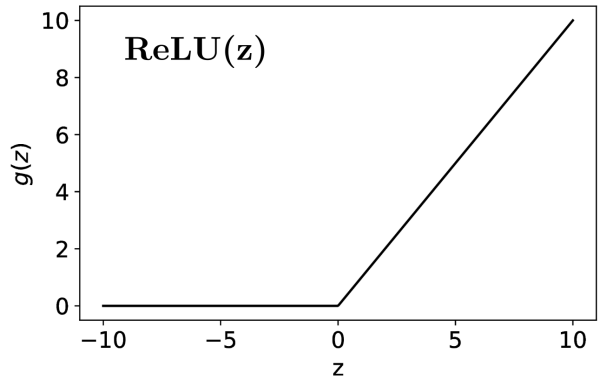
$$\mathbf{h}^{(l)} = g^{(l)}(\mathbf{h}^{(l-1)}) = \phi\big(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\big) = \phi(\mathbf{z}^{(l)}), \quad l = 1, \dots, L$$

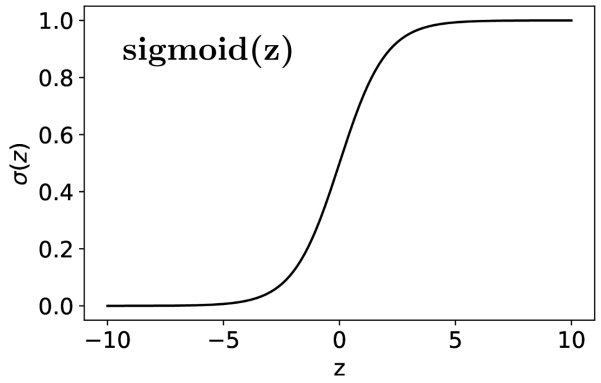
where L is the depth of the network, $\mathbf{h}^{(l)} \in \mathbb{R}^{m_l}$ is the $l-$th layer with m_l nodes ($\mathbf{h}^{(0)} = \mathbf{x}$ is the input data), biases $\mathbf{b}^{(l)} \in \mathbb{R}^{m_l}$ and weights $\mathbf{W}^{(l)} \in \mathbb{R}^{m_l \times m_{l-1}}$.

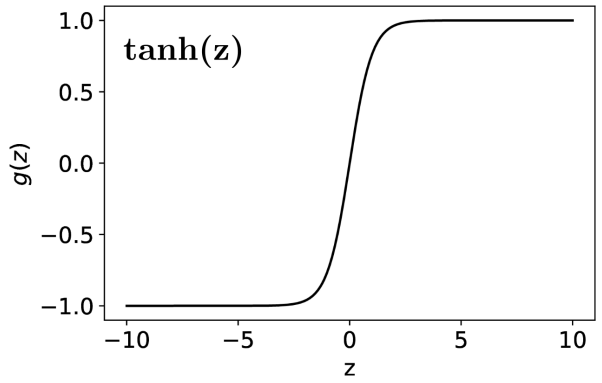
Nonlinearity: activations function ϕ, and layers $g^{(l)} : \mathbb{R}^{m_l} \to \mathbb{R}$.

Parameters: learnable weight matrices $\mathbf{W}^{(l)}$ and bias vectors $\mathbf{b}^{(l)}$.

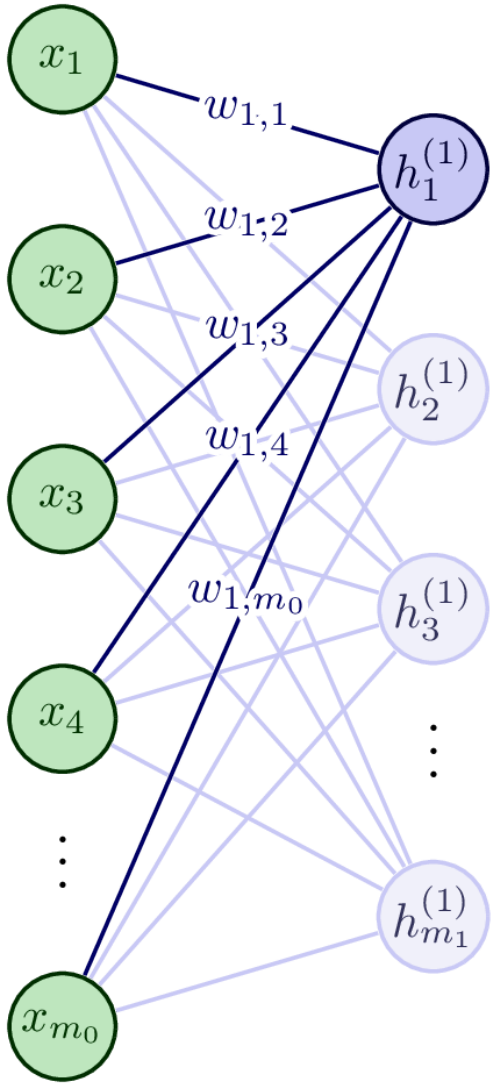
Remark: If $m_l = Q$, for all layers l, we learn $\mathcal{O}(DQ + Q^2L)$ parameters $\rightsquigarrow$ overparametrized NNs.







FNN component-wise



$$= g^{(1)}\left(w_{1,1}x_1 + w_{1,2}x_2 + \ldots + w_{1,m_0}x_{m_0} + b_1^{(1)}\right)$$

$$= g^{(1)}\left(\sum_{i=1}^{m_0} w_{1,i}x_i + b_1^{(1)}\right) = g^{(1)}\left((\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})_1\right)$$

$$\begin{pmatrix} h_1^{(1)} \\ h_2^{(1)} \\ \vdots \\ h_{m_1}^{(1)} \end{pmatrix} = g^{(1)}\left[\begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m_0} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m_0} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m_1,1} & w_{m_1,2} & \cdots & w_{m_1,m_0} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{m_0} \end{pmatrix} + \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_{m_1}^{(1)} \end{pmatrix}\right]$$

$$\mathbf{h}^{(1)} = g^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

Backpropagation

SGD algorithm: sample $k \in \{1, \dots, N\}$, compute the gradient of $\mathcal{L}_k$, update:

$$(w_{i,j}^{(l)})_{t+1} = (w_{i,j}^{(l)})_t - \gamma \frac{\partial \mathcal{L}_k}{\partial w_{i,j}^{(l)}}, \qquad (b_i^{(l)})_{t+1} = (b_i^{(l)})_t - \gamma \frac{\partial \mathcal{L}_k}{\partial b_i^{(l)}}.$$

Issue: applying chain-rules independently is inefficient due to compositions.

Solution: *backpropagation* exploiting chain-rule but reusing computations.

Task: given $\mathcal{L}_k = \frac{1}{2}(\mathbf{y}^k - g^{(L+1)} \circ \cdots \circ g^{(l+1)} \circ \phi(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}))^2$

$$\frac{\partial \mathcal{L}_k}{\partial w_{i,j}^{(l)}} = \sum_{q=1}^{Q} \frac{\partial \mathcal{L}_k}{\partial z_q^{(l)}} \frac{\partial z_q^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial \mathcal{L}_k}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial \mathcal{L}_k}{\partial z_j^{(l)}} \cdot x_i^{(l-1)} = \delta_j^{(l)} \cdot x_i^{(l-1)}$$

Forward: $\mathbf{x}^{(0)} = \mathbf{x}^k \in \mathbb{R}^D$, $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$, and $\mathbf{h}^{(l)} = \phi(\mathbf{z}^{(l)})$

Backward: $\boldsymbol{\delta}^{(L+1)} = \mathbf{z}^{(L+1)} - \mathbf{y}^k$, and $\boldsymbol{\delta}^{(l)} = \left[(\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)}\right] \odot \phi'(\mathbf{z}^{(l)})$

Architectural choices

Initialization: Improper parameter initialization can lead to the vanishing or exploding gradients
$\rightsquigarrow$ control layer-wise variance of neurons (He, Xavier).

Batch normalization: Normalize each layer's input with the mean and variance over the batch
$\rightsquigarrow$ stabilizes training, allows larger learning rates, reduces initialization impact.

Dropout: Randomly drop nodes in layer l at each training with probability $p^{(l)}$
$\rightsquigarrow$ better generalization, but typically requires more iterations to converge.

Data Augmentation: Generate new data from the available one (transformation or generative approaches)
$\rightsquigarrow$ encourages invariance, but task/dataset specific.

Weight decay: : Regularization for NNs by changing slightly the loss function as

$$\min_{\mathbf{w}} \mathcal{L} + \frac{\lambda}{2} ||\mathbf{W}^{(l)}||_F^2$$

$\rightsquigarrow$ favors small weights which can aid in generalization and optimization.

Supervised learning: error analysis

Goal: From a finite set of inputs and targets samples, we aim at finding a function that models the unknown relation between them, trying to predict the output once a new input is provided.
Ingredients:

- an *input space* $\mathcal{X} \subseteq \mathbb{R}^n$, a *target space* $\mathcal{Y} \subseteq \mathbb{R}^m$, and the space $\mathcal{Y}^{\mathcal{X}}$ of all the functions $\mathbf{f} : \mathcal{X} \to \mathcal{Y}$
- a *hypothesis space* $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$ for the model
- a *joint probability* distribution function $P_{\mathcal{X},\mathcal{Y}}$ defined on some σ-algebra on $\mathcal{X} \times \mathcal{Y}$ that represents the inputs distribution and the conditional probability of the target $\mathbf{y}$ being appropriate for an input $\mathbf{x}$
- a *loss metric* $d_M : \mathcal{Y} \times \mathcal{Y} \to [0, +\infty]$ measuring the mismatch between the output provided by the function $\mathbf{f}$ and the target,
- the *expected risk* $R(\mathbf{f}) = \mathbb{E}[d_M(\mathbf{y}, \mathbf{f}(\mathbf{x}))] = \int_{\mathcal{X}\times\mathcal{Y}} d_M(\mathbf{y}, \mathbf{f}(\mathbf{x}))dP_{\mathcal{X},\mathcal{Y}}(\mathbf{x}, \mathbf{y})$,
- a *training set* $S = \{(\hat{\mathbf{x}}_i, \hat{\mathbf{y}}_i),\ i = 1, \dots, N\}$ with $\hat{\mathbf{x}}_i \in \mathcal{X}$ and $\hat{\mathbf{y}}_i \in \mathcal{Y}$,
- the *empirical risk* $R_{S,N}(\mathbf{f}) = \frac{1}{N}\sum_{i=1}^{N} d_M(\hat{\mathbf{y}}_i, \mathbf{f}(\hat{\mathbf{x}}_i))$

Supervised learning: error analysis

Objective: Find a function $\hat{\mathbf{f}} \in \mathcal{Y}^{\mathcal{X}}$ that minimizes the *expected risk*, i.e.,

$$\hat{\mathbf{f}} = \operatorname{argmin}_{\mathbf{f} \in \mathcal{Y}^{\mathcal{X}}} R(\mathbf{f}),$$

representing the *optimal* input-output relationship corresponding to the joint probability distribution $P_{\mathcal{X},\mathcal{Y}}$.
Issue: the unknown model $\hat{\mathbf{f}}$ lives in the infinite dimensional space $\mathcal{Y}^{\mathcal{X}}$
Approximation: look for a model in a suitable hypothesis space $\mathcal{H}$ subset of $\mathcal{Y}^{\mathcal{X}}$, that is, computing

$$\hat{\mathbf{f}}_{\mathcal{H}} = \operatorname{argmin}_{\mathbf{f} \in \mathcal{H}} R(\mathbf{f}),$$

still unfeasible, since it requires both sampling the whole space $\mathcal{X} \times \mathcal{Y}$ and knowing the probability $P_{\mathcal{X} \times \mathcal{Y}}$.
Generalization: relying solely on the training set S we can define the model

$$\hat{\mathbf{f}}_{\mathcal{H},S} = \operatorname{argmin}_{\mathbf{f} \in \mathcal{H}} R_{S,N}(\mathbf{f}),$$

that is typically obtained via iterative *optimization* procedures, eventually finding an approximation $\hat{\mathbf{f}}^{*}_{\mathcal{H},S}$

Supervised learning: error analysis

Overall error: between the ideal model $\hat{\mathbf{f}}$ and the computed model $\hat{\mathbf{f}}^*_{\mathcal{H},S}$ can be decomposed as

$$
\begin{aligned}
|R(\hat{\mathbf{f}}) - R(\hat{\mathbf{f}}^*_{\mathcal{H},S})| \leq\ & |R(\hat{\mathbf{f}}) - R(\hat{\mathbf{f}}_{\mathcal{H}})| && (\text{approximation error}) \\
& + |R(\hat{\mathbf{f}}_{\mathcal{H}}) - R_{S,N}(\hat{\mathbf{f}}_{\mathcal{H}})| && (\text{generalization error}) \\
& + |R_{S,N}(\hat{\mathbf{f}}_{\mathcal{H}}) - R_{S,N}(\hat{\mathbf{f}}^*_{\mathcal{H},S})| && (\text{optimization error}) \\
& + |R_{S,N}(\hat{\mathbf{f}}^*_{\mathcal{H},S}) - R(\hat{\mathbf{f}}^*_{\mathcal{H},S})|
\end{aligned}
$$

where the last term can be bounded by the uniform generalization error given by

$$\sup_{\mathbf{f}\in\mathcal{H}} |R_{S,N}(\mathbf{f}) - R(\mathbf{f})|$$

Approximation error

Measures the expressivity of the model, i.e. how well a given input-output function can be approximated by functions belonging to the hypothesis space.

Shallow FNNs: with one hidden layer are universal approximators: under suitable assumptions, any continuous function on a compact set can be approximated by a shallow FNN up to an arbitrary precision. The following definition serves in the statement of Cybenko's theorem.

Definition: Let $K \subset \mathbb{R}^n$ be a compact set. A continuous function $\sigma : \mathbb{R} \to \mathbb{R}$ is said discriminatory with respect to K if, for every finite, signed, regular Borel measure μ on K, we have that

$$\left(\int_K \sigma(\mathbf{w} \cdot \mathbf{x} + b) d\mu(\mathbf{x}) = 0, \text{ for all } \mathbf{w} \in \mathbb{R}^n \text{ and } b \in \mathbb{R} \right) \Rightarrow \mu = 0.$$

Theorem (Universal Approximation Theorem) [Cybenko (1989)]: Let $K \subset \mathbb{R}^n$ be a compact set and $\sigma \in \mathcal{C}(\mathbb{R})$ a discriminatory activation function on K. Then, given any real-valued function $\hat{f}$ continuous on K and tolerance $\varepsilon > 0$, there exists a shallow FNN with weights and biases $\mathbf{w} = \{W^{[i]}, \mathbf{b}^{[i]}\}_{i=1}^2$ and real output $y = f(\cdot; \mathbf{w})$ such that $\|\hat{f} - f(\cdot; \mathbf{w})\|_\infty \leq \varepsilon$.

Complexity

Denoting by N_1 the number of hidden neurons, Cybenko's theorem ensures that the larger the number N_1 of parameters, the lower the tolerance ε. But, how does N_1 depend on ε?

Theorem (Lower and upper complexity bound for shallow NNs): Let $n \geq 2$, $s \in \mathbb{N}$, I an open real interval, $\sigma : \mathbb{R} \to \mathbb{R}$ such that $\sigma|_I \in \mathcal{C}^\infty(I)$ and $\sigma^{(k)}(x_0) \neq 0$ for some $x_0 \in I$ and all integers $k \geq 1$. Then, for any $\varepsilon > 0$ and for any $\hat{f} \in W^{s,2}((0,1)^n)$ with $\|\hat{f}\|_{W^{s,2}((0,1)^n)} \leq 1$, there exists a shallow FFNN with weights $\mathbf{w} = \{W^{[i]}, \mathbf{b}^{[i]}\}_{i=1}^2$ depending on $\hat{f}$ and ε, real output $y = f(\cdot; \mathbf{w})$, and N_1 hidden neurons s.t.

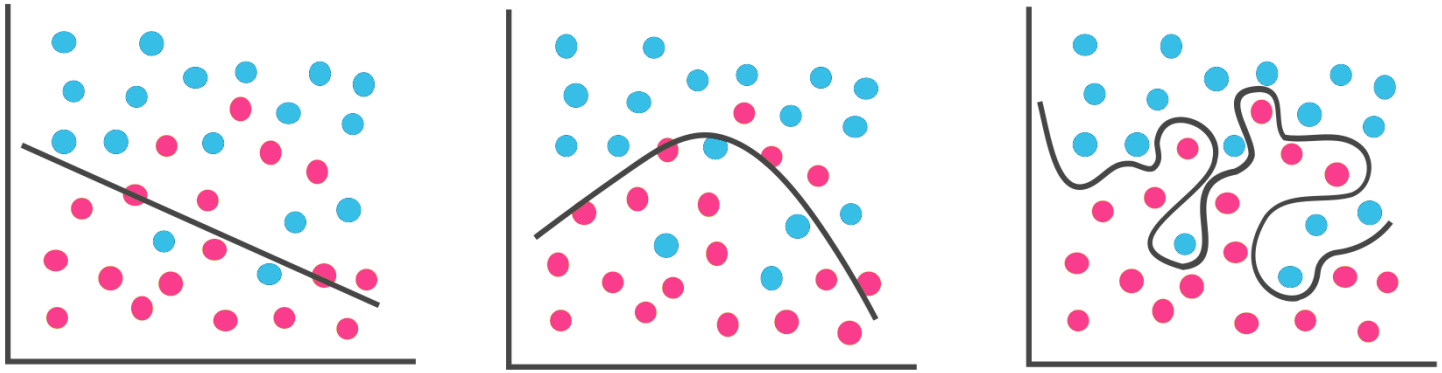
$$\|\hat{f} - f(\cdot; \mathbf{w})\|_2 \leq \varepsilon \qquad \text{and} \qquad \left(\frac{1}{\varepsilon}\right)^{(n-1)/s} \lesssim N_1 \lesssim \left(\frac{1}{\varepsilon}\right)^{n/s}.$$

- the smaller the tolerance ε, the larger the number N_1 of hidden neurons
- the higher the regularity s of $\hat{f}$, the lower the number N_1 of hidden neurons,
- the larger the input dimension n, the larger the number N_1 of hidden neurons
  $\rightsquigarrow$ FNNs suffer from the curse of dimensionality, i.e. the number of parameters of a shallow FNN grows exponentially in the input dimension n.

Generalization error

Goal: predict the target for unobserved inputs outside the training data, balancing two competing tasks

- minimizing the empirical risk, i.e. training error, $R_{S,N}(\hat{\mathbf{f}}_{\mathcal{H},S})$
  $\rightsquigarrow$ enriching the hypothesis space $\mathcal{H}$, if $\mathcal{H}$ fits too much the training set S we observe *overfitting*
- minimizing the gap $|R(\hat{\mathbf{f}}_{\mathcal{H}}) - R_{S,N}(\hat{\mathbf{f}}_{\mathcal{H}})|$ between the expected risk and the empirical risk for any $\hat{\mathbf{f}}_{\mathcal{H}}$
  $\rightsquigarrow$ if the hypothesis space $\mathcal{H}$ is too coarse, the model is not accurate, and we observe *underfitting*



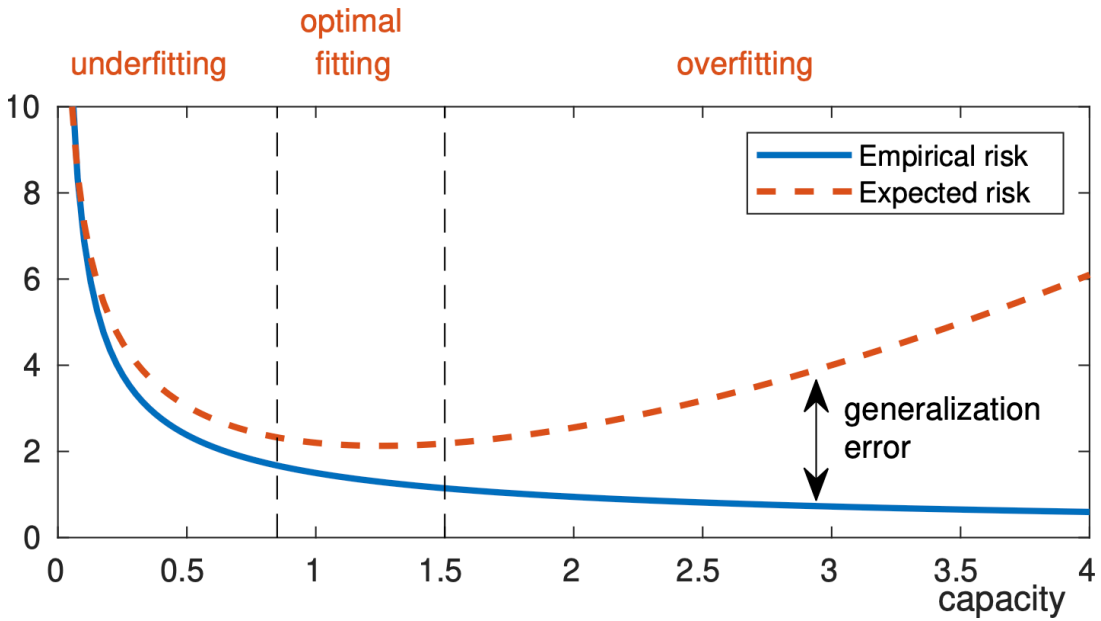
Capacity

Objective: understand how rich the hypothesis space $\mathcal{H}$ must be in the whole space $\mathcal{Y}^{\mathcal{X}}$ to model accurately the relation between inputs and targets, i.e. a possible indicator of the flexibility of a model

$\rightsquigarrow$ small capacity corresponds to underfitting, while high capacity may induce overfitting.

Features: the capacity is determined by the choice of the hyperparameters

$\rightsquigarrow$ the larger the capacity of the model, the smaller the empirical risk $R_{S,N}(\hat{\mathbf{f}}_{\mathcal{H},S})$, while the expected risk increases with the capacity when the latter is too large. The fitting is optimal at the minimum for $R(\hat{\mathbf{f}}_{\mathcal{H}})$.



Cross validation

A useful technique to combat overfitting, avoiding the evaluation of the generalization error
Idea: splitting a given data set $S = \{(\hat{\mathbf{x}}_i, \hat{\mathbf{y}}_i)\}_{i=1}^{N}$ into two distinct groups:

- the *training set* $S_t = \{(\hat{\mathbf{x}}_i, \hat{\mathbf{y}}_i)\}_{i=1}^{N_t}$, to update NN weights and biases minimizing the empirical risk
- the *validation set* $S_v = \{(\hat{\mathbf{x}}_i, \hat{\mathbf{y}}_i)\}_{i=N_t+1}^{N}$, to check the performance of the current network.

We define the *training error* and the *validation error* as

$$\mathcal{E}_t = R_{S_t,N_t}(\mathbf{f}) = \frac{1}{N_t}\sum_{i=1}^{N_t} d_M(\hat{\mathbf{y}}_i, \mathbf{f}(\hat{\mathbf{x}}_i;\mathbf{w})) \text{ and } \mathcal{E}_v = R_{S_v,N_v}(\mathbf{f}) = \frac{1}{N_v}\sum_{i=N_t+1}^{N} d_M(\hat{\mathbf{y}}_i, \mathbf{f}(\hat{\mathbf{x}}_i;\mathbf{w})).$$

- Large training errors $\mathcal{E}_t$ (underfitting) can be mitigated by increasing the model capacity, i.e. by augmenting the number of parameters.
- Large values for $\mathcal{E}_t - \mathcal{E}_v$ (overfitting) can be treated by increasing the cardinality of the set S, decreasing the capacity of the model, or employing regularization techniques
  $\rightsquigarrow$ Stop the optimization process at a stage where no improvement is seen in the validation error.

Convolutional Neural Networks

Issue: FNNs do not scale very well, need a lot of resources, prone to overfitting.

Example: A dataset with images with 32x32 pixels, 3 color channels (CIFAR-10).
$\rightsquigarrow$ A single fully-connected neuron in the first layer have 32x32x3 = 3072 weights.

Idea: Sharing the weights across different neurons for lighter architectures.

Architecture: *Convolutional Neural Networks* (CNNs) characterized by three main kinds of layers:

1. Convolutional layer: to learn the main features of the dataset
2. Pooling layer: to help in reducing the dimensionality
3. Fully-connected layer: to combine information and predict the output

CNN definition

Framework: $\mathbf{x} \in \mathbb{R}^{n \times m}$ is the image, $\mathbf{w} \in \mathbb{R}^{k \times l}$ is the kernel/filter.

Consider a single-channel convolutional layer (e.g. for grayscale image)

$$y_{i,j} = \sum_p \sum_q x_{i-p,j-q} \cdot w_{p,q}$$

$y_{i,j}$ is the output value, $x_{i-p,j-q}$ is the input value, and $w_{p,q}$ is the filter's weight.

Strategy: sliding (*stride*) the single filter over the input, performing an element-wise multiplication and sum within the filter's receptive field

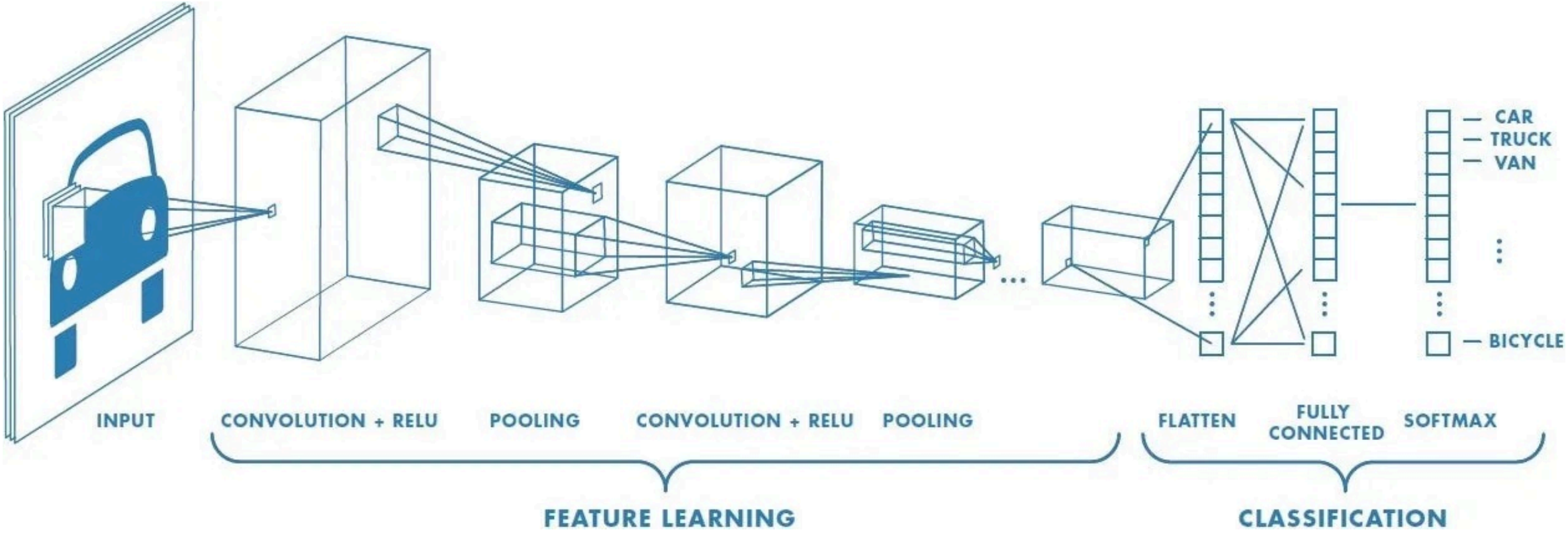
$$\text{Output Size} = \frac{\text{Input Size} - \text{Filter Size}}{\text{Stride}} + 1$$

Properties: local filters, $y_{i,j}$ only depends on points only depends on the values of $\mathbf{x}$ near (i, j).
$\rightsquigarrow$ $\mathbf{w}$ are learnable weights for all positions (*shared*).

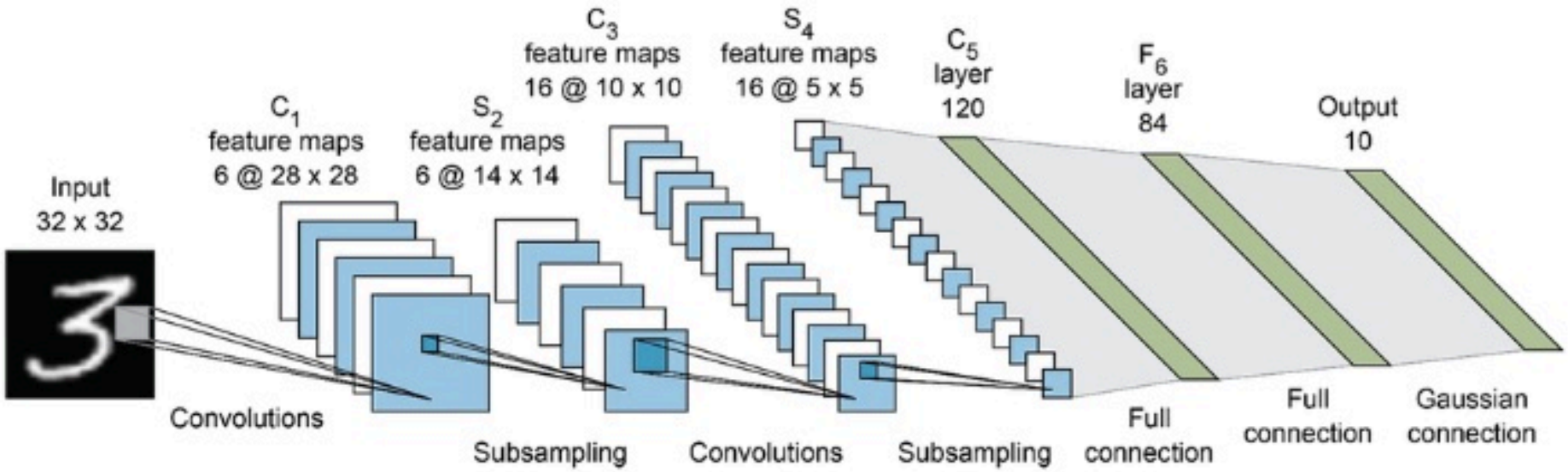
Translation equivariance: a shifted input results in a shifted output.

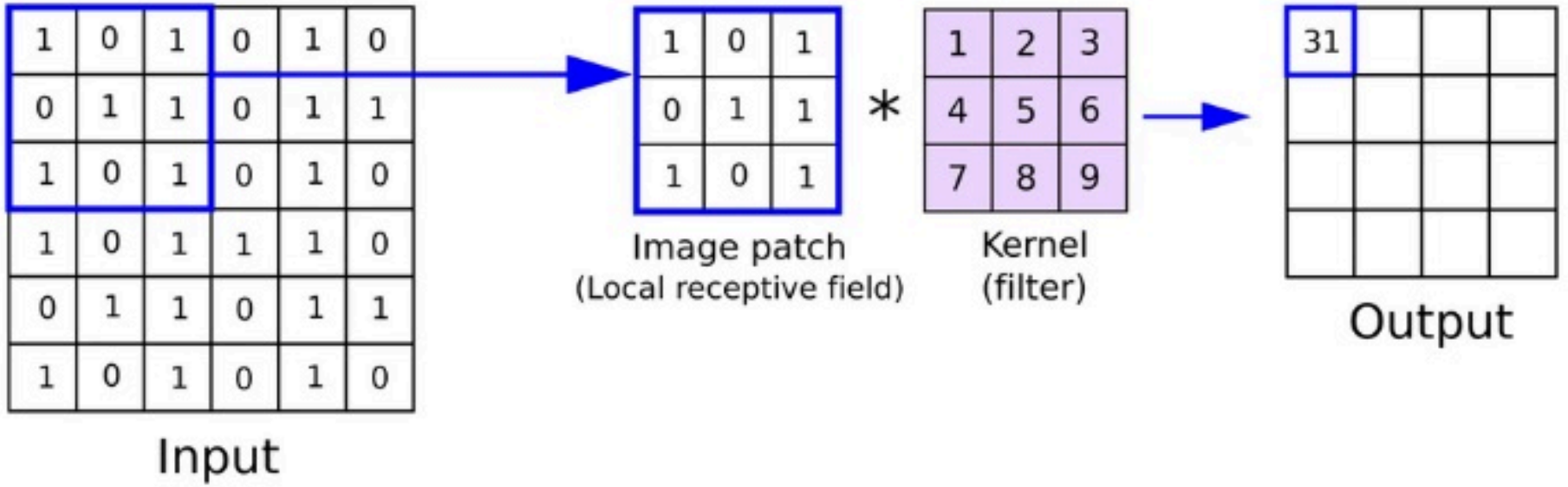
CNN architecture

Classification: CNNs can identify from simple features (colors, edges) to more complex objects (elements, shapes), finally classifying e.g. vehicles.



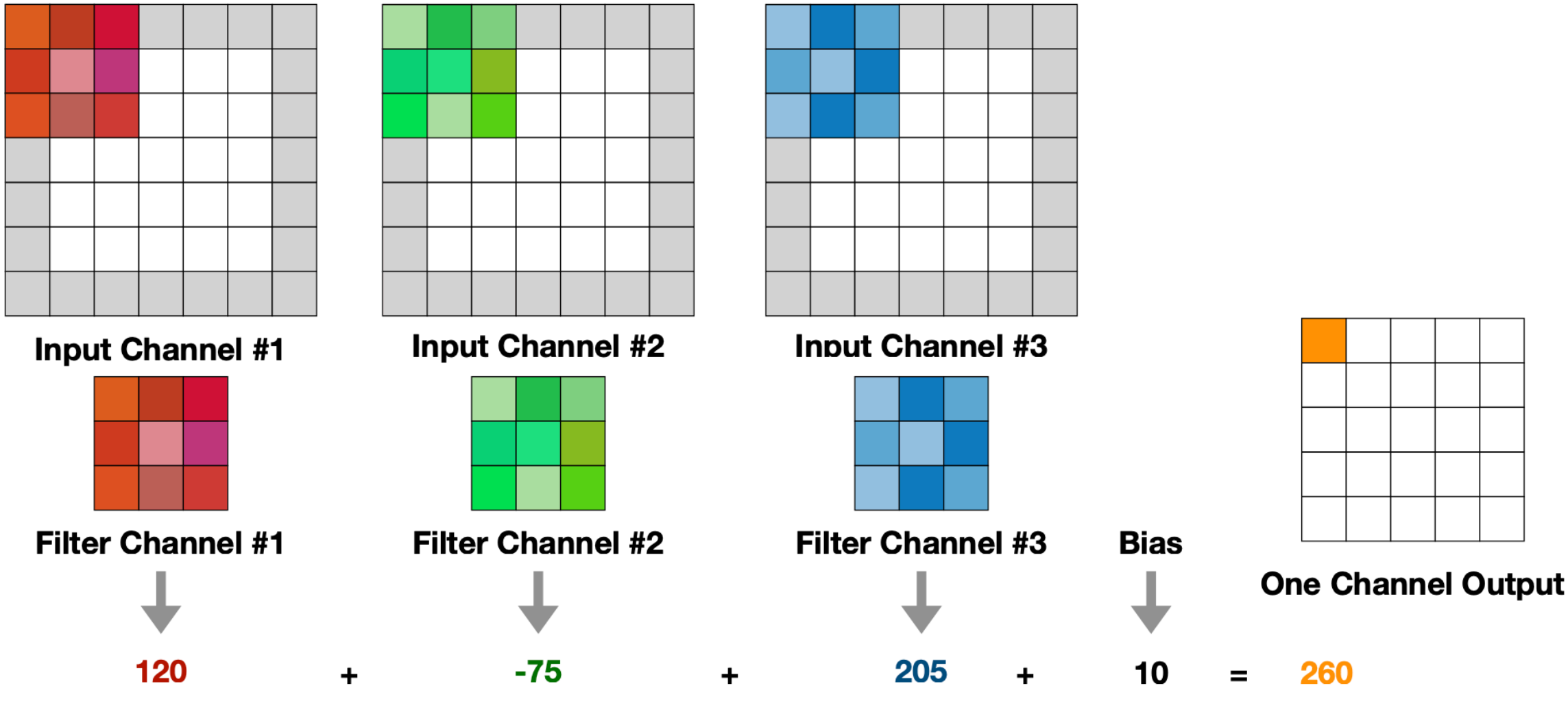
Convolutions





Multi-channel convolution

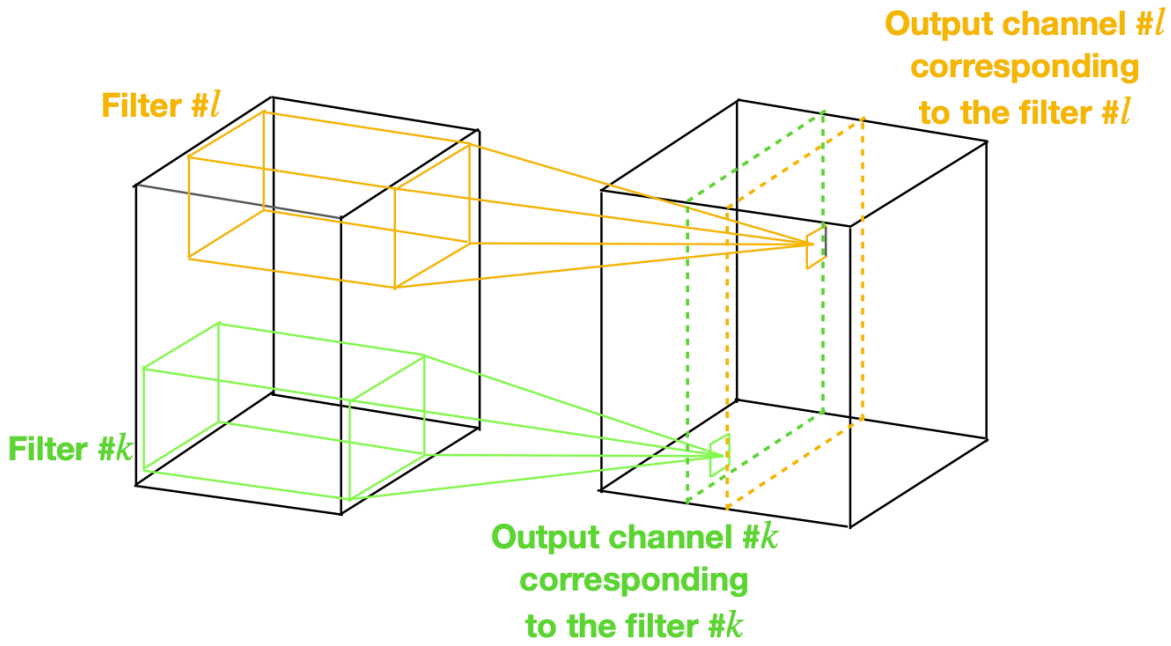
For multi-channel inputs, the filter has the same number of channels as the input



CNN architecture

Model: CNNs consist of sparsely connected convolutional layers requiring fewer parameters which are universal across different locations.

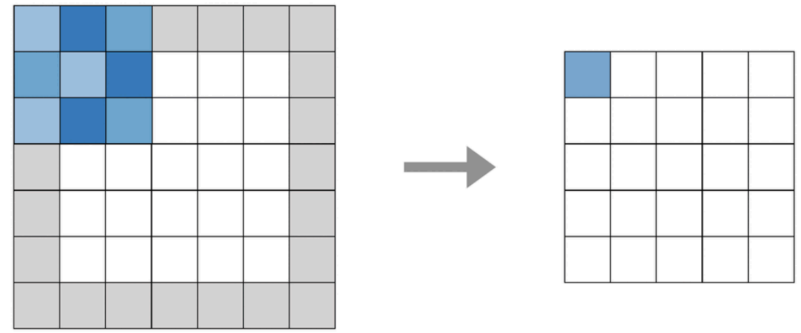
1. A convolutional layer is composed of multiple filters
2. Each output channel corresponds to its own independent filter
3. Hyper-parameters of the convolutional layer: size, padding, stride

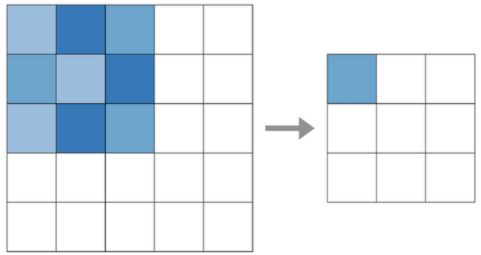


Padding

Goal: Padding help in dealing with borders, different strategies are available

1. **Zero padding**: Add zeros to each side of the input's boundaries
   ⇝ Output has same dimension
2. **Valid padding**: Convolutions only if the entire filter fits inside the image
   ⇝ Output has smaller dimension

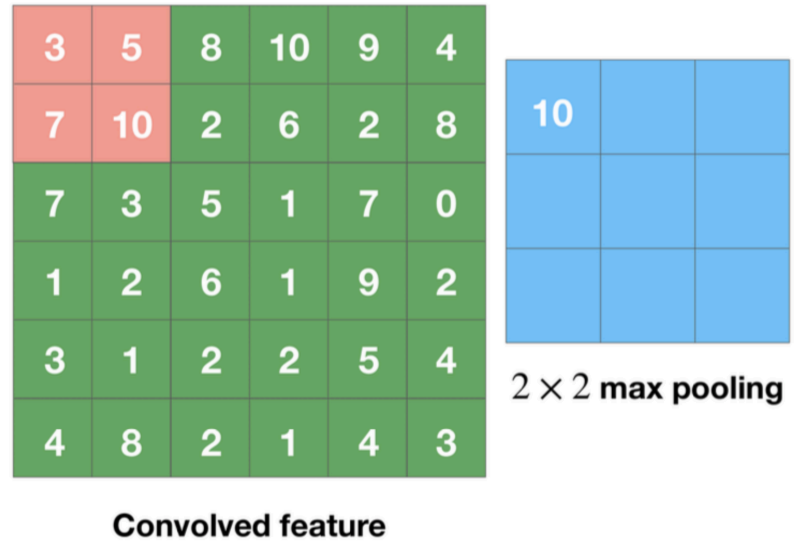


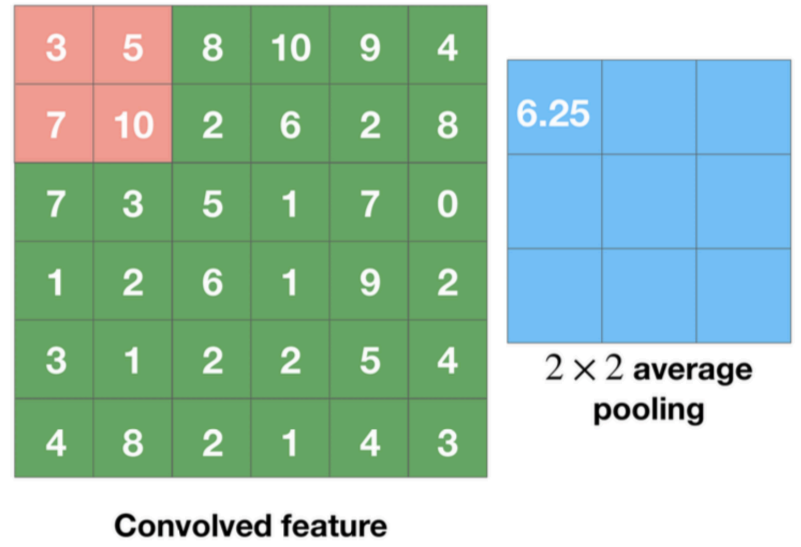


Pooling

Goal: Non-learnable downsampling operation that reduces the spatial dimensions of the image, hyperparameters are the size, type, and stride of the pooling.

1. **Max pooling**: Maximum value of the portion of the convolved feature that is covered by the kernel
2. **Average pooling**: Average value of the portion of the convolved feature that is covered by the kernel





References

- Hesthaven, J.S., Rozza, G., Stamm, B., 2015. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*, Springer International Publishing AG, Cham. https://doi.org/10.1007/978-3-319-22470-1
- Quarteroni, A., Manzoni, A., Negri, F., 2016. *Reduced Basis Methods for Partial Differential Equations: An Introduction*, Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-15431-2
- Rozza, G., Ballarin, F., Scandurra, L., Pichi, F., 2024. *Real Time Reduced Order Computational Mechanics: Parametric PDEs Worked Out Problems*, SISSA Springer Series. https://doi.org/10.1007/978-3-031-49892-3
- Quarteroni, A., Gervasio, P., Regazzoni, F., 2025. *Combining physics-based and data-driven models: advancing the frontiers of research with scientific machine learning*. Math. Models Methods Appl. Sci. 35, 905–1071. https://doi.org/10.1142/S0218202525500125
- Cuomo, S., Di Cola, V.S., Giampaolo, F., Rozza, G., Raissi, M., Piccialli, F., 2022. *Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next*. J Sci Comput 92. https://doi.org/10.1007/s10915-022-01939-z
- Prince, S. *Understanding Deep Learning*, 2025. MIT Press. https://mitpress.mit.edu/9780262048644/understanding-deep-learning/
- Bach, F. *Learning Theory from First Principles*, 2024. MIT Press. https://mitpress.mit.edu/9780262049443/learning-theory-from-first-principles/
- Peyré, G., 2022. *Mathematical foundations of data sciences*. https://www.numerical-tours.com
- Nielsen, M.A., 2015. *Neural Networks and Deep Learning*. http://neuralnetworksanddeeplearning.com