EPiC Series in Computing

Volume 79, 2021, Pages 159–170

Proceedings of ISCA 34th International Conference on Computer Applications in Industry and Engineering

Exploiting Joint Dependencies for Data-driven Inverse Kinematics with Neural Networks for High-DOF Robot Arms*

Meng-Ting Tsai, Chung-Ta King, and Chi-Kai Ho

Department of Computer Science, National Tsing Hua University, Taiwan

b123009558@gmail.com, king@cs.nthu.edu.tw, chikaiho@gmail.com

Abstract

Solving inverse kinematics (IK) has been an important problem in the field of robotics. In recent years, the solutions based on neural networks (NNs) are popular to handle the non-linearity of IK. However, the complexity of IK grows rapidly as the degree-of-freedom (DOF) of the robot arms increases. To address this problem, we exploit the dependencies among the joints of the robot arms, based on the observation that the movements of certain joints of a robot arm will affect the movements of other joints. We investigate the idea under a data-driven setting, i.e., the NN models are trained based on supervised learning through a given trajectory dataset. Several NN architectures are examined to exploit the joint dependencies of robot arms. A greedy algorithm is then presented to find a proper sequence of applying the joints to decrease the distance error. The experimental results on a 7-DOF robot arm show that the NN models using joint dependency can achieve the same accuracy as the single-MLP model but use fewer parameters.

1 Introduction

In the field of robotics, solving inverse kinematics (IK) is an important problem [1]. The IK problem is, given the position of the target point, to determine the angle of each joint of a robot arm to allow the end-effector to reach the target point.

IK has multiple solutions and is non-linear, which make the solution difficult to obtain [2]. Neural networks (NNs) are commonly used to solve the IK problem due to the abilities of modeling non-linear relationships [3]. In many NN-based solutions, the IK function is modeled by a single fully-connected feedforward neural network, the multi-layered perceptron (MLP), that directly takes the

* This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant MOST 110-2218-E-007-038 and by Information and Communications Research Laboratories of Industrial Technology Research Institute, Taiwan.

Cartesian coordinate of the target point as input and the joint angles as output [3, 4, 5, 6, 7]. However, when the degree-of-freedom (DOF) of the robot arm is high, the problem becomes complicated.

In general, the number of parameters required by a NN depends on the complexity of the problem. Moreover, the complexity of the IK problem is determined by the robot's geometry and the nonlinear equations that describe the mapping from the joint space to the Cartesian space [8]. It is usually proportional to at least the square of the number of DOF [9]. The higher the DOF of the robot, the higher the complexity of the IK problem.

Observing human arm motions, we can see that there are causality relationships between joints of a robot arm. Rotating a joint may drive the subsequent joints and the end effector of the robot arm, causing those joint angles to change correspondingly to reach the target position. Thus, if we apply these joints to the neural networks in sequence according to their dependencies instead of all in parallel, we may be able to use smaller neural networks to control the robot arm while maintaining similar performance.

To see how it may work, we assume that the NN models to solve the IK problem for a robot arm, which maps from Cartesian space into joint space, can be characterized by the IK function shown below. The IK function is decomposable into joint functions that determine the angle of a joint based on the target position and the angles of those depending joints. Each output joint is calculated from the previous joints. The symbols, *x*, *y* and *z* denote the coordinate of target position, and $\theta_1 \sim \theta_n$ denote the joint angles. We then train the NN models using data collected from human demonstrations to show that the assumption is workable.

$$\theta_1, \dots, \theta_n = IK(x, y, z) \longrightarrow \begin{cases} \theta_1 = IK_1(x, y, z) \\ \theta_2 = IK_2(x, y, z, \theta_1) \\ \vdots \\ \theta_n = IK_n(x, y, z, \theta_1, \dots, \theta_{n-1}) \end{cases} \quad \theta_k = IK_k(x, y, z, \theta_1 \sim \theta_{k-1})$$

We propose in this paper four different NN architectures to model the joint functions of IK:

- *Hierarchical neural network* (HNN): HNN is the simplest network that is composed of n MLPs, which respectively model the n joint functions.
- *Recurrent neural network* (RNN): RNN has a cyclic structure, so it can model the recurrent joint functions. We take advantage of this property to learn the causality relationship between joints, instead of time steps in original RNNs.
- *Recurrent neural network & self-attention* (RNN & SA): We use self-attention mechanism to improve the information loss of RNN.
- *Self-attention only* (SAO): We replace recurrent layers with self-attention layers.

To verify our architectures, we solve the IK problem of a 7-DOF robotic arm. We trained and tested the above models by the trajectories collected from human demonstration. Compared with the best MLP model, our models can reduce at best 44.6% distance error than the single MLP with a similar amount of parameters. To achieve a similar accuracy, our models only use 0.4% amount of parameters in the best case. To further improve our models, we also tried different orders of output joints and provide a novel method to optimize the output order of the joints. The result shows that the optimized output has an average 19.5% less distance error than the original output order.

Our contributions can be summarized as follows. (1) We have developed several neural network architectures considering joint dependency to solve the IK problem. (2) We have proposed an algorithm to optimize the joint output order of the hierarchical neural network. (3) We have applied our architectures and algorithm on a 7-DOF robotic arm and evaluated their effectiveness.

2 Related Works

2.1 Recurrent Neural Network

Recurrent neural network, a.k.a. RNN, is a class of artificial neural network which has a cyclic structure. Derived from feedforward neural network, RNN can keep information in their internal state to process variable length sequences of inputs. Hence, they are often used in the fields of natural language processing and speech recognition [14].

Backpropagation through time (BPTT) is commonly used for training RNNs. By unfolding a recurrent neural network in time, the backpropagation algorithm can be applied to update the network parameters [15].

2.2 Multi-Head Masked Self-Attention

Vaswani et al. proposed Transformer, a deep learning model primarily used for natural language processing (NLP) [16]. The major component in the transformer is the unit of multi-head self-attention mechanism. The self-attention is applying attention mechanism between the time steps in a sequence. It adopts the scaled dot-product attention written as

$$\text{Attention}(\text{Q}, \text{K}, \text{V}) = \text{softmax}(\frac{\text{Q}K^T}{\sqrt{d_k}})\text{V} \tag{1}$$

where *Q* is query, *K* is keys, *V* is value, and d_k is the dimension of *K*. The parameters *Q*, *K*, and *V* are the matrices composed of representation vectors of each timestep. The relationship between timesteps, which are words in NLP, is calculated with Q and K, and is used to calculate the weighted sum of V.

Transformer uses the multi-head mechanism to learn the relationship from different representation subspaces. Multi-head attention runs the scaled dot-product attention multiple times in parallel which is written as

$$\begin{aligned} &\text{MultiHead}(\text{Q}, \text{K}, \text{V}) = [head_1; \dots; head_h]W^o \\ &\quad \text{where } head_i = \text{Attention}(\text{Q}W_i^Q, \text{K}W_i^K, \text{V}W_i^V) \end{aligned} \tag{2}$$

The self-attention layer is only allowed to attend to previous positions in the sequence. It can be done by masking future positions, that is setting them to -inf, before the softmax step in the self-attention calculation.

3 Method

In this section, we introduce a NN-based IK solution with joint dependency. We use the IK problem of a 7-DOF robotic arm as an example (Section A), design several architectures to utilize joint dependency (Section B), and propose a novel method to optimize output order (Section C).

3.1 Inverse Kinematics on a 7-DOF Robotic Arm

Figure 1 shows the 7-DOF humanoid robotic arm. To compute forward kinematic, we build the Denavit-Hartenberg (DH) table, which consists of DH parameters extracted from the robotic arm, see Table 1, where θ is the joint angle between two adjacent x-axes measured around z-axis, d is the link offset that is the distance between two adjacent x-axes measured along z-axis, a is the link length that is the distance between two adjacent z-axes measured along x-axis, and α is the link twist angle between two adjacent z-axes measured around x-axis.

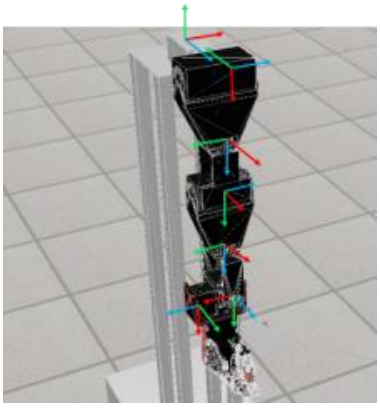

Figure 1: A 7-DOF robotic arm

joint	θ(degree)	d(mm)	a(mm)	α(degree)
1	-90	0	0	-90
2	-90	0	0	-90
3	0	250	0	90
4	0	0	0	-90
5	90	254	0	90
6	90	-20	197	90
7	0	1.5	0	0

Table 1: DH table for the robotic arm

With the DH table, we can write the homogeneous transform matrix from joint k to joint k+1 as

$$T_k^{k-1} = \text{Rot(z,}\theta\text{)Trans(0,0,d)Trans(a,0,0)Rot(x,}\alpha\text{)} = \begin{bmatrix} \cos\theta & -\sin\theta\cos\alpha & \sin\theta\sin\alpha & a\cos\theta \\ \sin\theta & \cos\theta\cos\alpha & -\cos\theta\sin\alpha & a\sin\theta \\ 0 & \sin\alpha & \cos\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

where Rot($\cdot$, $\cdot$) is the homogeneous rotation matrix and Trans($\cdot$, $\cdot$, $\cdot$) is the homogeneous translation matrix. Then, we can obtain the homogeneous transform matrix from the first joint to the last joint by

$$T_7^0 = T_1^0 T_2^1 T_3^2 T_4^3 T_5^4 T_6^5 T_7^6 = \begin{bmatrix} R_{11} & R_{12} & R_{13} & x \\ R_{21} & R_{22} & R_{23} & y \\ R_{31} & R_{32} & R_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

In this work, the forward kinematics is the mapping from the joint angles $\theta = \{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7\}$ to the end point coordinates $p = \{x, y, z\}$ by (3) and (4). On the other hand, the inverse kinematics is defined as $\theta = IK(p)$.

In traditional NN method for IK problem, we usually build a fully-connected feedforward neural network, a.k.a. MLP, which directly takes p as input and θ as output. By minimizing the error between output angle and ground truth angle, we can fit the NN model to solve the IK function. However, in the case of high DOF, MLP needs to use much more parameters to achieve the same accuracy as low DOF, but the resultant accuracy is often unsatisfactory. It is difficult to solve the high DOF IK problem by a single MLP.

3.2 Neural Network Architectures with Joint Dependency

To exploit the dependencies of joints, we divide the IK function into 7 joint functions as shown in Section I. The first joint function only takes the coordinates as input, while other joint functions take the previous output joint as input. To model the joint functions, we propose four different NN architectures: Hierarchical neural network (HNN), Recurrent neural network (RNN), Recurrent neural network & self-attention (RNN & SA), Self-attention only (SAO).

A. Hierarchical Neural Network

The most basic way to model all joint functions is to train an MLP for each joint function. All MLPs are composed of dense layers as shown in Figure 2. The MLPs' input and output are same as the joint function mentioned above. We use N to represent the number of neurons used in our models. In training phase (Figure 3), we use ground truth data collected from human demonstrations as the input to train each joint function. In inference phase (Figure 4), we combine the networks of joint functions to restore them to the original IK function. The network has a hierarchical structure and is thus called hierarchical neural network (HNN).

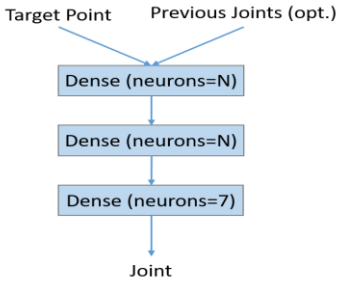

Figure 2: Architecture of MLPs for joint function

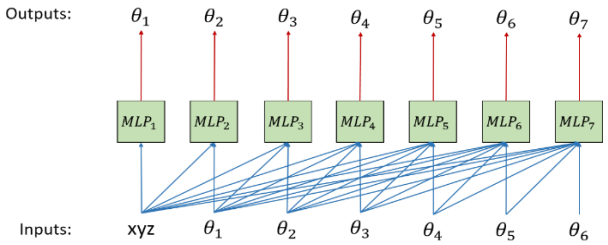

Figure 3: Training architecture of hierarchical neural

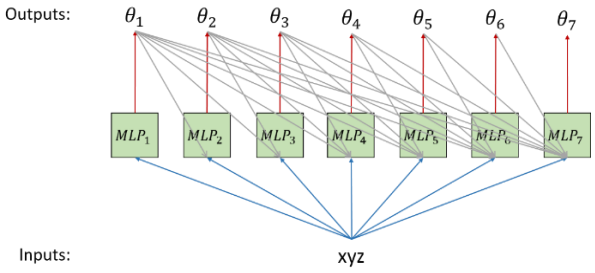

Figure 4: Inference architecture of hierarchical neural network

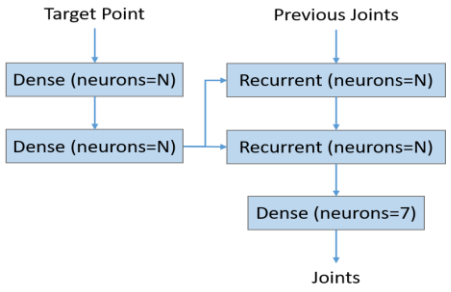

Figure 5: Architecture of RNN for joint function

B. Recurrent Neural Network

Note that joint functions repeat the same process, i.e., using the information of previous outputs to predict the next joint angle. This is very similar to recurrent neural networks (RNNs). Thus, we can model the recurrent joint functions, which reflect the causal relationship between joints, by using RNN. We thus use joint angles of the IK problem as the input of RNN's each time step and output one joint angle in each time step of RNN. The RNN model takes the coordinate of target point as the initial state and outputs one joint angle based on the previous joint angles at each time step, see Figure 5. RNN model uses different inputs during training and inference as shown in Figures 6 and 7.

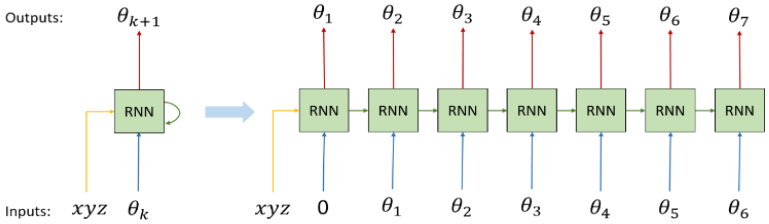

Figure 6: Training architecture of RNN. Left is the original network, and right is the unfolded network

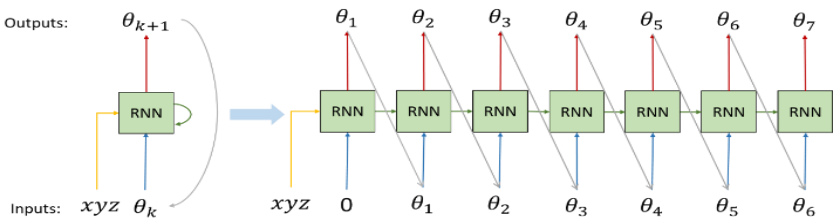

Figure 7: The inference architecture of RNN. Left is the original network, and right is the unfolded

C. Recurrent Neural Network & Self-Attention

Compared to HNN, RNN can only use one network to model all joint functions of IK. It may lose information of previous inputs, especially in long sequence. In order to solve this problem, we add self-attention mechanism in our RNN model.

To be exact, we use multi-head masked self-attention mechanism proposed by [16]. An attention layer makes the number of operations between all the time steps the same, so it can learn the relationship between current time steps more easily. In addition, multi-head attention allows the model to learn the time steps relationship from different representation space. Finally, the illegal connection in attention layer is masked to prevent joints from attending to subsequent joints.

In our model, we put a multi-head masked self-attention layer after the recurrent layer and concatenate their outputs as shown in Figure 8. Actually, we use 4 parallel attention layers, and the attention dimension is a quarter of the model dimension to keep the computational cost unchanged.

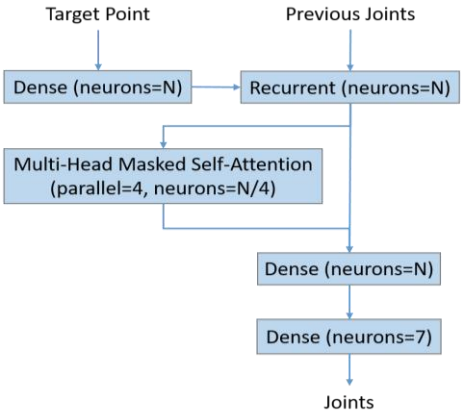

Figure 8: Architecture of RNN & SA

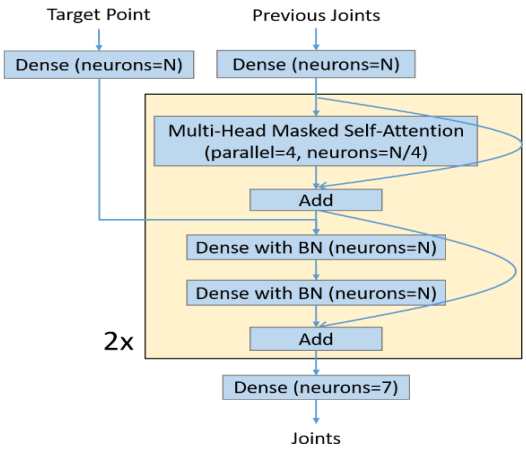

Figure 9: Architecture of SAO

D. Self-Attention Only

With attention mechanism, we can shorten the length of the path from input to output, and the network can compute the output of each time step in parallel. According to [16], we can even use only attention mechanism to achieve the effect of RNN. Hence, we design a network which consists of multi-head masked self-attention layers. To be more stable and easier to train the model, we use techniques such as residual block and batch normalization [17].

Figure 9 shows the architecture of the self-attention only (SAO) network. The SAO network is composed of 2 identical layers. Each layer has two residual blocks whose output is Layer(x)+x, where

Layer(x) is the function of the network in the block. The first is a multi-head masked self-attention mechanism, and the second is a feedforward network. The feedforward network concatenates input with the feature vector of target point and then connects it to two layers of dense and batch-normalization.

3.3 Output Order Optimization

In this section, we consider the problem of output ordering of the joints. Does the output order of joints affect the performance of the model? If so, how can we find a good output order? If we know which joint is more suitable for the later outputs, we may improve the model performance by choosing a better output order. We thus introduce a greedy algorithm to determine the order of the applying the joints with a goal to optimize the distance error of the dataset.

The algorithm is shown in Algorithm 1. Considering a n-DOF IK problem, we can divide the IK function into two joint functions. The first joint function directly outputs n-1 joints with target position, and the second joint function outputs the remaining joint with target position and the other joints. For n joints, there are n combinations of these two joint functions which output n-1 joints and one joint respectively, and each combination is modeled by an HNN. Then, we evaluate models by calculating the distance between end point and target point, and we choose the best combination for the next iteration as shown in Figure 10. This process is actually searching the most suitable joint to be the later outputs, and we repeat this process until we have an HNN which consists of n MLPs. This gives an output order which has lower distance error in testing dataset than the original order.

Algorithm 1: Optimize output order

n : the number of joints
θ_{unsort} : the collection of joints index to be sort
θ_{sorted}: the list of sorted joints index
θ_{hnn}: the list which is the output of HNN
θ_{best}: the joint which perform best

Initialize $\theta_{unsort} = \{1, \dots, n\}$, $\theta_{sorted} = [\,]$
for $i = 1 \sim n\text{-}1$ **do**
    foreach θ in θ_{unsort} **do**
        $\theta_{hnn} \leftarrow [\theta_{unsort} - \{\theta\}, \quad \theta] + \theta_{sorted}$
        build an HNN with the output order θ_{hnn}
        train the HNN and evaluate distance error
    end foreach
    $\theta_{best} \leftarrow$ choose the θ with the minimal distance error
    $\theta_{unsort} \leftarrow \theta_{unsort} - \{\theta_{best}\}$
    $\theta_{sorted} \leftarrow [\theta_{best}] + \theta_{sorted}$
end for
$\theta_{sorted} \leftarrow [\theta_{unsort}] + \theta_{sorted}$

Algorithm 1: Process to optimize output order of HNN for IK problems

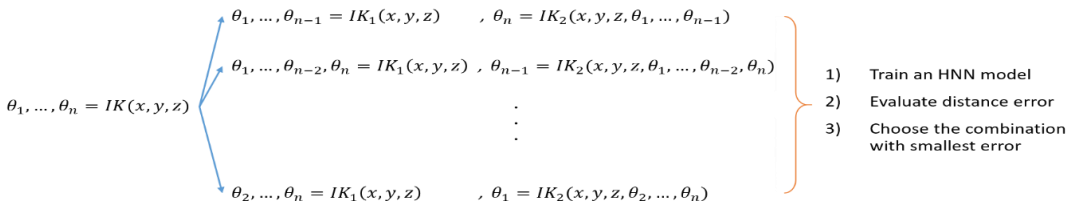

Figure 10: Choosing the best combination of output orders

4 The Experiments

4.1 Environment

In our experiments, we used a 7-DOF humanoid robotic arm to be the test subject. Due to the high cost in the real world, we simulated the robot in the virtual environment of Webots. To prepare the training data for the NN models, we used VR devices to control the robotic arm in the virtual environment and collected the data of human demonstration, which is to grasp the bottle on the table from the initial position, and the coordinates and joints' angles of all points in the path would be recorded. We collected 20 trajectories from each of 9 locations as training data and 1 trajectory at each of 8 locations as testing data.

We used Python deep learning library Keras to build NN model and use the tensor compiler PlaidML as Keras' backend. Our experiments were performed on AMD Radeon RX590 GPU and Intel i7-8700 six-core 3.2GHz CPU. The following is our training configuration: training epoch 2000, batch size 4096, optimizer Adam, and loss function MAE. For each model architecture, we used five different learning rates 0.00025, 0.0005, 0.001, 0.002, 0.004 to train models and chose the model with the best performance as the final result. We evaluated the models' performance by calculating the average distance error which is the coordinate difference between the target point and the predicted end point.

4.2 Comparison of Architectures

We evaluated the performance of single-MLP and our architectures under different numbers of parameters. We only changed the number of neurons of the hidden layers in the models and kept the other architecture setting unchanged. Figure 11 shows the experiment result.

Regardless of the architecture, the distance errors show a downtrend as the number of parameters increased. All our architectures outperform the single-MLP. Taking the best single-MLP as the baseline, our models can achieve 55.4% distance error of baseline. Moreover, we only used 0.4% number of parameters to achieve the accuracy of the baseline. The joint dependency is helpful for joint angle prediction, and allows NN models with fewer parameters.

The models with the self-attention mechanism has lower distance error than the RNN models and the HNN models when the number of parameters is larger than 100K. This shows that the self-attention mechanism is better than RNN to learn the joint dependency. However, the SAO models performed worse than the RNN&SA models when the number of parameters is smaller than 10K. A possible reason is that self-attention can learn the causal relationship better than RNN but needs more parameters.

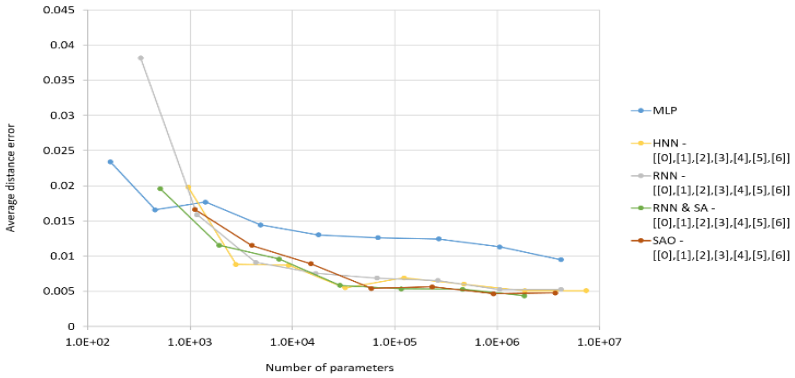

Figure 11: Performance comparison of architectures

4.3 Output Order Optimization

To further investigate the effect of joint output order on model performance, we used the proposed greedy algorithm to optimize the joint output order and compared the result with the original order as shown in Figure 12. The optimized output order outperforms the original output order under almost all number of parameters, resulting in an average 19.5% less distance error. In summary, the joint output order affects the performance of the models, and the proposed greedy algorithm can find a better output order to reduce the distance error of HNN models.

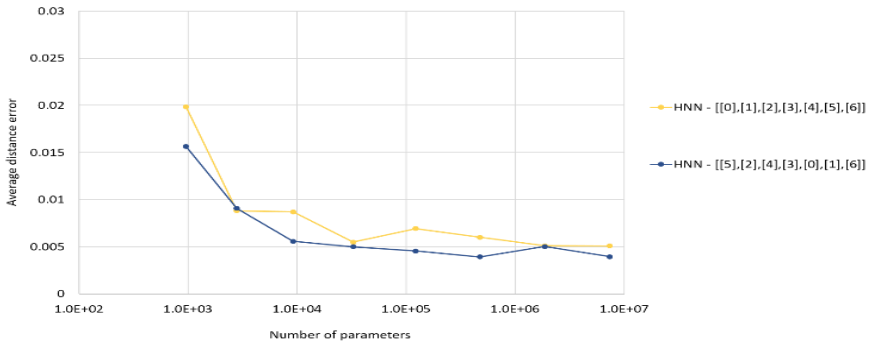

Figure 12: Performance comparison of output orders

4.4 Joint Analysis

The output order found by our algorithm is different from the intuitive joint order, which is from the body to the end. To know why, we analyze the training data. In Figure 13, it can be seen that the distributions of both joints 2 and 5 are very concentrated, meaning that they are almost unchanged in the trajectories of human demonstration. Hence, they are suitable for earlier outputs, because they have little effect on the distance errors. Table 2 shows the correlation coefficients of joints, where $\boldsymbol{R_{xyz}}$ takes target coordinate as input variables and $\boldsymbol{R_{xyz\&joints}}$ takes target coordinate and other joints as input variables. We found joint 6 has the lowest $\boldsymbol{R_{xyz}}$ and has the highest improvement by adding

other joints as input variables. Adding information about other joints can make joint 6 more interpretable and predictable, which explains why joint 6 is the last output in our result.

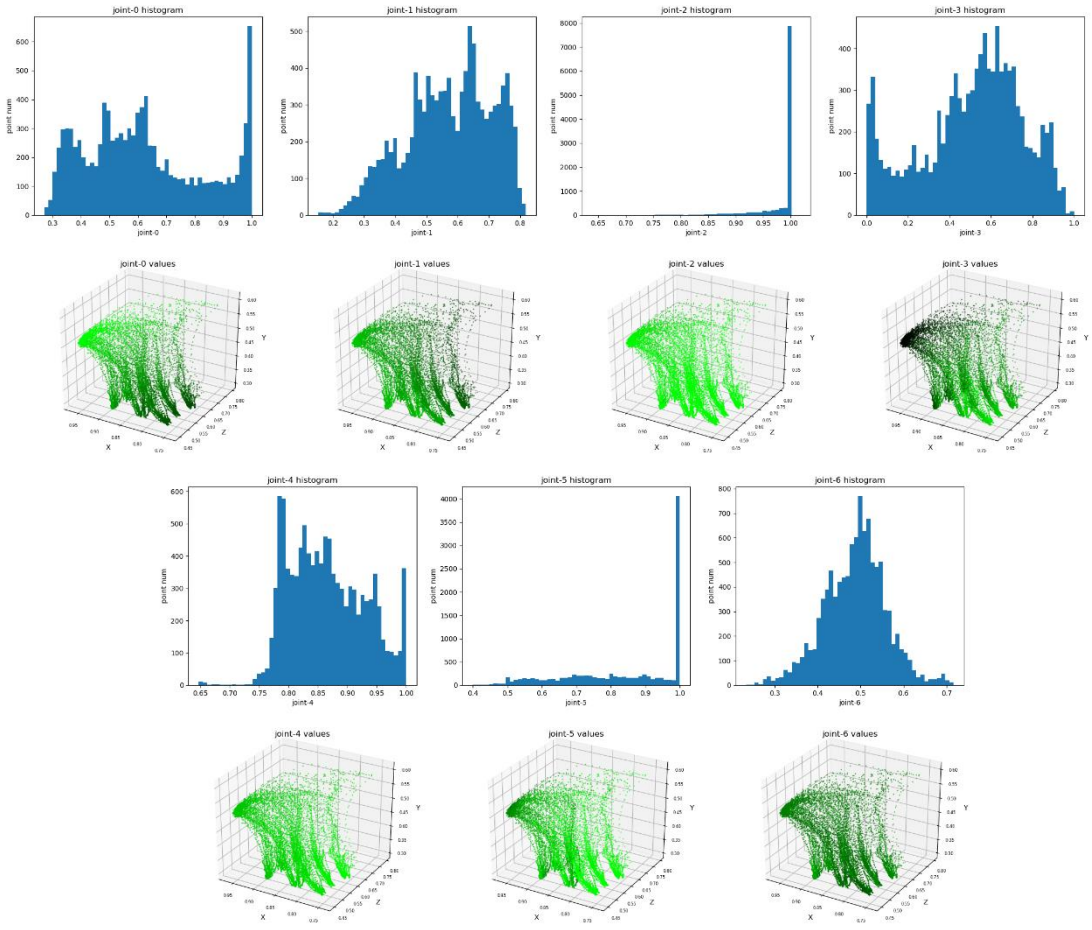

Figure 13: Joints value distribution. The upper row is the histogram of joint values, and the lower row is the 3D scatter diagram of joints value. From left-top to right-top is joint 0~3, and from left-bottom to right-bottom is joint 4~6.

Joint index	0	1	2	3	4	5	6
R_{xyz}	0.98	0.965	0.513	0.905	0.841	0.745	0.407
$R_{xyz\&joints}$	0.999	0.978	0.742	0.993	0.953	0.945	0.825
improvement	0.019	0.013	0.229	0.088	0.112	0.2	0.418

Table 2: Correlation coefficients of joints

5 Conclusions

In this paper, we proposed a NN-based IK solution with joint dependency to reduce parameters of NN models. Inspired by human arm motion, we consider the causality relationship among the joint angles in the IK problem. We designed four NN architecture to investigate the joint dependency and applied them on a 7-DOF robot. The experimental results show that the NN models with joint dependency use fewer parameters than the traditional single-MLP models when they have similar accuracy. With the joint dependency, we improved NN models for the IK problem. Furthermore, we also proposed a simple greedy algorithm to optimize the joint output order. The experimental results show that the optimized output order has better performance than the original output order.

References

[1] Montenegro, F. J. C., Grando, R. B., Librelotto, G. R., & da Silva Guerra, R., "Neural network as an alternative to the jacobian for iterative solution to inverse kinematics," *Proceedings 2018 Latin American Robotic Symposium,* 2018.

[2] Al-Qurashi, Z., & Ziebart, B., "Hybrid Algorithm for Inverse Kinematics Using Deep Learning and Coordinate Transformation," *Proceedings of Third IEEE International Conference on Robotic Computing (IRC)*, 2019.

[3] Duka, A. V., "Neural network based inverse kinematics solution for trajectory tracking of a robotic arm," *Procedia Technology*, vol. 12, no. 1, pp. 20-27, 2014.

[4] Aggarwal, L., Aggarwal, K., & Urbanic, R. J., "Use of artificial neural networks for the development of an inverse kinematic solution and visual identification of singularity zone (s)," in *Procedia Cirp*, *17*, 2014.

[5] Putra, R. Y., Kautsar, S., Adhitya, R. Y., Syai'in, M., Rinanto, N., Munadhif, I., ... & Soeprijanto, A., "Neural network implementation for invers kinematic model of arm drawing robot," in *2016 International Symposium on Electronics and Smart Devices (ISESD)*, 2016.

[6] Smirnov, P., Mikhalchenko, D., & Malov, D., "Neural Network Based Approach to Positioning Task for the End-Effector of a Four-Joint Manipulator," in *2018 International Russian Automation Conference (RusAutoCon)*, 2018.

[7] Lathifah, N., Handayani, A. N., Herwanto, H. W., & Sendari, S., "Solving inverse kinematics trajectory tracking of planar manipulator using neural network.," in *2018 International Conference on Information and Communications Technology (ICOIACT)*, 2018.

[8] Hasan, A. T., Al-Assadi, H. M. A. A., Isa, A. A. M., & Suzuki, K., "Neural networks' based inverse kinematics solution for serial robot manipulators passing through singularities," in *Artificial Neural Networks. Industrial and Control Engineering Applications*, 2011.

[9] Ayusawa, K., & Nakamura, Y., "Fast inverse kinematics algorithm for large DOF system with decomposed gradient computation based on recursive formulation of equilibrium," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[10] "Wikipedia-Kinematics," [Online]. Available: https://en.wikipedia.org/wiki/Kinematics.

[11] "Inverse kinematics (IK) algorithm design with MATLAB and Simulink," [Online]. Available: https://ww2.mathworks.cn/discovery/inverse-kinematics.html.

[12] "Wikipedia-Robot kinematics," [Online]. Available: https://en.wikipedia.org/wiki/Robot_kinematics.

[13] "Wikipedia-Inverse kinematics," [Online]. Available: https://en.wikipedia.org/wiki/Inverse_kinematics.

[14] "Recurrent Neural Networks cheatsheet," [Online]. Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[15] "Wikipedia-Backpropagation through time," [Online]. Available: https://en.wikipedia.org/wiki/Backpropagation_through_time.

[16] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I., "Attention is all you need," in *Advances in neural information processing systems*, 2017.

[17] He, K., Zhang, X., Ren, S., & Sun, J., "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[18] "Robot Kinematics in a Nutshell," [Online]. Available: https://robocademy.com/2020/04/21/robot-kinematics-in-a-nutshell/.

[19] "Deep Learning: Recurrent Neural Networks," [Online]. Available: https://medium.com/deeplearningbrasilia/deep-learning-recurrent-neural-networks-f9482a24d010.