

Advanced methodologies for the assessment of the fatigue response

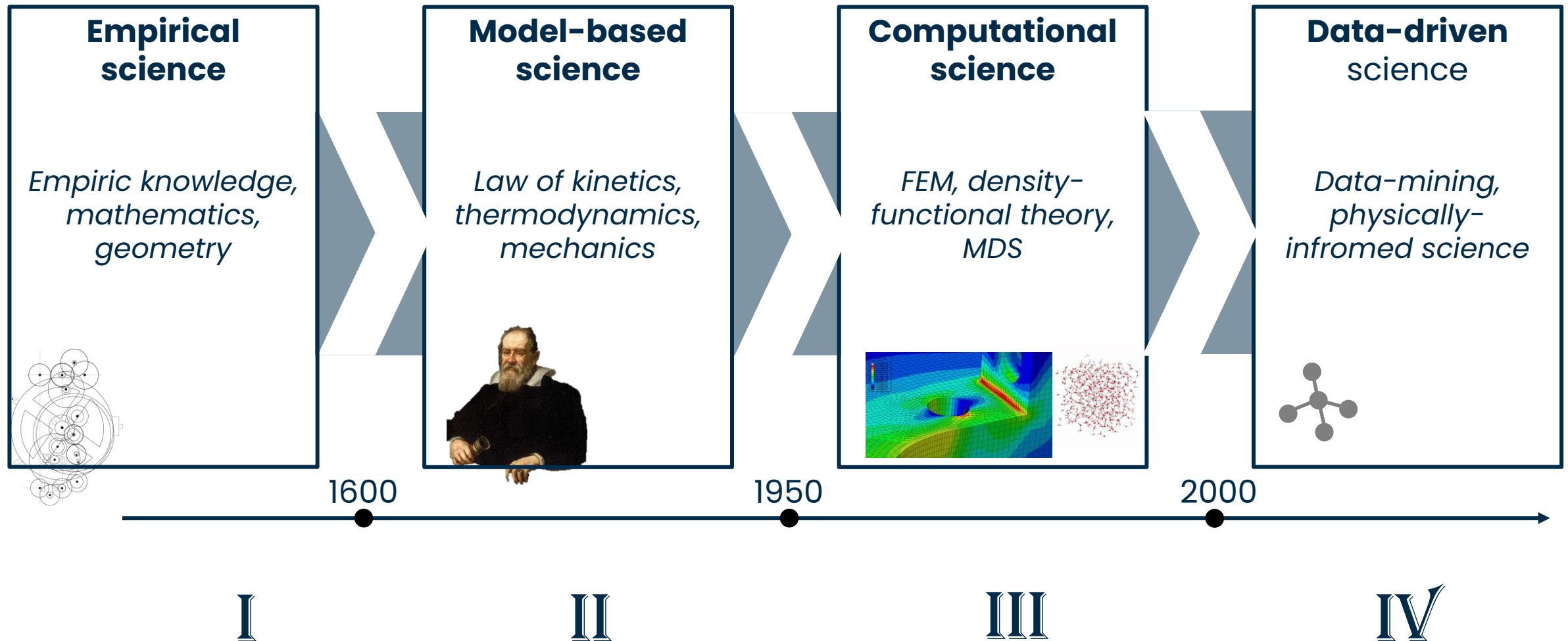
Fatigue and Machine learning: Introduction

A. Ciampaglia



**Politecnico
di Torino**

Paradigms of science

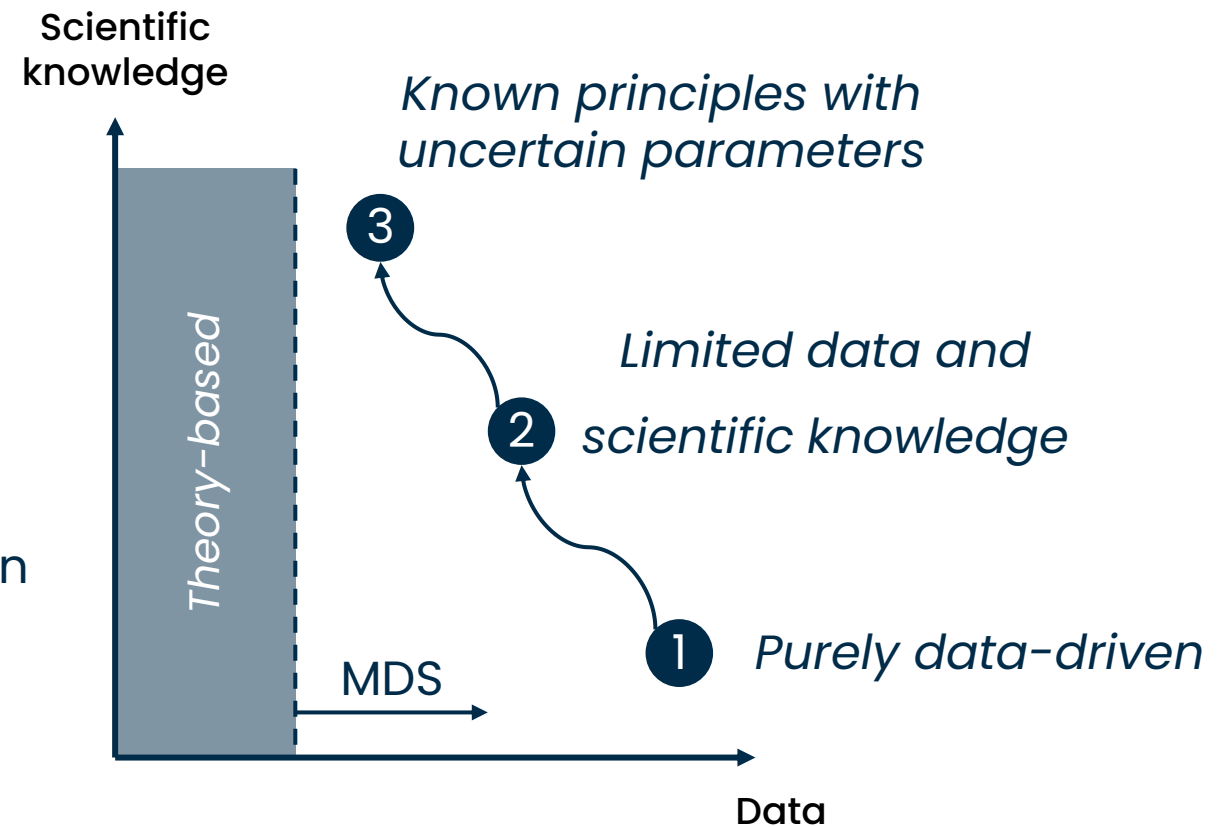


Mechanistic Data Science

«Mechanistic data science combines mechanistic calibrated principles and collected data to accelerate the knowledge extraction and improve predictive capacity»

The development of MDS models follows six steps:

1. Multimodal data generation/collection
2. Extraction of mechanistic features
3. Knowledge-driven dimension reduction
4. Reduced order surrogate models
5. Data science method for regression classification
6. System design



Course outline

Part I – Tuesday May 14th

- Introduction to Machine Learning
 - Definitions and classifications
 - Main architectures
- Multi-layer perceptron
 - Network and neurons
 - Training algorithm
 - The loss-function

Hands-on tutorial: build a NN with Google Tensorflow

Course outline

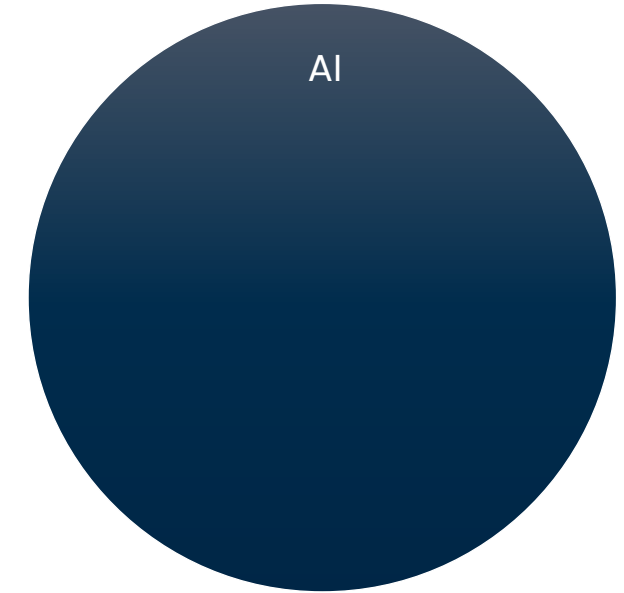
Part I – Tuesday May 14th

- Introduction to Machine Learning
 - Definitions and classifications
 - Main architectures
- Multi-layer perceptron
 - Network and neurons
 - Training algorithm
 - The loss-function

Hands-on tutorial: build a NN with Google Tensorflow

Definitions

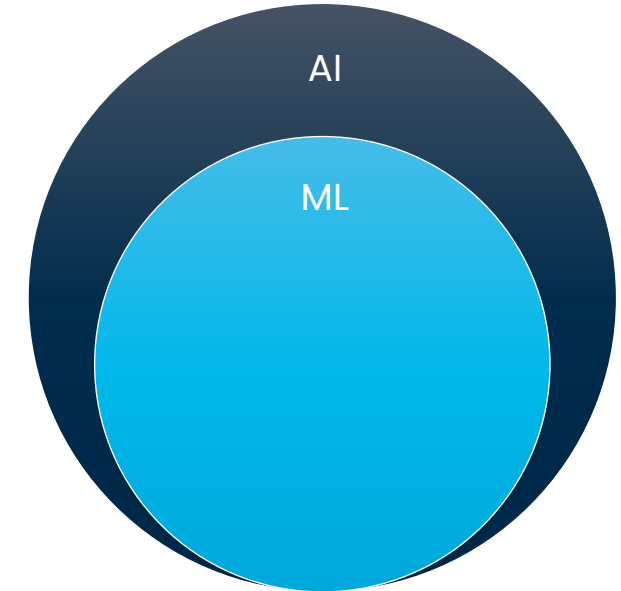
Artificial Intelligence is a technique which enables the machine to **mimic the human behaviour**.



Definitions

Artificial Intelligence is a technique which enables the machine to **mimic the human behaviour**.

Machine Learning is a type of Artificial Intelligence that provides computers with the ability to learn **without being explicitly programmed**.

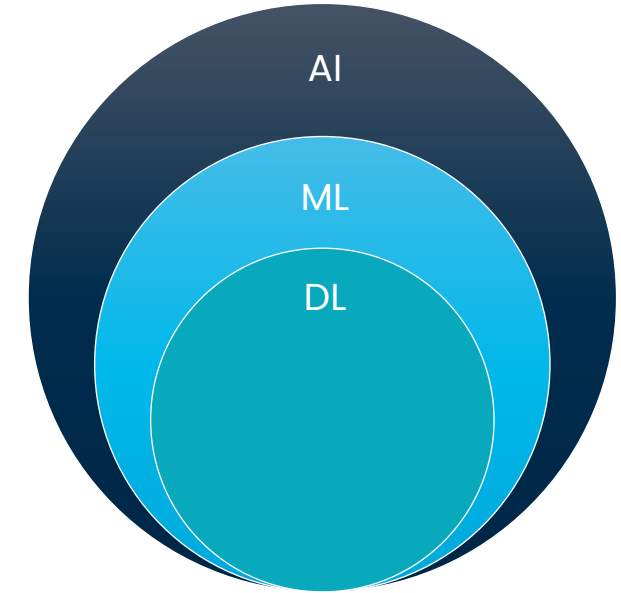


Definitions

Artificial Intelligence is a technique which enables the machine to **mimic the human behaviour**.

Machine Learning is a type of Artificial Intelligence that provides computers with the ability to learn **without being explicitly programmed**.

Deep Learning is a branch of Machine Learning that comprehends the techniques using **deep layered structures**.



Definitions

Artificial Intelligence is a technique which enables the machine to **mimic the human behaviour**.

Machine Learning is a type of Artificial Intelligence that provides computers with the ability to learn **without being explicitly programmed**.

Deep Learning is a branch of Machine Learning that comprehends the techniques using **deep layered structures**.

Learning approaches

Supervised learning

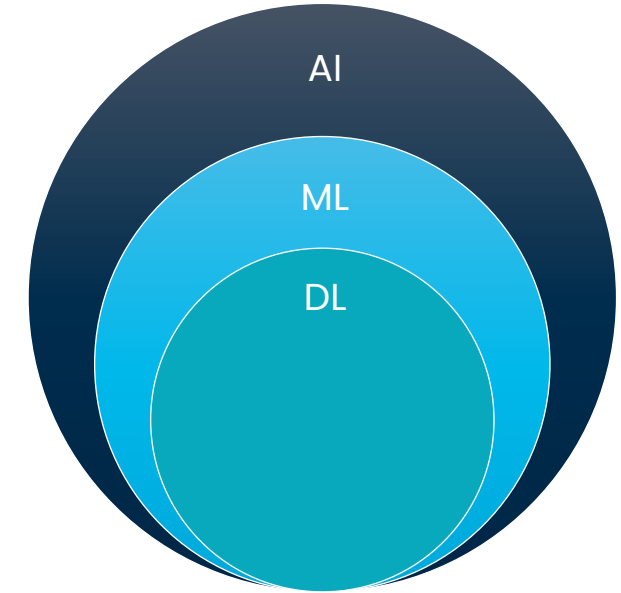
Learn to predict an output \mathbf{y} from an input \mathbf{x} from a set of **labelled data** (i.e., training dataset).

Unsupervised learning

Discovers **patterns** in unlabeled data \mathbf{x} using similarity metrics.

Reinforcement learning

Learn to best react to an input \mathbf{x} with a decision \mathbf{y} based on the effect of \mathbf{y} on a defined **scoring** function.



Identify CATs and DOGs trained on labelled pictures

Cluster similar documents based on the text content

Learn to play chess by winning or losing

Type of problems

Regression

Learn a **continuous** function that **predict** the features y from the inputs x on the **labelled data** (X, Y) :

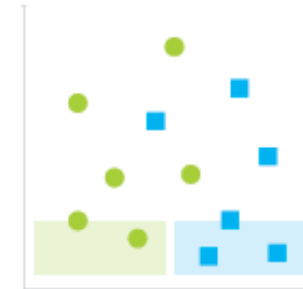
$$f: f(x) = y \mid \min_f E(Y, f(X))$$



Classification

Learn a discrete function that **predicts** the **class k** from the input x on the **labelled data** (X, Y) :

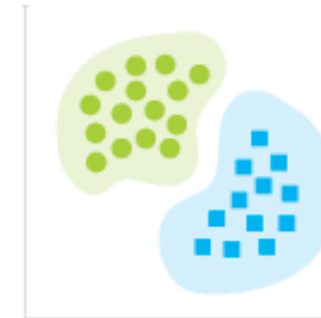
$$f: f(x) = k \mid \min_f E(Y, f(X))$$



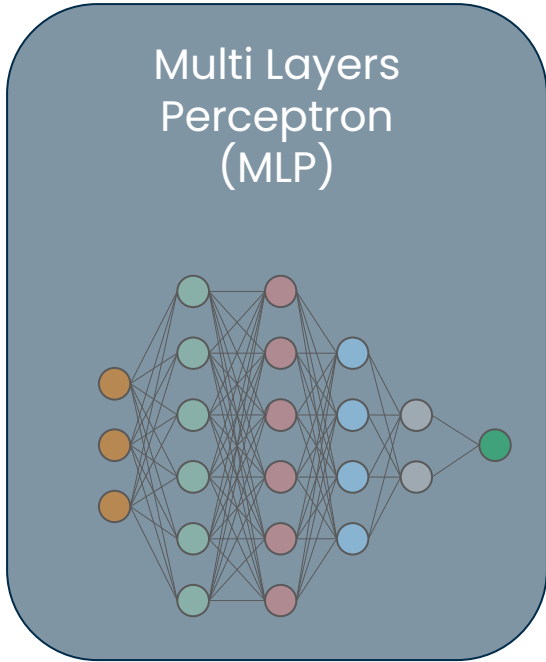
Clustering

Cluster a set of **unlabelled observations** X into k cluster by maximizing a similarity metric L :

$$k \mid \max_k L(X_k)$$



Main Architectures



Regression
Classification
Supervised
training

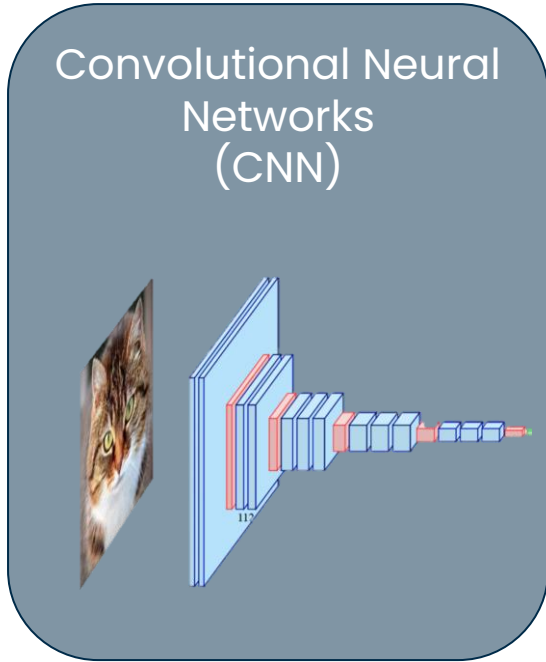
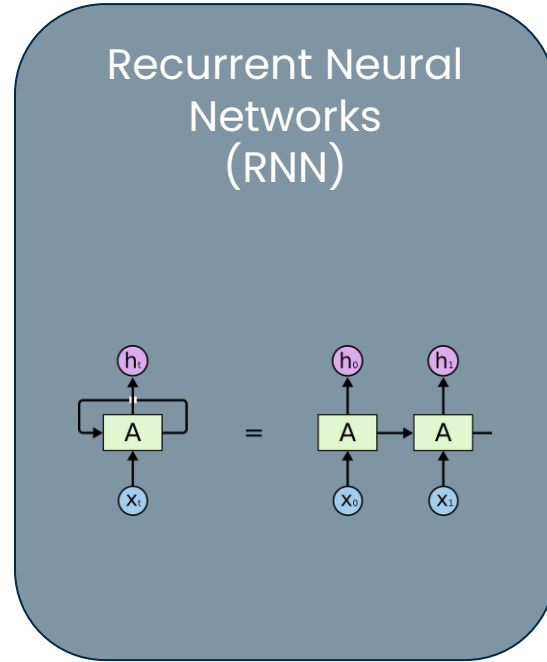
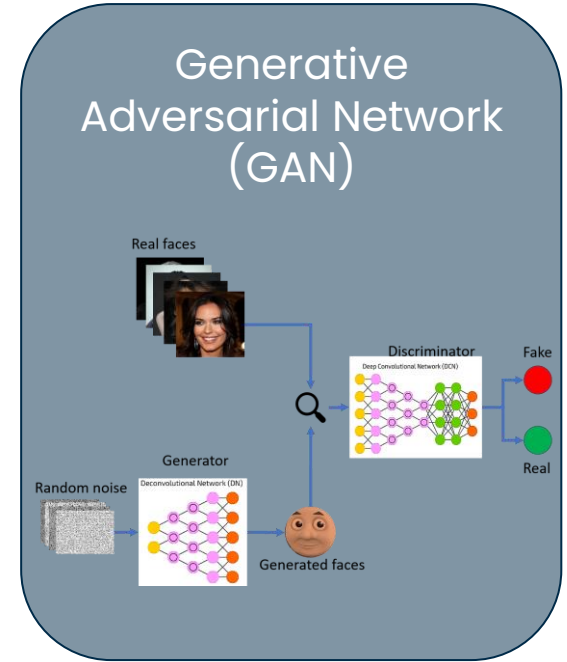


Image processing
Classification
Supervised
training



Time history processing
Classification
Regression
Supervised
training



Generative
algorithm
Adversarial
training

Course outline

Part I – Tuesday May 14th

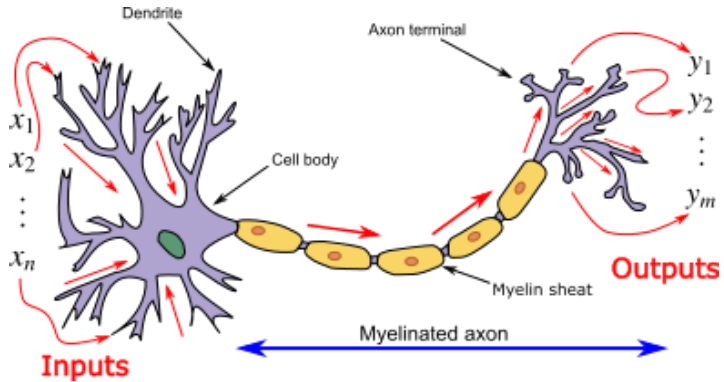
- Introduction to Machine Learning
 - Definitions and classifications
 - Main architectures
- Multi-layer perceptron
 - Network and neurons
 - Training algorithm
 - The loss-function

Hands-on tutorial: build a NN with Google Tensorflow

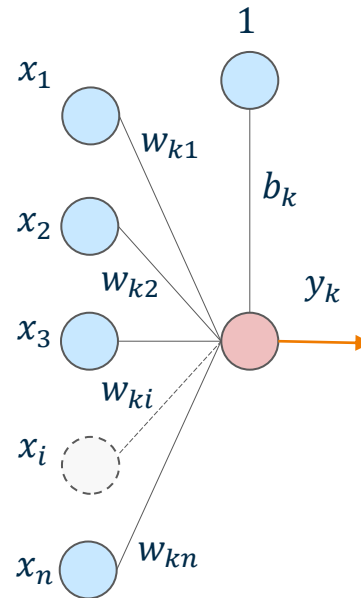
MLP: Network and neurons

Is based on the idea of an artificial neuron:

Human neuron



Artificial neuron



Mathematical Expression

$$y_k = \mathcal{A} \left(b_k + \sum_i w_{ki} x_i \right)$$

or

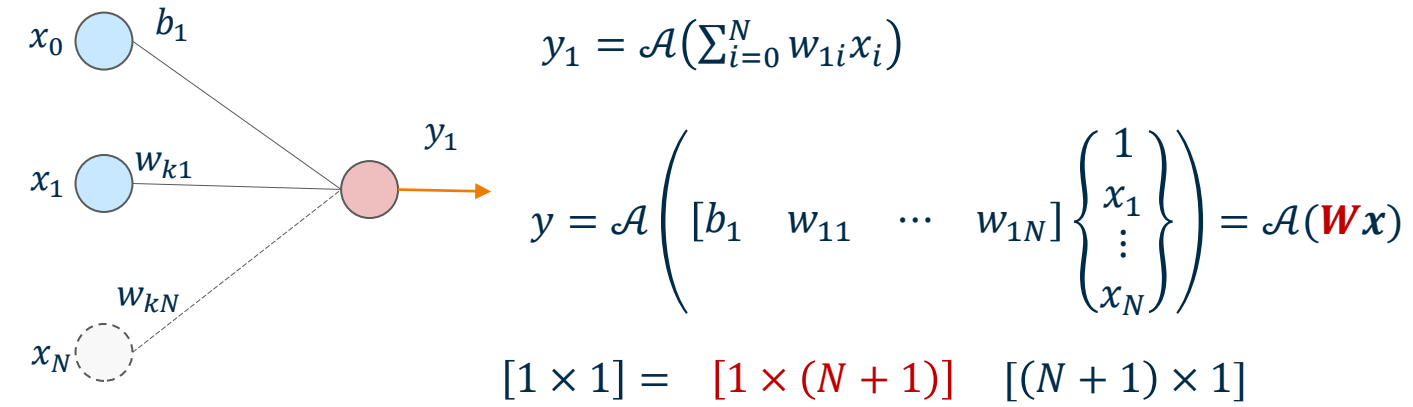
$$\begin{cases} y_k = \mathcal{A} \left(\sum_{i=0}^n w_{ki} x_i \right) \\ x_0 = 1, w_{k0} = b_k \end{cases}$$

[1] Mcculloch, Warren and Pitts, Walter. "A Logical Calculus of Ideas Immanent in Nervous Activity." *Bulletin of Mathematical Biophysics* 5 (1943): 127--147.

MLP: Network and neurons

The Multi Layer Perceptron (i.e., a Neural Network) is a stacked architecture of interconnected artificial neurons.

Single output

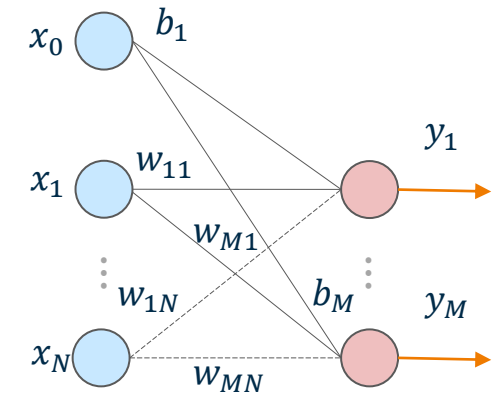


N+1 Parameters

MLP: Network and neurons

The Multi Layer Perceptron (i.e., a Neural Network) is a stacked architecture of interconnected artificial neurons.

Multiple output



$$y_i = \mathcal{A} \left(\sum_{i=0}^N w_{1i} x_i \right)$$

$$\begin{Bmatrix} y_1 \\ \vdots \\ y_M \end{Bmatrix} = \mathcal{A} \left(\begin{bmatrix} b_1 & w_{11} & \cdots & w_{1N} \\ \vdots & \cdots & \cdots & \vdots \\ b_N & w_{M1} & \cdots & w_{MN} \end{bmatrix} \begin{Bmatrix} 1 \\ x_1 \\ \vdots \\ x_N \end{Bmatrix} \right) \rightarrow \mathbf{y} = \mathcal{A}(\mathbf{W}\mathbf{x})$$

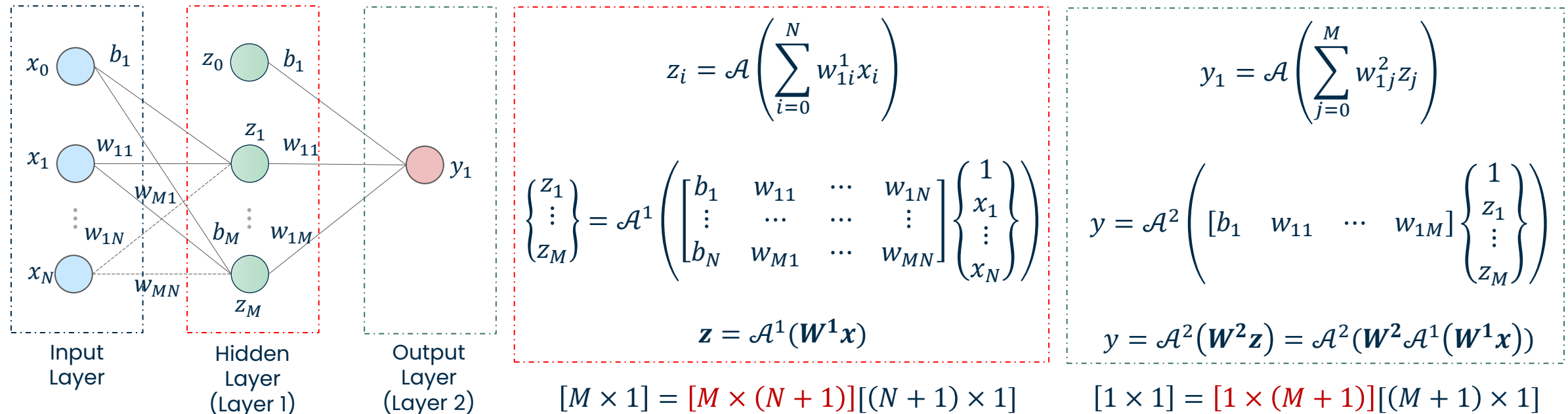
$$[M \times 1] = [M \times (N + 1)][(N + 1) \times 1]$$

$M(N+1)$ parameters

MLP: Network and neurons

The Multi Layer Perceptron (i.e., a Neural Network) is a stacked architecture of interconnected artificial neurons.

Multiple layers single output



Single Hidden layer: «shallow»
Multiple hidden layers: «Deep»

M(N+1) parameters

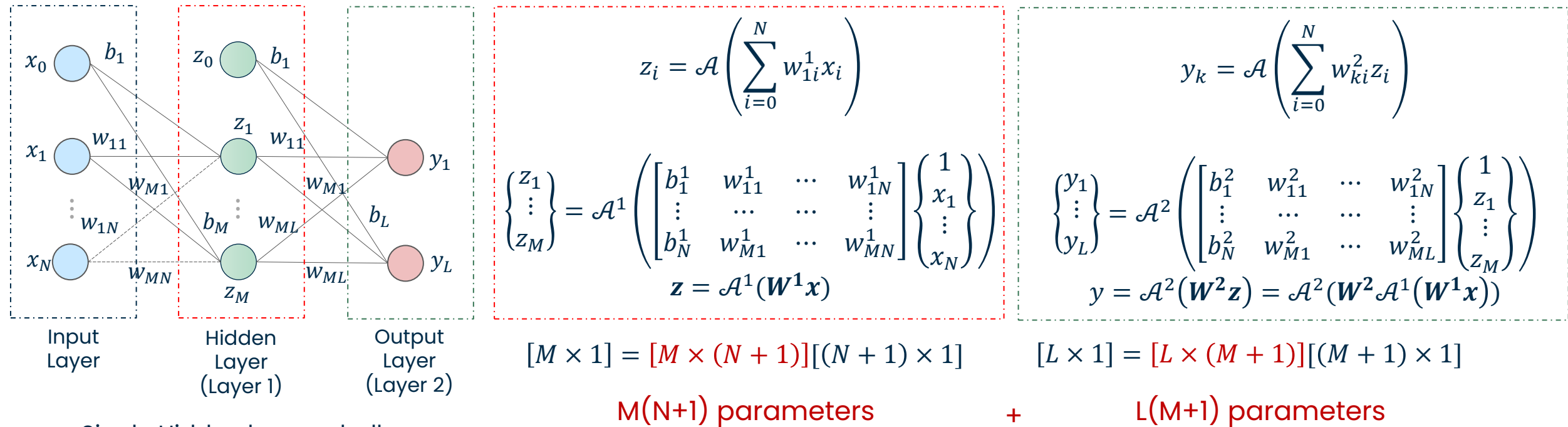
+

M+1 parameters

MLP: Network and neurons

The Multi Layer Perceptron (i.e., a Neural Network) is a stacked architecture of interconnected artificial neurons.

Multiple layers multiple output



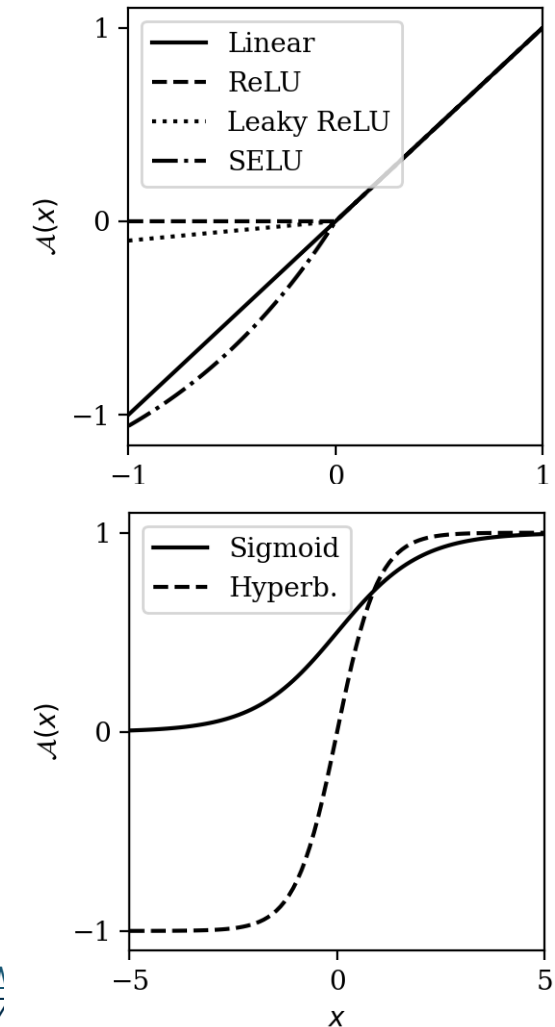
Single Hidden layer: «shallow»
Multiple hidden layers: «Deep»

MLP: Activation functions

Activation functions mimic the “firing” of neuron, let them de-activate (i.e., reduce the influence of their results on the neuron output), using mathematical functions. The activation function must be:

- Non-constant²
- Bounded² (at least its combination)
- Differentiable
- Monotonic (rare exceptions, e.g., “swish”)

Name	Expression	Params	Models
Linear	$\mathcal{A}(x) = x$	-	Regr.
Rectified Linear Unit (ReLU) ^[3]	$\mathcal{A}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	-	Regr.
Leaky ReLU	$\mathcal{A}(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	α	Regr.
Scaled Exponential Linear Unit (SELU) ^[4]	$\mathcal{A} = \lambda \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	α, λ	Regr.
Hyperbolic Tangent	$\mathcal{A}(x) = \tanh(x)$	-	Class.
Sigmoid	$\mathcal{A}(x) = \frac{x}{1 + e^{-x}}$	-	Class.



[2] Kurt Hornik, *Approximation capabilities of multilayer feedforward networks*, *Neural Networks*, Volume 4, Issue 2, 1991.

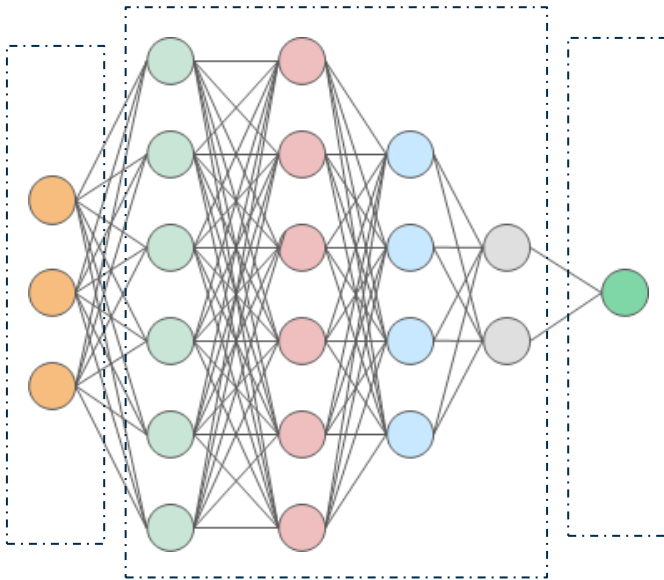
[3] K. Fukushima, *Cognitron: A self-organizing multilayered neural network*, *Biol Cybern.* 20 (1975) 121–136. <https://doi.org/10.1007/BF00342633/h>

[4] G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter, *Self-Normalizing Neural Networks*, *Adv Neural Inf Process Syst.* 2017-December (2017) 97

Network and neurons

The Multi Layer Perceptron (i.e., a Neural Network) is a stacked architecture of interconnected artificial neurons.

Multiple hidden layers



Parameters

tunable parameters of the model (i.e. bias and weights of the network)

Hyper-parameters

Non-tunable parameters defining the model architecture and training strategy (number of layers, activation functions, ..)

Main takeouts:

- Neural networks are tensor product combined with operators (\mathcal{A}) -> explicit computation, extremely fast
- The number of parameters rapidly increase with increasing number of neurons or number of layers

Loss function

The MLP can be treated as a parametric function with parameters $\theta = \{w_{ij}^k, b_j\}$:

$$y = MLP(x; \theta) : \mathbb{R}^N \rightarrow \mathbb{R}^M$$

The parameters (i.e., weights and bias) are defined by minimizing a loss function:

$$L(\bar{x}, \bar{y}; \theta) = err(MLP(\bar{x}), \bar{y})$$

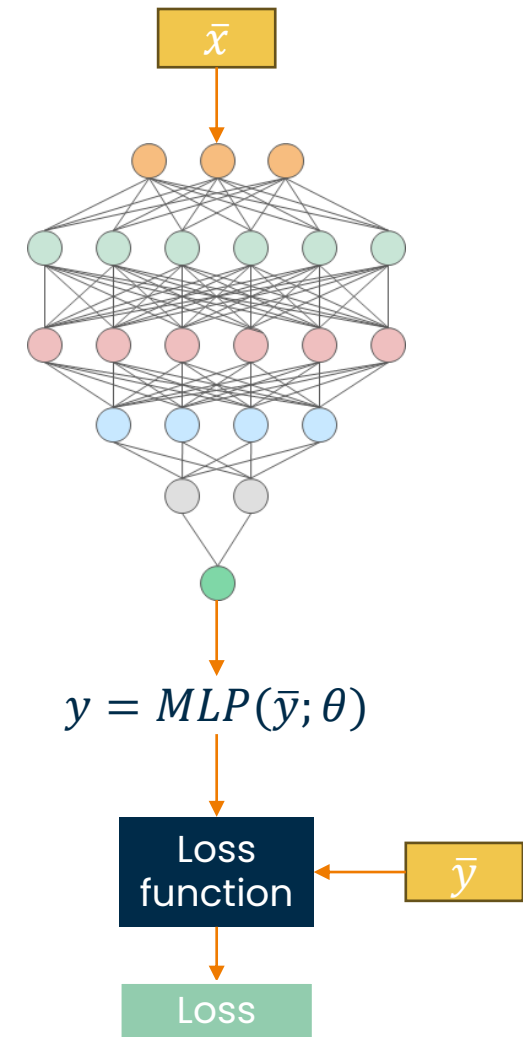
where (\bar{x}, \bar{y}) is the training dataset containing a set of observation.

The training process is defined as an optimization problem:

$$\theta_{trained} = \arg \min_{\theta} L(\bar{x}, \bar{y}; \theta)$$

How do we choose the loss function?

Problem	Output	Final \mathcal{A}	Loss
Regression	Numerical	Linear	MSE
Classification	2 Class	Sigmoid	Binary Cross Entropy
Classification	Multiple class	SoftMax	Cross Entropy
Regression	Numerical (physical qoi)	Linear	MSE + physics constr.
Probabilistic regr.	Probabilistic	Linear	Maximum Likelihood



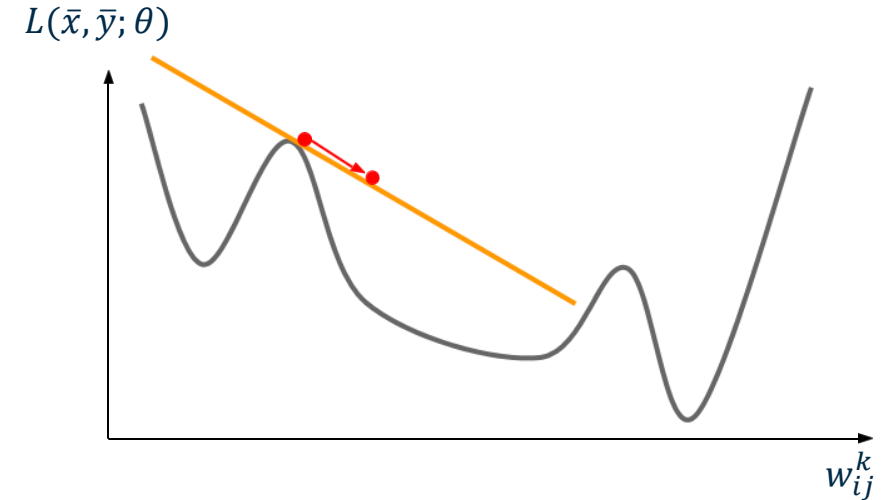
Supervised Training

The loss is used to optimize the weights using the gradient:

For each weight w_{ij}^k :

1. Compute $\partial L / \partial w_{ij}^k$ to find a linear approximation
2. Update w_{ij}^k with a step in the decreasing direction

The derivative of the loss w.r.t. each weights is a gradient vector:



$$\nabla L(\bar{x}, \bar{y}, \theta) = \begin{Bmatrix} \frac{\partial L}{\partial \theta_i} \\ \frac{\partial L}{\partial y_j} \\ \frac{\partial L}{\partial \bar{x}_k} \end{Bmatrix} \in \mathbb{R}^{N+M+H}; i \in [1, N], j \in [1, M], k \in [1, H]$$

Need for optimizing θ

Derivative w.r.t. inputs (PINN)

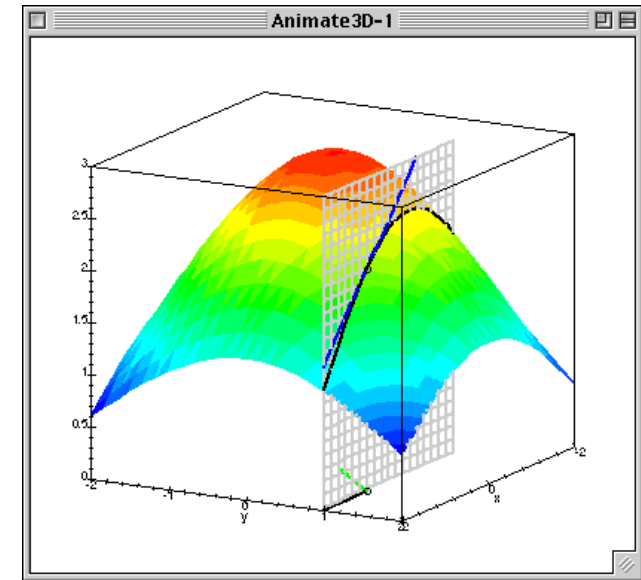
N: n° inputs
M: n° outputs
H: n° hyperparams

Into N dimensions, the multidimensional gradient vector indicates the descending direction.

$$\theta^{i+1} = \theta^i + \mu^i f^i$$

$$f^i = -\nabla L(\theta^i)$$

μ is the learning rate defining the size of the step at each i^{th} iteration.



Back-propagation

How do we define the term $\partial L / \partial w_{ij}^k$?

The MLP is equivalent to:

$$y = h(g(f(x))), h, g, f, \text{ with } h, g, f \text{ the operator of each layer}$$

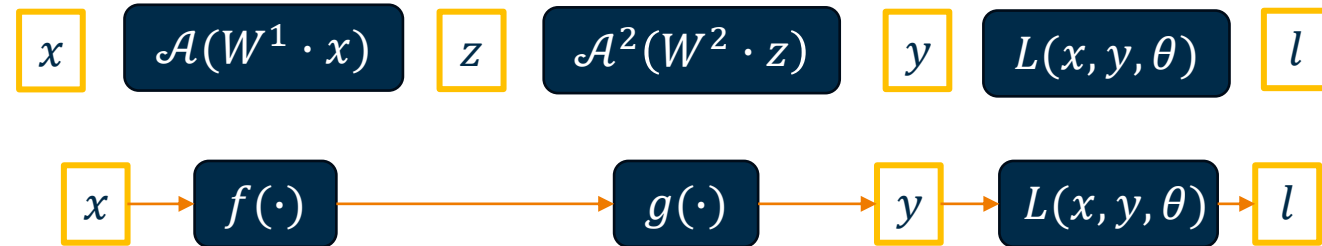
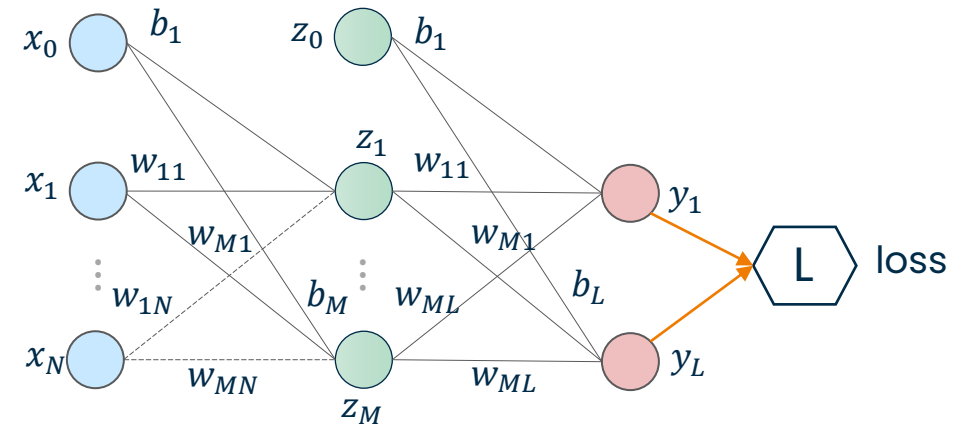
The loss is equivalent to:

$$l = L(\bar{x}, \bar{y}; \theta) = L(h(g(f(x))))$$

The derivative of the loss can be computed with the chain rule:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial h} \frac{\partial h}{\partial g} \frac{\partial g}{\partial f} \frac{\partial f}{\partial x}$$

Computing the derivative from the last layer to the first one.

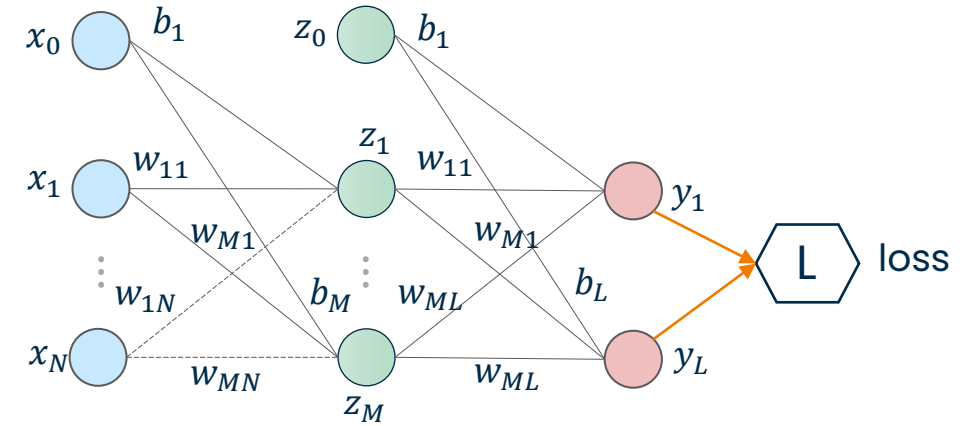
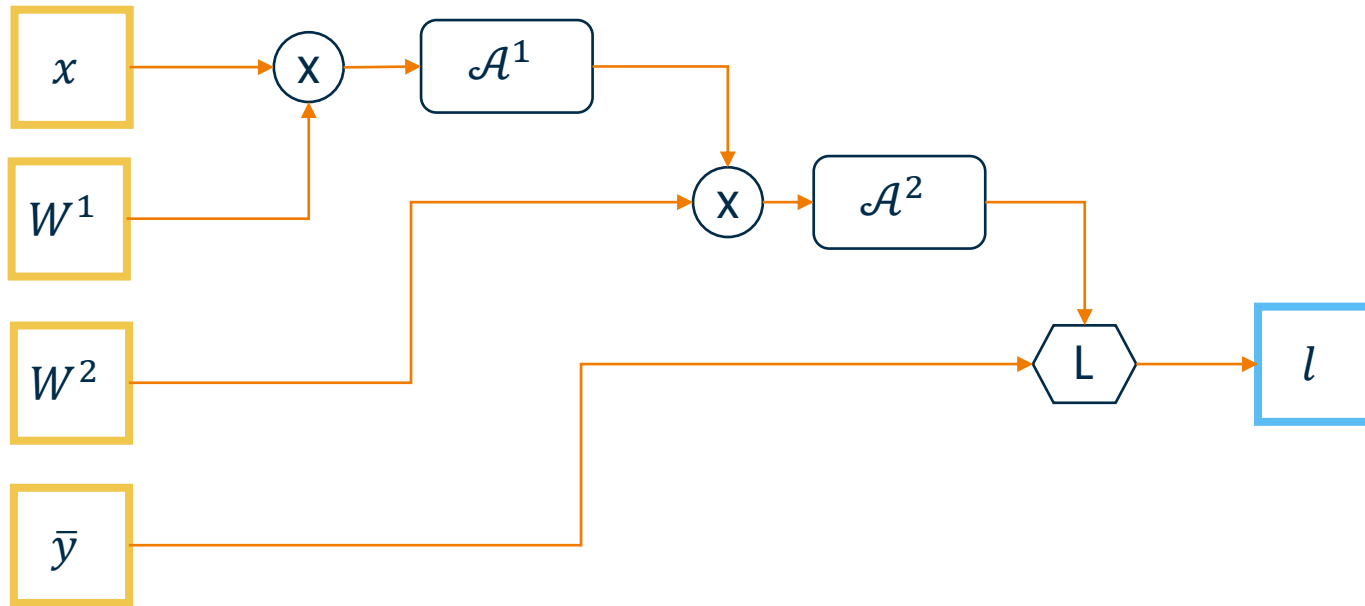


$$y = g(f(x)) \rightarrow l = L(g(f(x)))$$

This operation is done on the **computational graph** and is called **back-propagation** (backwards in the network).

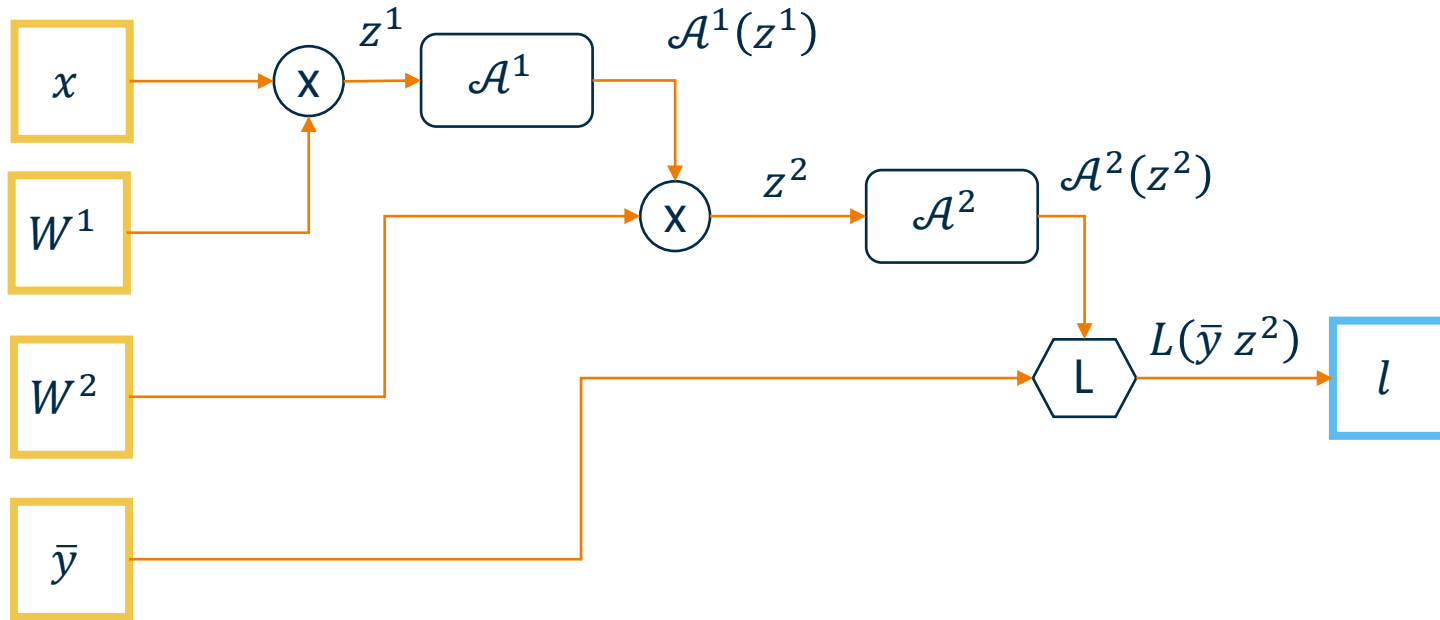
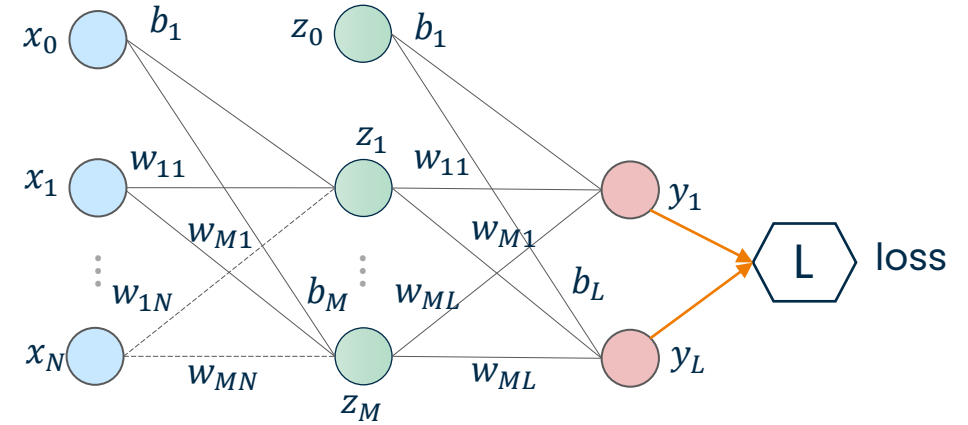
Back-propagation

1. Define the computational graph of the network



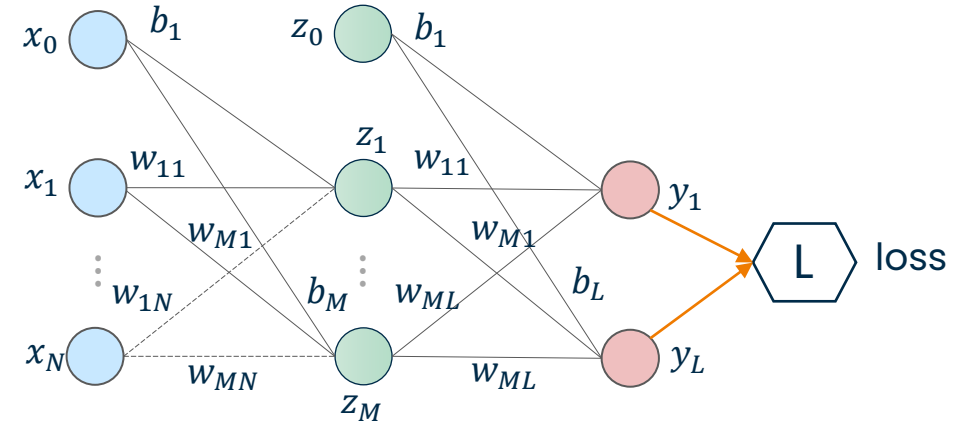
Back-propagation

1. Define the computational graph of the network
2. Calculate the forward pass
 1. Variables



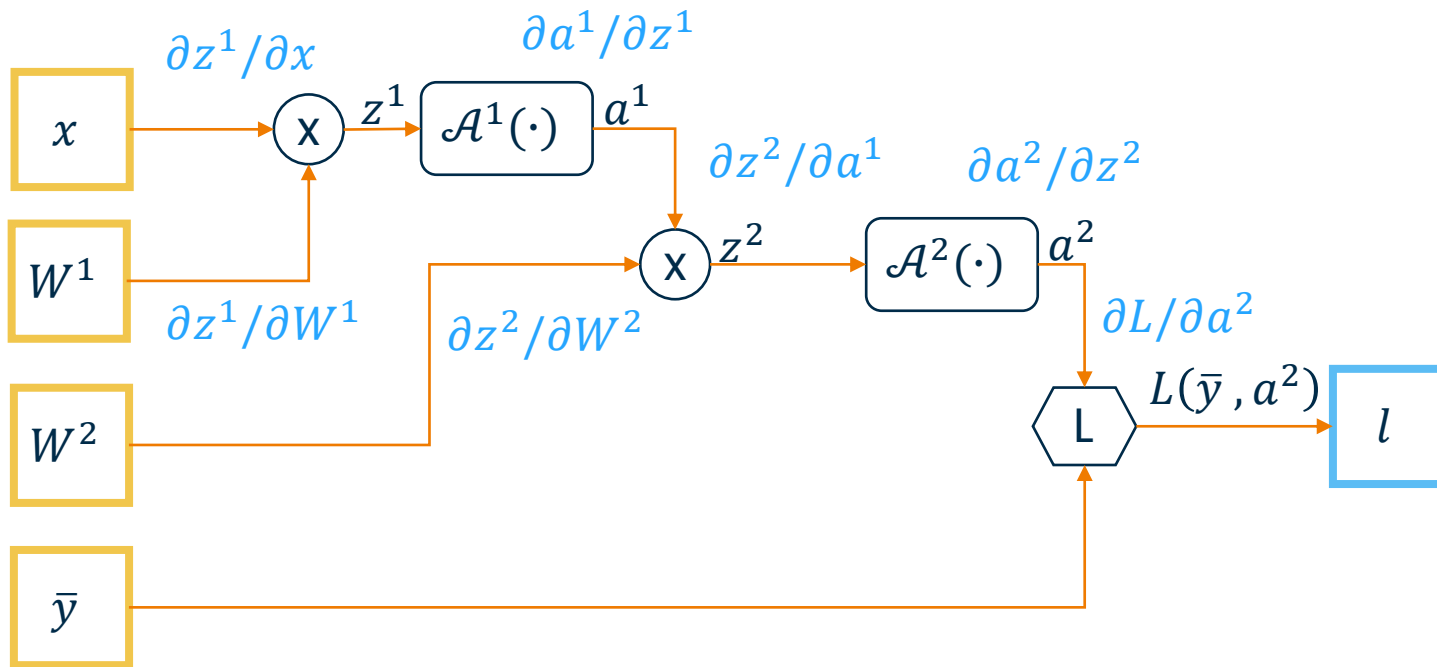
Back-propagation

1. Define the computational graph of the network
2. Calculate the forward pass
 - a) Variables
 - b) Local gradients



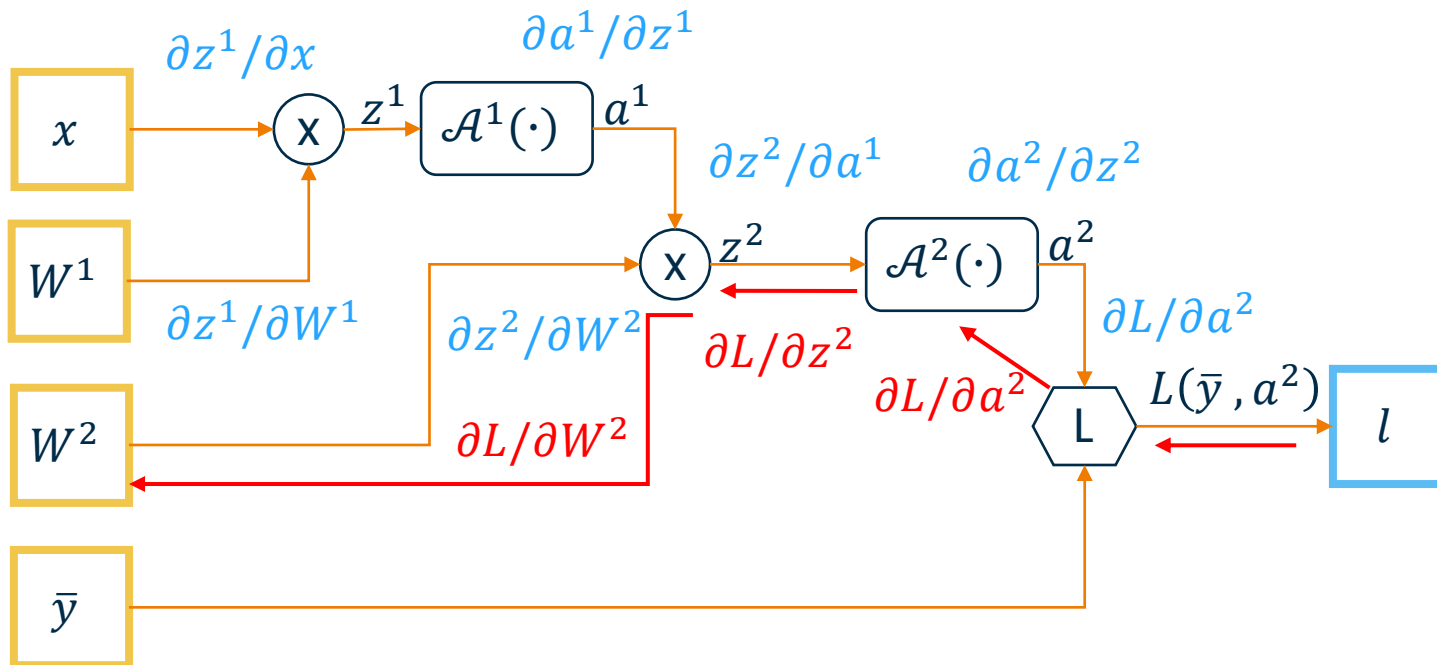
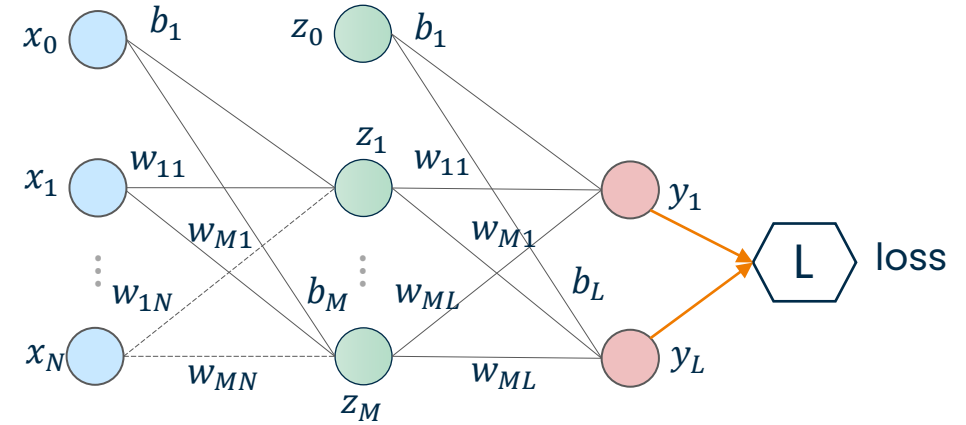
$$a^1 = \mathcal{A}^1(z^1)$$

$$a^2 = \mathcal{A}^2(z^2)$$



Back-propagation

1. Define the computational graph of the network
2. Calculate the forward pass
 - a) Variables
 - b) Local gradients
3. Calculate the backward pass
 - a) Back-propagated error



$$a^1 = \mathcal{A}^1(z^1)$$

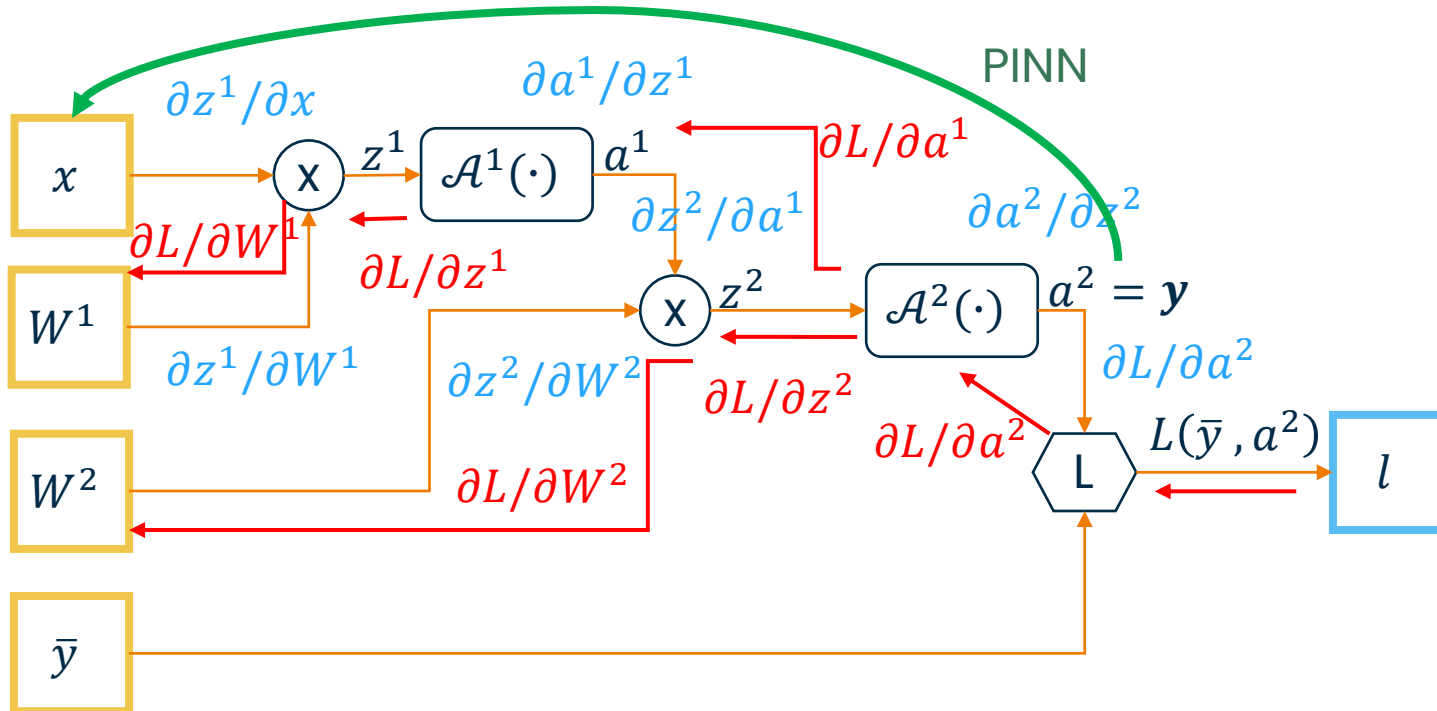
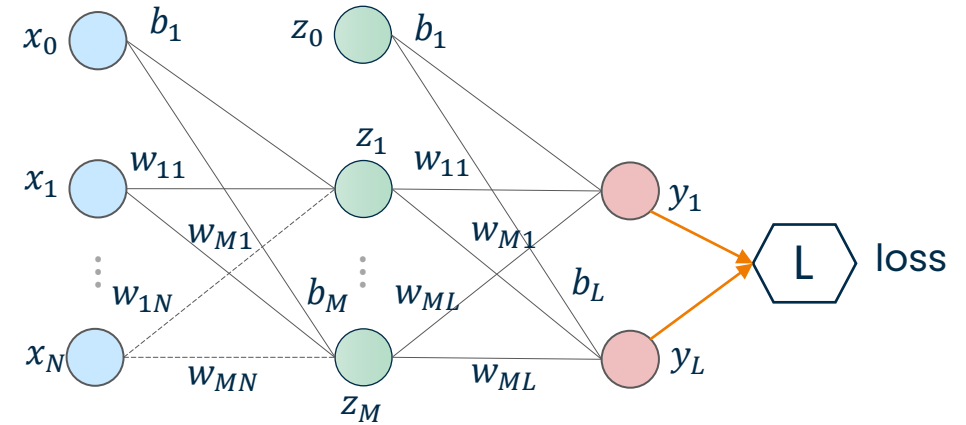
$$a^2 = \mathcal{A}^2(z^2)$$

$$\frac{\partial L}{\partial a^2} \frac{\partial a^2}{\partial z^2} = \frac{\partial L}{\partial z^2}$$

$$\frac{\partial L}{\partial z^2} \frac{\partial z^2}{\partial W^2} = \frac{\partial L}{\partial W^2}$$

Back-propagation

1. Define the computational graph of the network
2. Calculate the forward pass
 - a) Variables
 - b) Local gradients
3. Calculate the backward pass
 - a) Back-propagated error



$$a^1 = \mathcal{A}^1(z^1)$$

$$a^2 = \mathcal{A}^2(z^2)$$

$$\frac{\partial L}{\partial a^2} \frac{\partial a^2}{\partial z^2} = \frac{\partial L}{\partial z^2}$$

$$\frac{\partial L}{\partial z^2} \frac{\partial z^2}{\partial a^1} = \frac{\partial L}{\partial a^1} \rightarrow \frac{\partial L}{\partial a^1} \frac{\partial a^1}{\partial z^1} = \frac{\partial L}{\partial z^1}$$

$$\frac{\partial L}{\partial z^1} \frac{\partial z^1}{\partial W^1} = \frac{\partial L}{\partial W^1}$$

Optimizer

Once the gradient is defined, the weights are iteratively updated with gradient-based optimizers. The most common are:

1. Batch gradient descent

$$w^{(i+1)} = w^{(i)} - \alpha \nabla_w L(\bar{x}, \bar{y}, w^{(i)}) \text{ for } i \in (1, n_{epochs})$$

α is the learning rate, it can be fixed or varying with the epochs.

2. Mini-batch (or Stochastic) gradient descent

$$w^{(i+1)} = w^{(i)} - \alpha \nabla_w L(\bar{x}_k, \bar{y}_k, w^{(i)}) \text{ for } i \in (1, n_{epochs}), k \in (1, n_b)$$

Data are shuffled and divided into n_b mini batches. If the batch size is one, it is called *Stochastic Gradient Descent* (SGD).

The **number of epochs** and the **batch size** are hyper-parameters of the model.

The number of epoch can be automatically selected with a **early stopping** algorithm that interrupts the optimization at convergence.

- ✓ Always converge toward - local or global-minimum
- ✓ Fixed α can be used
- × Slow
- × Redundant

- ✓ Faster iterations
- ✓ Non redundant calculations
- ✓ Add noise due to random sampling helping generalization
- × May not converge and stuck in local minima
- × Loss oscillation may require variable α (e.g., exponential decay)

Optimizer: momentum and adaptivity

The **momentum** is defined as a fraction of the previous step:

$$m = \mu(w^{(i)} - w^{(i-1)})$$

m is used to compute the velocity:

$$v^{(i+1)} = \rho v^{(i)} - \alpha \nabla L(w^{(i)})$$

$$w^{(i+1)} = w^{(i)} + v^{(i+1)}$$

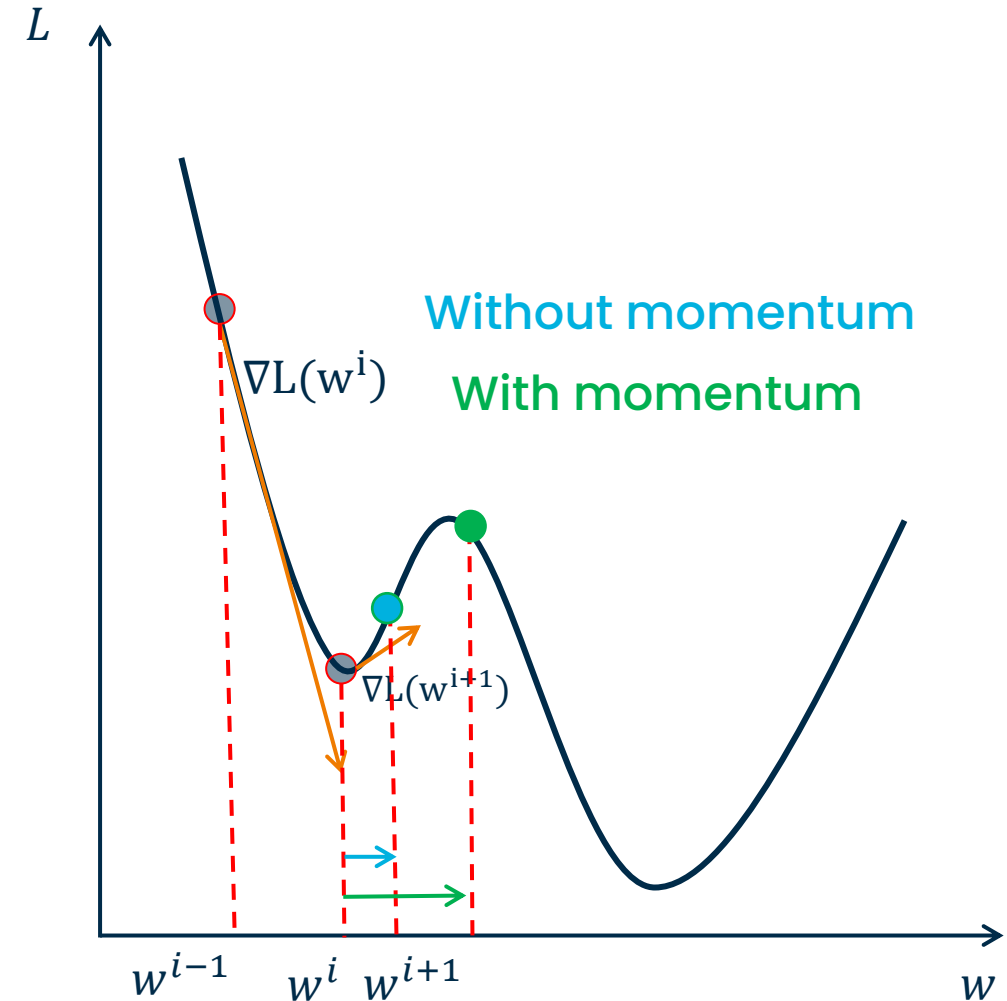
With **adaptivity**, the step is scaled w.r.t. the (scaled) history of the gradient, so:

$$w^{(i+1)} = w^{(i)} - \frac{\alpha \nabla L(w^{(i)})}{\sum_j^i \sqrt{(\nabla L(w^j))^2}}$$

AdaGrad

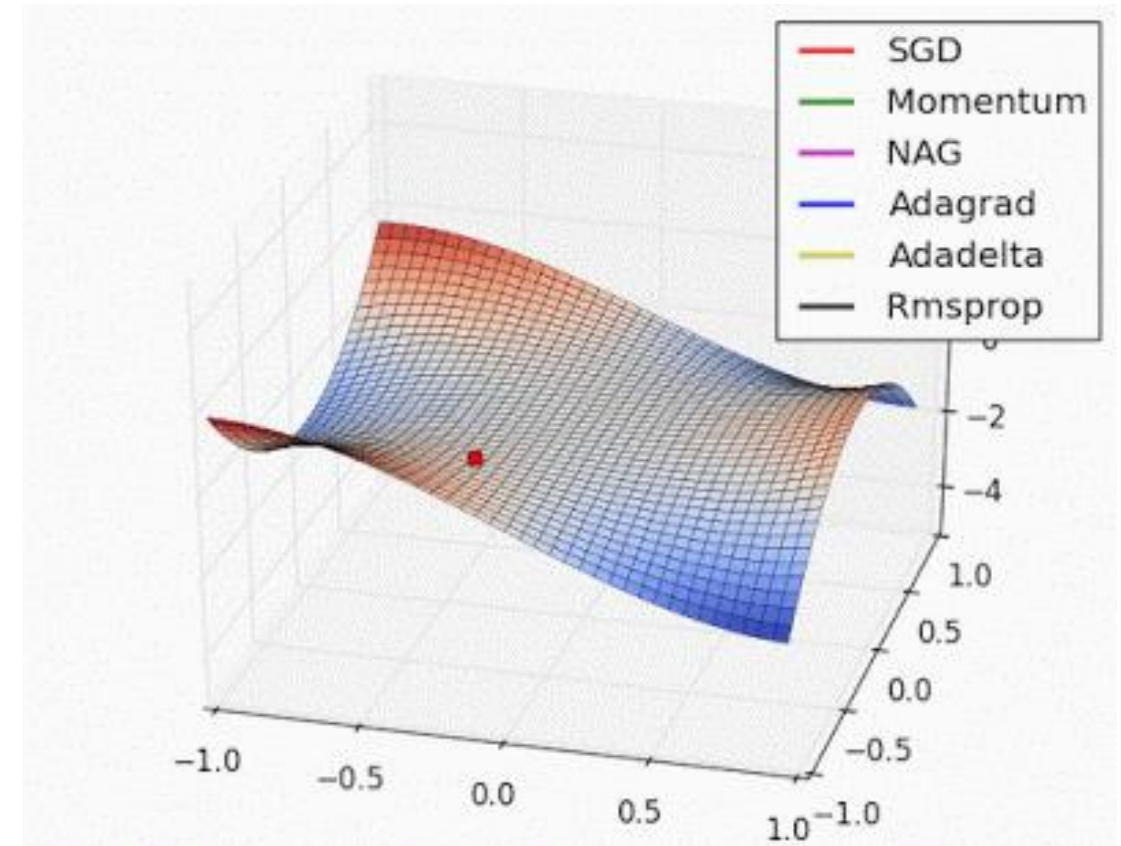
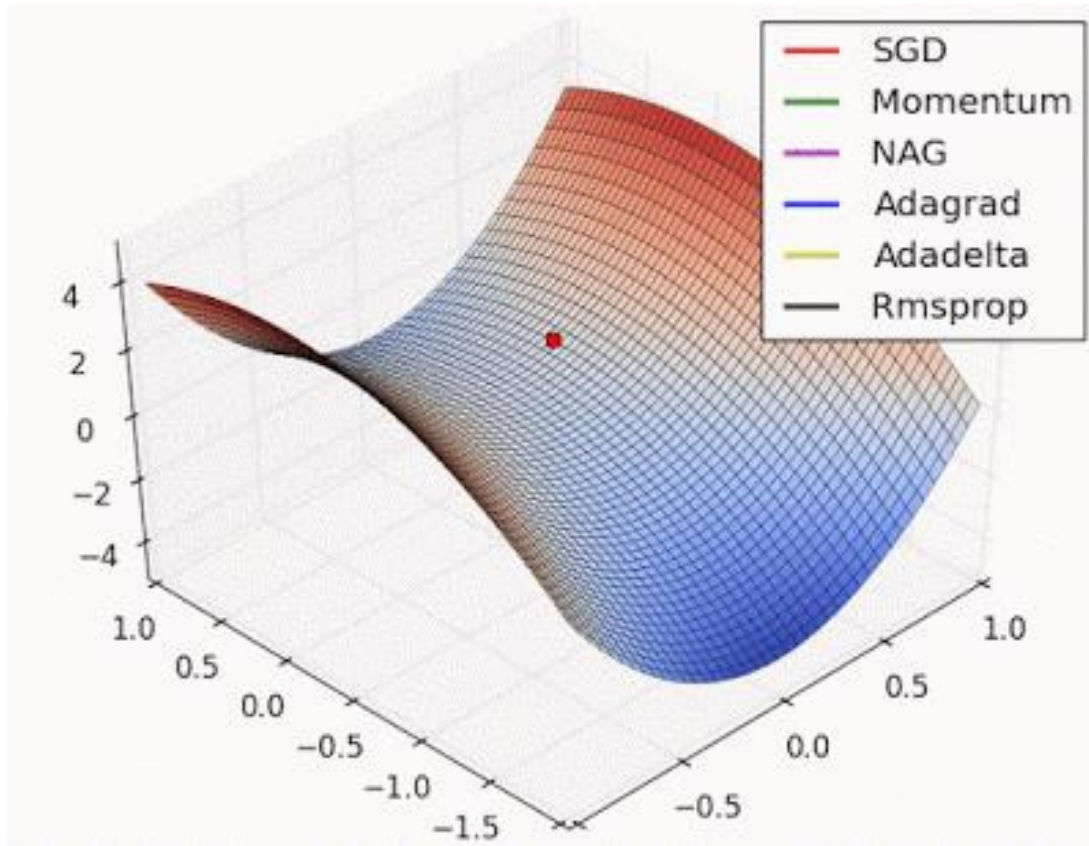
$$w^{(i+1)} = w^{(i)} - \frac{\alpha \nabla L(w^{(i)})}{\sqrt{\gamma (\nabla L)^2 + \sum_j^{i-1} (1 - \gamma)^j \nabla L(w^j)}}$$

RMSProp

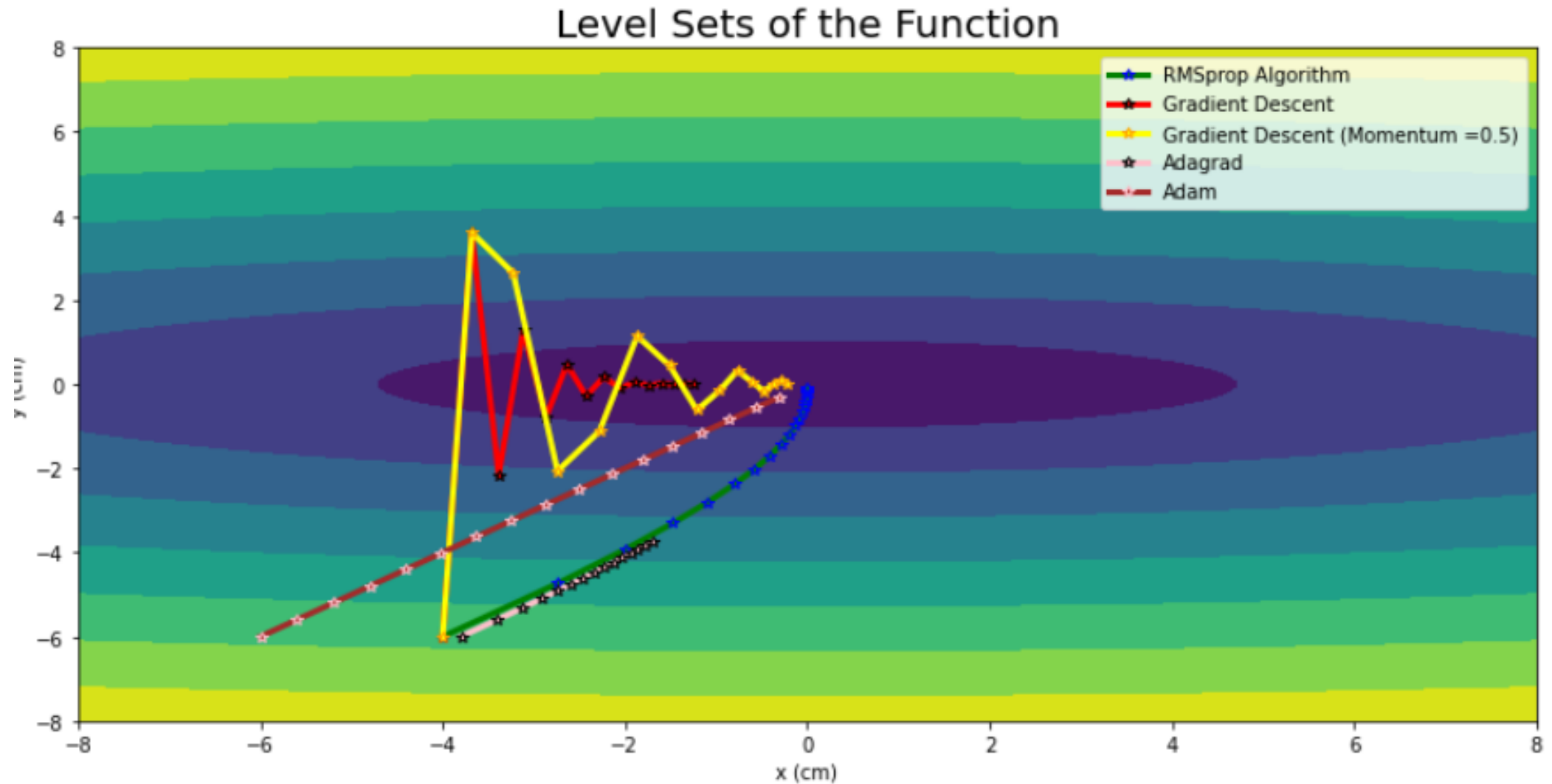


RMSProp + Momentum = Adam

Optimizer: momentum and adaptivity



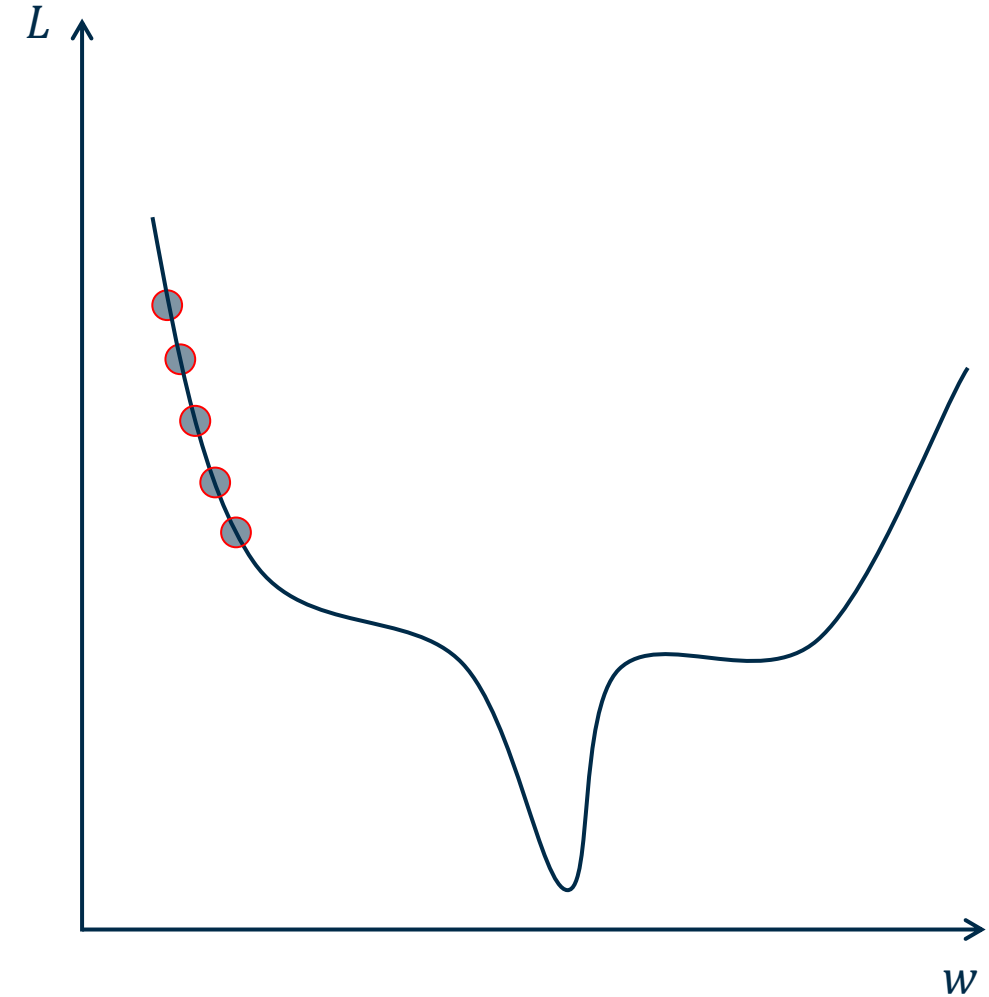
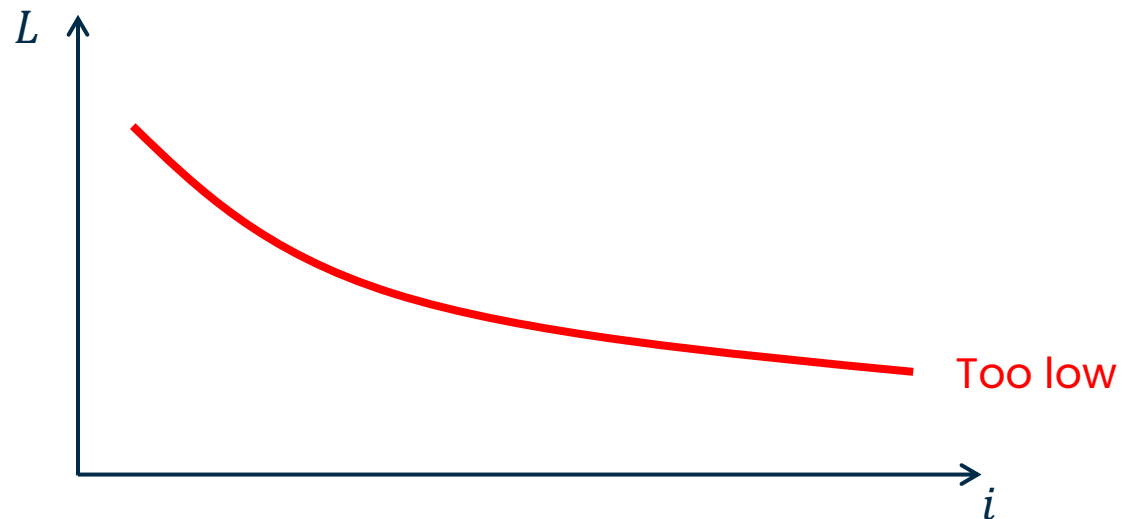
Optimizer: momentum and adaptivity



Optimizer: learning rate

Every optimizer presents a learning rate, which affects the training procedure.

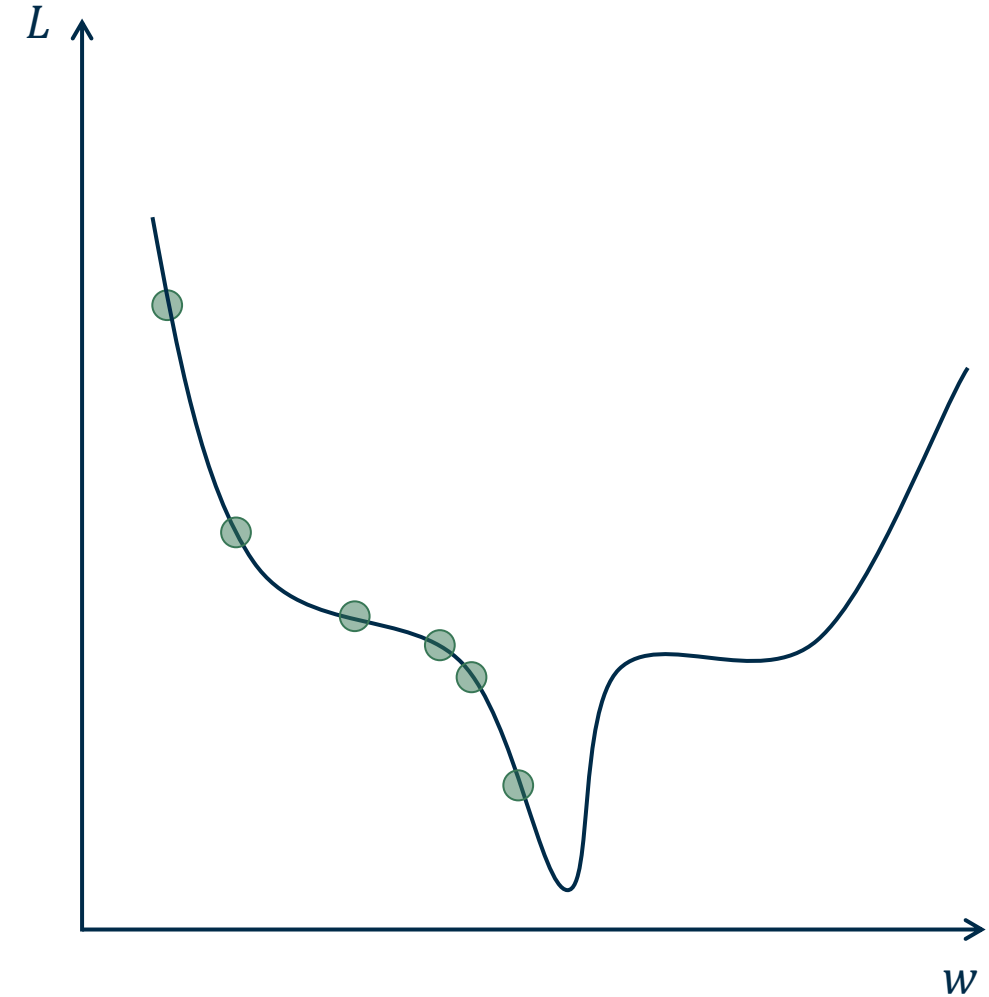
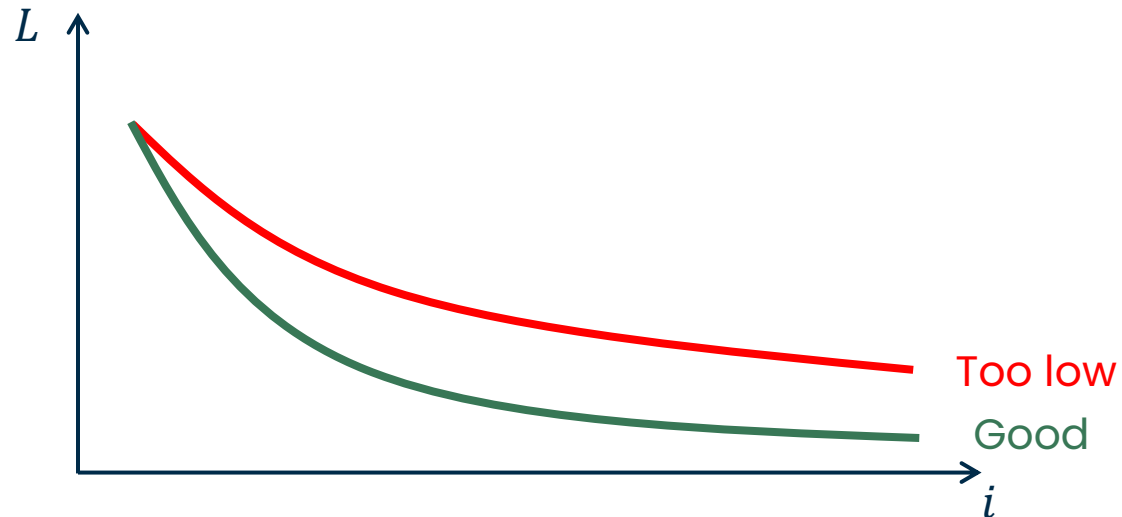
- Small $lr \rightarrow$ slow convergence



Optimizer: learning rate

Every optimizer presents a learning rate, which affects the training procedure.

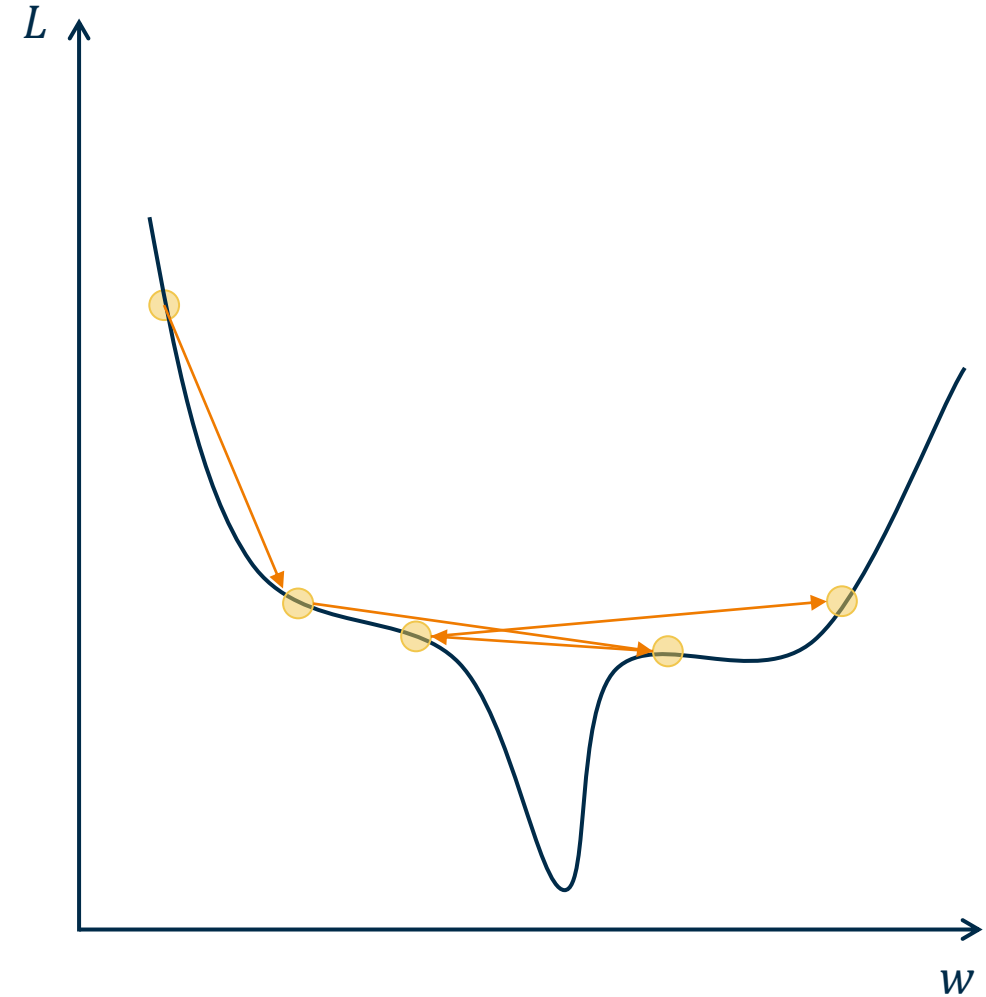
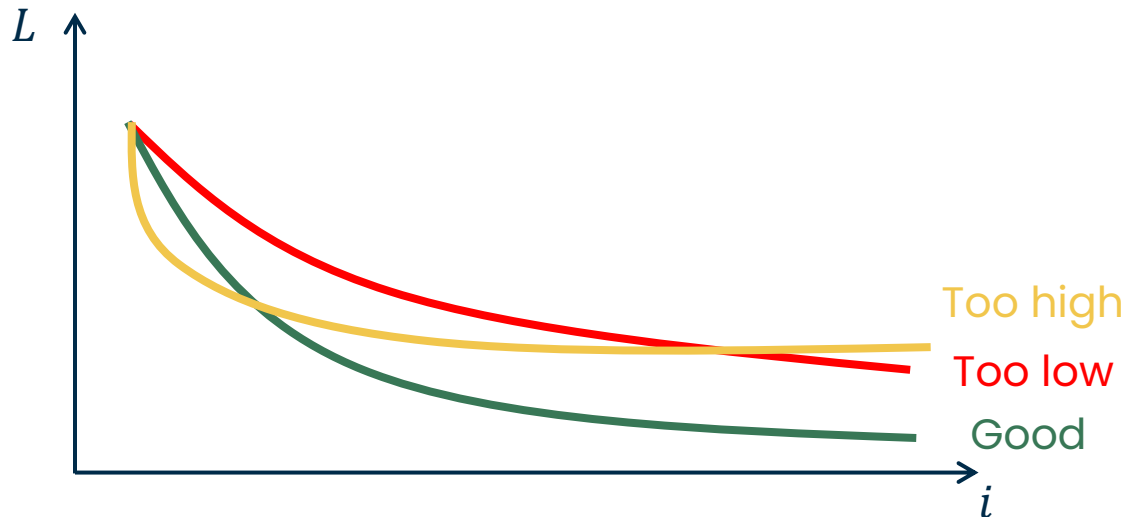
- Small $lr \rightarrow$ slow convergence
- Good learning rate \rightarrow optimal



Optimizer: learning rate

Every optimizer presents a learning rate, which affects the training procedure.

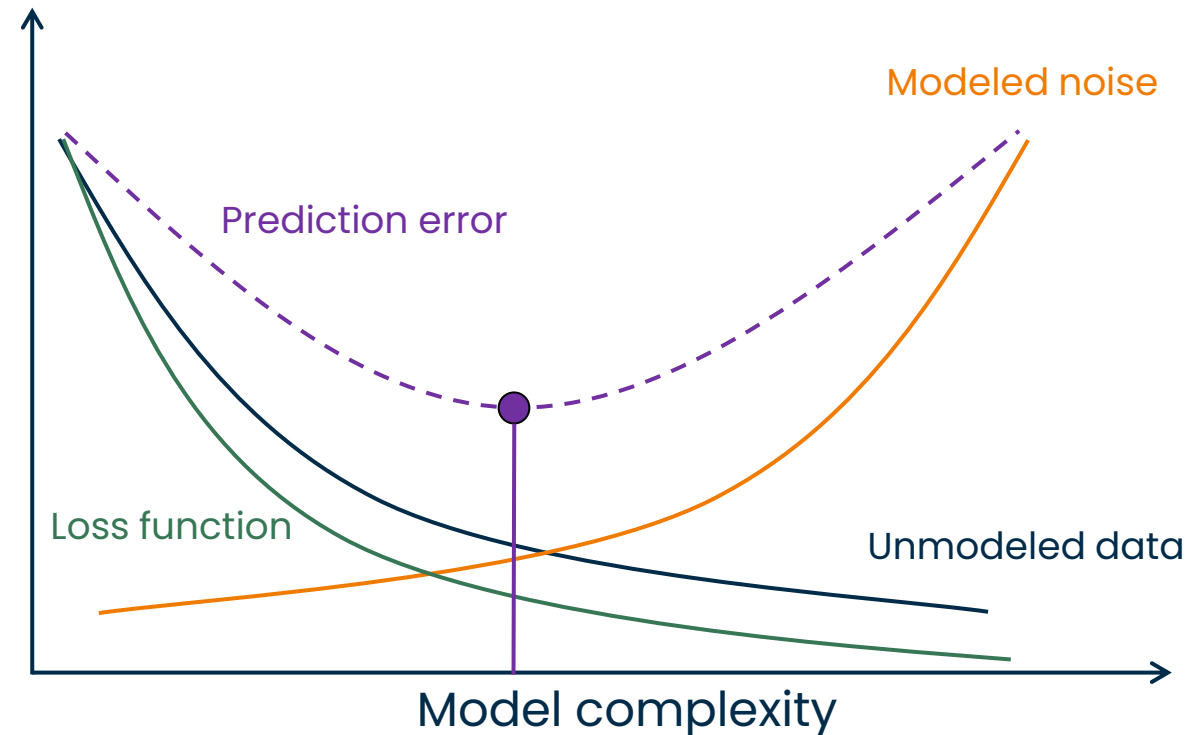
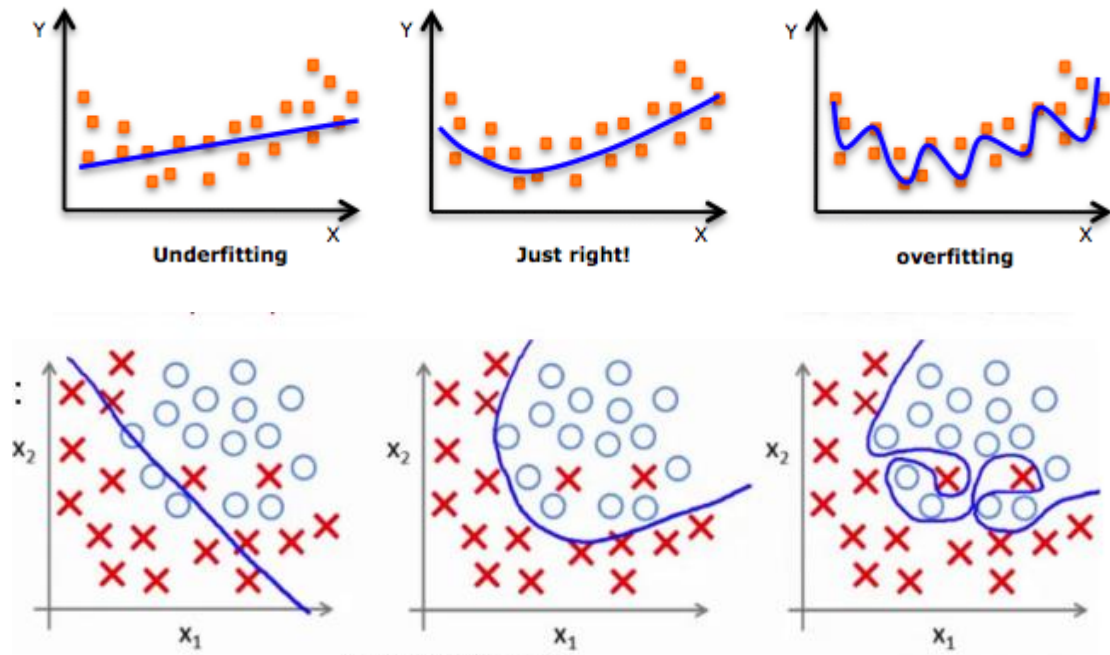
- Small $lr \rightarrow$ slow convergence
- Good learning rate \rightarrow optimal
- High $lr \rightarrow$ not converge to minima



Generalization: under-fitting and over-fitting

When bounded and non-constant activation functions are used, a MLP with a sufficient number of hidden units is a universal approximator²

- Not enough hidden units \rightarrow can not approximate \rightarrow **underfitting**
- Enough units but noisy data \rightarrow approximates the noise \rightarrow **overfitting**

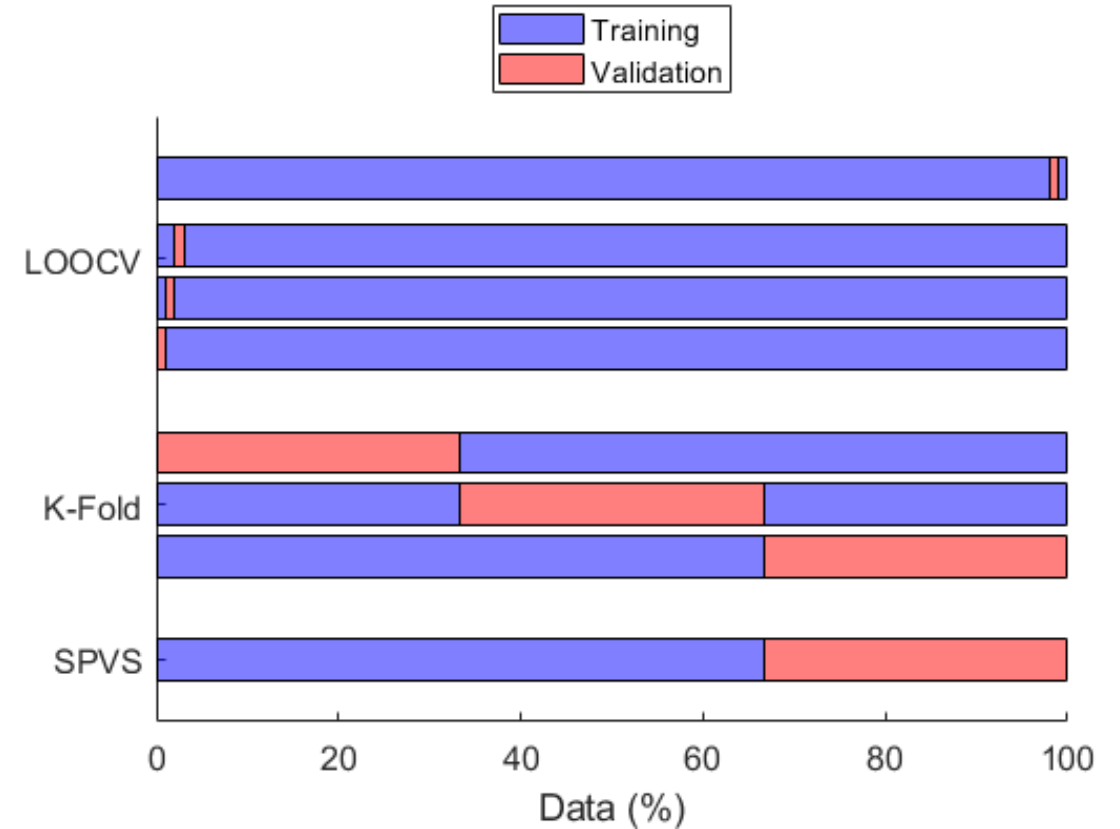


Generalization: under-fitting and over-fitting

How can I check the prediction error? Data splitting!

Splitting techniques:

- Single-split: divide dataset into validation and split
- Leave one out: multiple training with one sample out each training
- K-Fold: split dataset in k-fold and repeat training k times with kth subset as validation



Generalization: under-fitting and over-fitting

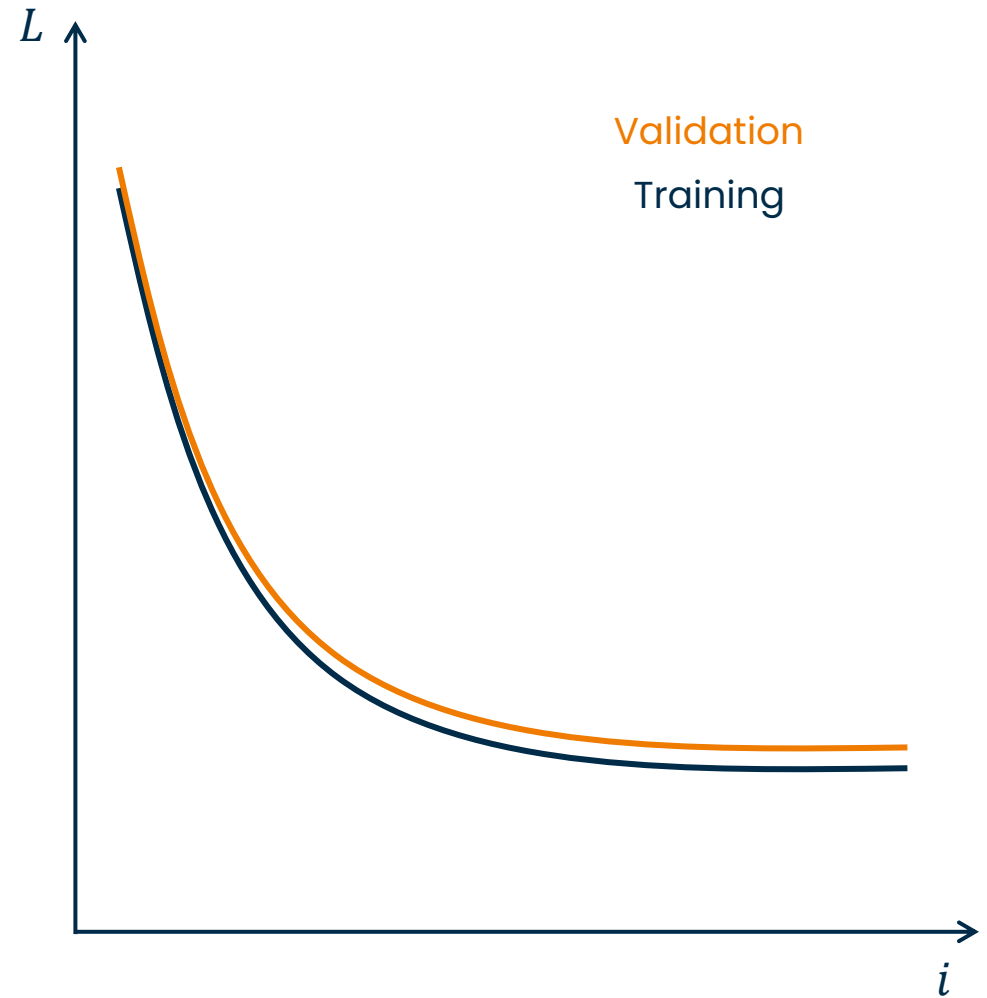
How can I check the prediction error? Data splitting!

Splitting techniques:

- Single-split: divide dataset into validation and split
- Leave one out: multiple training with one sample out each training
- K-Fold: split dataset in k-fold and repeat training k times with kth subset as validation

Train on training and predict on validation at each epochs:

Underfitting: training loss tracks validation loss



Generalization: under-fitting and over-fitting

How can I check the prediction error? Data splitting!

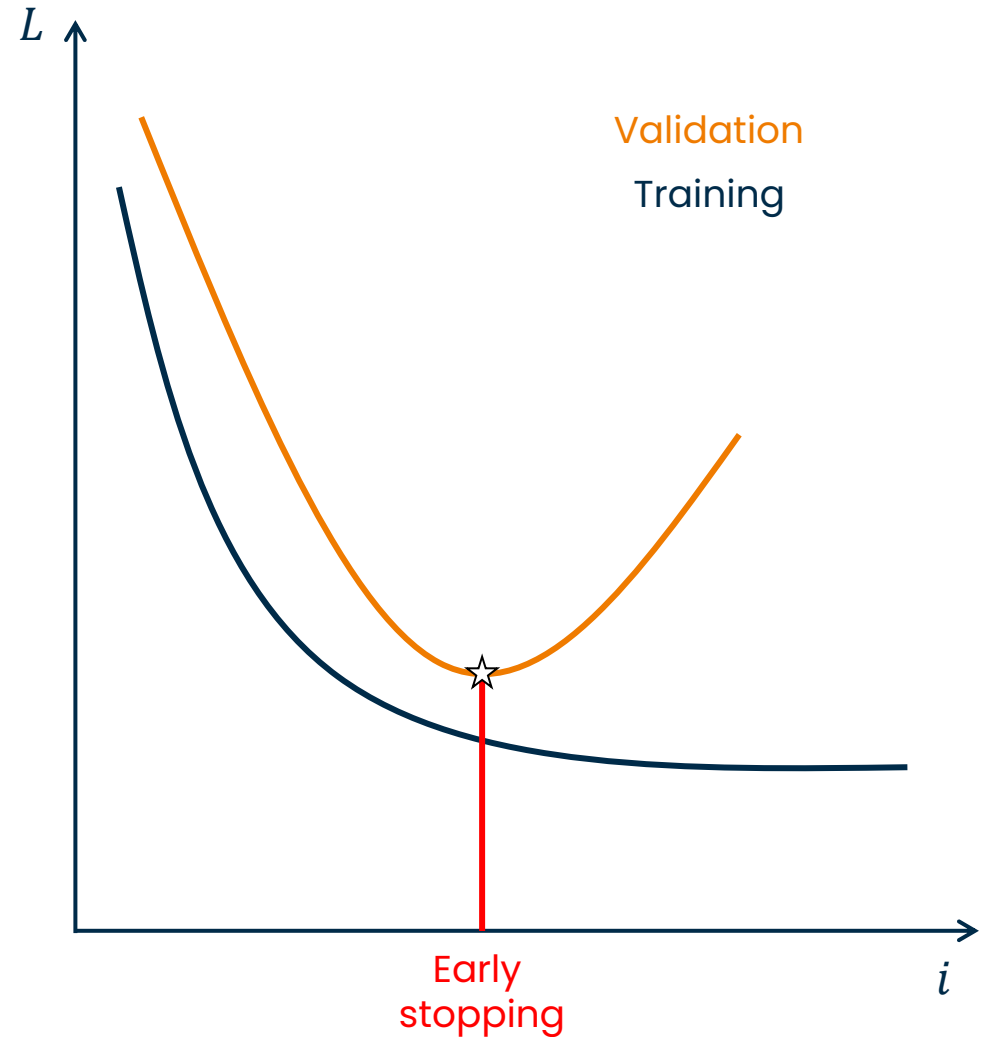
Splitting techniques:

- Single-split: divide dataset into validation and split
- Leave one out: multiple training with one sample out each training
- K-Fold: split dataset in k-fold and repeat training k times with kth subset as validation

Train on training and predict on validation at each epochs:

Underfitting: training loss tracks validation loss

Overfitting: Validation loss increase while training keep decreasing



Generalization: under-fitting and over-fitting

There are algorithmic solutions to overfitting:

- **Regularization**

Add a weight regularization term to the loss function (L1= 'lasso', L2 = 'ridge')

$$L = MSE + \lambda |w^{(i)}|^k, k = 1 \text{ or } 2$$

- **Dropout**⁶

Drop MLP neuron in hidden layers at each epochs.

Define a dropout probability p_i at each i^{th} layer

At each iteration

Extract $p_k^i \in [0,1]$ for each k^{th} neurons in i^{th} layer

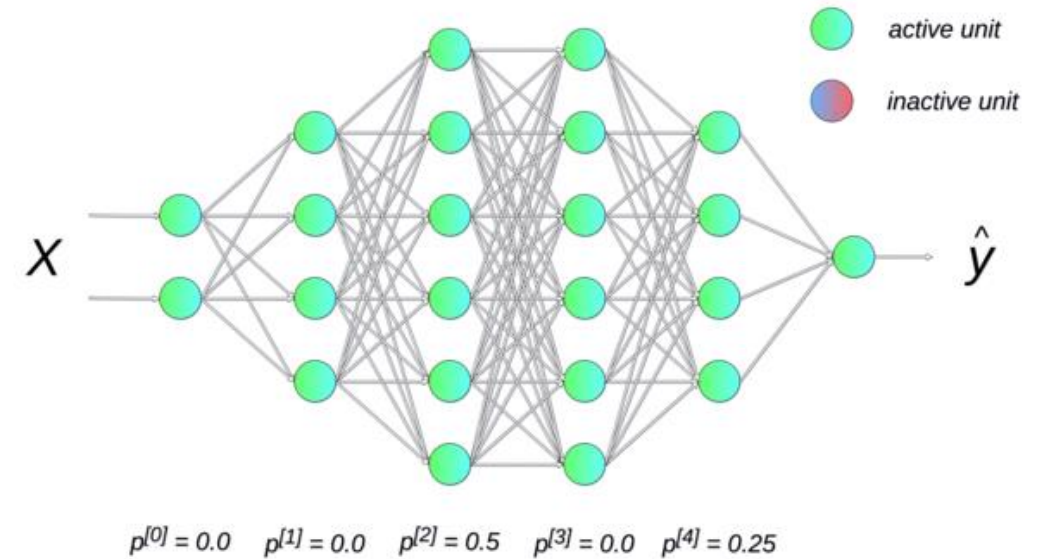
Remove neurons with $p_k^i < p^i$

Remove connection $w_{drop} = w_{jk}^i \forall j$

Compute the loss $L(w \cap w_{drop})$

Backpropagate

Update weights $w \cap w_{drop}$



[6] Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Hands-on tutorial

Google Colab:

Here you have a simple code for training a regression MLP:

[https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/linear regression with synthetic data.ipynb](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/linear_regression_with_synthetic_data.ipynb)

Suggested material:

Here you can find an online course on how to use Python:

<https://www.youtube.com/watch?v=rfscVS0vtbw>

Here you can find a crash course on ML provided by Google that will teach you the basic principles together with the basic steps with the Tensorflow library in Keras:

<https://developers.google.com/machine-learning/crash-course/ml-intro>

Since organizing and preprocessing the data is a fundamental aspect of the ML projects, take a look at this short course about the most common data processing techniques:

<https://developers.google.com/machine-learning/data-prep>

TUTORIAL



**Politecnico
di Torino**

Course outline

Part I – Tuesday May 14th

- Introduction to Machine Learning
 - Definitions and classifications
 - Main architectures
- Multi-layer perceptron
 - Network and neurons
 - Training algorithm
 - The loss-function

Hands-on tutorial: build a NN with Google Tensorflow

Part II – Friday May 17th

- ML applied to structural integrity
 - General MDS framework
 - State of the art
- ML model for scientific applications
 - Overview of PINN
 - Overview of DeepONet
 - Overview of BNN

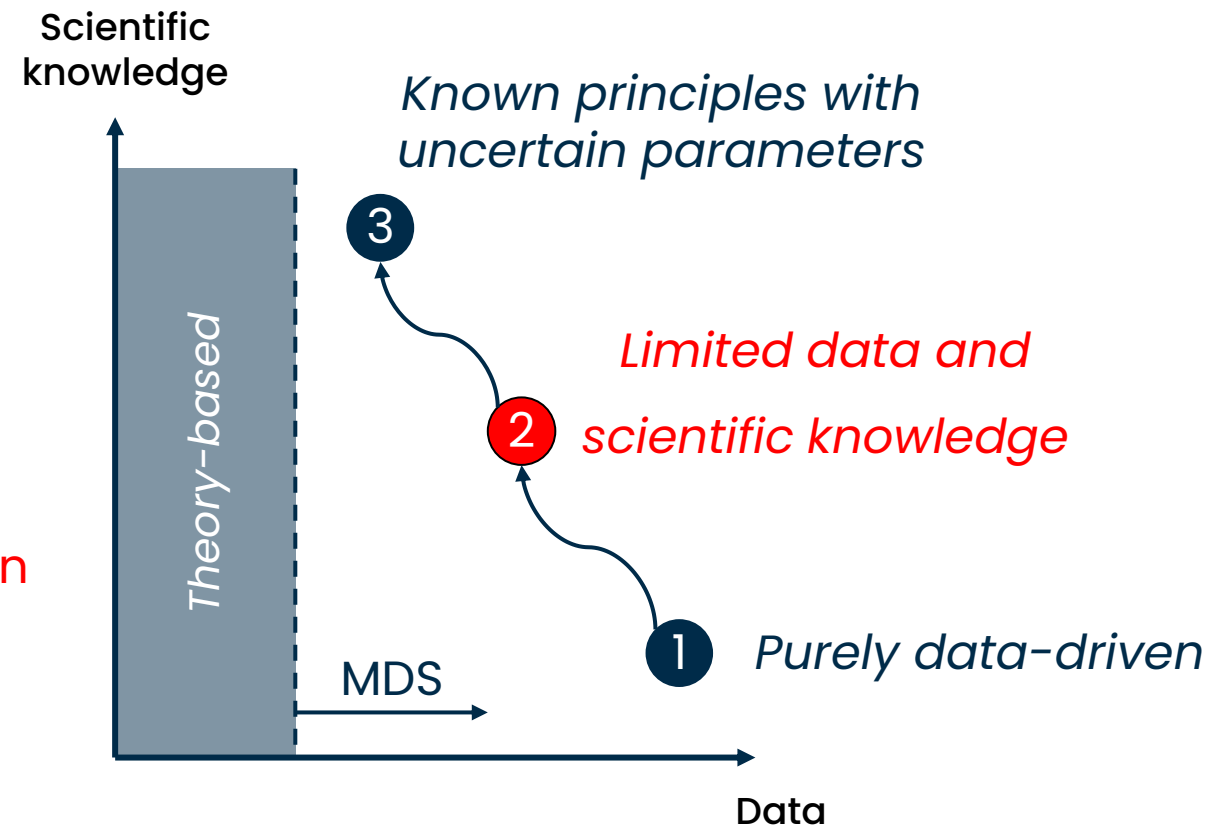
Hands-on tutorial: build a BNN with Google Tensorflow Probabilistic

Mechanistic Data Science

«Mechanistic data science combines mechanistic calibrated principles and collected data to accelerate the knowledge extraction and improve predictive capacity»

The development of MDS models follows six steps:

1. Multimodal data generation/collection
2. Extraction of mechanistic features
3. Knowledge-driven dimension reduction
4. **Reduced order surrogate models**
5. **Data science method for regression classification**
6. System design



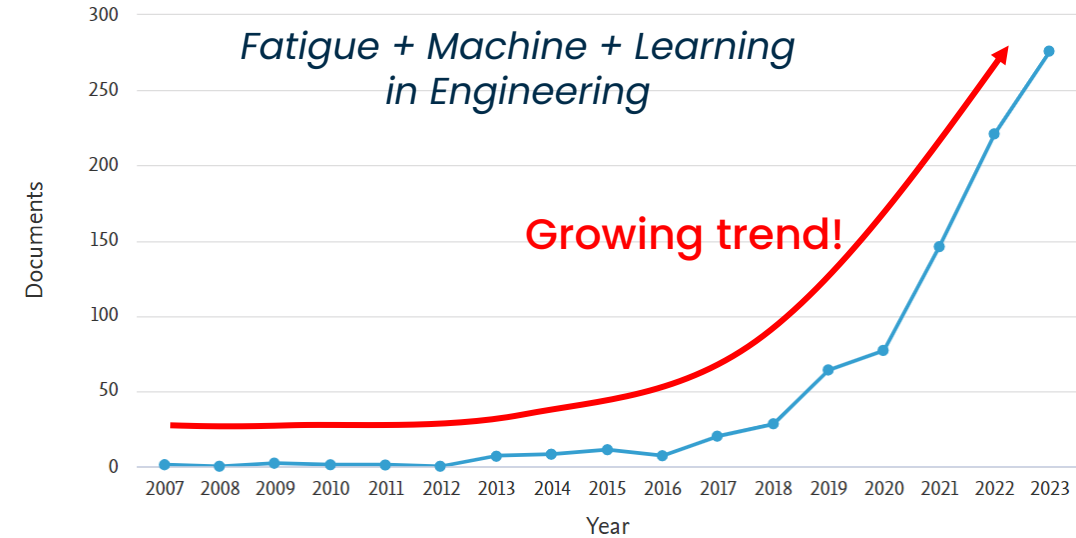
ML for fatigue: state of the art

Growing research interests towards ML applied to fatigue:

- Data-driven problem from the beginning
- Phenomenological modes derived from observations
- Few physics equations are known (e.g., LEFM)

Main goals:

- Predict the effect of design variables on the fatigue response from data (e.g. manufacturing, environmental)
- Discover governing models from observation and expand the knowledge
- Discover effect of defects and porosity on fatigue response



ML applied to structural integrity

When applying ML to scientific problems we want to:


1. Merge the data with physical knowledge
2. Understand the confidence of the prediction
+ in case of stochastic phenomena (e.g., fatigue):
3. Have a probabilistic prediction

Physics-informed neural networks

[7]

Journal of Computational Physics 378 (2019) 686–707

Contents lists available at [ScienceDirect](#)

 ELSEVIER


Journal of Computational Physics

www.elsevier.com/locate/jcp

Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations

M. Raissi^a, P. Perdikaris^{b,*}, G.E. Karniadakis^a

^a Division of Applied Mathematics, Brown University, Providence, RI, 02912, USA
^b Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA, 19104, USA



[8] Blundell, Charles, et al. "Weight uncertainty in neural network." International conference on machine learning. PMLR, 2015.

Physics-Informed Neural Networks

Neural networks that are trained to solve supervised learning tasks while respecting any given laws of physics described by general nonlinear partial differential equations.

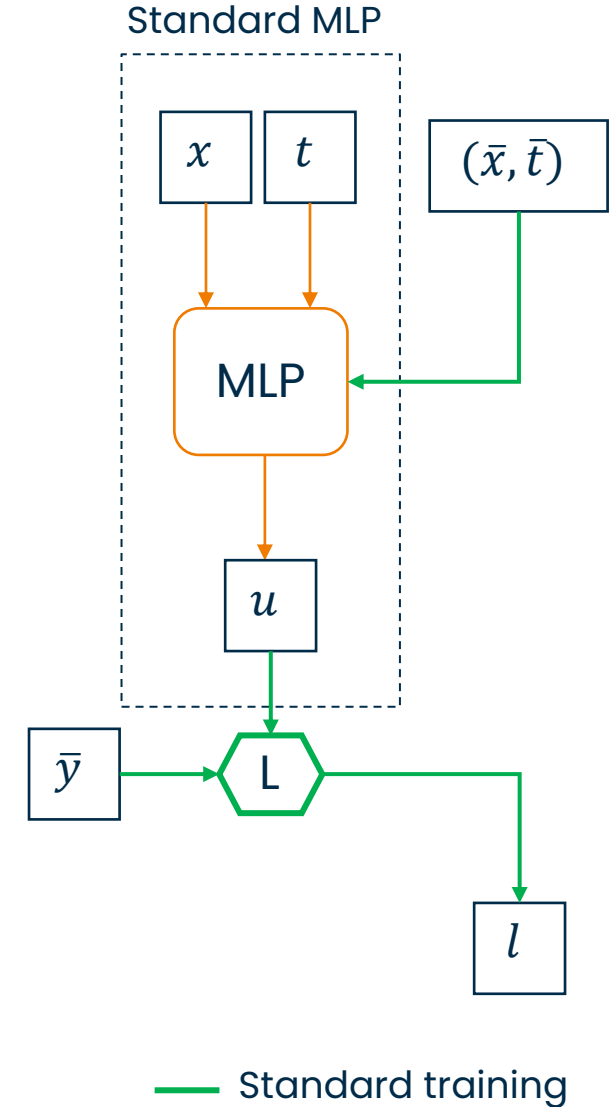
A general PDE over a domain Ω , in a time interval $[0, T]$, with non-linear operator \mathcal{N} is introduced:

$$u_T + \mathcal{N}[u] = 0, x \in \Omega, t \in [0, t]$$

The solution $u(x, t)$ is learned by a MLP, defined as $MLP(t, x; \theta)$, from a set of observed data $(\bar{x}, \bar{t}, \bar{u})$.

With a standard approach:

$$\theta_{trained} = \arg \min_{\theta} L(\bar{x}, \bar{t}, \bar{u}) = \arg \min_{\theta} \boxed{MSE(MLP(\bar{x}, \bar{t}; \theta), \bar{u})}$$



Physics-Informed Neural Networks

Neural networks that are trained to solve supervised learning tasks while respecting any given laws of physics described by general nonlinear partial differential equations.

A general PDE over a domain Ω , in a time interval $[0, T]$, with non-linear operator \mathcal{N} is introduced:

$$u_T + \mathcal{N}[u] = 0, x \in \Omega, t \in [0, t]$$

The solution $u(x, t)$ is learned by a MLP, defined as $MLP(t, x; \theta)$, from a set of observed data $(\bar{x}, \bar{t}, \bar{u})$.

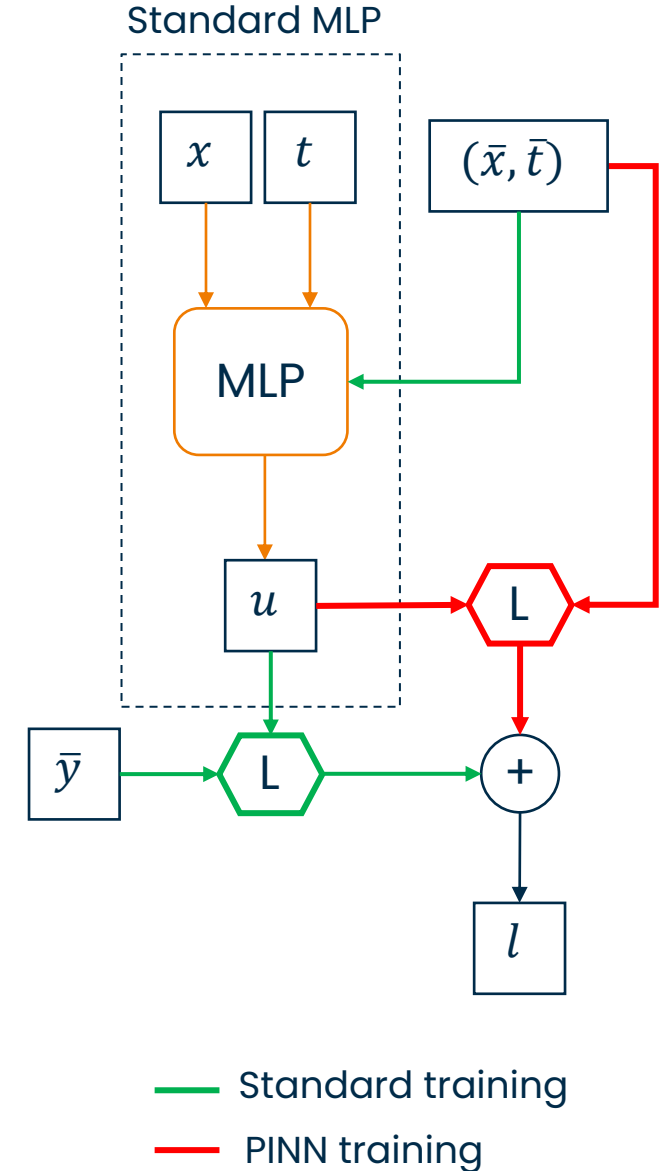
With a standard approach:

$$\theta_{trained} = \arg \min_{\theta} L(\bar{x}, \bar{t}, \bar{u}) = \arg \min_{\theta} \boxed{MSE(MLP(\bar{x}, \bar{t}; \theta), \bar{u})}$$

Since the PDE must hold for a valid solution, it should embed in the loss function:

$$\theta_{trained} = \arg \min_{\theta} [L(\bar{x}, \bar{t}, \bar{u}) + \boxed{MLP_T(\bar{x}, \bar{t}; \theta) + \mathcal{N}[u]}]$$

$$u_T + \mathcal{N}[u] = 0$$



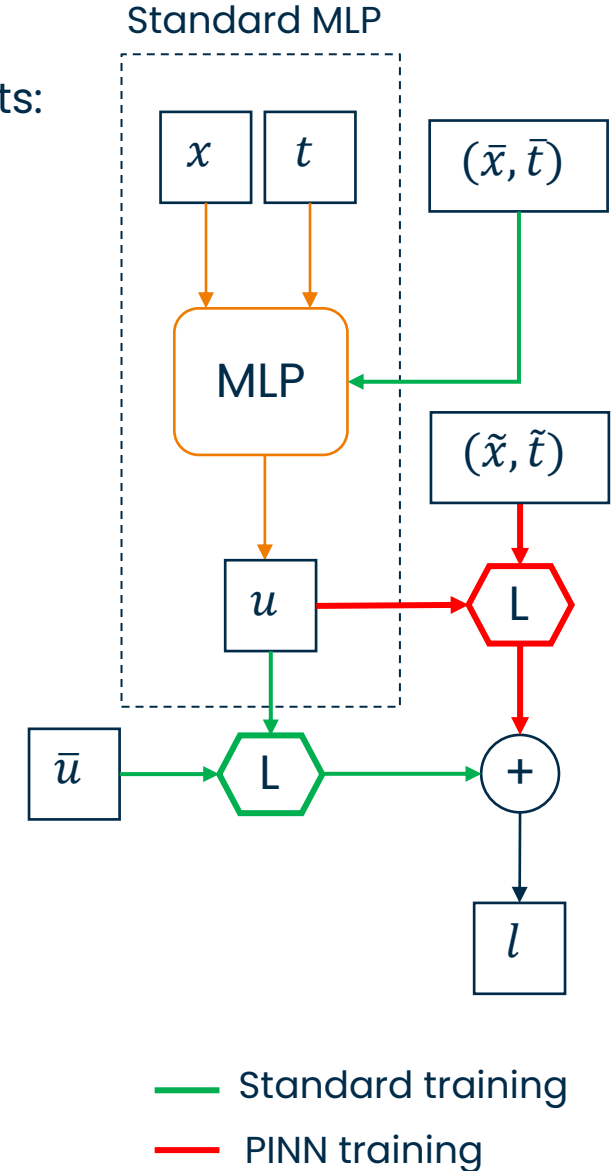
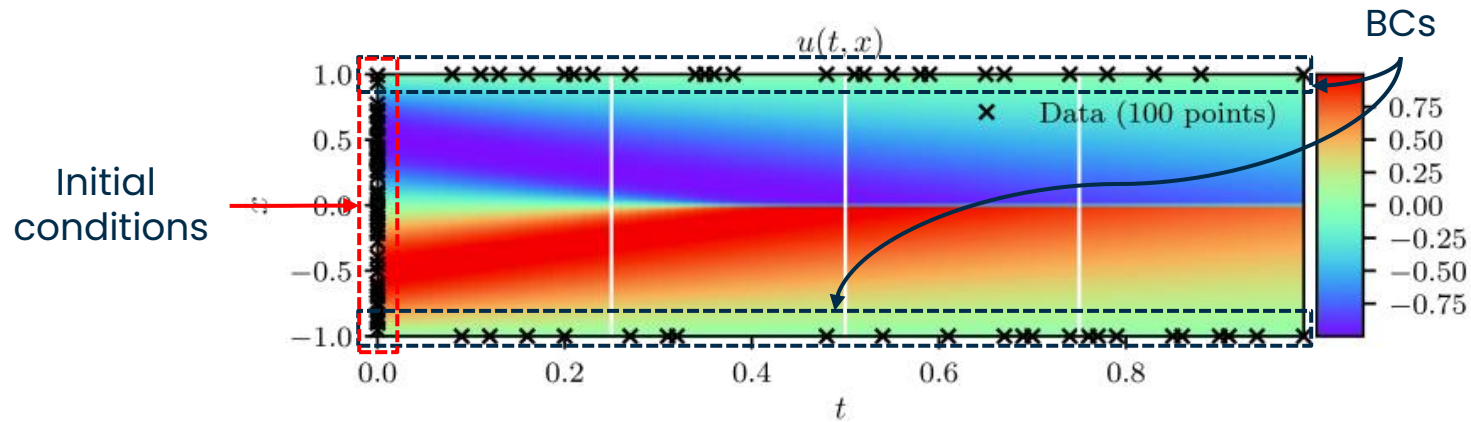
Physics-Informed Neural Networks

Physics constraints can be expanded out of the training data using collocation points:

$$\theta_{trained} = \arg \min_{\theta} L_{MSE}(\bar{x}, \bar{t}, \bar{u}; \theta) + L_{phy}(\tilde{x}, \tilde{t}; \theta), (\tilde{x}, \tilde{t}) \text{ in } \mathbb{R} \setminus \Omega$$

		(\tilde{x}, \tilde{t})						
		N_f	2000	4000	6000	7000	8000	10000
(\bar{x}, \bar{t})	N_u							
	20		2.9e-01	4.4e-01	8.9e-01	1.2e+00	9.9e-02	4.2e-02
	40		6.5e-02	1.1e-02	5.0e-01	9.6e-03	4.6e-01	7.5e-02
	60		3.6e-01	1.2e-02	1.7e-01	5.9e-03	1.9e-03	8.2e-03
	80		5.5e-03	1.0e-03	3.2e-03	7.8e-03	4.9e-02	4.5e-03
	100		6.6e-02	2.7e-01	7.2e-03	6.8e-04	2.2e-03	6.7e-04
	200		1.5e-01	2.3e-03	8.2e-04	8.9e-04	6.1e-04	4.9e-04

Increasing collocations points leads to better prediction -> PINN added value!



Physics-Informed Neural Networks

Collocation help extrapolation.

Interpolation vs extrapolation:

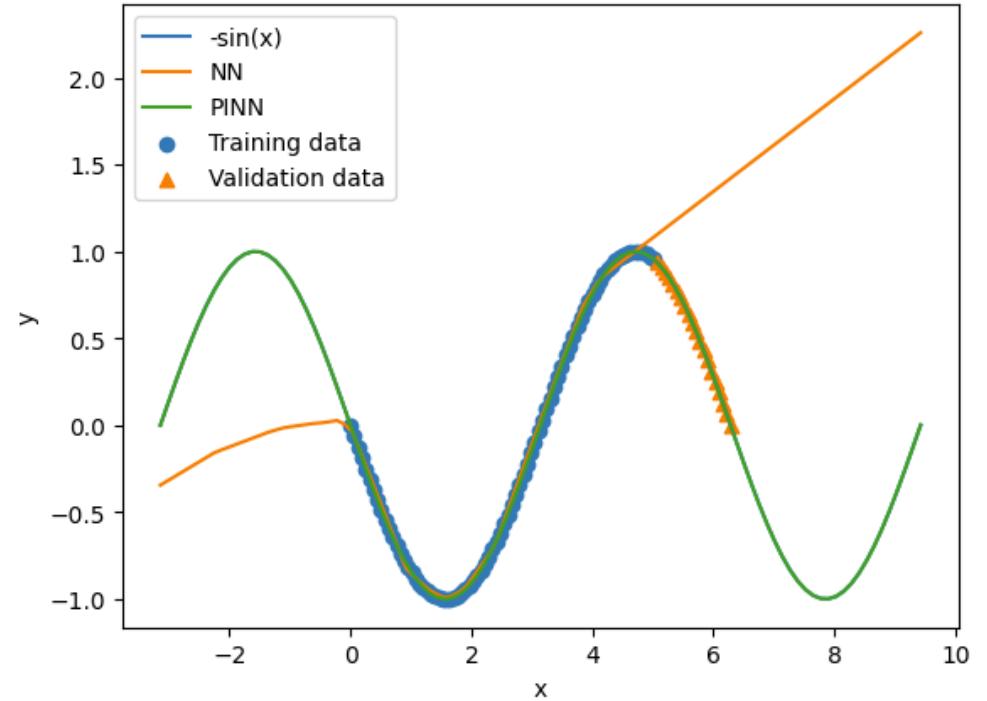
Interpolation: inside the training data domain

Extrapolation: outside the training data domain

Toy example:

$$y = f(x) = \sin(x)$$

$$MLP(x; \theta) = y_{pred}$$



The PINN training enforce the correct derivative:

$$L(\bar{x}, \bar{y}, \theta) = \frac{1}{N} \sum_{i=1}^N (MLP(\bar{x}_i, \theta) - \bar{y}_i)^2 + \frac{1}{M} \sum_{j=1}^M \left(\frac{d}{dx} MLP(\tilde{x}_j, \theta) - \cos(\tilde{x}_j) \right)^2$$

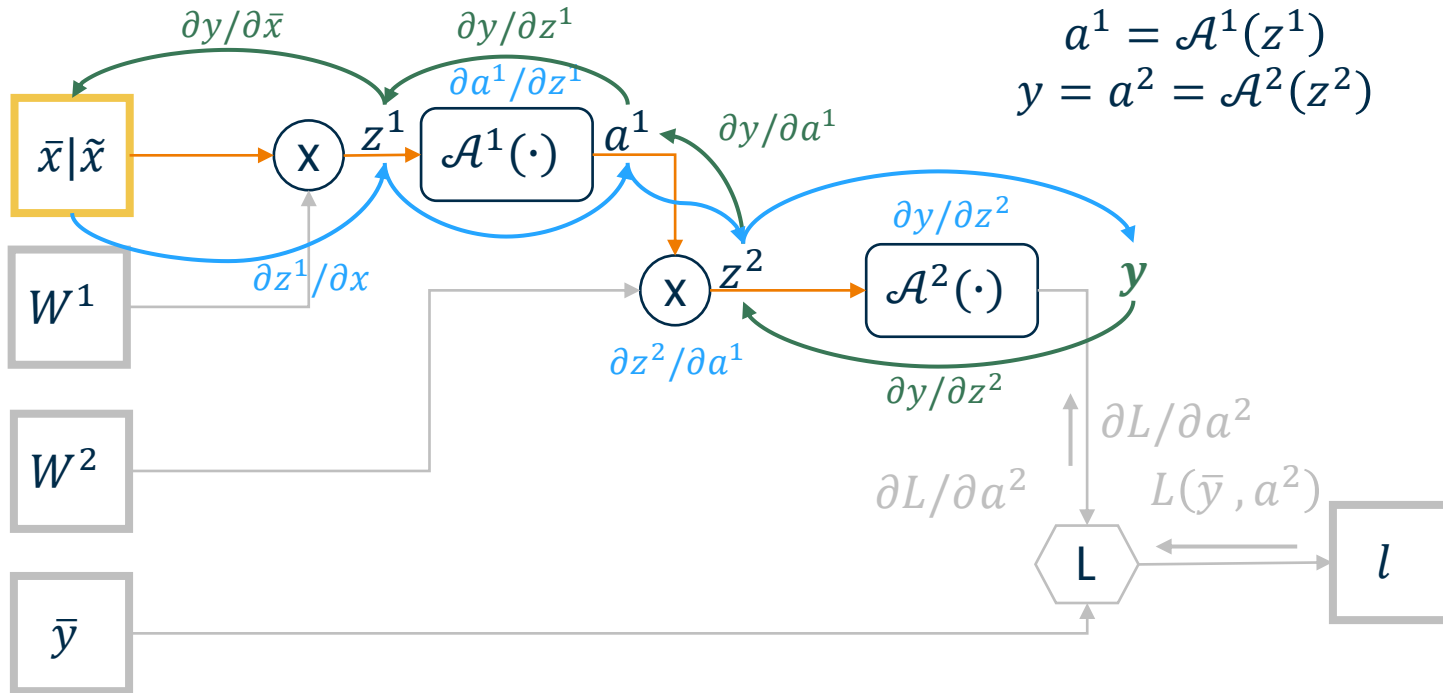
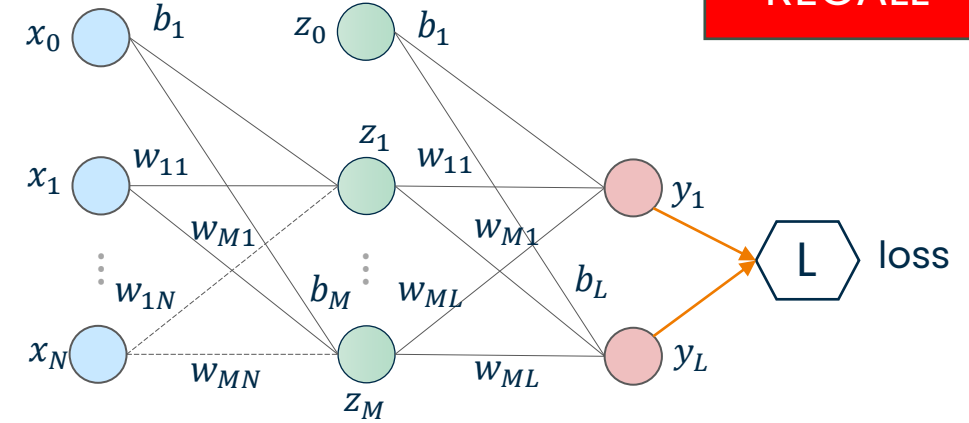
MSE

PI

Back-propagation in PINN

1. Define the computational graph of the network
2. Calculate the forward pass
 - a) Variables
 - b) Local gradients
3. Calculate the backward pass
 - a) Get the gradient $\partial y / \partial \bar{x}$

RECALL



$$\frac{\partial y}{\partial a^1} = \frac{\partial y^2}{\partial z^2}$$

$$\frac{\partial y}{\partial z^2} = \frac{\partial y^2}{\partial z^2} \frac{\partial a^1}{\partial z^2}$$

$$\frac{\partial y}{\partial z^1} = \frac{\partial y^2}{\partial z^2} \frac{\partial a^1}{\partial z^1} \frac{\partial z^1}{\partial z^2}$$

$$\frac{\partial y}{\partial \bar{x}} = \frac{\partial y^2}{\partial z^2} \frac{\partial a^1}{\partial z^1} \frac{\partial z^1}{\partial x}$$

PINN outtakes

It can be concluded that:

- PINN has a standard architecture
- The way the network is trained makes it physics-informed
- All method applied to MLP can be extended to PINN (Dropout, regularization, ..)
- It can be extended behind PDEs, anywhere there is an apriori knowledge coupling the input with the output with a mathematical expression
- Thanks to the MLP differentiability, this expression can contain derivatives

PINN applications

Prediction of the SIF of interacting ellipsoidal defects from defect characteristics [8]:

- Input: defect features
- Output: defect stress intensity factor ΔK_{rel}
- Physics knowledge: Murakami equation:

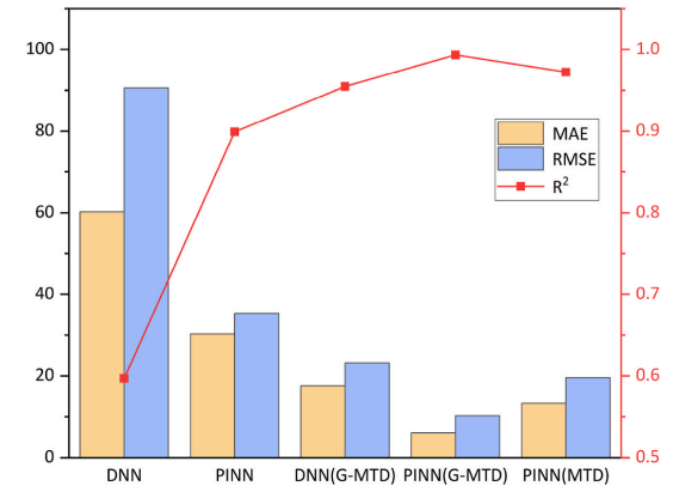
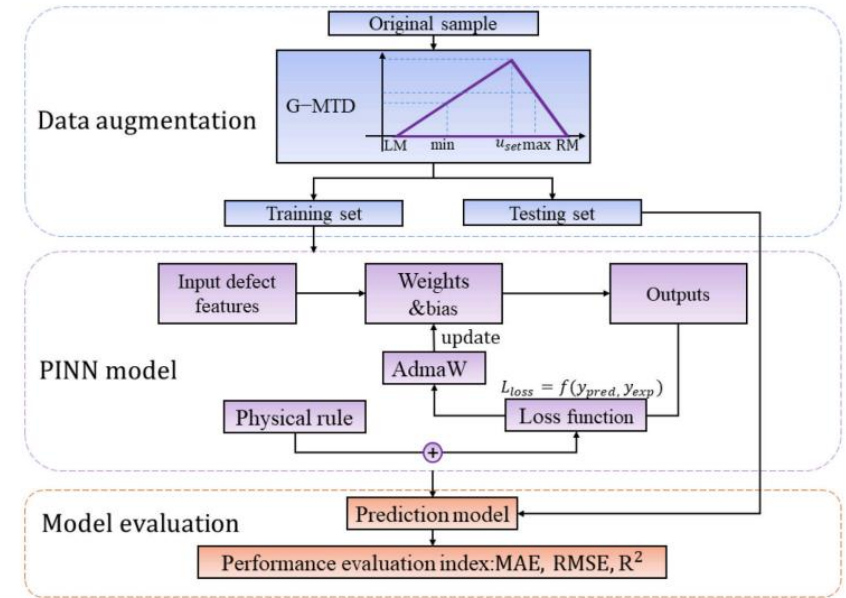
$$\Delta K_{rel} = F \cdot S \cdot K_t(AR) \cdot K_t(FR) \cdot K_t(interaction) \cdot Y \cdot \sqrt{\pi(area)^{0.5}}$$

$$\frac{d\Delta K_{rel}}{darea} = C(area)^{-\frac{3}{4}}$$

Physics-informed loss:

$$L = MSE + \left| \frac{\partial \Delta \hat{K}_{rel}}{\partial area} - \frac{d\Delta K_{rel}}{darea} \right|$$

- Improved results training on expanded database



[8] Wang et al, Fatigue life prediction driven by mesoscopic defect data, Engineering Applications of Artificial Intelligence (2024)

PINN applications

Prediction of multi-axial fatigue response^[9]:

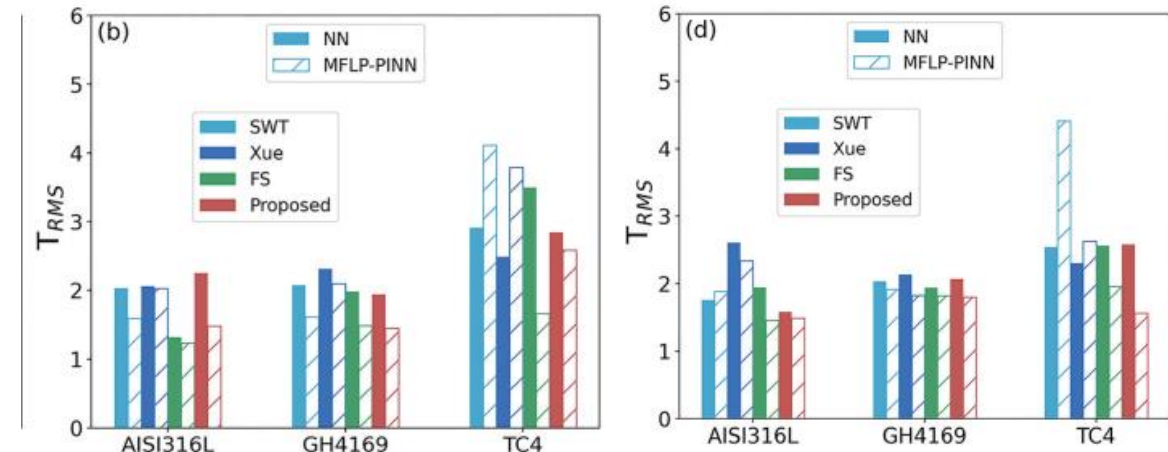
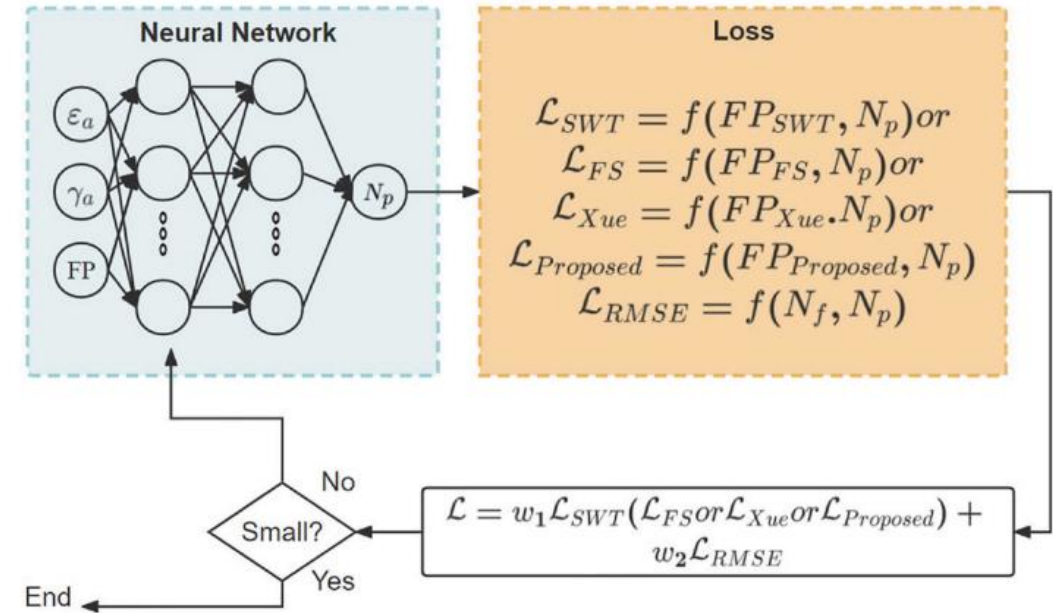
- Input: shear and normal strain amplitude, model parameters
- Output: fatigue life
- Physics knowledge: multi-axial fatigue model

$$\left(\frac{\tau_f'}{G} (2N_f)^{b_0} + \gamma_f' (2N_f)^{c_0} - a \right) = 0$$

Physics-informed loss:

$$L = MSE + \left| \left(\frac{\tau_f'}{G} (2N_{pi})^{b_0} + \gamma_f' (2N_{pi})^{c_0} - a \right) \right|$$

- Improved results w.r.t. standard NN
- Tested over different models



[9] He, Zhao, Yan, MFLP-PINN: A physics-informed neural network for multiaxial fatigue life prediction, European Journal of Mechanics and Solids (2023)

PINN applications

Prediction of fatigue curves^[10] of Additively Manufactured metals from process parameters:

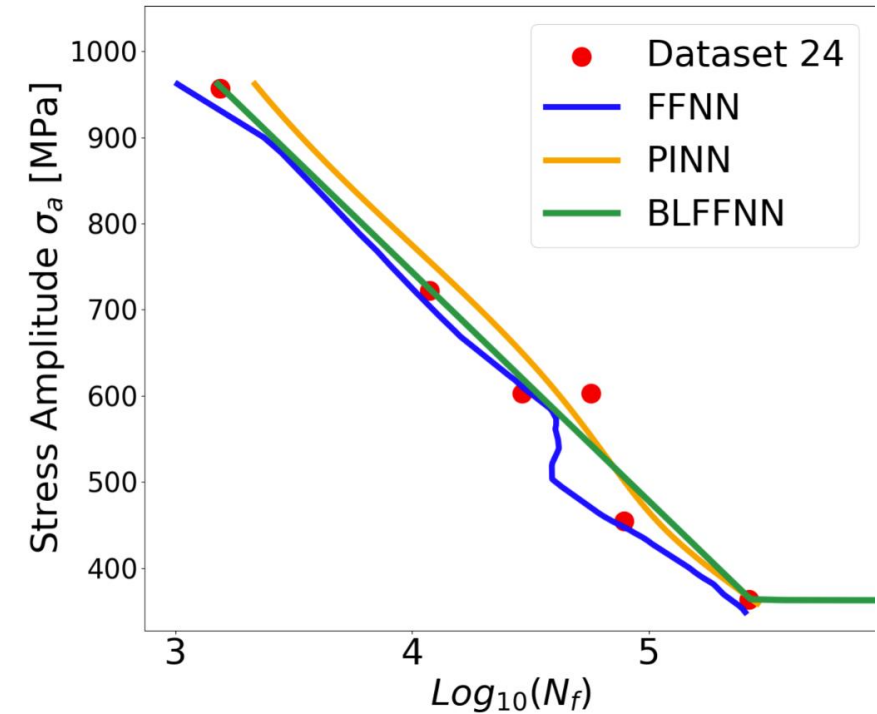
- Input: manufacturing parameters
- Output: fatigue life
- Physics knowledge: Murakami, negative slope, positive curvature:

$$\frac{dN_f}{dS_a} < 0; \frac{d^2N_f}{dS_a^2} > 0;$$

Physics-informed loss:

$$L = MSE + \frac{-\frac{dN_f}{dS_a} + \left|\frac{dN_f}{dS_a}\right|}{2}$$

- Improved results w.r.t. standard NN
- Tested over different models



[10] Centola, Ciampaglia, Tridello, Paolino. Machine learning methods to predict the fatigue life of selectively laser melted Ti6Al4 components, FFEMS (2024)

TUTORIAL

[Click here](#)



**Politecnico
di Torino**

ML applied to structural integrity


When applying ML to scientific problems we want to:

1. Merge the data with physical knowledge
 2. Understand the confidence of the prediction
- + in case of stochastic phenomena (e.g., fatigue):
3. Have a probabilistic prediction

Novel methods properly developed to account for:

- The physics of the modelled problem
- The stochastic nature of the problem
- The uncertainty of the model

Physics-informed neural networks [7]



Journal of Computational Physics 378 (2019) 686–707

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations

M. Raissi^a, P. Perdikaris^{b,*}, G.E. Karniadakis^a

^a Division of Applied Mathematics, Brown University, Providence, RI, 02912, USA
^b Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA, 19104, USA

Bayesian neural networks [8]

Weight Uncertainty in Neural Networks

Charles Blundell
Julien Cornebise
Koray Kavukcuoglu
Daan Wierstra
 Google DeepMind

CBLUNDELL@GOOGLE.COM
 JUCOR@GOOGLE.COM
 KORAYK@GOOGLE.COM
 WIERSTRA@GOOGLE.COM

[8] Blundell, Charles, et al. "Weight uncertainty in neural network." International conference on machine learning. PMLR, 2015.

Bayesian Neural Networks

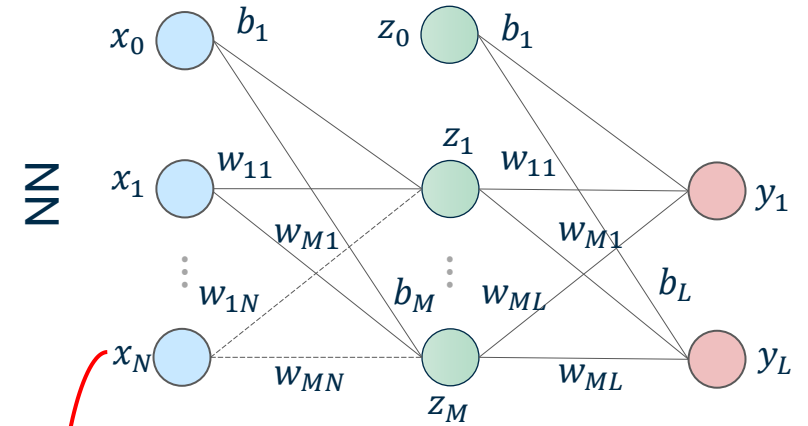
“Minimizing MSE of the mean is equivalent to maximizing the log likelihood of the data under the assumption of Gaussian noise.”

It is hence true that:

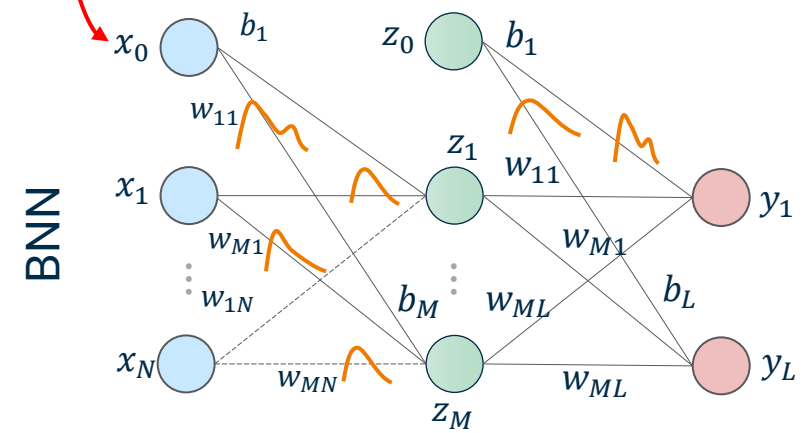
$$MLP(\mathbf{x}, \mathbf{w}) = \mathbf{y}, \mathbf{x} \in \mathbb{R}^N, \mathbf{y} \in \mathbb{R}^M$$

$$\mathbf{w} = \arg \max_{\mathbf{w}} \log(P(\bar{\mathbf{y}}|\bar{\mathbf{x}}, \mathbf{w})) = \arg \max_{\mathbf{w}} \sum_i \log P(\bar{x}_i, \bar{y}_i, \mathbf{w})$$

This approach can be extended to a probabilistic neural network, where the weights \mathbf{w} are random functions -> Bayesian Neural Network. The BNN is random and will give a different results every time is used.



w_{ij}^k is a random variable



Bayesian Neural Networks

We want to find the distribution of weights w given the training dataset

$\mathcal{D} = (\bar{x}, \bar{y})$, which maximize $P(w|\mathcal{D})$.

Using Bayes theorem:

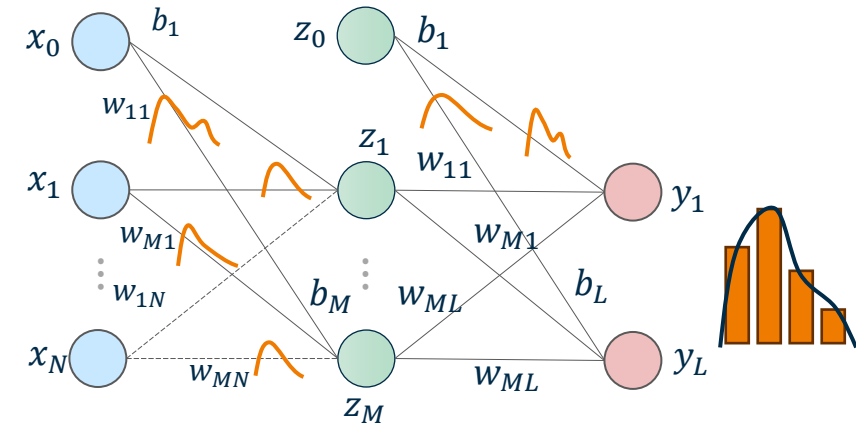
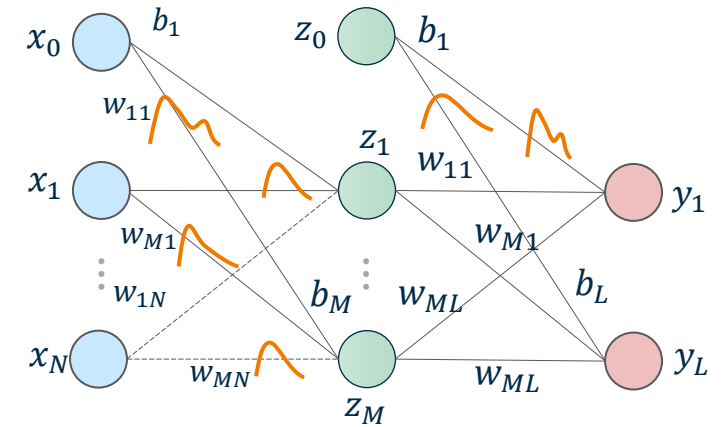
$$P(w|\mathcal{D}) = \frac{P(\mathcal{D}|w)P(w)}{P(\mathcal{D})}$$

Posterior distribution
Prior distribution
Prediction On \mathcal{D}

The predicted expectation can be computed with multiple predictions:

$$P(\bar{y}|\bar{x}) = \mathbb{E}_{P(w|\mathcal{D})}[P(\bar{y}, \bar{x}, w)] \approx \sum_i p(w|\mathcal{D}) BNN^i(\bar{x}, \bar{y}, w)$$

With a variational approach, the posterior distribution is a parametric function (e.g., Gaussian, Bernoulli, ..) $q(w, \theta)$ with parameters θ .



Bayesian Neural Networks: training

The problem becomes:

$$\theta = \arg \min_{\theta} KL[q(w|\theta)||P(w|\mathcal{D})]$$

Where KL is the Kullback-Leibler divergence between two distributions:

$$KL[q(w|\theta)||P(w|\mathcal{D})] = \int q(w|\theta) \log \left(\frac{q(w|\theta)}{P(w|\mathcal{D})} \right) dw = \int q(w|\theta) \log \left(\frac{q(w|\theta)}{P(w)P(\mathcal{D}|w)} \right) dw$$

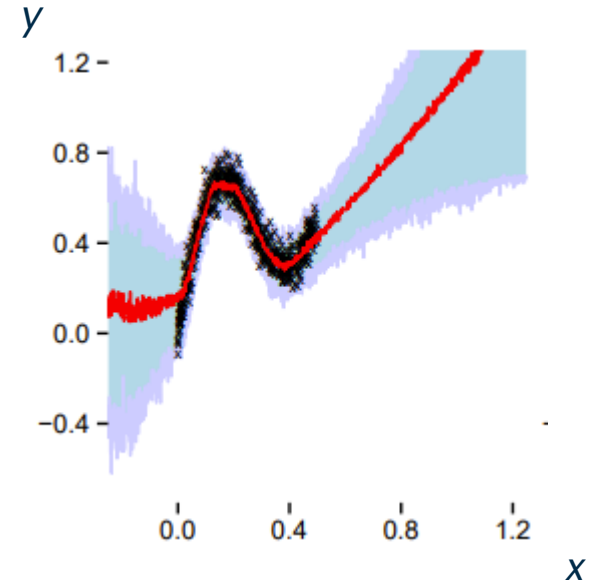
That is equal to:

$$KL[q(w|\theta)||P(w|\mathcal{D})] = KL[q(w|\theta)||P(w)] - \int \log(P(\mathcal{D}|w)) q(w|\theta) dw$$

$$L(\mathcal{D}, \theta) = KL[q(w|\theta)||P(w)] - \mathbb{E}_{q(w|\theta)}[\log(P(\mathcal{D}|w))] \quad \mathbb{E}_{f(x)}[g(x)] = \int g(x)f(x)dx$$

$$L(\mathcal{D}, \theta) = \mathbb{E}_{q(w|\theta)}[\log(q(w|\theta)) - \log(P(w)) - \log(P(\mathcal{D}|\theta))]$$

We recognize a data-related term and a complexity term in the loss function



Bayesian Neural Networks: training

To minimize the loss we need its gradient. Using the proposition:

$$\frac{\partial}{\partial \theta} \mathbb{E}_{q(w|\theta)} [\log(P(\mathcal{D}|w))] = \mathbb{E}_{q(\epsilon)} \left[\frac{\partial f(w, \theta)}{\partial w} \frac{\partial w}{\partial \theta} + \frac{\partial f(w, \theta)}{\partial \theta} \right]$$

With ϵ a small random number so that $q(\epsilon)d\epsilon = w(w|\theta)d\theta$.

$$\mathbb{E}_{q(w|\theta)} [\log(q(w|\theta)) - \log(P(w)) - \log(P(\mathcal{D}|w))] = \mathbb{E}_{q(w|\theta)} [l(\theta, \mathcal{D})]$$

$$\approx \sum_i^n \log(q(w^{(i)}|\theta)) - \log(P(w^{(i)})) - \log(P(\mathcal{D}|w^{(i)}))$$

Monte-Carlo sampling from the variational posterior

In case of Gaussian distribution of weights $q(w|\theta) = \mathcal{N}(w|\mu, \sigma)$, $\theta = (\mu, \sigma)$:

$$w = \mu + \log(1 + \exp(\sigma)) \cdot \epsilon$$

$$\frac{\partial}{\partial \mu} \mathbb{E}_{q(w|\theta)} [\log(P(\mathcal{D}|w))] = \mathbb{E}_{q(\epsilon)} \left[\frac{\partial l(w, \theta)}{\partial w} \cdot 1 + \frac{\partial l(w, \theta)}{\partial \mu} \right] = \nabla_{\mu} L$$

Gradient of BNN

$$\frac{\partial}{\partial \sigma} \mathbb{E}_{q(w|\theta)} [\log(P(\mathcal{D}|w))] = \mathbb{E}_{q(\epsilon)} \left[\frac{\partial l(w, \theta)}{\partial w} \frac{\epsilon}{1 + \exp(-\sigma)} + \frac{\partial l(w, \theta)}{\partial \sigma} \right] = \nabla_{\sigma} L$$

Gradient of back propagation

Bayesian Neural Networks: Bayes by Backpropagation

$$\frac{\partial}{\partial \mu} \mathbb{E}_{q(w|\theta)} [\log(P(\mathcal{D}|w))] = \mathbb{E}_{q(\epsilon)} \left[\frac{\partial l(w, \theta)}{\partial w} \mathbf{1} + \frac{\partial l(w, \theta)}{\partial \mu} \right] = \nabla_{\mu} L$$

$$\frac{\partial}{\partial \sigma} \mathbb{E}_{q(w|\theta)} [\log(P(\mathcal{D}|w))] = \mathbb{E}_{q(\epsilon)} \left[\frac{\partial l(w, \theta)}{\partial w} \frac{\epsilon}{1 + \exp(-\sigma)} + \frac{\partial l(w, \theta)}{\partial \sigma} \right]$$

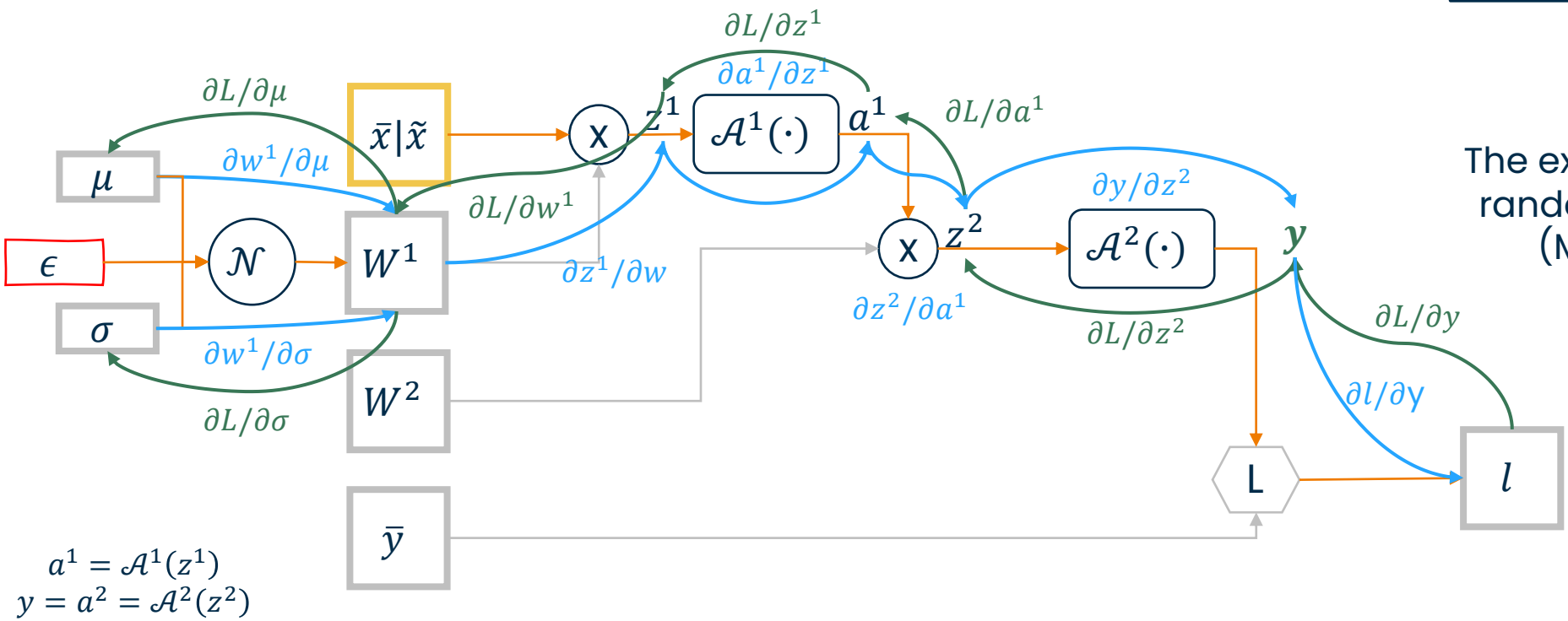
$$\frac{\partial l(w, \theta)}{\partial w^1} = \frac{\partial l}{\partial y} \frac{\partial y}{\partial z^2} \frac{\partial z^2}{\partial a^1} \frac{\partial a^1}{\partial z^1} \frac{\partial z^1}{\partial w^1}$$

$$\frac{\partial l(w, \theta)}{\partial \mu} = \frac{\partial l}{\partial y} \frac{\partial y}{\partial z^2} \frac{\partial z^2}{\partial a^1} \frac{\partial a^1}{\partial z^1} \frac{\partial z^1}{\partial w^1} \frac{\partial w^1}{\partial \mu}$$

$$\frac{\partial l(w, \theta)}{\partial \sigma} = \frac{\partial l}{\partial y} \frac{\partial y}{\partial z^2} \frac{\partial z^2}{\partial a^1} \frac{\partial a^1}{\partial z^1} \frac{\partial z^1}{\partial w^1} \frac{\partial w^1}{\partial \sigma}$$

From the computational graph

The expectation is computed by randomly sampling ϵ n times (Monte Carlo sampling)



Bayesian Neural Networks

- Allow to estimate the model uncertainty
- It is a modified architecture with probabilistic weights
- It requires a modified back-propagation
- The number of parameters **increases** (double for diagonal normal, escalate for multivariate distributions with weights co-variance)
- Can be used for adaptive sampling strategies: sample data in uncertain regions on domain
- Can be used to suppress connections with high noise-to-signal ratio (e.g., σ/μ)

Thank you all for your kind attention!

email: alberto.ciampaglia@polito.it



Politecnico
di Torino

