«KOLMOGOROV-ARNOLD NETWORK (KAN) FOR ROBOT CONTROL»

ROBOTICS PROJECT – TUTOR: DR.ENRICO DONATO

Università degli Studi di Pisa, Master's Degree in Computer Science – Artificial Intelligence

Course: «Robotics» (387AA)

Gaetano Nicassio (658073)

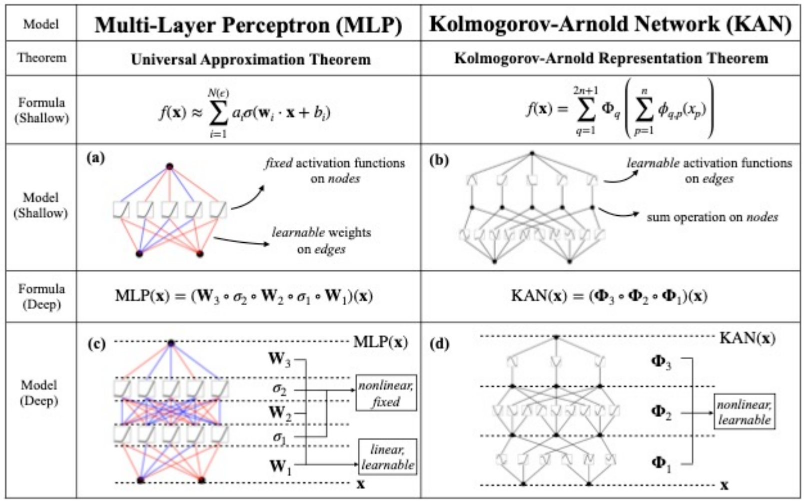
MOTIVATION

Continuum robots, pose significant challenges in modeling and control due to their infinite degrees of freedom and complex dynamics.

- Traditional modeling approaches: Cosserat rod theory, Piecewise Constant Curvature model
- Data-driven methods:
  - **Neural networks** like **Multi-Layer Perceptrons** (**MLPs**) : *Statics learning* involves understanding the relationship between actuator inputs (e.g., cable tensions, pressures) and the resultant static configurations of the robot
  - **Recurrent Neural Networks** (**RNNs**): *Dynamics learning* addresses how the robot's state evolves over time under dynamic conditions, including inertia and external forces

Kolomogorov-Arnold Networks (**KANs**) [1] a new learning paradigm, promising to outperform MLPs:

- Learning of activation functions instead of learning of weights
- Continual Learning capabilities
- Interpretability through pruning and refinement mechanisms of the network



OBJECTIVE

Question: Can KANs improve accuracy for robot static?

Learn direct and inverse static models on continuum robots by using KANs and compare the results with MLP on:

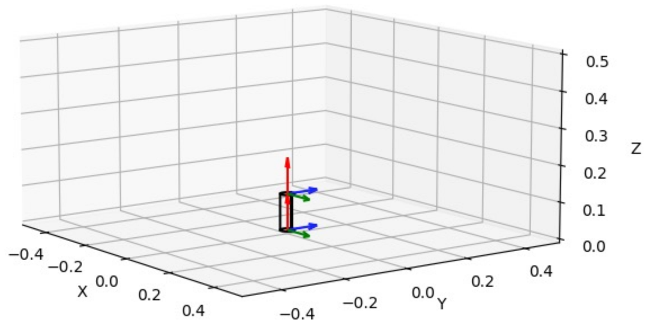
- Accuracy (based on Loss MSE)
- Computational complexity
- Continual Learning

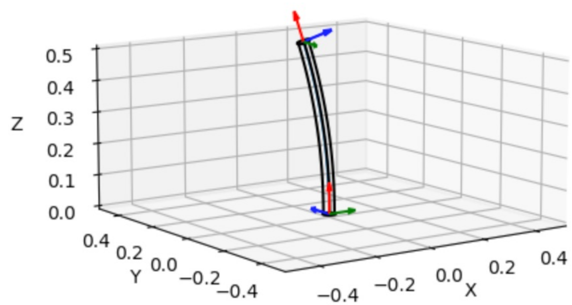
CONTINUUM ROBOT SIMULATOR

To generate our dataset we use a Robot Simulator based on PCC assumption:

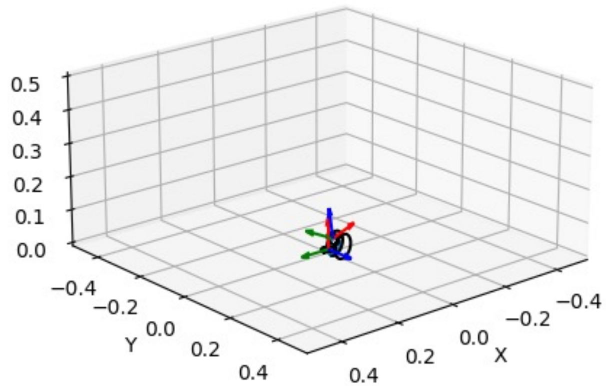
- Provide inputs in the actuation space (3 actuators/segments at 120°)
- Outputs the base and tip

Actuators:
[[.1 .1 .1]]
Poses:
 [[0. 0. 0. 0. 0. 0.]
 [0. 0. 0.1 0. 0. 0.]]





Training dataset



STATIC MODELING

Forward Kinematics: compute a generic function that maps the joint positions to the pose of the end-effector:

- Denavit-Harteberg, Homogeneous Transformation Matrix, ecc.

$$x = k(q)$$

$$\mathbf{T}_n^0 = \mathbf{T}_1^0 \cdot \mathbf{T}_2^1 \cdot \mathbf{T}_3^2 \cdot \dots \cdot \mathbf{T}_n^{n-1}$$

$$T_n^0 = Final\ Transformation\ Matrix, n = numbers\ of\ joints$$

$$x = \begin{bmatrix} x \\ y \\ z \\ \varphi \\ \vartheta \\ \psi \end{bmatrix} \quad q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{bmatrix}$$

STATIC MODELING

Inverse Kinematics: given a desire trajectory for EE, moving the joint to reach the desired position

$$q = k^{-1}(x)$$

$$\mathbf{T}_n^0(\theta_1, \theta_2, \ldots, \theta_n) = \mathbf{T}_{\text{desiderata}} \qquad Given\ T_n^0\ find\ \theta_i$$

CONS:

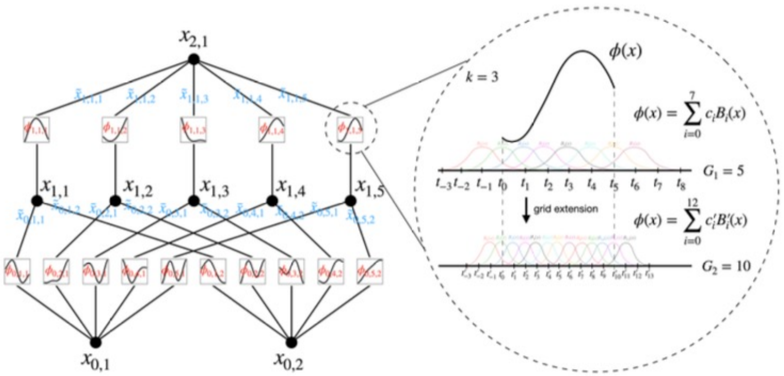
- **Nonlinearity**: Equations involve trigonometric functions.
- **Multiple Solutions**: There are often multiple solutions (configurations) for the same desired T
- **Singularity**: Points where the robot loses degrees of freedom and the solutions become indeterminate.

KAN

Based on **Kolmogorov-Arnold Represenation Theorem [1]:**
«If f is multivariate continuous function on a bounded domain, the f can be written as a finite composition of continuous functions of a single variable and the binary operation of addition»

$$f : [0,1]^n \to \mathbb{R} \qquad f(\mathbf{x}) = f(x_1, \cdots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$$

Use ***B-Spline Curve*** with learneable coefficient of local B-Spline basis functions.



Implementation Details

- **Residual activation function**: the activation function is the sum of the basis function $b(x)$ (silu in the case of KAN) and the spline function:

$$\phi(x) = w_b b(x) + w_s \text{spline}(x).$$

- $spline(x)$ **parametrized as a linear combination** of B-splines such that $c_i s$ are trainable $\text{spline}(x) = \sum_i c_i B_i(x)$
- **Update of spline grids** during the training according to its input activations

KAN

Parameter count:

- Consider a network of L layers, with equal width N
- Each spline of order k on G intervals (grid)

The **complexity** of KAN is $O(N^2L(G+k) \approx O(N^2LG)$ while the complexity of MLPs is $O(N^2L)$ but KANs require much smaller N than MLPs:

- Saves parameters
- Facilitates interpretability

Small KANs generalize better; MLPs need to be deep (scaling law)

Sparsification techniques (L1 Norm, Pruning defined on a score on computation contribution)

KAN can work in **continual learning without catastrophic forgetting** due to the leverage of locality of spline (since spline bases are local, a sample will only affect a few nearby spline coefficient)

SETTING THE STAGE

Two different problems: **forward** and **inverse** kinematics
Two different models to compare: MLPs and KANs

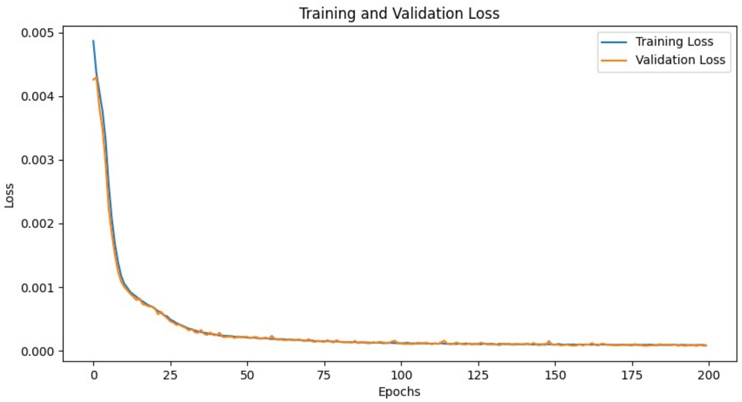
Both the model were trained on the same **dataset** of 100k samples **randomly generated** from PCC Simulator

Parameters for both models were found by grid search, resulting in the following architectures:

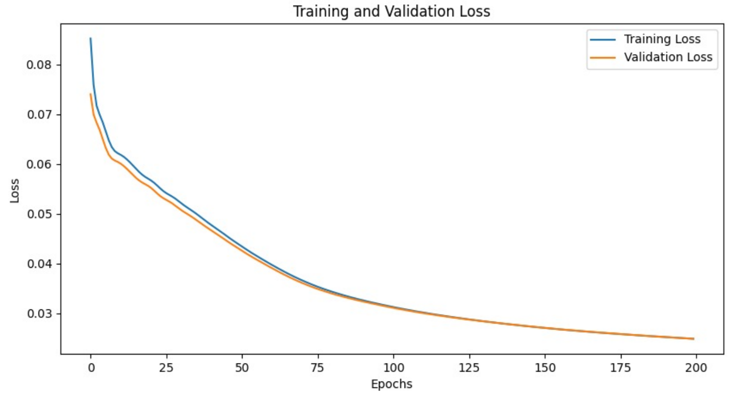
- MLPs:
  - Forward kinematics: 3 layers network [256,256,64] with Tanh as activation function
  - Inverse kinematics: 3 layers network [256,128,64] with Tanh as activation function
- KANs:
  - Forward kinematics: 1 layer network [22] with k=4 and grid=15
  - Inverse kinematics: 1 layer network [19] with k=3 and grid=30

RESULTS ON DIRECT MODELING

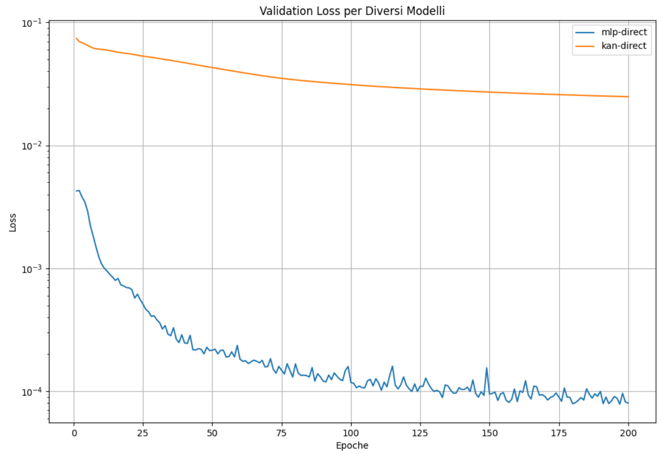
MLP

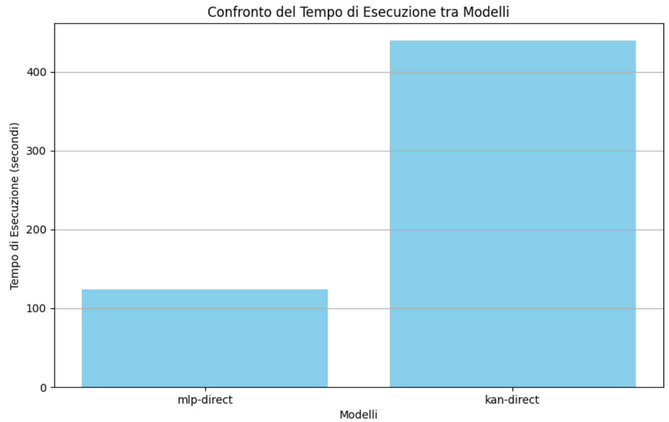


KAN

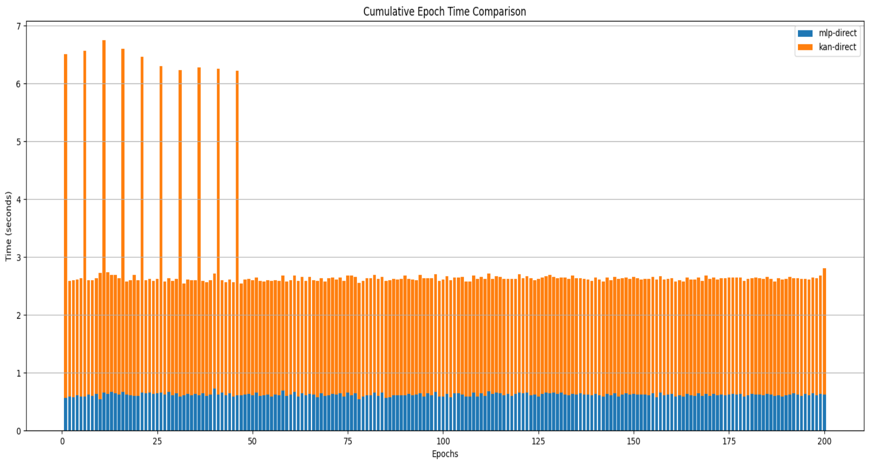


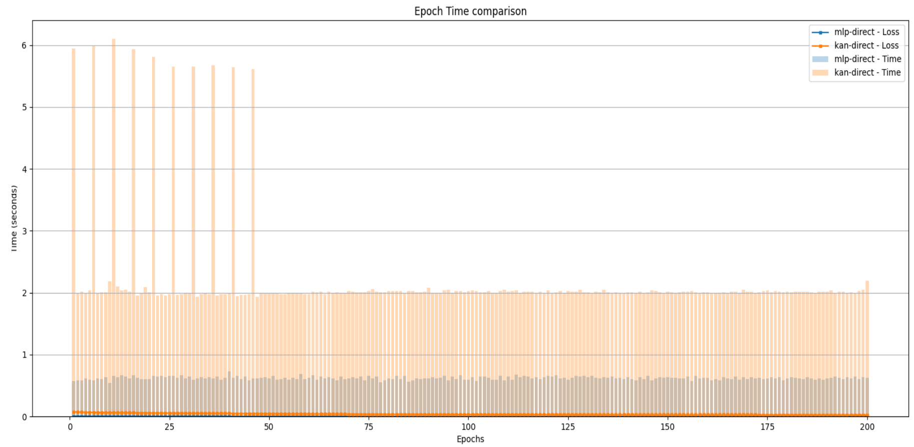
COMPARISON





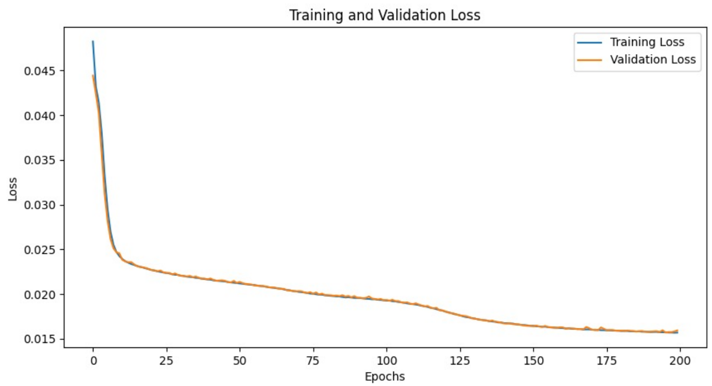
RESULTS ON DIRECT MODELING



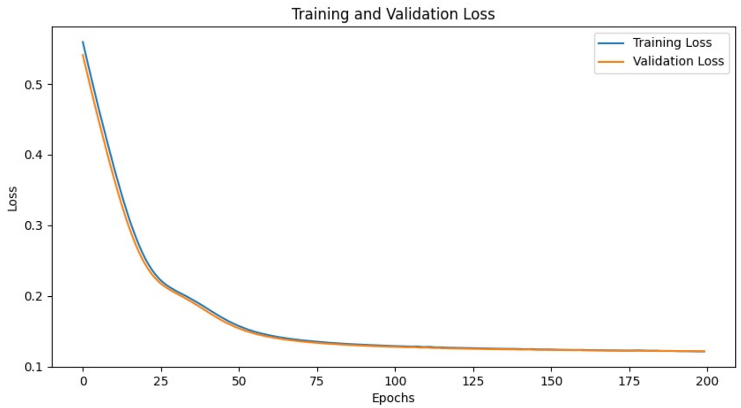


RESULTS ON INVERSE MODELING

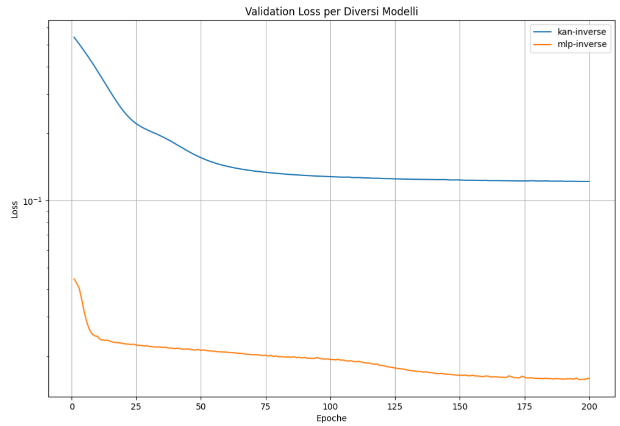
MLP

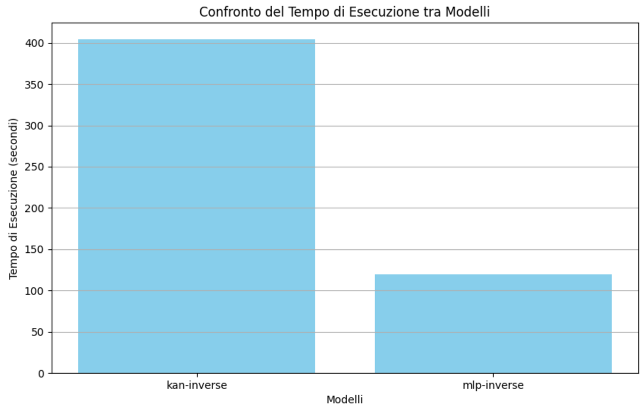


KAN

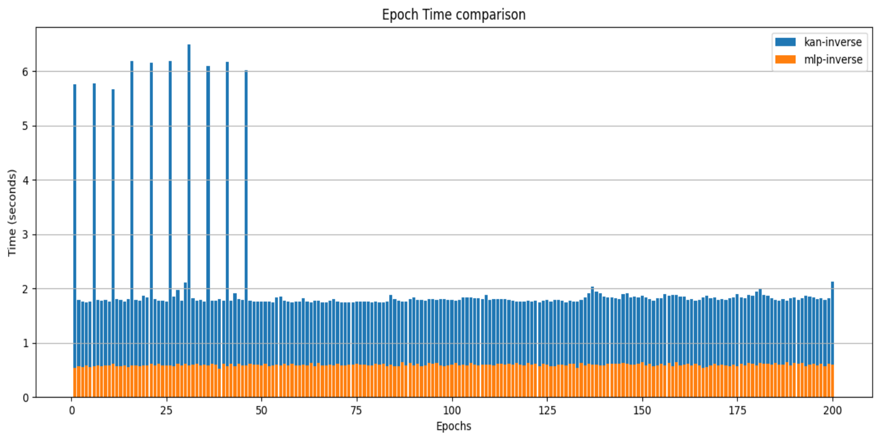


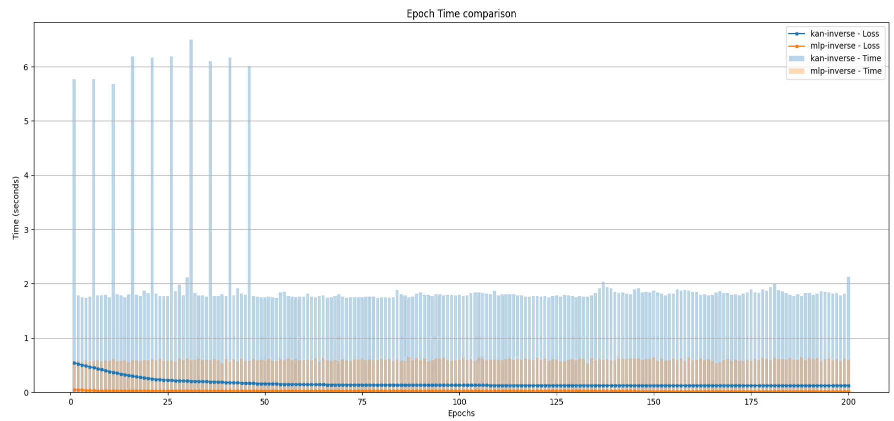
COMPARISON





RESULTS ON INVERSE MODELING





CONTINUAL LEARNING ON INVERSE MODEL

No continual learning techniques used as data reply, weights layer freeze, regularization

Following this **strategy** for each kind of model:

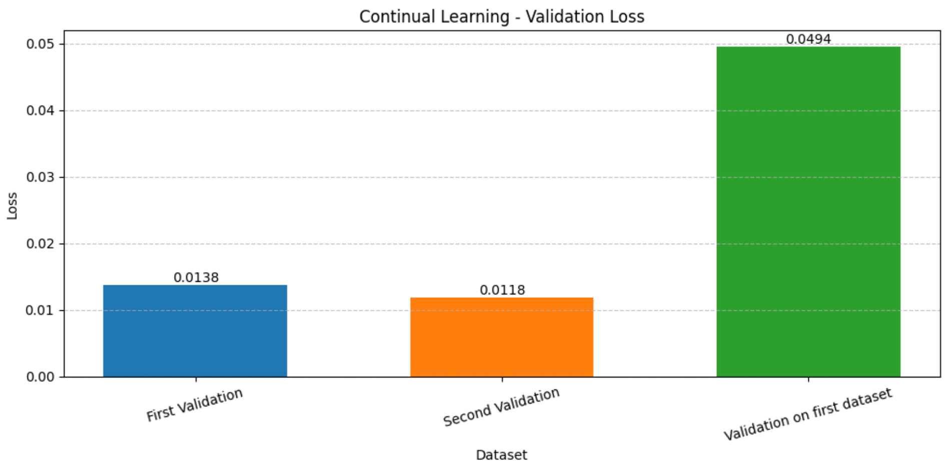
- Create a new randomly generated dataset and split into 2 quadrants of a cartesian space ($x>0$ and $x<0$)
- Train the model on the dataset representing the first quadrant
  - Measure performance (training and validation loss on test dataset)
- Train the model output from previous step on the train dataset representing the second quadrant
  - Measure performance (training and validation loss on test dataset)
- Measure performance of the model output from previous step on the test dataset representing the first quadrant

What do we expect:

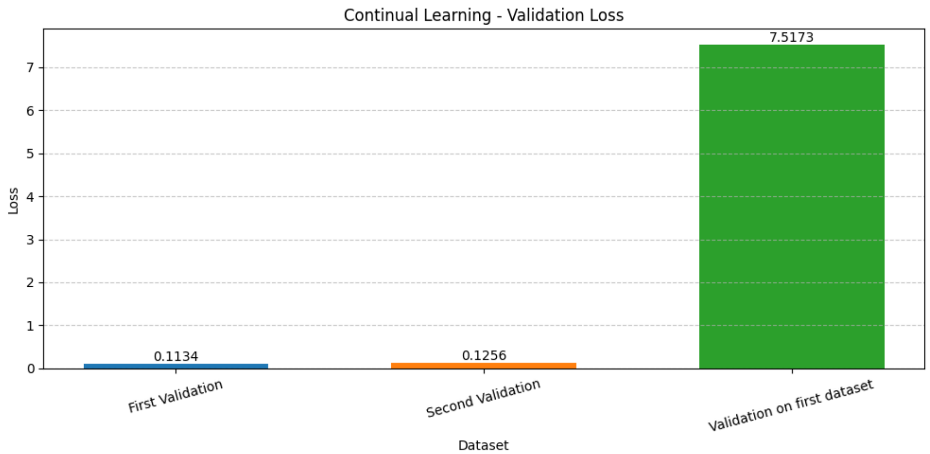
- MLPs catastrophic forgetting
- KANs continual learning with improvements

CONTINUAL LEARNING ON INVERSE MODEL

Continual Learning Validation on MPL



Continual Learning Validation on KAN



SUMMARY OF RESULTS

The experiment refused the initial hypothesis.

Results shows that:

- KANs is not applicable in all kind of problems except of small AI+Science tasks [1], [7]
- Like MLPs it suffer of catastrophic forgetting
- Execution time (and overall performance) are worse than MLPs due to the computation of different activation functions

It seems that KANs are not usefull, but…

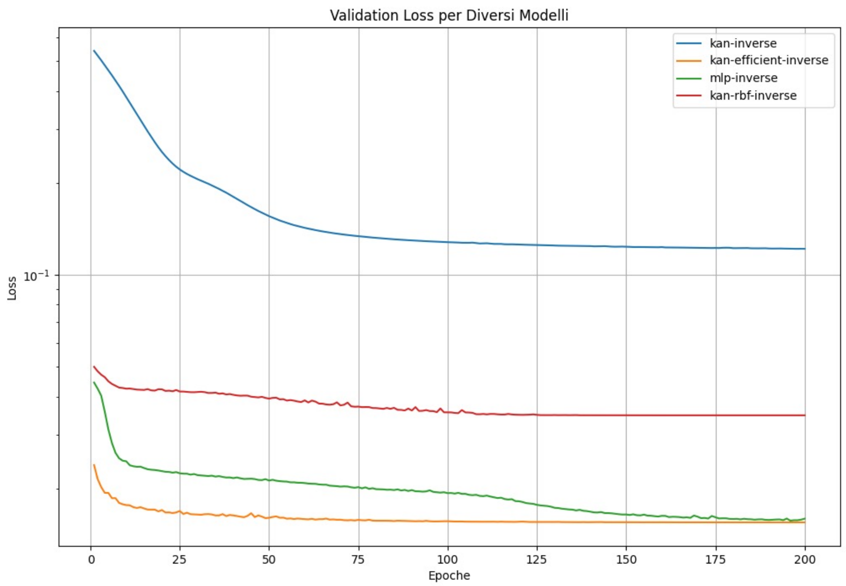
ALTERNATIVE KAN

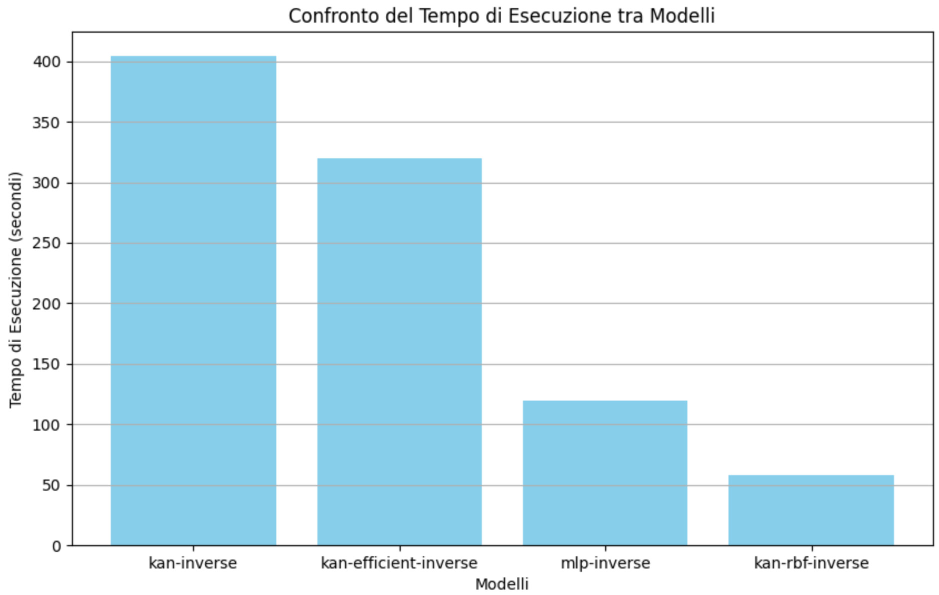
Researcher have done more improvements proposing different KANs implementation based on the same concepts:

- **KAN 2.0 [2]**: some nodes (addition nodes) are copied from corresponding subnodes, while other nodes (multiplication nodes) perform multiplication on k subnodes from the previous layer.
- **Efficient-KAN [3]**: starting from the original implementation, the author instead of using all activation functions as linear combination of a fixed set of basis functions which are B-splines, reformulate the computation as activate the input with different basis functions and then combine them linearly.
- **FastKAN [4]**: using Gaussian Radial Basis Function to approximate B-Spline with easy calculation as they have a uniform grid
- **BSRBF_KAN[5]:** combining B-Spline with Radial Basis Function
- **KAN-SGAN, Kformers, Deep-KAN, GraphKAN [6]**….

Playing with some of them…

RESULTS ON INVERSE MODELING – ALTERNATIVE KAN

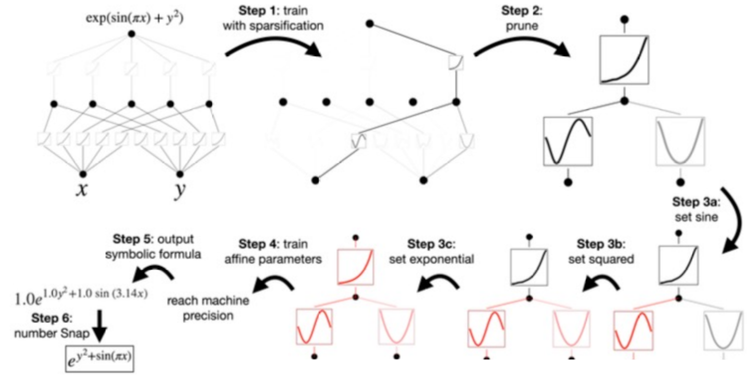




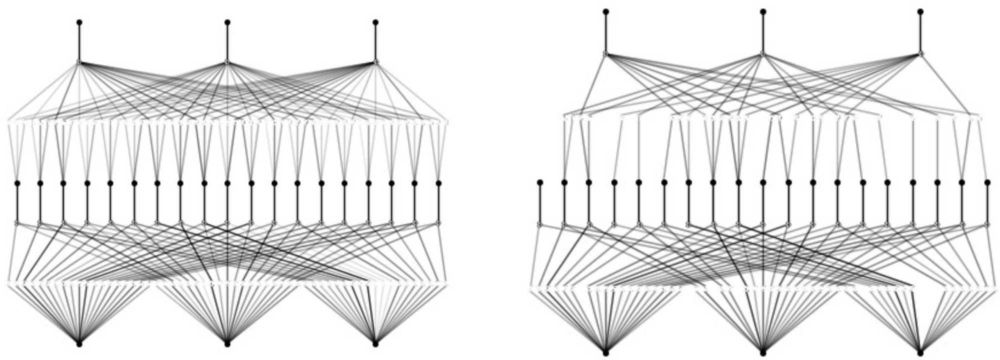
INTERPRETABILITY

KANs promise to be more interpretable than MLPs

- **Sparsification**: KAN can be trained with sparsification using L1 norm and entropy regularization
- **Pruning**: each node has a score on computation contribution. Pruning those under a certain threshold makes KAN more interpretablee
- **Symbolification**: a set of symbolic functions (i.e. sin, cos, exp) are provided in the case some activations are attributable to them.



We tried to pruning also our newtwork



Pre-pruning KAN

Post-pruning KAN

CONCLUSION AND FUTURE WORKS

This project aims to explore if and how KANs can outperform MLPs in a robot static problem

- Experiments refused the hypothesis
- More KANs have been proposed and there is room to explore

This approach can be exetended to other areas like:

- Robot Dynamics
- Multi-segment robots
- Using real robot data
- Include presence of external forces
- Explore model composition (i.e. KANs + Reinforcement Learning, GraphNN + KAN) to exploit the interpretability features of the network.

REFERENCES

[1]: **KAN: Kolmogorov-Arnold Networks** - (https://arxiv.org/abs/2404.19756)

[2]: **KAN 2.0: Kolmogorov-Arnold Networks Meet Science -** (https://arxiv.org/abs/2408.10205)

[3]: **An Efficient Implementation of Kolmogorov-Arnold Network -** (https://github.com/Blealtan/efficient-kan)

[4]: **FastKAN: Very Fast Kolmogorov-Arnold Network via Radial Basis Functions -** (https://github.com/ZiyaoLi/fast-kan)

[5]: **BSRBF-KAN: A combination of B-splines and Radial Basis Functions in Kolmogorov-Arnold Networks** (https://arxiv.org/abs/2406.11173)

[6]: **Awesome KAN(Kolmogorov-Arnold Network) -** (https://github.com/mintisan/awesome-kan?tab=readme-ov-file)

[7]: **KAN or MLP: A Fairer Comparison** (https://arxiv.org/pdf/2407.16674)

THANK YOU