Machine Learning

Homework I 2024/2025

Daniele Sabatini

matricola:

1890300

Contents

1 Introduction

The goal of this homework is to create a model for compute the Forward Kinematic of 3 different Robots: 2R, 3R and 5R. All my code was write on Google colab and I use Pytorch framework

2 Robot 2R

2.1 100K Samples

For solve the 2R's task at begin what I have done is consider the dataset with 100k samples. The dataset was created using the code present in this link, in particular I create a dataset using `seed = 1000` and `num_samples = 100000`, the file generated has 10 colums correspond `j0` and `j1` which are the angles of the robot's arms `cos(j0), sin(j0), cos(j1), sin(j1)` which the corresponding *sin* and *cos* of angles `j0` and `j1` and the remain 4 colums are `ft_x, ft_y, ft_qw, ft_qz` which represent respectively the position and the orientation of finger-tip of robot. I save them on google drive in order to use it to google colab.

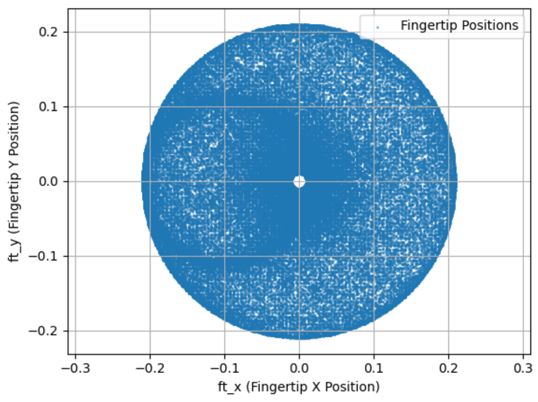


Figure 1: Plot of positions of finger-tip

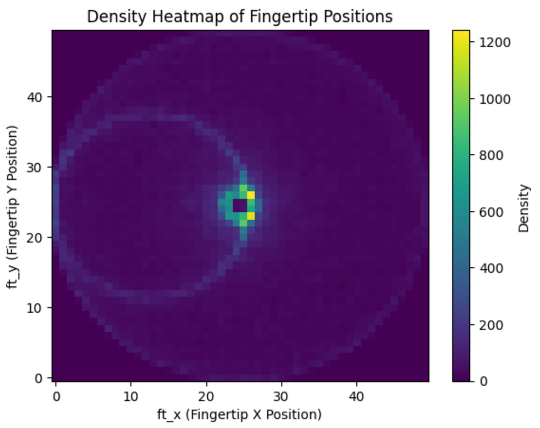


Figure 2: Frequency of positions of finger-tip

Figure 3: Exploration of 100K datsate

Before start to train models I explore the dataset in particular I plot the position of end-effector and the frequency of positions of end-effector as we can see in Figure 3, from the dataset plots, we can observe that the points are very close to each other, this sug-

gests that models like KNN, which rely on point-to-point distance, are likely to perform well. However, this is not necessarily because they generalize well, but rather due to the inherent structure of the dataset itself.
To perform the train of model, I decided to consider from the dataset only the robot's angles (`j0, j1` and `j2`) and the fingertip position (`ft_x` and `ft_y`).
After I diveded the dataset in train, validation and test in particular 60K samples for traing 20K for validation and 20K for test, I done this using `train_test_split` of `sklearn` library.
The model that I use are: Linear Regression with `MultiOutputRegressor` from `sklearn` library because we have multiple output, Decision Tree, Support Vector Regressor, Random Forest, KNN regressor and Neural Networks.
Except for Linear Regression for the other model I perform grid search in particular:

- Decision Tree: `'splitter': ['best', 'random']`, `'max_depth': [5, 10, 20]`
- SVR: `'kernel': ['linear', 'rbf']`, `'C': [0.1, 1, 10]`, `'epsilon': [0.01, 0.1, 1]`
- Random Forest: `'n_estimators': [100, 150]`,`'max_depth': [5, 12]`
- KNN: `'n_neighbors': [3, 5, 10]`, `'weights': ['uniform', 'distance']`, `'p': [1, 2]`

For perform a Hyperparameter search of Neural Network I define it in this way:

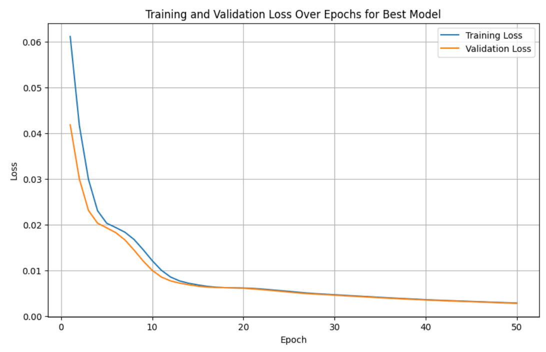
```python
class NeuralNetwork(nn.Module):
    def __init__(self, hidden_sizes):
        super(NeuralNetwork, self).__init__()
        self.layers = nn.ModuleList()
        # Input layer
        input_size = 2  # two input features: j0 and j1
        self.layers.append(nn.Linear(input_size, hidden_sizes[0]))
        # Hidden layers
        for i in range(1, len(hidden_sizes)):
            self.layers.append(nn.Linear(hidden_sizes[i-1],
    hidden_sizes[i]))
        # Output layer
        output_size = 2  # output: ft_x and ft_y
        self.layers.append(nn.Linear(hidden_sizes[-1], output_size))
    def forward(self, x):
        for layer in self.layers[:-1]:
            x = torch.relu(layer(x))
        x = self.layers[-1](x)
        return x
```

Listing 1: Definition of Neural Network

Thanks to the for loop between the input layer and the output layer, I was able to create a parameter grid where, by varying different hidden sizes and hidden layers, I could test which network structure performed best. Additional hyperparameters I used included the learning rate `[0.01, 0.001]` and different optimizers such as `Adam`, `SGD` and `RMSprop` and also I use different Hidden layer and Hidden size `[[8, 16, 8], [8, 16, 32, 16, 8], [16, 32, 64, 32, 16]]`. As shown, I chose `ReLU` as the activation function for the network and `MSELoss` as the loss function.

Model	**RMSE**	**R^2**
Linear Regression	0.0898	0.0674
Decision Tree	0.0022	0.9994
SVR	0.0198	0.9520
Random Forest	0.0053	0.9968
KNN	0.0010	0.9999
NN	0.0192	0.9571

(a) Performance metrics of different model for 100K dataset for 2R robot

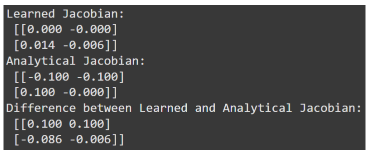


(b) Train and Validation loss of Neural Network during the training

Figure 4: Table and plot for the 100K dataset

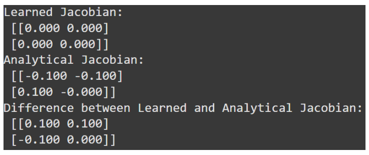
As we can see in Table [4a] the best model is KNN but as we already say this high value is due the fact the position are very close each other, execept for linear regression, all the models achieved very high performance, which can be attributed to the fact that the task in this case is not particularly challenging, and we also have a large number of samples for training.

Another method to verify which model performs best is to calculate the Jacobian matrix and compare it with the analytical one. For calculating the Jacobian matrix using the trained models, I used the functions provided by the professor (adapted for the PyTorch framework). For the analytical Jacobian, I first calculated the direct kinematics of a 2R robot and then computed the partial derivatives.



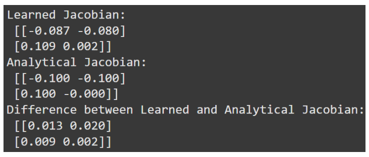
```
Learned Jacobian:
 [[0.000 -0.000]
 [0.014 -0.006]]
Analytical Jacobian:
 [[-0.100 -0.100]
 [0.100 -0.000]]
Difference between Learned and Analytical Jacobian:
 [[0.100 0.100]
 [-0.086 -0.006]]
```

(a) Linear regression



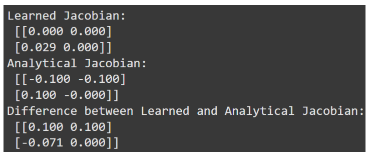
```
Learned Jacobian:
 [[0.000 0.000]
 [0.000 0.000]]
Analytical Jacobian:
 [[-0.100 -0.100]
 [0.100 -0.000]]
Difference between Learned and Analytical Jacobian:
 [[0.100 0.100]
 [-0.100 0.000]]
```

(b) Decision Tree



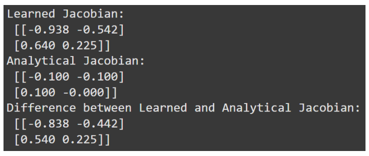
```
Learned Jacobian:
 [[-0.087 -0.080]
 [0.109 0.002]]
Analytical Jacobian:
 [[-0.100 -0.100]
 [0.100 -0.000]]
Difference between Learned and Analytical Jacobian:
 [[0.013 0.020]
 [0.009 0.002]]
```

(c) SVR



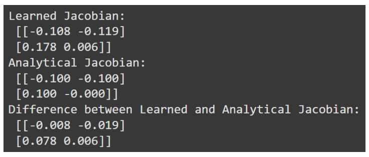
```
Learned Jacobian:
 [[0.000 0.000]
 [0.029 0.000]]
Analytical Jacobian:
 [[-0.100 -0.100]
 [0.100 -0.000]]
Difference between Learned and Analytical Jacobian:
 [[0.100 0.100]
 [-0.071 0.000]]
```

(d) Random Forest



```
Learned Jacobian:
 [[-0.938 -0.542]
 [0.640 0.225]]
Analytical Jacobian:
 [[-0.100 -0.100]
 [0.100 -0.000]]
Difference between Learned and Analytical Jacobian:
 [[-0.838 -0.442]
 [0.540 0.225]]
```

(e) KNN



```
Learned Jacobian:
 [[-0.108 -0.119]
 [0.178 0.006]]
Analytical Jacobian:
 [[-0.100 -0.100]
 [0.100 -0.000]]
Difference between Learned and Analytical Jacobian:
 [[-0.008 -0.019]
 [0.078 0.006]]
```

(f) NN

Figure 5: Comparison of Analytic Jacobian with Learned Jacobian

As we can see from Figure [5] model that during train and test phase perform very well (e.g. Decision Tree) when they are used to compute the Jacobian matrix, their performance deteriorates. This could be because the predicted output of the model might be close to the real value on average (indicated by a good RMSE), but the local behavior (variations with respect to the inputs) might not be consistent with reality. This results in a Jacobian that deviates from the analytical one.

Since the models that computed the Jacobian most accurately are SVR and NN, I decided to perform a more detailed analysis. Using the `.sample()` function, I randomly selected 100 points from the test dataset and computed 100 Jacobians. I then evaluated their differences using the Frobenius norm. The results are shown in the figure [8][11].

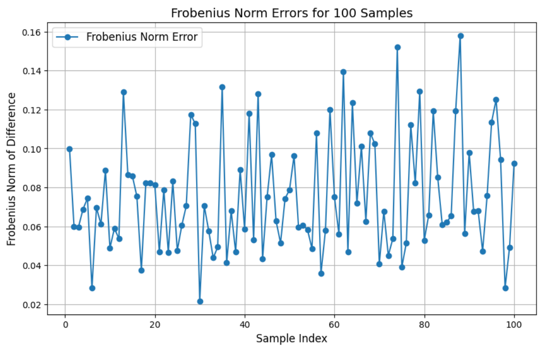


Figure 6: Frobenius Norm Errors for 100 Samples

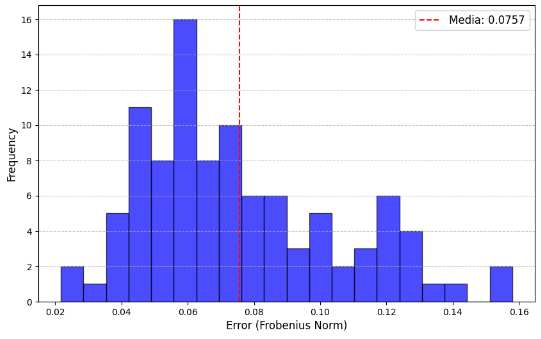


Figure 7: Error Distribution in Frobenius Norm: A histogram visualizing the frequency of errors across 100 samples the red dashed line indicating the mean error

Figure 8: Neural Network

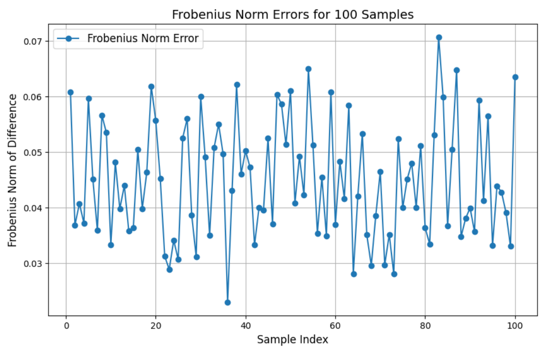


Figure 9: Frobenius Norm Errors for 100 Samples

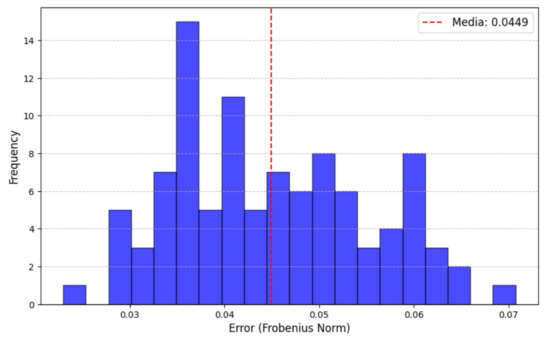
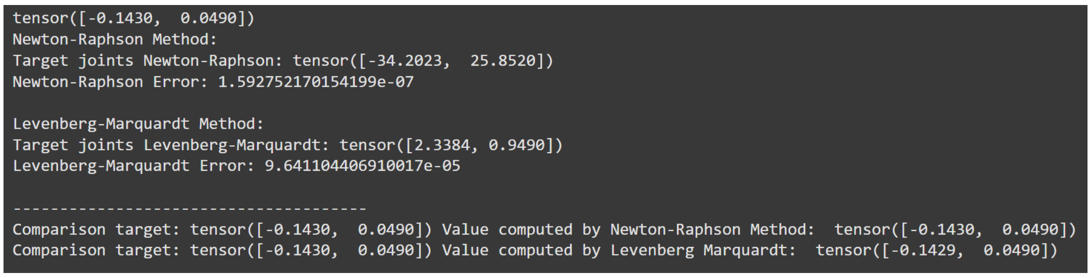


Figure 10: Error Distribution in Frobenius Norm: A histogram visualizing the frequency of errors across 100 samples the red dashed line indicating the mean error

Figure 11: Support Vector Regressor

An optional task in the homework is to compute the inverse kinematics. To achieve this, I used the models that best computed the Jacobian, namely NN and SVR.
The algorithms employed for the inverse kinematics calculation are Newton-Raphson and Levenberg-Marquardt. Specifically, I initialized the robot's configuration at $(0, \pi/4)$ and sampled a target fingertip position from the test set. Then, I used both algorithms to determine which one better reached the target position. The results are shown in the figure [12][13]

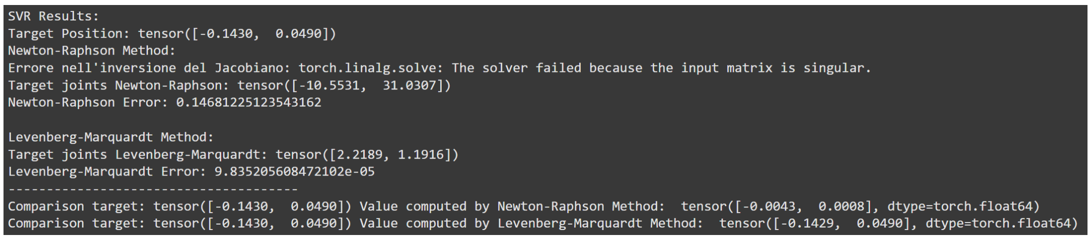


```
tensor([-0.1430,  0.0490])
Newton-Raphson Method:
Target joints Newton-Raphson: tensor([-34.2023,  25.8520])
Newton-Raphson Error: 1.592752170154199e-07

Levenberg-Marquardt Method:
Target joints Levenberg-Marquardt: tensor([2.3384, 0.9490])
Levenberg-Marquardt Error: 9.6411044069100l7e-05

-------------------------------------
Comparison target: tensor([-0.1430,  0.0490]) Value computed by Newton-Raphson Method:  tensor([-0.1430,  0.0490])
Comparison target: tensor([-0.1430,  0.0490]) Value computed by Levenberg Marquardt:  tensor([-0.1429,  0.0490])
```

Figure 12: Inverse Kinematics using Neaural Network as model



```
SVR Results:
Target Position: tensor([-0.1430,  0.0490])
Newton-Raphson Method:
Errore nell'inversione del Jacobiano: torch.linalg.solve: The solver failed because the input matrix is singular.
Target joints Newton-Raphson: tensor([-10.5531,  31.0307])
Newton-Raphson Error: 0.14681225123543162

Levenberg-Marquardt Method:
Target joints Levenberg-Marquardt: tensor([2.2189, 1.1916])
Levenberg-Marquardt Error: 9.835205608472102e-05
-----------------------------------
Comparison target: tensor([-0.1430,  0.0490]) Value computed by Newton-Raphson Method:  tensor([-0.0043,  0.0008], dtype=torch.float64)
Comparison target: tensor([-0.1430,  0.0490]) Value computed by Levenberg-Marquardt Method:  tensor([-0.1429,  0.0490], dtype=torch.float64)
```

Figure 13: Inverse Kinematics using Support Vector Regressor as model

For both models and I use the same parameter in particular for Newton-Raphson `max_iters=10000, tolerance=1e-4` and for Levenberg-Marquardt `max_iters=10000, tolerance=1e-4, lambda_=0.05`

To conclude, I would like to add that, as evident from the results, the task appears to be particularly simple. In fact, except for Linear Regression and KNN, all models achieve excellent performance. Among them, SVR stands out as particularly effective.
A possible explanation for this behavior could be that NN tends to capture global relationships in the data but may fail to accurately represent local behavior if not sufficiently regularized. In contrast, SVR is designed to minimize errors within a defined tolerance (ϵ), making it more stable in capturing local variations between inputs and outputs.
This approach allows SVR to better approximate derivatives (Jacobian), as it focuses on a precise representation of local behavior rather than on global adaptation to the data.

2.2 Train Test separated

As I mentioned in the previous paragraph, the models were trained using a dataset created from a single file that was split into train, test, and validation sets. Since the robot is a 2R robot and the number of samples is very large, it is possible that the fingertip positions in the test set are extremely close to those in the training set.
For this reason, I decided to generate two separate files (using different seeds): one file for training and validation and a second file exclusively for testing. Both files contain 100k samples. As we can see from the results [1], similar to the previous case, KNN achieved

Model	**RMSE**	**R^2**
Linear Regression	0.0891	0.0696
Decision Tree	0.0018	0.9996
SVR	0.0186	0.9584
Random Forest	0.0054	0.9965
KNN	0.0008	0.9999
NN	0.0418	0.7913

Table 1: Performance metrics of models: RMSE and R^2 scores.

excellent performance due to the data points being very close to each other. However, the performance of the NN model has decreased, especially the R^2 score. Another reason for the drop in the R^2 score is that, in this case, we have 100K test samples instead of 20K as in the previous scenario.

2.3 10K and 1K samples

As demonstrated by the previous results, the models with the 100k dataset perform well in solving the problem. I decided to attempt the same task with fewer samples, first with 10k and finally with 1k, to observe how the performance changes with smaller datasets. In order to obtain the 10K and 1K datasets, I started with the 100K dataset and randomly removed samples until the dataset reached the desired number of samples.

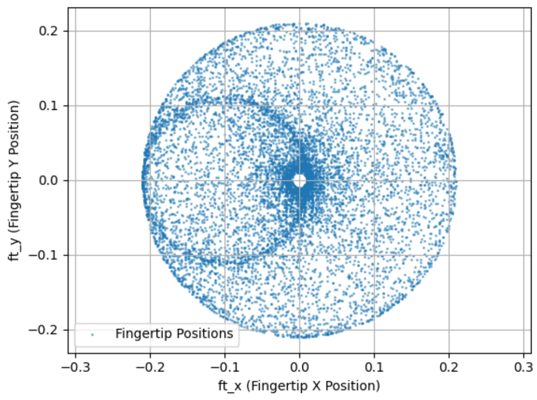


Figure 14: Plot of positions of finger-tip for 10K

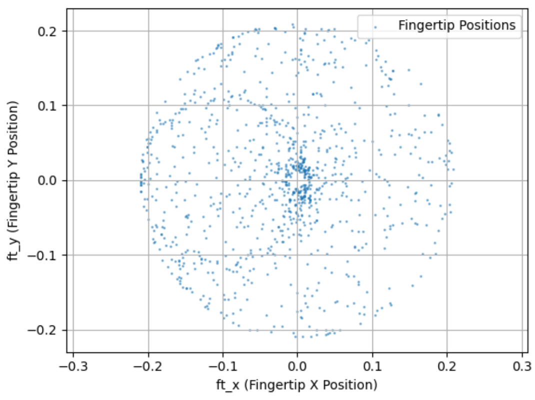


Figure 15: Plot of positions of finger-tip for 1K

Figure 16: Exploration of 10K and 1K dataset

As with the 100K dataset, the first step for the 10K and 1K datasets was to plot the positions and some information about the datasets figure [25][26]. Also in this case, I considered only `j0, j1, ft_x` and `ft_y`, and split the dataset into train, validation, and test sets, as done for the 100K dataset. Afterward, I trained the models, and the results are reported in the table [3]

Model	**RMSE**	$\mathbf{R^2}$
Linear Regression	0.0903	0.0754
Decision Tree	0.0069	0.9945
SVR	0.0213	0.9453
Random Forest	0.0066	0.9948
KNN	0.0029	0.9990
NN	0.0329	0.8770

Table 2: Results of models with 10K dataset

Model	**RMSE**	$\mathbf{R^2}$
Linear Regression	0.0901	0.0568
Decision Tree	0.0199	0.9536
SVR	0.0156	0.9715
Random Forest	0.0138	0.9776
KNN	0.0088	0.9907
NN	0.0374	0.8366

Table 3: Results of models with 1K dataset

As shown by the data in the tables, reducing the number of samples also decreases the performance of the models. In particular, this reduction in data significantly affected

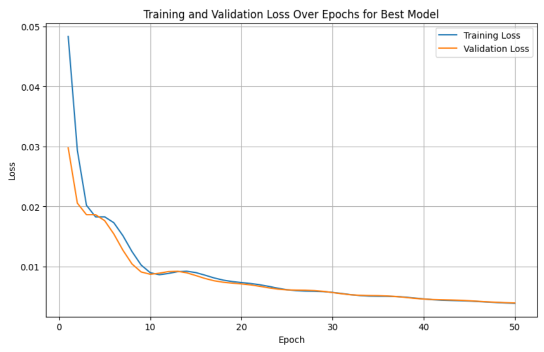


Figure 17: Train and Validation loss 10K dataset

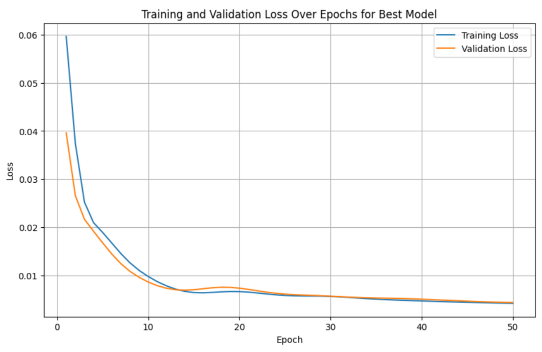


Figure 18: Plot of positions of finger-tip for 1K

the neural network. This is because, although neural networks have a strong capacity for generalization and data representation, they require a large amount of data to achieve optimal performance.

For the models trained on the 10K and 1K datasets, I also computed the Jacobian and compared it with the analytical one. Once again, the best results were achieved by the SVR and Neural Network models, as shown in the images below

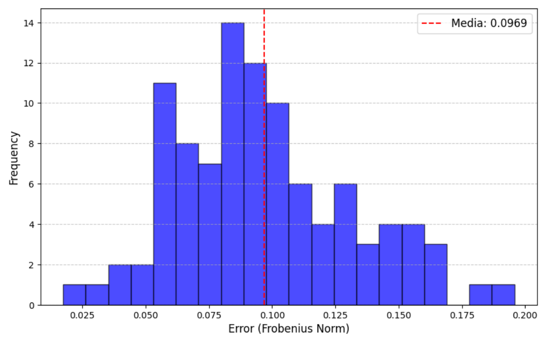


Figure 19: model Neural Network 10K dataset

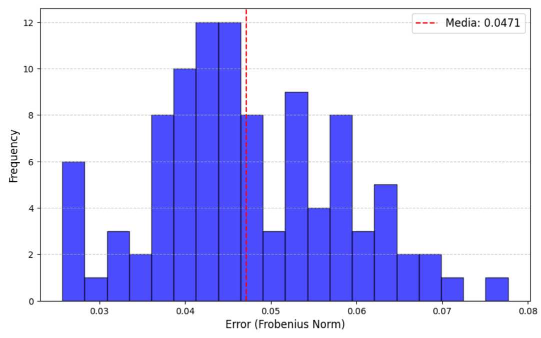


Figure 20: model SVR 10K dataset

Figure 21: Error Distribution in Frobe- nius Norm: A histogram visualizing the frequency of errors across 100 samples the red dashed line indicating the mean error

In conclusion, I also calculated the Inverse Kinematics using the SVR and Neural Network models and the same methods applied to the models trained on the 100K dataset. One notable difference is that, for the models trained on the 10K dataset, the Newton-Raphson method achieved better results compared to the Levenberg-Marquardt method.

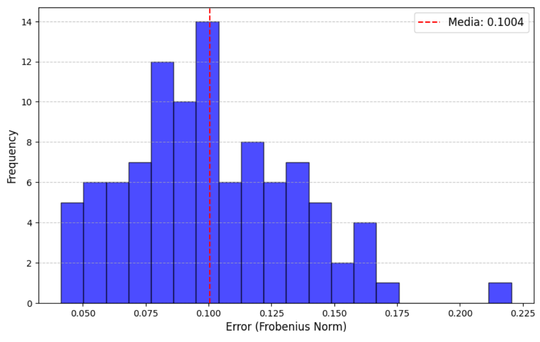


Figure 22: model Neural Network 1K dataset

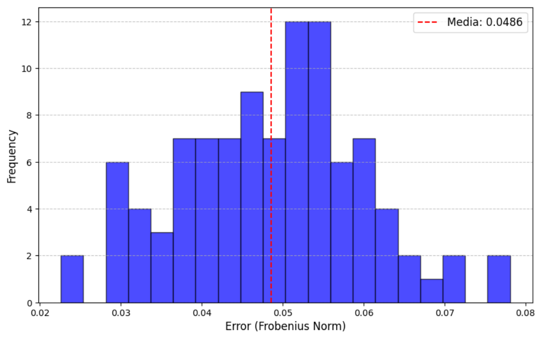


Figure 23: model SVR 1K dataset

Figure 24: Error Distribution in Frobenius Norm: A histogram visualizing the frequency of errors across 100 samples the red dashed line indicating the mean error

2.4 Final Consideration

In conclusion, I would like to add that solving the task with a 100K dataset appears to be particularly easy for the models, especially for Neural Networks and SVR. One characteristic I observed is that, as the number of samples decreases, what significantly improves the performance of Neural Networks is not only the width and depth of the network but also the number of training epochs, which plays a crucial role.

Through the analysis of Jacobians, I found that, regardless of the number of samples used to train the model, SVR consistently performed the best.

Future studies could focus on improvements in the calculation of inverse kinematics. In my implementation, I sampled a single point from the dataset and randomly selected the initial position. Since these methods are sensitive to initial positions, the results I obtained may not be entirely generalizable. A possible solution could involve using various target positions and initial configurations to ensure more robust and generalized results.

3 Robot 3R

3.1 100K samples

In this case as well, the dataset was generated using the code provided by the professor. To create the datasets for training, testing, and validation, only the samples related to the robot's angles (j0, j1, j2) and the fingertip positions (ft_x, ft_y) were extracted.

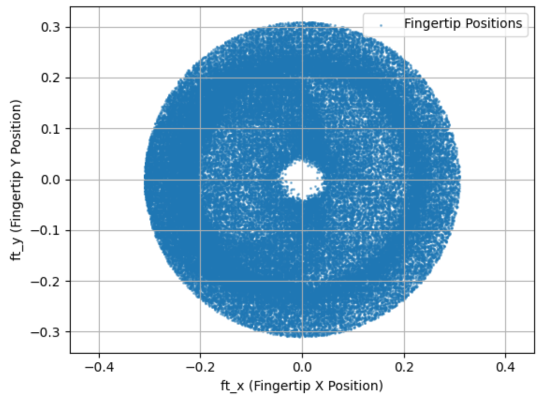


Figure 25: Plot of positions of finger-tip

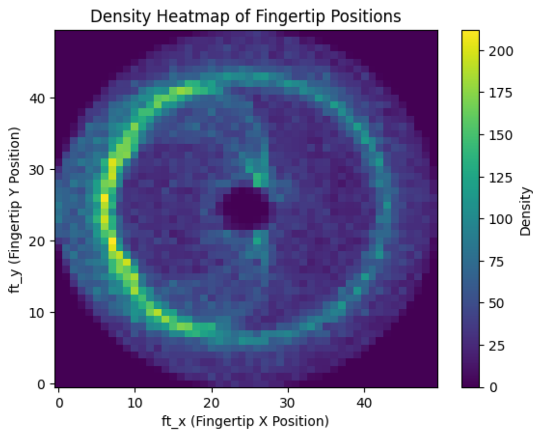


Figure 26: Frequency of positions of finger-tip

Figure 27: Exploration of 100K dataset of 3R robot

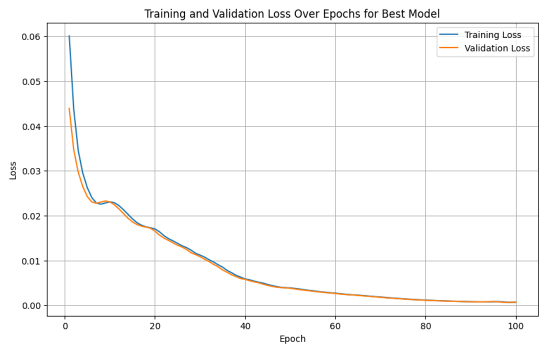
As we can see from the images, the points are very close to each other in this case as well. Consequently, the KNN model is expected to achieve particularly high performance. The models used are the same as those employed for the 2R robot: Linear Regression, Support Vector Regressor, Random Forest, and Neural Network. However, the hyper parameters for the grid search were adjusted:

- Decision Tree Regressor: `'splitter': ['best', 'random']`,`'max_depth': [5, 10, 20]`
- Support Vector Regressor: `'kernel': ['linear', 'rbf']`, `'C': [0.1, 1, 10]`,`'epsilon': [0.01, 0.1, 1]`
- Random Forest: `'n_estimators': [100, 150]`,`'max_depth': [10, 15]`
- KNN: `'n_neighbors': [3, 5, 10]`, `'weights': ['uniform', 'distance']` `'p': [1, 2]`

Going into more detail about the Neural Network, it was defined in the same way as for the 2R robot. However, in this case, I adjusted the hyperparameters and the number of training epochs, specifically:`'learning_rate': [0.01, 0.001]`,`'hidden_sizes': [[8, 16, 32, 16, 8], [16, 32, 64, 32, 16], [32, 64, 80, 64, 32]]`,`'optimizer': ['Adam', 'SGD']` and I train it for 100 epochs.
The reason for this change lies in the increased complexity of the task. Since this robot has an additional arm compared to the previous one, making the task more challenging, a more extensive exploration of hyper parameters was necessary to achieve optimal results.

Model	**RMSE**	**R^2**
Linear Regression	0.14639	0.09123
Decision Tree	0.01334	0.99246
SVR	0.03350	0.95191
Random Forest	0.00963	0.99607
KNN	0.00518	0.99886
NN	0.01738	0.98708

(a) Performance metrics of different models for 100K dataset for 3R robot



(b) Train and Validation loss of Neural Network during the training

Figure 28: Table and plot for the 100K dataset

As we can see from the results [28], the R2 and RMSE values are very similar to those obtained for the 2R robot. This demonstrates that adjusting the hyperparameter values for a different grid search and increasing the number of epochs, despite the more complex task, improves the overall performance of the models.

In this case as well, another method to determine the best overall model was to evaluate the Jacobians calculated by the models and compare them with the corresponding analytical Jacobian. Initially, I calculated a single Jacobian for each model to identify which one produced the Jacobian most similar to the analytical one. From this preliminary analysis, as in the 2R task, SVR and NN emerged as the best models for the 3R task as well. To more precisely assess how closely the models' Jacobians resembled the analytical Jacobian, I sampled 100 points from the test dataset and evaluated the 100 Jacobians against their analytical counterparts using the Frobenius norm as show in figure [31]

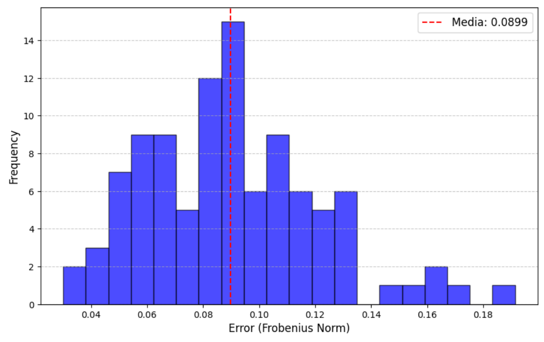


Figure 29: Neural Network with dataset 100K

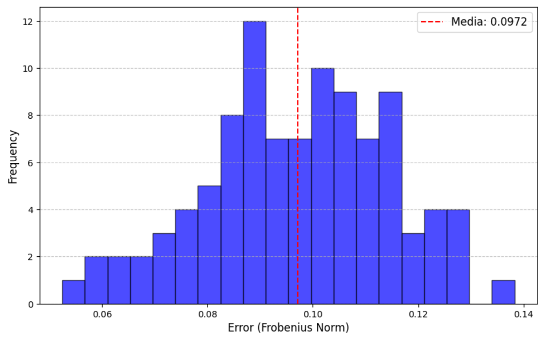


Figure 30: SVR with dataset 100K

Figure 31: Error Distribution in Frobenius Norm: A histogram visualizing the frequency of errors across 100 samples the red dashed line indicating the mean error

From the histograms [31], two key observations can be made. First, even for this task, the SVR model outperformed the Neural Network. Second, both models achieved worse performance compared to the 2R task. This confirms the increased difficulty the models encountered in solving this task.
The last task I performed was the optional point of calculating the Inverse Kinematics. As in the previous case, I used the Newton-Raphson and Levenberg-Marquardt methods. For both methods, I set the initial configuration to $(0, \pi/4, 0)$ and chose a target position randomly sampled from the test dataset. One difference compared to the 2R case was the number of `max_iterations`, which I increased. Without this adjustment, both methods struggled to accurately determine the target position for that specific initial configuration.

3.2 10K and 1K samples

In this case as well, the results show that the models with 100K samples achieved excellent performance. Since the 2R robot task demonstrated that reducing the number of samples leads to a general decline in performance for all models, I decided to conduct a further analysis for the 3R task using datasets with 10K and 1K samples.
As with the 2R robot, for the 3R robot, I used the same 100K dataset and randomly removed samples until the desired sizes were achieved.

The models used were the same as in the 100K case, with identical hyperparameters and the same number of iterations for training (in the case of SVR and NN), allowing

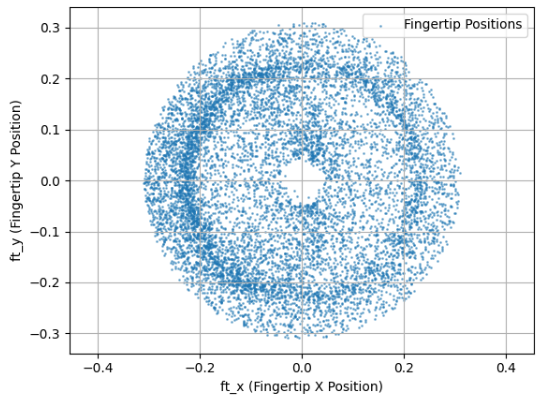


Figure 32: Plot of positions of finger-tip for 10K

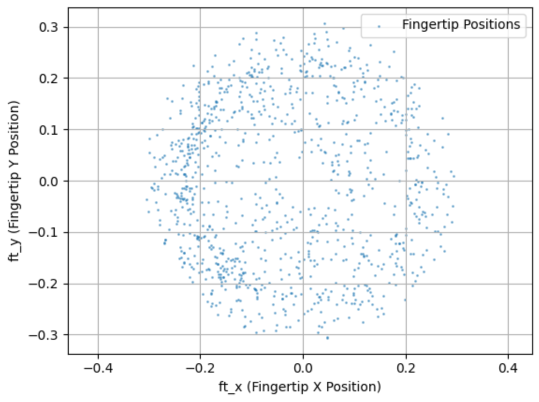


Figure 33: Plot of positions of finger-tip for 1K

Figure 34: Exploration of 10K and 1K dataset

for a direct comparison with the 100K dataset. In this case as well, as shown by the results [5], the general performance of the models worsens as the number of samples decreases. However, one characteristic we can observe is that, unlike the 2R robot, the Neural Network achieved good performance despite the reduction in the dataset size

Model	**RMSE**	$\mathbf{R^2}$
Linear Regression	0.14711	0.09747
Decision Tree	0.02794	0.96741
SVR	0.02831	0.96657
Random Forest	0.01570	0.98969
KNN	0.01181	0.99414
NN	0.02140	0.98089

Table 4: Results of models with 10K dataset

Model	**RMSE**	$\mathbf{R^2}$
Linear Regression	0.14569	0.08016
Decision Tree	0.05541	0.86588
SVR	0.02478	0.97263
Random Forest	0.03775	0.93819
KNN	0.02617	0.97031
NN	0.01911	0.98412

Table 5: Results of models with 1K dataset

Further confirmation of the good performance achieved by the Neural Network comes from the test conducted with the Jacobians, where the Neural Network achieved a lower error compared to the SVR model figure [42].

In this case as well, for the models trained on the 10K and 1K datasets, I performed inverse kinematics starting from the same initial configuration used for the models trained on the 100K dataset and employed the same models.

For the inverse kinematics calculation, in the case of models trained on the 10K dataset, the Newton method worked better for SVR, while the Levenberg-Marquardt method performed better for the Neural Network. On the other hand, for the models trained on the 1K dataset, the Levenberg-Marquardt method worked better for both

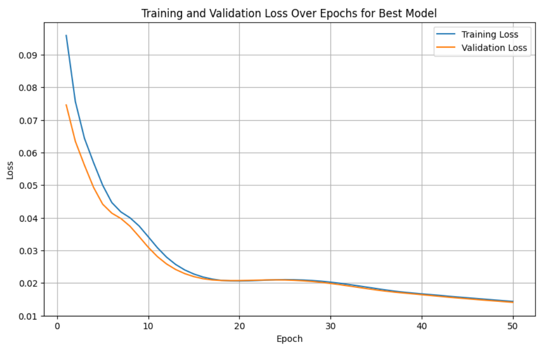


Figure 35: 10K

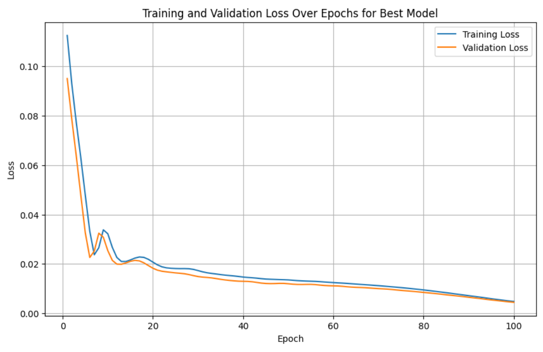


Figure 36: 1K

Figure 37: train and Validation loss

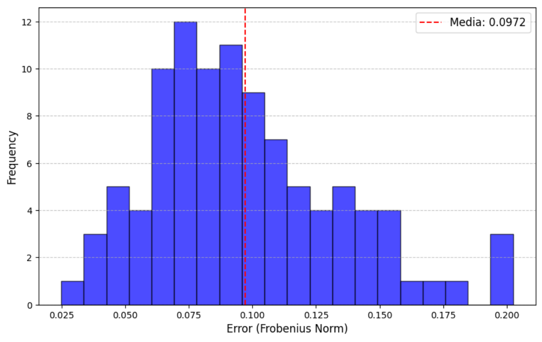


Figure 38: model Neural Network 10k dataset

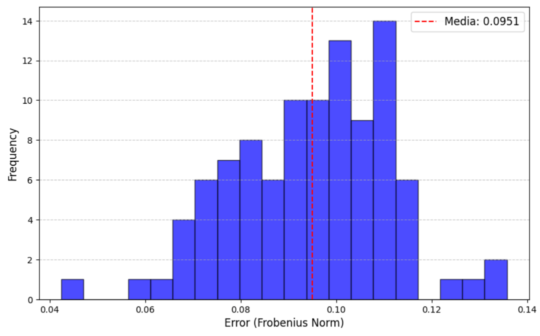


Figure 39: model SVR 10k dataset

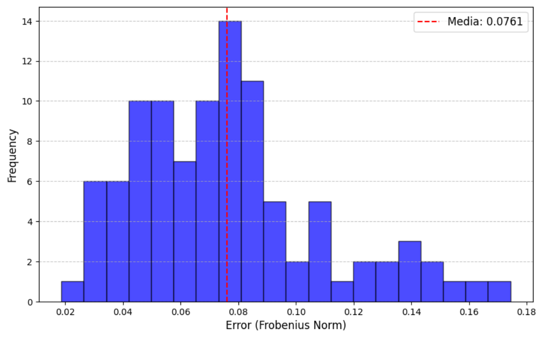


Figure 40: model Neural Network 10k dataset

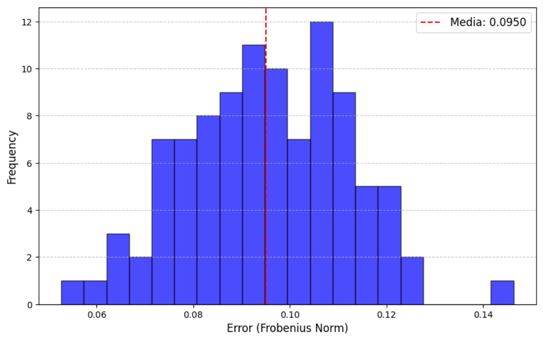


Figure 41: model SVR 1k dataset

Figure 42: Error Distribution in Frobe- nius Norm: A histogram visualizing the frequency of errors across 100 samples the red dashed line indicating the mean error

3.3 Final Consideration

In conclusion, the 3R task proved to be more challenging for the models to solve, as shown by the results. A key difference compared to the 2R robot was the performance of the Neural Network, which achieved excellent results when using the 1K dataset. This could be due to the fact that PyTorch's nn.Module initializes the network weights randomly (sampling from a uniform distribution). In this case, it is possible that the weights were initialized in a way that allowed the network to learn particularly efficiently from just 1K samples.

Further studies that could be applied to the 3R robot task include testing various initial configurations and target positions for the inverse kinematics calculations, providing more data to understand how well the models can solve this task. Another area of study could involve exploring additional hyperparameters to determine whether performance improvements can be achieved, especially for the 10K and 1K datasets.

4 Robot 5R

4.1 100K samples

The final task of the homework involves designing models capable of calculating the fingertip position given an initial configuration of the 5R robot. Unlike the previous tasks, where the robots operated in a 2D environment, this robot operates in a 3D space.

This dataset was also generated using the code provided by the professor.

As I did for the 2R and 3R robots, I began by exploring the dataset. However, due to the limited range of dataset values, the large number of samples, and the fact that the robot operates in a three-dimensional environment, the plot of the fingertip positions appears as a cloud of points (Figure [43]).

For the creation of the training, testing, and validation datasets, I considered only the angular positions of the joints (`j0, j1, j2, j3, j4`) and the final positions of the fingertip (`ft_x, ft_y, ft_z`).

An additional difference compared to the 2R and 3R robots was the application of `StandardScaler()` to test whether it could improve performance.

The models used were the same as in the previous tasks. However, given the com-

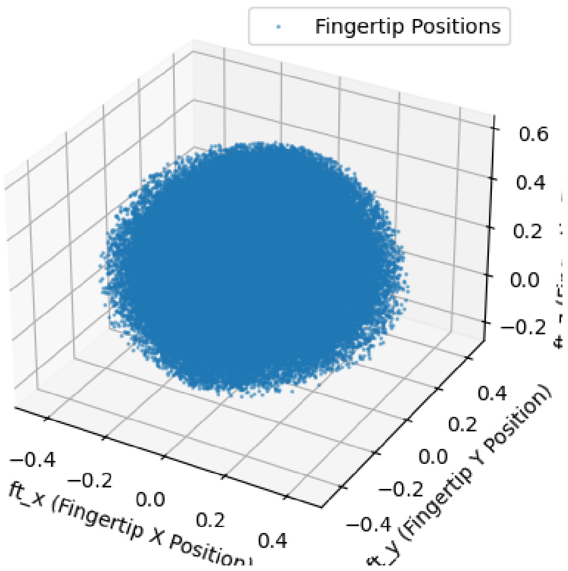


Figure 43: 3D Plot of Fingertip Positions

plexity of this task, I decided to introduce new hyperparameters to further improve performance. Specifically:

- Decision Tree: `'splitter': ['best', 'random']`, `'max_depth': [20, 25, 30]`
- SVR: `'kernel': ['linear', 'rbf']`, `'C': [0.1, 1, 10]`, `'epsilon': [0.01, 0.1, 1]`
- Random Forest: `'n_estimators': [100, 150]`, `'max_depth': [12, 15]`
- KNN: `'n_neighbors': [3, 5, 10]`, `'weights': ['uniform', 'distance']`, `'p': [1, 2]`

The Neural Network (defined as in the previous tasks) also had new hyperparameters: `'learning_rate': [0.01, 0.001]`, `'hidden_sizes': [[16, 32, 64, 32, 16], [32, 64, 80, 64, 32], [16, 32, 64, 128, 64, 32, 16]]`, `'optimizer': ['Adam', 'SGD']`. Additionally, I increased the number of training epochs compared to the 2R and 3R tasks, setting `epochs=100`.

As we can see from the results in Tables [6][7], the models trained on the 100K dataset achieve excellent results in solving the task. One notable observation is that standardization, in this case, did not bring any benefit to the model's performance.

Since the structure of the 5R robot is more complex than that of the 2R and 3R robots, I was unable to calculate the analytical Jacobian matrix for the 5R. As a result, I could

Model	RMSE	R^2
Linear Regression	0.16564	0.11512
Decision Tree	0.04514	0.93342
SVR	0.03159	0.96546
Random Forest	0.02379	0.97986
KNN	0.01861	0.98862
NN	0.02341	0.98213

Table 6: Results of models without standardization dataset

Model	RMSE	R^2
Linear Regression	0.16577	0.11369
Decision Tree	0.04523	0.93314
SVR	0.03697	0.95226
Random Forest	0.02379	0.97985
KNN	0.01861	0.98863
NN	0.02484	0.97944

Table 7: Results of models with standardized dataset

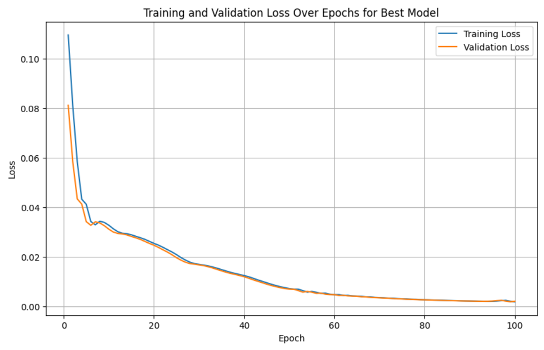


Figure 44: Train and Validation loss with No standardization

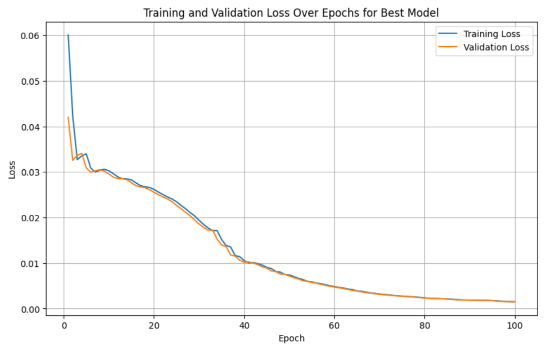


Figure 45: Plot of positions of finger-tip for 1K

Figure 46: Train and validation loss with standardization

not compare the Jacobians computed by the models with their analytical counterparts. I considered an alternative approach to verify the correctness of the Jacobians by using them to compute the inverse kinematics and checking whether the methods (Levenberg-Marquardt and Newton-Raphson) could approximate the target position. However, this approach cannot be considered a rigorous evaluation method. Comparing with the analytical Jacobian allows for a direct assessment of the accuracy of the Jacobian estimated by the model relative to the analytical one, using metrics such as the Frobenius norm. This provides a quantitative and objective measure of the model's ability to approximate local variations between inputs and outputs. In contrast, evaluating through inverse kinematics does not directly assess the quality of the Jacobian but rather the overall effectiveness of the optimization method (Newton or Levenberg-Marquardt). The accuracy of the Jacobian is only one of many factors influencing the result.

4.2 1K samples

For the 5R robot, I also decided to perform a comparison with models trained on a reduced dataset, specifically with 1K samples. The dataset, as in the case of the 2R and 3R robots, was obtained by starting from the 100K dataset and randomly removing samples until reaching the desired size. In this case, the features considered are the same as those in the 100K dataset (`j0, j1, j2, j3, j4, ft_x, ft_y`, and `ft_z`). Since the fingertip positions are always in a three-dimensional space, it remains challenging to plot them [47].

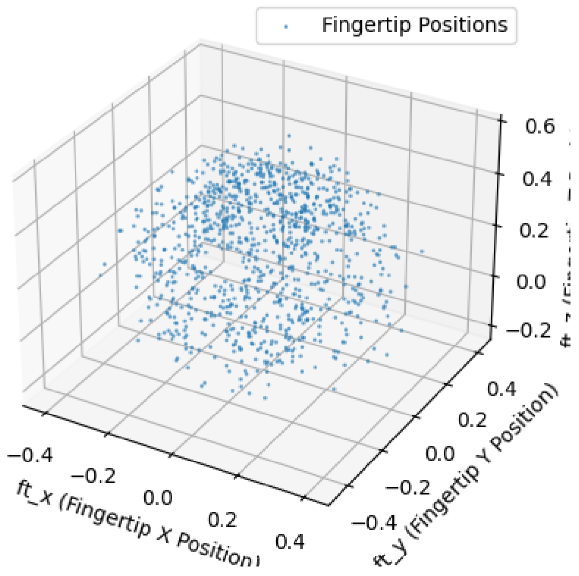


Figure 47: Positions of the fingertip

The models used are the same as those for the 100K dataset, with identical hyperparameters, allowing for a direct comparison. For this analysis, I used both a dataset with standardized values (using `StandardScaler()`) and one without standardization to evaluate whether it brought any improvements.

Model	RMSE	R^2
Linear Regression	0.16721	0.08932
Decision Tree	0.11989	0.51921
SVR	0.03789	0.94978
Random Forest	0.07548	0.80431
KNN	0.06545	0.86065
NN	0.02666	0.97528

Table 8: Results of models with 1K dataset without standardization

Model	RMSE	R^2
Linear Regression	0.16758	0.08501
Decision Tree	0.12182	0.50408
SVR	0.03445	0.95670
Random Forest	0.07546	0.80437
KNN	0.06533	0.86117
NN	0.03085	0.96805

Table 9: Results of models with 1K dataset with standardization

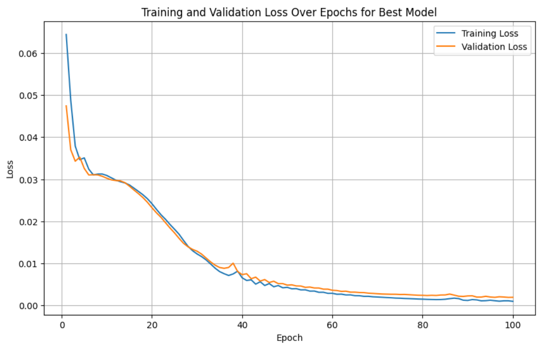


Figure 48: Train and Validation loss with No standardization

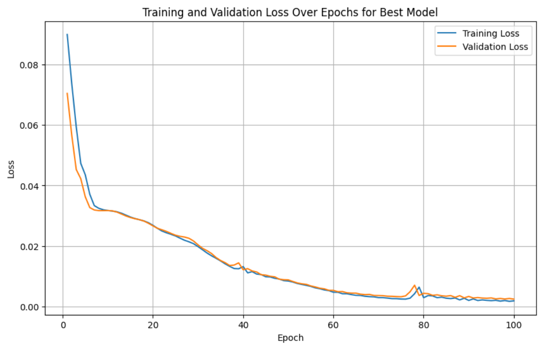


Figure 49: Plot of positions of finger-tip for 1K

Figure 50: Train and validation loss with standardization

As with the 2R and 3R robots, the overall performance of the models deteriorated for the 5R robot as well. One observation we can derive from the tables [9] is that, similar to the 100K dataset, standardization does not improve model performance in this case either. Subsequently, I calculated the Jacobians from the trained models. However, for the same reasons described in the previous section, I was unable to evaluate their accuracy

5 Final Consideration

In conclusion, I would like to add some final considerations. From the results of the models trained with the 100K dataset [7][6] and the 1K dataset [8][9], we observed that standardization did not improve performance. This can be explained by the fact that models like Decision Tree and Random Forest partition the data space into regions based on decision thresholds, so the scale of the features does not affect performance. Similarly, for models based on ordinal distances, if the model uses only relative positions or order relationships, standardization does not alter the results. Furthermore, in this dataset, the features have similar ranges, so standardization offers no significant advantage.

A potential future development of this work could involve calculating the analytical Jacobian matrix and performing a more in-depth comparison with the Jacobians computed by the models.

Additionally, as explained in Section 100, the inverse kinematics was calculated starting from a single initial configuration, and the target position was sampled from the test dataset. Since the methods are sensitive to the initial configuration, a more consistent comparison could be achieved by using multiple initial configurations.

Finally, to improve performance, it would be beneficial to explore a wider range of hyperparameters and conduct larger grid searches to find more precise model configurations.

6 Note

Additional images can be found in the `.ipynb` files, which include the code used to train and test the models.
I would also like to specify that when creating the datasets with fewer samples, I mentioned that samples were randomly removed until the desired size was reached. This did not create unbalanced datasets because the same number of samples was removed from each feature, albeit randomly.