First Homework Report on Robot Forward Kinematics

Giuseppe Misuraca, 1944376
misuraca.1944376@studenti.uniroma1.it

Academic year 2024-25

University of Rome "La Sapienza"

Abstract

This assignment addresses the problem of computing the forward kinematics of robots using machine learning models, focusing on three different robot configurations: a 2-joint planar, a 3-joint planar and a 5-joint 3D robot. The goal is to predict the position and orientation of the robot's end-effector based on the joint angles. Three different datasets are used, each corresponding to one of the robot configurations. This report presents the methodology employed to achieve optimal performance in the resolution of forward kinematics through the utilization of diverse machine learning models.

1 Dataset

To solve the given task, three separate datasets were created, each corresponding to a specific robot configuration: R2, R3 and R5. In particular, the datasets were generated by executing the provided programs from the given Dockerfile. This involved launching a simulation for each robot, whereby the robots assumed different joint angles, thus resulting in different positions and orientations of the end-effectors. In order to ensure a representative sample, the number of configurations assumed by each robot was set to 100000. Incorporating a greater variety of data allows models to generalise more effectively and reduces the likelihood of overfitting. To assess the model's performance under different conditions, an experiment was conducted utilising both a comprehensive dataset and a smaller subset comprising 1000 samples. Furthermore, to enhance the diversity of the analysis, the training was conducted on a specific dataset and the testing was performed on a distinct dataset generated with an alternative seed. This approach, based on testing with diverse distribution samples, offers insights into the model's capacity to generalize beyond specific configurations, ensuring that the model does not merely memorize the training data but learns the underlying patterns essential for accurate predictions. Consequently, this approach ensures robustness in the training process. It should additionally be noted that, in each task, a data split into training and testing sets was employed, whereby 70% of the data was allocated for training and the remaining 30% for testing.
To facilitate the work, the datasets were uploaded to Google Drive, which allowed for easy integration within a Google Colab environment. In particular, a basic mount function in Google Colab was employed to access the data stored on the drive, which streamlined the process of importing and utilizing the datasets for model training.

1.1 Features Selection

The dataset for each task was characterized by the following features: joint angles and their values expressed through sine and cosine. Prior to initiating the analysis, a simple feature selection was computed, resulting in the decision to utilize a subset of the provided features. This decision was made to prioritize the raw input values, specifically the joint angles, as they are the most fundamental representation of the robotic configuration. It was thought that including extra features, like trigonometric functions of the joint angles, might make the learning process more complex without improving the model's predictions. These features, which are derived from the joint angles, lacked information, so by excluding them the input feature space was made simpler. This reduction in dimensionality often has several benefits, including faster training times. During the training phase, the models are capable of learning complex transformations, including trigonometric functions. This enables them to deduce these relationships without the need for explicit feature engineering, by providing the raw joint angles as input.

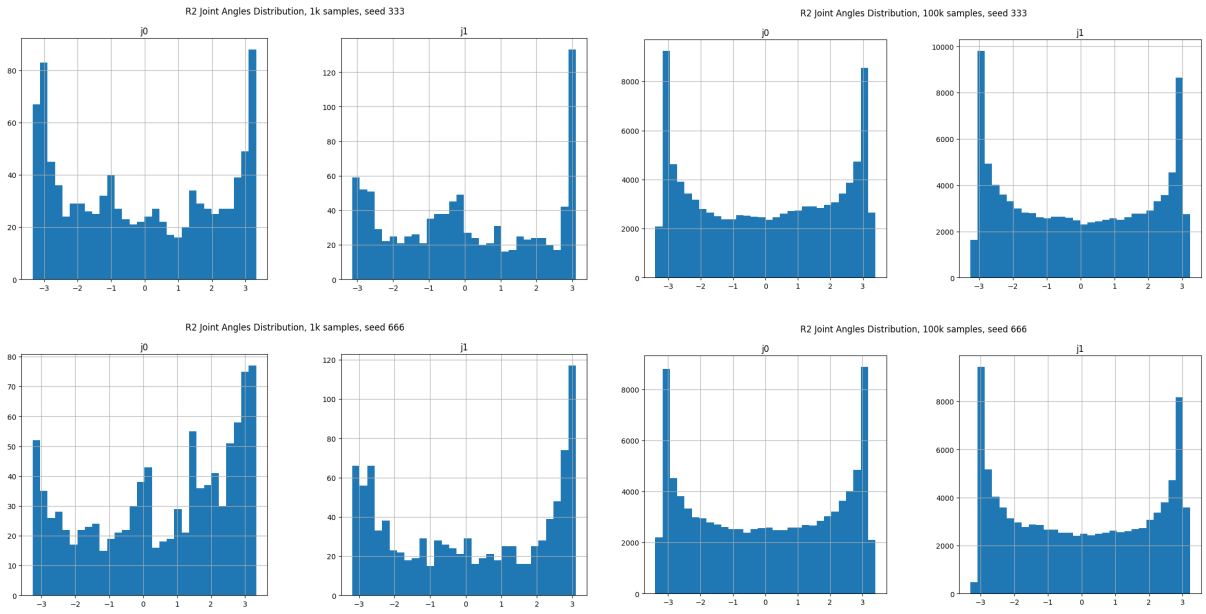

Figure 1: Comparison of data distributions for two dataset lengths and two random seeds.

1.2 Normalization Steps

Another significant design decision was to exclude the normalization of the joint angles. The normalization is employed to scale input features to a comparable range, which can facilitate the convergence of certain models. However, in this specific instance, normalization was not required since the joint angles are already within a defined numerical range. Moreover, given that the model will be trained and tested on analogous data sets (robot configurations), the inclusion of normalization would not notably enhance the model's performance. Consequently, the omission of this step permitted the creation of a more straightforward preprocessing pipeline and the avoidance of unnecessary transformations.

2 Jacobian Evaluation

The task involves also estimating the Jacobian matrix using the trained direct kinematics model. The evaluation of the Jacobian is a crucial step in the process of validating the performance and reliability of the model. By comparing the estimated Jacobian with the analytical Jacobian (derived from the robot's kinematic equations), it is possible to assess the accuracy and physical consistency of the model.
The importance of the Jacobian matrix cannot be overstated. It provides a fundamental link between joint space and Cartesian space, describing how infinitesimal changes in joint angles affect the end-effector's position and orientation. Consequently, it is crucial to obtain low differences between the two. The learned Jacobian was computed in two ways, depending on the model used. Either it was computed numerically by differentiation, or it was obtained by utilizing the gradients provided by the neural network during training.

2.1 Jacobian calculation using gradients

A dedicated function was supplied for the purpose of calculating the Jacobian through the use of TensorFlow's GradientTape. This function computes the Jacobian matrix by recording the operations on the input during the forward pass and then differentiating the output with respect to the input.
The function in question was modified and subsequently extended to accommodate the R5 configuration, which involves five joint angles as input. This was necessary because the R5 task, as well as the R2, was solved using neural network models.

2.2 Jacobian calculation using numerical differentiation

As will be explained in subsequent sections, the R3 problem was solved by implementing an Adaboost model. The limitation of the AdaBoost model is that it is not based on gradients. Therefore, the standard gradient-based approach to compute the Jacobian was not applicable in this case. Instead, the learned Jacobian for the R3 task was calculated using numerical differentiation (1). This approach involves calculating the difference between the output of the forward kinematics function with slightly perturbed inputs (added eps to the input) and dividing them by the double of the perturbing element. This approach works, but is generally less precise than the gradient-based approach. This is because the accuracy of the numerical differentiation depends also on the choice of the perturbation value (eps).

$$f'(x) \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon} \tag{1}$$

2.3 Jacobian comparison

This section of the report delineates the methodology employed to ascertain the congruence between the Jacobian matrix derived from the machine learning model and the analytically derived Jacobian matrix. The approach commences with the selection of a subset of random test samples from the dataset. Each sample is associated with a specific set of joint angles and is employed to calculate both the learned and analytical Jacobians. The Jacobians are computed in accordance with the methodology outlined in the preceding paragraphs. Subsequently, only the rows corresponding to the end-effector position (omitting orientation) were extracted from both Jacobians. The degree of similarity between the two matrices is quantified using the Frobenius distance, which is computed by taking the square root of the sum of the squared differences between corresponding elements of the two matrices (2). This was calculated for each pair of Jacobians generated by a sample, and to obtain a single score, the mean of that distance was calculated for all samples under consideration. This metric provides an intuitive sense of the discrepancy between the learned and analytical Jacobians, with smaller distances indicating higher accuracy. Additionally, the mean and standard deviation of these distances were printed over the test samples to provide a comprehensive view of the model's consistency.

$$d_F(A, B) = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(A_{ij} - B_{ij})^2} \tag{2}$$

3 R2 Task

To address the first task, a prediction model leveraging a deep ensemble learning approach was developed, specifically designed to transform input parameters into accurate positional and orientational predictions. The decision was made to obtain a model that could achieve high performance on small datasets while effectively leveraging the complementary strengths of multiple models to enhance predictive accuracy. This ensemble-based approach ensures robustness and precision, even when training data is limited. Notably, it demonstrated strong performance even on subsets as small as 1000 samples, highlighting its ability to maintain reliability and efficiency under the constraints of data scarcity.

3.1 Network Architecture

The foundational neural network model, on which the deep ensemble is based, was implemented as a shallow neural network in order to achieve an optimal balance between computational efficiency and learning capacity. The input layer consists of two nodes, representing the two joint angle measurements. A single hidden layer, that employs the Rectified Linear Unit (ReLU) activation function, has 64 nodes, chosen as the optimal architecture through empirical testing. The output layer comprises four nodes, representing the description of the robotic end-effector, both position and orientation. The estimated number of parameters for this network is derived by summing the parameters of each layer. The first dense layer has an input size of 2 and an output size of 64. Therefore, it has 2 x 64 = 128 weights and 64 biases, resulting in a total of 192 parameters. The second dense layer has an input size of 64 and an output size of 4, leading to 64 x 4 = 256 weights and 4 biases, for a total of 260 parameters. The total number of trainable parameters in the network is the sum of parameters from both dense layers: 192 + 260 = 452.

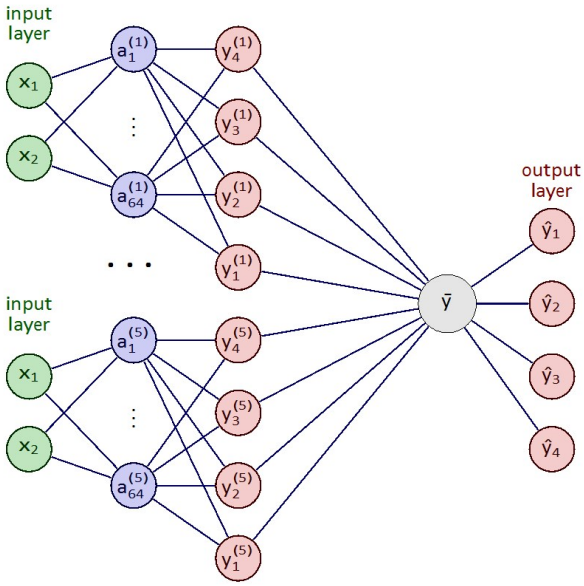

Figure 2: Overview of the Deep Ensembles that solves the R2 task

3.2 Training Strategy

The model's training strategy is based on an ensemble learning approach. By instantiating multiple neural network models, specifically five independent instances, the methodology mitigates individual model limitations and reduces predictive variance. Each model is initialized with a unique random seed, resulting in diverse weight configurations. This diversity enables the ensemble to explore a broader solution space and mitigates the risk of overfitting or convergence to suboptimal solutions. In order to optimize the performance of the models that comprise the ensemble, a five fold grid search cross-validation strategy is employed. This process systematically splits the training data into five subsets, using four for training and one for validation in each iteration. In addition, a different seed is used for each k-fold cross-validation to further increase the diversity. The mean squared error (MSE) was selected as the evaluation metric and the Adam optimizer was employed for training due to its efficient parameter convergence.

The exploration of hyperparameters in this study focused exclusively on the number of epochs, batch size, and learning rate. This choice was due to their significant impact on the outcome and the limited resources available for further exploration. In particular, for the model trained on 1000 samples a combination of 12 hyperparameters was explored. The hyperparameter space explored in this study included the following: the number of training epochs (with values of 150 and 300), the batch size (with values of 16, 32, and 64), and the optimizer's learning rate (with values of 0.001 and 0.01). The use of a fivefold cross-validation approach resulted in a total of 60 training instances per model. Considering that the deep ensemble comprises five base models, it was necessary to train the method 300 times in order to achieve optimal performance.
The following outcomes were derived from the cross-validation phase of the model trained on 1000 samples:

- Model 1 Best: 0.000121 using {`batch_size: 16, epochs: 300, learning_rate': 0.001`}
- Model 2 Best: 0.000114 using {`batch_size: 16, epochs: 300, learning_rate': 0.001`}
- Model 3 Best: 0.000130 using {`batch_size: 16, epochs: 300, learning_rate': 0.001`}
- Model 4 Best: 0.000114 using {`batch_size: 16, epochs: 300, learning_rate': 0.001`}
- Model 5 Best: 0.000106 using {`batch_size: 16, epochs: 300, learning_rate': 0.001`}

It can be observed that the various models yield comparable scores, which will consequently result in a similar prediction score when computed in two distinct ways (simple mean and weighted mean).

Due to the considerable computational expense associated with training five distinct models on 100,000 samples across numerous configurations, a comprehensive grid search could not be conducted for the model trained on the whole dataset. The five networks that constitute the deep ensemble were trained on the empirical identified optimal parameters, which were as follows: 50 epochs, 16 batch size and a learning rate of 0.001. As with the previous models, within the deep ensembles the resulting performances were found to be analogous (MSE: ~ 0.000020).

3.3 Prediction Aggregation

Two complementary prediction aggregation methodologies were implemented to enhance predictive accuracy. The first approach employs a simple arithmetic mean, aggregating predictions from all ensemble members by averaging their outputs. This method is straightforward and effectively reduces noise by leveraging the ensemble's collective intelligence. A more sophisticated methodology employs a weighted mean prediction, wherein the individual contributions of the models are scaled dynamically based on their cross-validation performance. This weighted approach prioritizes predictions from more accurate models, thereby introducing a bias towards instances of high-performing models.
Despite the degree of diversity introduced to train the ensembles models as independently as possible, the performances obtained during the various cross validations, as seen, were similar. Even the best values of the hyperparameters yielded identical results. Consequently, the discrepancy between the score attained through the utilization of the weighted mean and that achieved through the application of the simple mean was insignificant. The observed similarity in performance across the different cross-validation runs, as well as the identical results obtained using both methods, suggests that the ensemble models have converged to similar solutions.

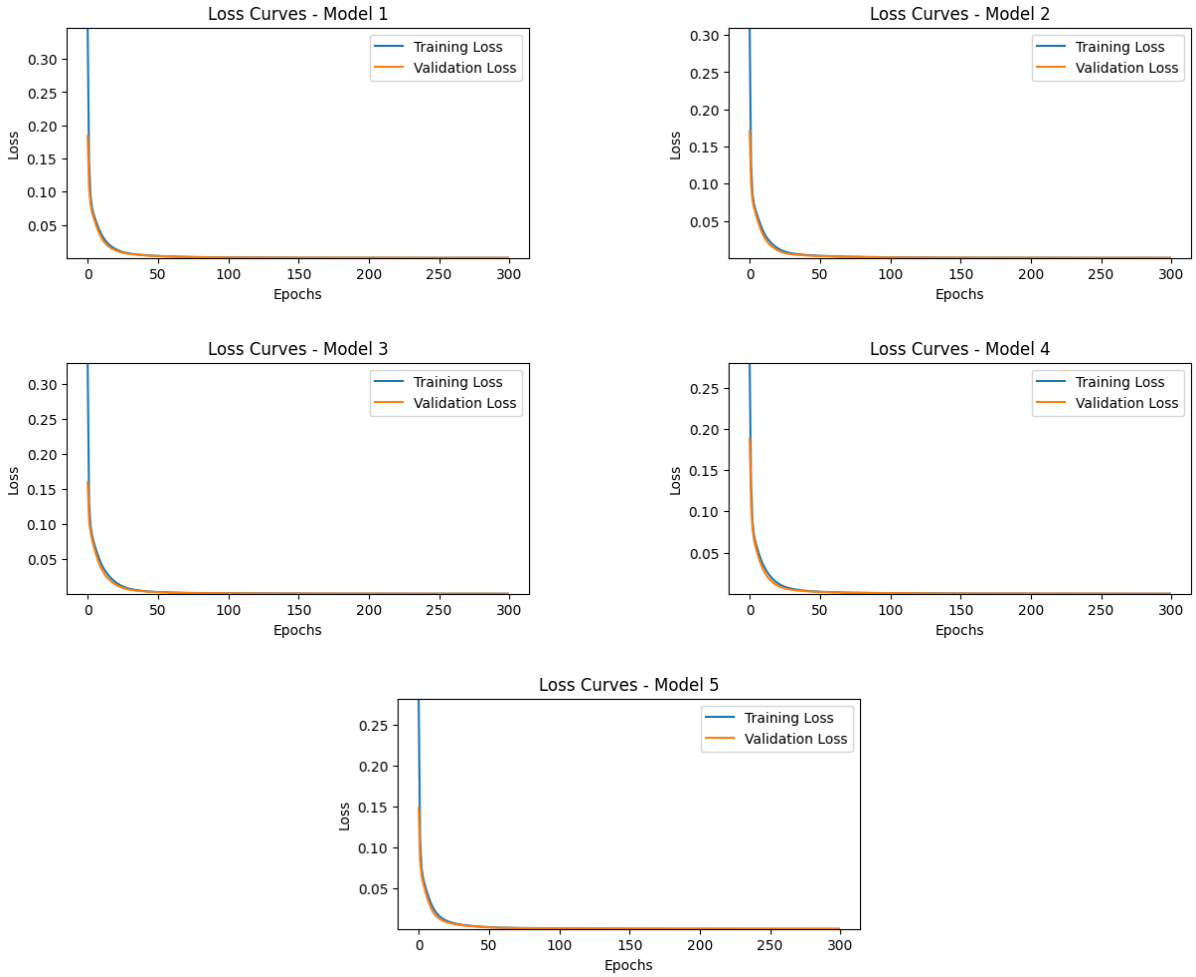

Figure 3: Evolution of losses for the five models in the deep ensemble, trained on 1000 samples.

3.4 Testing Results

The results obtained by this model are highly satisfactory, both with 1000 samples and with 100000 samples used for training. As previously stated, the performances of the models that constitute the deep ensembles during the grid search cross validation were remarkably similar. Consequently, the results obtained were identical when computed with the mean and the weighted prediction. In particular, the mean square error (MSE) measured during the testing phase was 0.00004 for the model trained on the subset of the dataset, compared to 0.000005 for the model trained on the full dataset. By leveraging a substantial number of samples, the model demonstrated a notable enhancement in its performance, exhibiting an increase of up to an order of magnitude.

3.5 Jacobian Comparison

In regard to the calculation of the Jacobian, the results were also highly satisfactory. The outcomes presented below were obtained through the methodologies described in the preceding section on a random subset of 50 samples belonging to the testing set.
In particular, the model that was trained on a restricted subset demonstrated a mean value of 0.0356 for the Frobenius distance, which is basically identical to the weighted mean value. The variance among the results was 0.0140.
The model trained on 100k samples achieved the following results. With weighted prediction, the mean Frobenius distance was 0.0234 and the Frobenius distance standard deviation was 0.0080. Using mean prediction, the mean Frobenius distance was 0.0229 and the Frobenius distance standard deviation was 0.0077.

Model	**Mean Frobenius Distance**		**Frobenius Distance Std. Dev.**	
	Simple	**Weighted**	**Simple**	**Weighted**
Trained on 1k	0.0357	0.0356	0.0142	0.0140
Trained on 100k	0.0229	0.0234	0.0077	0.0080

Table 1: Jacobian distance metrics for different models, R2 task.

4 R3 Task

In the second task, the objective was to predict the position and orientation of the end-effector for a robot with three degrees of freedom. The R3 robotic system prediction challenge was addressed through a classical ensemble learning approach, specifically utilizing AdaBoost regression with DecisionTreeRegressor base estimators. The decision to employ AdaBoost was motivated by the desire to explore an alternative approach to neural networks, which had already been utilized in the other task. This provided an opportunity to assess the performance of a different machine learning paradigm in addressing the prediction problem. By leveraging AdaBoost's ability to sequentially improve weak learners, insights were gained into how boosting could handle the complexity of robotic forward kinematics.

4.1 Model Description

The AdaBoost model utilizes a DecisionTreeRegressor as its base estimator, selected for its superior performance compared to alternative models evaluated in the project's initial phases. The boosting mechanism involves an iterative training process where subsequent estimators are trained to address the residual errors of preceding estimators, ultimately creating a robust and accurate aggregate model. To accommodate the multi-output regression task, the MultiOutputRegressor wrapper was employed, generating independent regressors for each output dimension and enabling AdaBoost to effectively handle multi-dimensional output spaces. This approach facilitates the model's ability to accurately predict both positional and orientational data concurrently.

4.2 Training Phase

A comprehensive grid search cross-validation strategy was employed to determine the optimal configuration of the AdaBoost Regressor model. The process started with the definition of a parameter grid, encompassing key hyperparameters such as the maximum depth of the base estimators, the number of boosting estimators and their learning rate. These hyperparameters were carefully selected as they govern critical aspects of the model: the complexity of the individual decision trees, the size of the ensemble and the contribution of each estimator to the overall prediction. The hyperparameter values investigated for the 1000 samples model included a maximum depth for the base estimator ranging from 8 to 14 in increments of 2, with a total of 250 and 500 estimators considered and a learning rate varied across 0.09, 0.3, and 0.9. In total, 24 combinations were thus explored. Once more, the number of possible combinations of hyperparameters for the model trained on the entire dataset was limited due to the high cost. Therefore, I only explored 4 combinations (max depth [12, 14], number of estimators [500], learning rate [0.3, 0.9]). To guarantee a reliable assessment of the model, a five-fold cross-validation methodology was employed. The GridSearchCV function was utilized to automate the process of hyperparameter tuning. The Mean Squared Error (MSE) was selected as the scoring metric to evaluate the predictive performance during cross-validation.
The grid search phase was concluded with the identification of the optimal parameters for the 1000 model: estimator max depth of 12, learning rate of 0.9 and estimator number of 500. This resulted in a mean squared error of 0.0049 in the training-validation phase. In contrast, the optimal parameters for the other model were an estimator max depth of 14, a learning rate of 0.9 and an estimator number of 500. With these parameters, an MSE of 0.00025 was achieved.

4.3 Testing Results

The results of the two models, trained on the 1000 subset and the full dataset respectively, reveal an expected but important trade-off between the size of the training data and the resulting predictive performance. The model trained on the subset, despite being significantly constrained by data availability, demonstrated a commendable performance with an MSE of 0.004041 on the test set. This highlights AdaBoost's ability to extract meaningful patterns even from limited data by iteratively refining weak learners. However, the model trained on the full dataset achieved a notably lower MSE of 0.000225, emphasizing the advantage of leveraging larger datasets in enhancing the generalization capabilities and overall accuracy of the ensemble. This analysis reinforces the importance of tailoring machine learning strategies to the available data. Ensemble methods like AdaBoost offer flexibility and effectiveness even with smaller datasets, but the inclusion of larger datasets unlocks their full potential, significantly improving the precision and reliability of predictions.

4.4 Jacobian Comparison

The Jacobian calculation for this task revealed markedly inferior performance outcomes in comparison to those attained in the previous task. As previously stated, the gradient method was not applicable, necessitating the approximation of the derivatives to calculate the Jacobian. This approach was identified as a contributing factor to the suboptimal performance. Another key factor contributing to the suboptimal performance in the calculation of the Jacobian is the limited size of the training dataset, comprising only 1000 samples. Despite being trained on the same length dataset, the deep ensembles for the R2 task exhibited enhanced performance in Jacobian estimation in comparison to the method mentioned before. This is attributed to the deep ensemble's capacity to capture intricate, non-linear relationships, thereby facilitating superior generalization even with restricted data. Although a small dataset inherently restricts a model's ability to generalise, the more powerful deep ensemble model was still able to create a closer approximation of the analytical Jacobian than the adaboosting method. It is important to note that the R2 task, with its lower dimensionality compared to the R3 task, presented a simpler modelling challenge. This inherent difficulty may have further contributed to the failure of adaboosting in approximating the analytical Jacobian with limited data.
The results of the comparison on the model trained on the subset are as follows: the mean Frobenius distance was found to be 0.338, with a standard deviation of the Frobenius distance of 0.22. In contrast, for the model trained on 100000 samples, the mean Frobenius distance was 0.077963, with a standard deviation of the Frobenius distance of 0.032074.

5 R5 Task

The third task involved predicting the three-dimensional position and orientation of the end-effector of a R5 robot, with the input consisting of the angles of its joints. Given the greater complexity of this task compared to the earlier ones, a more substantial model was constructed. In particular, a neural network model consisting of three wide layers was employed.

5.1 Network Structure

During the experimental phase, it was observed that the model exhibited marginally superior performance when the network structure was wider rather than deeper. One possible explanation for this behavior is that a wider architecture provides a larger capacity for learning representations across the feature space, which may better suit tasks involving high-dimensional outputs such as position and orientation. Deeper networks often necessitate a greater quantity of data to effectively learn hierarchical representations. In the event that the training data is limited, the networks may fail to converge to an optimal solution. Given the structured nature of the direct kinematics problem, the mapping from joint angles to the end-effector's position and orientation is relatively straightforward and does not necessitate the use of deeper models. A wider network is sufficient to capture the required relationships without relying on the hierarchical feature transformations typically facilitated by deeper architectures. The neural network has been designed to accept a five-dimensional feature vector, which represents the joint angles. Subsequently, the input flows through three hidden layers, each of which employs the ReLU activation function. The first hidden layer comprises 64 neurons, while the second contains 128. The third hidden layer then reduces the dimensionality back to 64 neurons, preparing the data for the output layer. The output layer includes seven neurons, which are mapped to the position coordinates and the quaternion components.

In order to ascertain the total number of parameters in the neural network, it was necessary to calculate the contributions of each layer separately. This was achieved by considering the connections between the units in adjacent layers and the biases associated with each unit. The input layer, which takes data with a shape of (5,), has no trainable parameters. The first dense layer has 64 units, with each of the 5 input features connected to all 64 units. This results in $5 \times 64 = 320$ weights. In addition, each unit has a bias term, contributing 64 biases. Therefore, the total number of parameters in this layer is $320 + 64 = 384$. The second dense layer contains 128 units. Each of the 64 units in the previous layer is connected to all 128 units, resulting in $64 \times 128 = 8192$ weights. Including 128 biases, the total number of parameters for this layer is $8192 + 128 = 8320$. The third dense layer reduces the number of units to 64. Each of the 128 units in the previous layer is connected to all 64 units, resulting in $128 \times 64 = 8192$ weights. Adding the 64 biases gives a total of $8192 + 64 = 8256$ parameters for this layer. The fourth and final dense layer outputs seven units. Each of the 64 units from the previous layer is connected to each of the 7 units, giving $64 \times 7 = 448$ weights. Including the 7 biases, the total for this layer is $448 + 7 = 455$. By summing the parameter totals from all the layers, the total number of trainable parameters in the model is calculated to be $384 + 8320 + 8256 + 455 = 17415$.

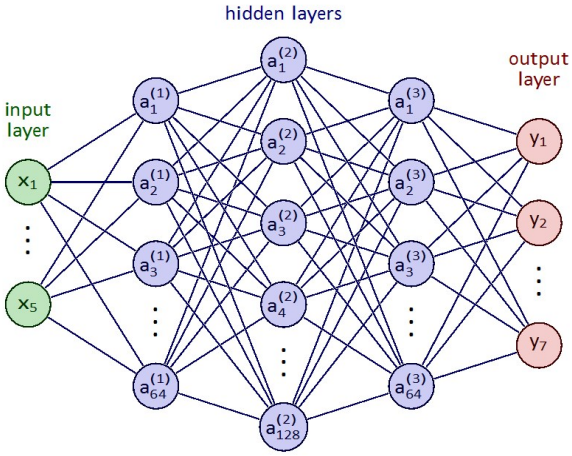

Figure 4: Overview of the Forward Neural Network that solves the R5 task

5.2 Layer Design Explored

During the model development process, experiments were conducted incorporating dropout and batch normalization layers with the objective of enhancing the network's generalization and training stability. However, contrary to initial expectations, these additional regularization techniques resulted in a reduction in model performance. The implementation of dropout layers with varying dropout rates between the hidden layers was conducted with the objective of preventing overfitting and enhancing the model's capacity for generalization. However, the experimental outcomes indicated a decline in performance: increase in prediction error. Batch normalization layers were integrated to normalize the inputs to each layer. Despite the widely reported benefits in various deep learning applications, the implementation demonstrated that the additional computational overhead did not correspond to a corresponding performance gain.

5.3 Model Training

As well as the other tasks, the search for optimal hyperparameters via grid search is also conducted here. In this context, the parameters that were examined during the grid search phase were epochs, batch size and optimizer learning rate. The faster training of the model on 1000 samples permitted the exploration of a wider set of combinations, specifically 18 combinations of the following hyperparameters: epochs (250, 500), batch size (16, 32, 64) and optimiser learning rate (0.01, 0.001, 0.0001). The optimal configuration was identified as a batch size of 16, an epoch number of 500 and a learning rate of 0.001, which yielded a mean squared error (MSE) of 0.000282 during the training phase. Training on a larger set (100000 samples) necessitates a considerable investment of time, which in practice limits the number of combinations of the parameters that can be explored. Specifically, the following ranges were considered: epochs (100), batch size (16, 32), and learning rate (0.001, 0.0001). The optimal configuration, as determined by minimizing the mean squared error (MSE), was found to be: batch size 16, epochs 100 and learning rate 0.0001. This configuration yielded an MSE of 0.000043 in the training batch.

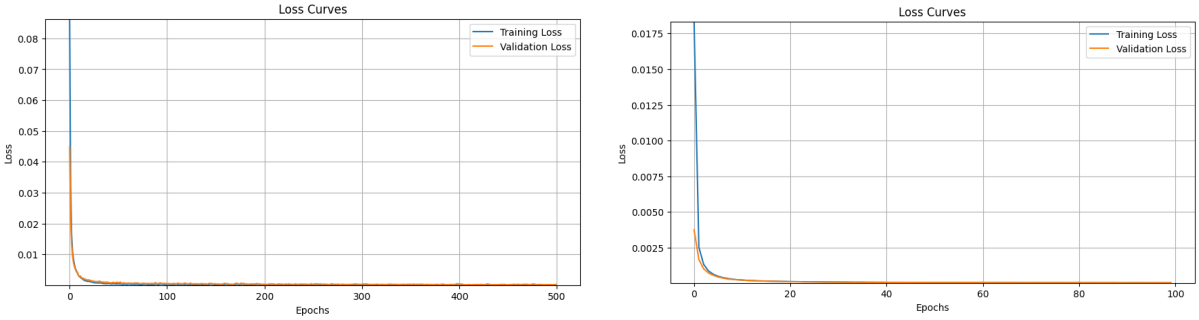

Figure 5: Evolution of loss for both models, respectively, for the subset and for the full dataset.

5.4 Testing Results

As observed in the preceding task, the results once again illustrate the clear benefit of training on a larger dataset. This is evidenced by the markedly lower mean square error (MSE) of 0.000028 recorded during the testing phase. In contrast, the model trained on the smaller subset attained a testing MSE of 0.000185. While the smaller model performed commendably given the limited data, the larger dataset allowed the network to capture more intricate patterns, resulting in superior accuracy.

5.5 Jacobian Comparison

In order to solve the third task, an ANN was employed, thus a model based on accumulating gradients was utilised. The computation of the learned Jacobian was conducted through the gradient method, as opposed to the numerical method employed in the preceding task. The comparison began with the retrieval of 50 random samples from the test set, upon which both the Jacobian were computed and subsequently compared using the Frobenius distance. The results of the comparison are as follows: for the model trained on a subset, the mean Frobenius distance was found to be 0.531923, with a standard deviation of 0.132849.

In contrast, for the model trained on 100000 samples, the mean Frobenius distance was determined to be 0.492613, with a standard deviation of 0.107868. The results indicate that the Frobenius distances for the Jacobian evaluation are higher, indicating a worse evaluation, in comparison to the estimation of the Jacobian for simpler robots, such as the lower robots (R2 and R3). This discrepancy is likely due to the increased complexity of the third task, which involves more intricate dynamics. The increased complexity makes the Jacobian estimation more challenging, resulting in higher errors when approximating it.

6 Conclusion

This report has demonstrated the application of machine learning models to solve the forward kinematics problem for three robotic configurations: a two-joint planar robot (R2), a three-joint planar robot (R3) and a five-joint three-dimensional robot (R5). By analyzing these diverse systems, the study explored the interplay between data complexity, model architecture and prediction performance. Through the use of neural networks, ensemble learning techniques and a systematic approach to hyperparameter tuning, the models achieved satisfactory results in predicting the position and orientation of the end-effectors for each robot configuration.

In the case of relatively simple systems such as the R2 robot, neural network models based on a deep ensemble structure have been shown to achieve excellent results even when trained on limited datasets. In contrast, for the R3 task, AdaBoost regressors represented an alternative to neural networks, demonstrating that boosting techniques can effectively address forward kinematics while providing competitive predictive performance. Notably, the research exposed significant methodological challenges inherent in high-dimensional robotic systems. The R5 task, being the most complex, required a more sophisticated neural network architecture with wide layers. The results confirmed the necessity of leveraging larger datasets for high-dimensional tasks, as the model trained on the full dataset significantly outperformed its counterpart trained on a smaller subset. However, the estimation of the Jacobian matrix, particularly for the R5 task, revealed some limitations in capturing intricate dynamics, thereby underscoring the challenge.

Looking forward, there are several avenues for extending this research. One promising direction is the exploration of more advanced neural network architectures, such as transformers or graph neural networks, which could be particularly well-suited for modelling the spatial relationships in high-dimensional robotic systems. These models may enhance the ability to capture non-linearities and complex dependencies more effectively than traditional feedforward networks. Furthermore, future work could delve deeper into improving the estimation of the Jacobian matrix. Physics-informed machine learning models (PIML), which explicitly incorporate the kinematic equations of the robots into the training process, could help bridge the gap between learned and analytical Jacobians.

In conclusion, this report serves as a foundation for applying machine learning to robotic kinematics, demonstrating both the potential and the challenges of this approach. By extending the methodologies explored here and incorporating advanced techniques, future research can further enhance the accuracy and generalizability of machine learning models in robotics.