参赛队员姓名： _____ 廖培元 _____

中学： _____ Kent School _____

省份： _____ CT _____

国家/地区： _____ USA _____

指导教师姓名： _____ Matthew Austin _____

论文题目： ___Deep Neural Network Based Subspace Learning of Robotic Manipulator Workspace Mapping__

Deep Neural Network Based Subspace Learning of Robotic Manipulator Workspace Mapping

Peiyuan Liao

Kent School

Kent, Connecticut 06757

Email: liaop20@kent-school.edu

Abstract—**The manipulator workspace mapping is an important problem in robotics and has attracted significant attention in the community. However, most of the pre-existing algorithms have expensive time complexity due to the reliance on sophisticated kinematic equations. To solve this problem, this paper introduces subspace learning (SL), a variant of subspace embedding, where a set of robot and scope parameters is mapped to the corresponding workspace by a deep neural network (DNN). Trained on a large dataset of around 6×10^4 samples obtained from a MATLAB® implementation of a classical method and sampling of designed uniform distributions, the experiments demonstrate that the embedding significantly reduces run-time from 5.23×10^3 s of traditional discretization method to 0.224 s, with high accuracies (average F-measure is 0.9665 with batch gradient descent and resilient backpropagation).[1]**

I. INTRODUCTION

As an essential feature of robotic manipulators, manipulator workspace is the set of poses they can attain by their end effectors. Since the last few decades, the its study has gained much attention by the field because the workspace of a manipulator foreshadows its functionality and understanding it aids in the process of trajectory planning. Thus, extensive research has been done on developing an algorithm to calculate the workspace [1]–[15]. Most of the current methods are focused on three-dimensional (3D) subspace which is projected from the original higher-dimensional manifold, because 3D is the maximum dimension that can be visualized [4].

The focus of this paper is on improving the discretization method, one of the three major types of workspace algorithms. Discretization method involves dividing the desired spaces into partitions of equal volume, and further applying either forward or inverse kinematics. More specifically, the binary representation method, a variant of the family of discretization methods, involves the approximation of the workspace through the form of a binary map. The three-dimensional space is divided into multiple cuboid partitions with equal shape and volume, and an array is created to represent them. If a particular node is contained in the actual workspace, the value of the corresponding element will be set to 1, otherwise 0. A sequential initialization on all elements in the array generates the output.

However, this method time-consuming when the number of partitions is substantialy large. Take [3]'s method for constant

orientation workspace of all-revolute serial-link manipulators for example. Seeing the workspace algorithm as a mapping from a vector x that defines the robot and the scope to a bit vector $y \in \mathbb{B}^n$ describing the workspace, the input vector is defined as the concatenation of the Denvait-Hartenberg parameters, coordinate values of all nodes on the partitions as well as the chosen orientation.

$$y = f\left(\begin{bmatrix} r_d & r_a & r_\alpha & n_x & n_y & n_z & n_i \end{bmatrix}^\top\right) \qquad (1)$$

The first part of the vector $r_d, r_a, r_\alpha \in \mathbb{R}^d$ defines a manipulator regarding link offset, link length and link twist [16], where d is the degree-of-freedom (DOF). $n_x \in \mathbb{R}^{c_x+1}, n_y \in \mathbb{R}^{c_y+1}, n_z \in \mathbb{R}^{c_z+1}, n_i \in \mathbb{S}^3$ describes the scope of the algorithm, where c_x, c_y and c_z are the number of elements in n_x, n_y and n_z respectively. To promote memory efficiency, the technique of coordinate mapping is proposed: n_x, n_y, n_z respectively describe the Cartesian coordinate values an arbitrary node can take in the first, second and third dimension. To extract the specific coordinate values, one simply has to find all possible permutations of elements in n_x, n_y, n_z, thus the name. n_i describes the chosen orientation in terms of displacement around x, y and z axis, and the transformation from which to $SO(3)$ is also straightforward [17]:

$$R = R_x(n_{i_1})R_y(n_{i_2})R_z(n_{i_3}) \qquad (2)$$

In the implementation, the programmer could either choose to implement the forward kinematic equation or the inverse according to the specific situation. Due to the difficulty of the calculation of $\mathcal{K}^{-1}(\xi)$ and $\mathcal{K}(q)$, the algorithm has an expensive time complexity as x increases in length. For the inverse kinematics method, there are two factors that contributes to the time complexity. Either the robot itself becomes more complex (resulting in a longer vector describing it), or the number of points in the bit tensor is increasing (increasing scope or precision). Thus, the classical algorithm has a time complexity of

$$O((c(n))^3) + O(n^3) \qquad (3)$$

, where $O(c(n))$ denotes the time complexity of the inverse kinematics algorithm as when the length of the vector describing the robot increases. The $O(n^3)$ term is resulted from the coordinate mapping method. Due to the abundance

of matrix inversions, differential motion calculations and Jacobian computations in the inverse kinematic equation, the overall complexity is obviously worse than $O(n^3)$. Appendix VIII-B shows a possible complexity of the algorithm when the method is implemented in MATLAB® with depedent library created by [17], and the time complexity for a general case is $O(n^{36})$. For the forward kinematics case, it is even harder to discuss the time complexity, since the process of sampling a joint configuration from probability distributions is unstable and time-consuming per se.

Thus, it would be useful if an algorithm can compute the workspace without using either coordinates mapping or manipulator kinematic equations. Artificial neural networks, as universal approximators for high-dimensional non-convex functions [18], [19], are found to be extremely helpful in this situation. In the following sections, the possibilities of developing an algorithm for manipulator workspace generation using techniques from deep learning are being explored. First, the problem of manipulator workspace will be formally defined. Then, a new technique termed subspace learning is introduced, exhibiting desirable characteristics from both classical methods and general deep learning algorithms. Finally, case studies on a class of 6-DOF manipulators is presented to demonstrate the feasibility of SL.
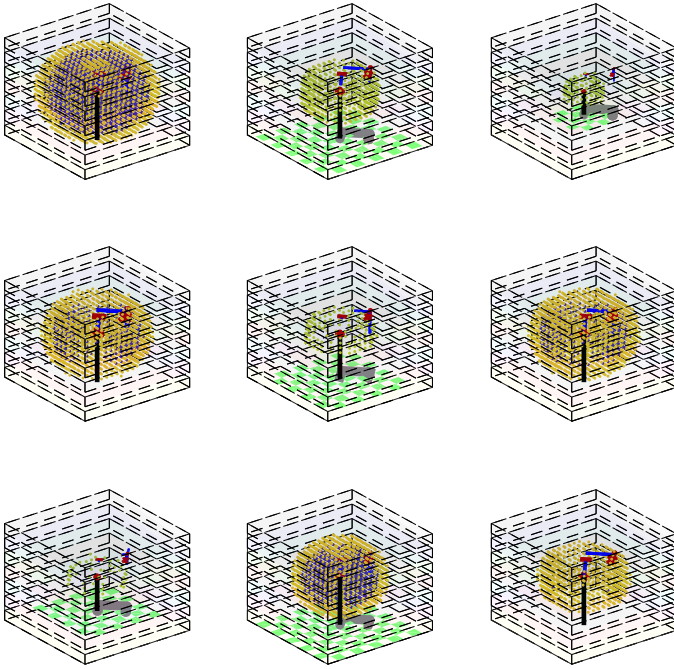

Fig. 1. DNN-generated constant orientation workspace of 6-DOF manipulators with spherical wrist

II. RELATED WORKS

In the literature, the development of a neural network for workspace generation has been already tested on both serial-link and parallel mechanisms, utilizing several model architectures and optimization methods [20], [21]. The feasibility of using an artificial neural network for workspace analysis is first proposed and investigated by [9], where a two-layer perceptron learns to generate the orientation workspace of a 6-3 SPM parallel mechanism based on a 3-tuple input. In a similar approach, [8] constructs a three-layer deep network that takes the input joint angles of a 2-DOF parallel manipulator and gives the workspace of the mechanism in a particular Cartesian plane. Levenberg-Marquardt method is used for optimization, and the lowest mean squared error obtained by the algorithm is 0.026.

III. PROBLEM STATEMENT

A. Inverse Kinematics

Since the purpose of this study is not to discuss different ways of obtaining the relationship between joint angle and end-effector velocities, and how it relates to existence of solutions (or lack thereof), only a general numerical solution is presented in Appendix VIII-C [22]. For more information on this topic, one can refer to works of [23] and [17], where a wider spectrum of methods are discussed more in depth.

B. Orientation and constant orientation workspace

First, let $\boldsymbol{q}$ denote the joint configuration of a serial-link manipulator (for all-revolute manipulator $\boldsymbol{q} \in \mathbb{S}^n$) with a degree-of-freedom of n. $\boldsymbol{a} \in \mathbb{R}^3$ denotes the Cartesian coordinate of a point in 3-D space and $\boldsymbol{R} \in SO(3)$ denotes its orientation. The pose of a manipulator (or any rigid body in general) encodes two different sets of information: position and orientation. Thus, the special Euclidean group $SE(3)$ is helpful in this context. It is clear in the form below that $SE(3)$ can store both types of information [4]:

$$SE(3) = \left\{ (\boldsymbol{a}, \boldsymbol{R}) \right\} = \mathbb{R}^3 \times SO(3) \tag{4}$$

And the complete six-dimensional manipulator workspace:

$$\mathbb{W} \subset SE(3) = \left\{ (\boldsymbol{a}, \boldsymbol{R}) | \exists \boldsymbol{q}(\mathcal{K}^{-1}(\xi(\boldsymbol{a}, \boldsymbol{R})) = \boldsymbol{q}) \right\} \tag{5}$$

However, to store the workspace in a presentable manner, different variants of the complete workspace exist that put constraints either on position or orientation. the orientation and constant orientation workspace of the manipulator, $\mathbb{W}_{ow}$ and $\mathbb{W}_{cow}$ respectively, can be defined as below:

$$\mathbb{W}_{ow} = \left\{ \boldsymbol{R} \in SO(3) | \exists \boldsymbol{q}(\mathcal{K}^{-1}(\xi(\boldsymbol{a}_{const.}, \boldsymbol{R})) = \boldsymbol{q}) \right\} \tag{6}$$

$$\mathbb{W}_{cow} = \left\{ \boldsymbol{a} \in \mathbb{R}^3 | \exists \boldsymbol{q}(\mathcal{K}^{-1}(\xi(\boldsymbol{a}, \boldsymbol{R}_{const.})) = \boldsymbol{q}) \right\} \tag{7}$$

It is also possible to convert the workspace sets to tensors ($\mathbf{W}, \mathbf{W}_{ow}$, or $\mathbf{W}_{cow}$), for the sake of portability.

C. Supervised Learning

The problem of generating a workspace given manipulator and the scope parameter can be classified as a type of supervised learning, where input predicts the "label". In machine learning, instead of using an analytic form of mapping from x to y for every value in a specific set (like $\mathbb{R}^n$), the rationale is probabilistic in essence. This means that knowledge about the input is presented as a probability distribution $p(\mathbf{x})$, and the mapping is now the probability distribution of the targets y given x ($p(\mathbf{y} \mid \mathbf{x})$).

Thus, viewing the workspace as a vectorized random variable, we can arrive at a probabilistic model version of the workspace mapping:

Find $p_{ideal}(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\theta})$, where $p(\mathbf{x})$ describes the probability distribution of manipulator construction parameters and pose information of the scope. A state of $p_{ideal}(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\theta})$ has elements describing whether a pose is within the true workspace and can be converted first to a bit tensor $\mathbf{P}$ and then to $\mathbb{W}$ defined above.

IV. METHODOLOGY

A. Redefining the problem

The problem with the machine learning problem formulated above is that both $\mathbf{y}$ and $\mathbf{x}$ are infinite-dimensional. The probability distribution is designed to be capable of describing any manipulator with infinite precision, so the number of elements in the vectors will also be infinite. This is clearly unfeasible and unnecessary for real-world applications, so the first layer of simplification is to find instead a different distribution $p_s(\mathbf{y}'$ $\mid \mathbf{x}')$, that has random variables of finite dimensions. The new conditional distribution only presents the approximated workspace with finite precision given vectorized parameters of a fixed type of manipulators. To further make the problem suitable for deep learning, an altruistic method implemented in this paper is to treat the "scope" as concatenation of parameters regarding nodes on the finite partitions of either $\mathbb{R}^3$ or $SO(3)$. By taking advantage of Equation (1) [3], $\mathbf{x}'$ can be defined in the same way, except in terms of a random variable. $p_s(\mathbf{y}' \mid \mathbf{x}')$ would then be a multivariate Bernoulli distribution describing whether what nodes are inside the real workspace. Thus, the discretization method mentioned in the introduction can now be seen as obtaining the state of $\mathbf{y}'$ given a state of $\mathbf{x}'$. Direct assembly from a state of $\mathbf{x}'$ to the bit tensor $\mathbf{P}$ is impossible however, yet still feasible with informations from $\mathbf{y}'$ and specific code instructions. This new distribution is in per se a different distribution compared to p_{ideal}, yet it is learnable by an artificial neural network, and it offers useful insights that enable us to extract information from the obtainable distribution.

B. Experience Generation

It is possible to obtain observations of the new objective distribution $p_s(\mathbf{y}' \mid \mathbf{x}')$ by conventional workspace algorithm. The only modification is to disassemble the bit tensor $\mathbf{P}$ to a vector, so that the training of the dataset in the context of deep learning is possible. To design $\mathbf{x}'$, Equation (1) is used

where altruistic precision, scope, and manipulator kinematic structure are chosen. In implementation, the process is online where values of specific elements are sampled from designed distributions. In this subspace learning scenario, which will be explained below, values are observations from different uniform distributions. The details of this can be found in Appendix VIII-D.

C. Subspace Learning

Another problem about the application of deep learning in industrial scenarios is that artificial neural networks are merely approximators [18], [19]. In other words, it is impossible for deep learning algorithms to thoroughly learn the mapping [24], and often challenging for them to reach high precision with a hard task and a limited computing capability.

A solution to the problem is subspace learning (SL), a deep learning algorithm based on subspace embedding that reduces the dimensionality of the distribution before the process of learning. The rationale behind SL is that sometimes not only are input-output pairs available to us, but also $p(\mathbf{x})$ or even the true data generation process. Hence, learning the entire conditional distribution is often not necessary, and only some specific cases (or "subspaces," since sometimes they are in fact lower-dimensional subspaces of the original higher-dimensional space) of it are considered as the objective. So, instead of struggling to find the complete $p_s(\mathbf{y}' \mid \mathbf{x}')$, it is simplified to

$$\left\{ \tilde{p}_1(\mathbf{y}_1'|\mathbf{x}_1'), \tilde{p}_2(\mathbf{y}_2'|\mathbf{x}_2'), \ldots, \tilde{p}_i(\mathbf{y}_i'|\mathbf{x}_i') \right\} \tag{8}$$

, where $\tilde{p}_1(\mathbf{y}_1'|\mathbf{x}_1'), \tilde{p}_2(\mathbf{y}_2'|\mathbf{x}_2'), \ldots$ are unnormalized modifications of the original $p_s(\mathbf{y}' \mid \mathbf{x}')$ generated by altruistically altering $p(\mathbf{x}_1'), p(\mathbf{x}_2') \ldots, p(\mathbf{x}_1', \mathbf{y}_1'), p(\mathbf{x}_1', \mathbf{y}_2') \ldots$ or themselves. These modifications include but are not limited to: removing elements,

$$\begin{aligned} \mathbf{x}_i' &\subset \mathbf{x}' \\ \mathbf{y}_i' &\subset \mathbf{y}' \end{aligned} \tag{9}$$

fixing elements,

$$\begin{aligned} \mathbf{x}_i' = \left\{ \mathbf{x}_a', \cdots \sim \delta(\mu_a) \ldots | p_s(\mathbf{x}_a' = \mu_a) \neq 0, \ldots \right\} \\ \cup \left\{ \mathbf{x} \in \mathbf{x}' | \mathbf{x} \neq \mathbf{x}_a, \ldots \right\} \end{aligned} \tag{10}$$

and changing parameters of a distribution, like the lower bound and the upper bound of a uniform distribution.

$$\begin{aligned} \mathbf{x}_i' = \left\{ \mathbf{x}_a', \cdots \sim \mathcal{U}(a_{min}, a_{max}), \ldots | a_{min}, a_{max} \in \mathbb{R}, \ldots \right\} \\ \cup \left\{ \mathbf{x} \in \mathbf{x}' | \mathbf{x} \neq \mathbf{x}_a, \ldots \right\} \end{aligned} \tag{11}$$

The last two modifications introduced here are not applicable to $\mathbf{y}'$ since $p(\mathbf{y}')$ is mostly unobtainable, while $p(\mathbf{x}'|\mathbf{y}')$ is the goal of learning. Thus, it is not logical to fix elements or change distribution type of some unknown distributions.

This reduces the dimensionality of the learning task even though the altered distributions are not used a priori. Although

the changes are mostly done on $\mathbf{x}'$, in some cases the structure of $\mathbf{y}'$ might also be modified heuristically. For Equation (10), the new $\tilde{p}_i(\mathbf{y}'_1|\mathbf{x}'_1)$ is literally a geometrical subspace of the original high dimension distribution. While for examples like Equation (9) and Equation (11), the new distributions only share elusive characteristics with its origin, and does not satisfy the literal meaning of "subspace."

Traditionally, only one approximator function $f_{original}(\boldsymbol{x}; \boldsymbol{\theta})$ is created to learn and approximate the distribution, and sometimes the required capacity of $f_{original}$ is too large to be constructed and trained on existing platforms. Subspace learning, on the other hand, does not require the optimization of such function. A set of functions related to the subspaces are constructed instead that not only as a whole are analogous to the original function, but also have lower capacities.

$$
\begin{aligned}
&f_{original}(\boldsymbol{x}; \boldsymbol{\theta}) \\
&\left\{ f_1(\boldsymbol{x}_1; \boldsymbol{\theta}_1), f_2(\boldsymbol{x}_2; \boldsymbol{\theta}_2), ... f_i(\boldsymbol{x}_i; \boldsymbol{\theta}_i) \right\}
\end{aligned}
\tag{12}
$$

This is where the trade-off comes in. Intuitively, the total time of optimizing $f_{original}$ is much less than optimizing all possible subspace functions $f_1, f_2 \ldots f_n$, if the means to create a subspace is properly defined. However, for certain tasks the majority of values in $\boldsymbol{x}$ is never used to calculate $f_{original}(\boldsymbol{x}; \boldsymbol{\theta})$, but whatever are being used does belong to the same distribution. Thus, the subspace learning alternative can be used to fix the values of those elements. Moreover, if a specific machine learning task only requires a small number of subspaces, the sequential training of subspaces is more feasible and require less time than training the original function.
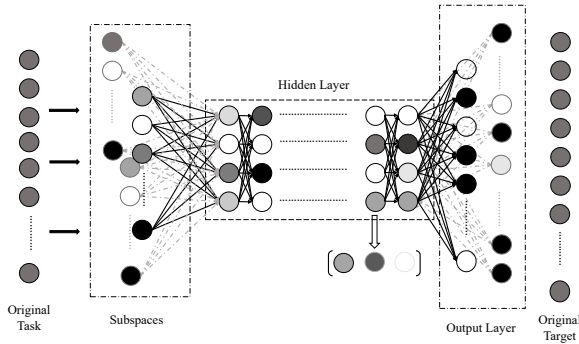

Fig. 2. Network Architecture

Figure (2) presents the DNN model of subspace learning. There are two remarks on the architecture. First, the original samples are never used for the actual training of the algorithm. Instead, the optimization process is done on various extracted subspaces. The second remark is that all subspace models use the same hidden layer architecture. Apparently, the values of parameters are different if the same model is optimized to suit specific subspaces. To address the problem, all neurons will store multiple sets of parameters. During the learning process,

different subspaces are optimized by different altruistic methods and then stored in the same model. During run-time, the system will go through a linear-search to make the neurons take the specific set of parameters tuned especially for the input.

1) Distinctions between Subspace Learning and Generic Feedforward Networks: There are a few nuances between the DNN model for subspace learning and a typical deep feedforward network. First, there are multiple goals for optimization in subspace learning, although optimizing the parameters of a specific subspace is the same as that of a feedforward network. The cost function is now the average loss over the subspace instead of the original training set, which can be written as (for the i-th subspace)

$$
J(\boldsymbol{\theta}_i) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p_i} L(f_i(\boldsymbol{x}; \boldsymbol{\theta}_i), \boldsymbol{y})
\tag{13}
$$

And, applying the concept of empirical risk minimization with m examples [25],

$$
\begin{aligned}
&\mathbb{E}_{(\boldsymbol{x}_i,\boldsymbol{y}_i)\sim \hat{p}_i} L(f_i(\boldsymbol{x}_i; \boldsymbol{\theta}_i), \boldsymbol{y}_i) \\
&= \frac{1}{m} \sum_{i=1}^{m} [L(f_i(\boldsymbol{x}_i^{(i)}; \boldsymbol{\theta}_i), \boldsymbol{y}_i^{(i)})]
\end{aligned}
\tag{14}
$$

$$
\begin{aligned}
&\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p} L(f_{original}(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y}) \\
&= \mathbb{E}_{(\boldsymbol{x}_n,\boldsymbol{y}_n)\sim p_n} [\mathbb{E}_{\mathbf{m}\sim p_{subspace}} [L(f_n(\boldsymbol{x}_n; \boldsymbol{\theta}_n), \boldsymbol{y}_n)]] \\
&\approx \mathbb{E}_{(\boldsymbol{x}_n,\boldsymbol{y}_n)\sim \hat{p}_n} [\mathbb{E}_{\mathbf{m}\sim p_{subspace}} [L(f_n(\boldsymbol{x}_n; \boldsymbol{\theta}_n), \boldsymbol{y}_n)]]
\end{aligned}
\tag{15}
$$

Equation (15) presents a loosely defined expectation of loss expectations of all possible subspaces taken over first the random variable parameters defining the subspace $\mathbf{m}$ then the model distribution (and thus the empirical distribution). It is believed that if the ways to create subspace is limited to a few well-defined procedures, the expectation is equal to the original expectation of the loss of the would-be created function. This is, however, not proven mathematically, and hence more theoretical works beyond the example given in Appendix VIII-A are required. The only certainty is that all subspaces provide useful characteristics that are analogous to that of the original one. Notice that the expectation is taken from the subspace empirical distribution instead of the entire data generating empirical distribution $\hat{p}_s$.

For regularized models of a subspace,

$$
J'_i(\boldsymbol{\theta}_i; \mathbb{X}_i, \mathbb{Y}_i) = J_i(\boldsymbol{\theta}_i; \mathbb{X}_i, \mathbb{Y}_i) + \beta_i \Omega_i(\boldsymbol{\theta}_i)
\tag{16}
$$

The regularization function $\Omega(\boldsymbol{\theta})$ and the hyperparameter $\beta \in [0, +\infty)$ can be the same for every subspace, or it can be altered altruistically to adapt to specific situations.

Intuitively, all existing optimization and regularization methods are applicable to train the subspace, and in the case studies section results between some of the most popular optimization methods are compared to show some of the nuances in subspace optimization.

The second major distinction is that training a SL algorithm includes the integration of different models. Once traditional

optimization methods are done for every subspace and optimal parameters are obtained, it is possible to construct a single model to store all the information. Traditionally, two graphs will be created for a feed-forward neural network, one graph $\mathcal{G}$ for forward propagation and a subgraph of the former $\mathcal{B}$ for back-propagation [25]. In subspace learning, two computational graphs are also created for a subspace, except for the following differences. First, instead of storing weight matrices for the hidden layer, tensors are created where $\mathbf{W}^{(j)}_{:,:,i}$, $j \in \{1, 2, ...l\}$ denotes the i-th subspace weight matrices of a model with a depth of l. The same is true for biases, where

$$\boldsymbol{B}^{(j)}_{:,i}, j \in \{1, 2, ...l\} \tag{17}$$

denotes the bias parameters of the i-th subspace in the same model. The information of input/output parameters are stored in other altruistic data structure. The second difference has to do with the connections between the hidden layer and the input/output layers. When a specific subspace is used or being trained, a new graph $\mathcal{G}_{new}$ will be created at runtime that incorporates the input/output layer and the hidden layer. Since different subspaces may have different input and output vector length, $Pa_{\mathcal{G}_{new}}(\boldsymbol{u}^{(2)})$ differs from subspaces to subspaces. The same is also true for the last layer $\boldsymbol{u}^{(l)}$. Except for the first and last layers, the new graph's layers contain exactly one edge for an edge connecting two arbitrary nodes in $\mathcal{G}$. Rest of the connections are made exactly like a generic fully-connected feed-forward network.

2) Context in Manipulator Workspace problem: The motivation of creating a subspace learning network is mainly to addresses needs in the field of robotics, so it is especially useful in manipulator workspace calculation for the following. First, although Denvait-Hartenberg parameters are capable of describing a broad spectrum of kinematic structures, only a few have real-world applications. For example, spherical wrists significantly enhance the manipulability of robots because it allows rotation of the end-effector around a single point in three-dimensional space. Now consider an element in the input vector x_i that represents a link twist between two links of a manipulator. Instead of learning the complete distribution $x \sim \mathcal{U}_{\mathbb{S}}$, which is unnecessary because the algorithm has to adjust for peculiar angles like $\frac{2\pi}{13}$, three subspaces are created to the algorithm where for a subspace the algorithm observes a fixed state of x (like x=$\frac{\pi}{2}$, 0 or $-\frac{\pi}{2}$). Graphically, the algorithm is learning a hyperplane in the original higher-dimensional space. This subspace will fail miserably if the x_i in the input does not equal to any of the recorded values since optimization and regularization measures only tailored the algorithm given chosen inputs. However, because most of the input in real-world applications will be one of the recorded values, this induced-overfit method makes the algorithm easier to train with higher precision. Figure (3) presents the structure of subspace learning algorithm when applied to workspace calculation.

3) Time Complexity: The time complexity of subspace learning is defined as the asymptotic behavior of the runtime
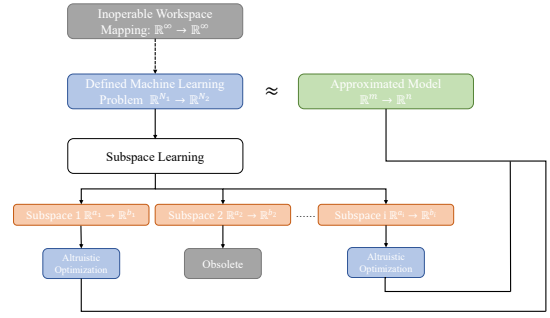

Fig. 3. Structure of subspace learning in manipulator workspace problem

of the forward prorogation graph as the length of the vector describing a subspace increases. It does not concern with the training time. Since the connections within the hidden layer are constant, assuming all operations between two nodes take the same time, a linear increase in the length of input vector only causes a linear increase in the connections between the first two and a cubic increase for the last two layers. More specifically, if k connections already exist in $\mathcal{G}_{new}$ and a_1 new elements are added to the input vector that leads to a_2 new elements added to the output layer. If there are m_1 nodes in the first layer and m_2 nodes in the last layer of the graph, the number of connections after the increase can be expressed as:

$$k_{new} = k + a_1 m_1 + a_2 m_2 \tag{18}$$

Assuming constant time of the computations between nodes in the graph, the run-time of the algorithm can be also expressed as a function of the input vector length n:

$$f(n) = c_{core} + c_1 n + c_2 g(n) \tag{19}$$

Where $g(n)$ is a polynomial in n with a degree of 3. This exists because subspace learning still has characteristic analogous to coordinate mapping used in classical methods.

Over the long run, this has a time complexity of $O(n^3)$. Although on the first sight it's still expensive, locally subspace learning drastically decreases the run time by substituting complex operations by the simplest ones like addition, multiplication and exponentiation.

V. Experimental Setup: 6-DOF Serial-link Manipulator with Spherical Wrist

The 6-DOF serial-link manipulator with a spherical wrist is arguably the most prevalent type of serial-link manipulators in the industry due to its high flexibility and the existence of an analytic solution to the inverse kinematics. The kinematic structure of this type of robot is included below:

j	θ	d	a	α
1	q_1	0	0	$\frac{\pi}{2}$
2	q_2	0	a_1	0
3	q_3	d_1	a_2	$-\frac{\pi}{2}$
4	q_4	d_2	0	$\frac{\pi}{2}$
5	q_5	0	0	$-\frac{\pi}{2}$
6	q_6	0	0	0

In the table, d_1, d_2 and a_1 describes the length of three major parts of the manipulator, and a_2 describes the shoulder offset that may not apply to some of the variants. To transfer the manipulator worksapce problem to a machine learning problem, the design of the input distribution $p(\mathbf{x})$, and its corresponding subspaces are important. The remark here is that there are two separate steps in this process: defining a solvable machine learning problem from the inoperable infinite workspace mapping, and creating its subspaces. The former includes setting the general scope (coordinate information of the nodes on the cuboid partitions and precision) and the degree-of-freedom of the manipulator (the length of $\begin{bmatrix} \mathbf{r}_d & \mathbf{r}_a & \mathbf{r}_\alpha \end{bmatrix}$), while the latter specifies the exact kinematic structure and sometimes crop part of the original output for the sake of dimensionality reduction in training.

For the specification of the kinematic model:

$$\mathbf{x} = \begin{bmatrix} \mathbf{r}_d & \mathbf{r}_a & \mathbf{r}_\alpha & \mathbf{n}_x & \mathbf{n}_y & \mathbf{n}_z & \mathbf{n}_i \end{bmatrix}^\top$$
$$r_{d_i} \sim \mathcal{U}(0,\beta)|i \in \{3,4\}$$
$$r_{d_i} \sim \delta(0)|i \in \{1,2,5,6\}$$
$$r_{a_i} \sim \mathcal{U}(0,\beta)|i \in \{2,3\}$$
$$r_{a_i} \sim \delta(0)|i \in \{1,4,5,6\}$$
$$r_\alpha \sim \begin{bmatrix} \delta(\frac{\pi}{2}) & \delta(0) & \delta(-\frac{\pi}{2}) & \delta(\frac{\pi}{2}) & \delta(-\frac{\pi}{2}) & \delta(0) \end{bmatrix}^\top$$

(20)

For constant orientation workspace,

$$n_{x_k} \sim \delta(n_{x_k})$$
$$n_{y_k} \sim \delta(n_{y_k})$$
$$n_{z_k} \sim \delta(n_{z_k})$$
$$n_{i_k} \sim \delta(n_{i_k})$$
$$\mathbf{n}_x \in \mathbb{R}^{l_x}| -2\beta \le n_{x_i} \le 2\beta, n_{x_{i+1}} - n_{x_i} = \Delta x$$
$$\mathbf{n}_y \in \mathbb{R}^{l_y}| -2\beta \le n_{y_i} \le 2\beta, n_{y_{i+1}} - n_{y_i} = \Delta y$$
$$\mathbf{n}_z \in \mathbb{R}^{l_z}| -2\beta \le n_{z_i} \le 2\beta, n_{z_{i+1}} - n_{z_i} = \Delta z$$
$$\mathbf{n}_i \in \mathbb{S}^3$$

(21)

All permutation of states in $\mathbf{n}_x$, $\mathbf{n}_y$ and $\mathbf{n}_z$ defines a point in $\mathbb{R}^3$, and the states in $\mathbf{n}_i$ defines a point in $SO(3)$ according to Equation (2).

β is an arbitrary value in $\mathbb{R}$. It does not effect the algorithm since scaling the D-H parameter of a manipulator will result in the scaling of the $\mathbb{R}^3$ part of the workspace in the same manner. In the training sessions of this paper, β is set to be 0.5.

For orientation workspace, the input vector is defined in a similar way except for a few differences shown below:

$$\mathbf{n}_x \in \mathbb{S}^{l_x}| -2\beta \le n_{x_i} \le 2\beta, n_{x_{i+1}} - n_{x_i} = \Delta x$$
$$\mathbf{n}_y \in \mathbb{S}^{l_y}| -2\beta \le n_{y_i} \le 2\beta, n_{y_{i+1}} - n_{y_i} = \Delta y$$
$$\mathbf{n}_z \in \mathbb{S}^{l_z}| -2\beta \le n_{z_i} \le 2\beta, n_{z_{i+1}} - n_{z_i} = \Delta z$$
$$\mathbf{n}_i \in \mathbb{R}^3$$

(22)

The design of the input vector in the spherical wrist case can be seen as a subspace of the machine learning problem defined in a generic 6-DOF manipulator, where states of $\mathbf{r}_d, \mathbf{r}_a$ are in $\mathbb{R}$ and states of $\mathbf{r}_\alpha$ are in $\mathbb{S}$.

Deep neural networks are constructed in MATLAB® R2017b with functions of feedforward networks in the Neural Network Toolbox. Experience are generated in the same environment as well with the implementation of algorithm in Appendix C. Results are compared between different subspaces of the two types of workspaces, different depth and width, different levels of stochasticity, and different methods of optimization and regularization. Since the outputs of the network are in $\mathbb{R}^n$ while the ideal states are in $\mathbb{B}^n$, a simple fine-tuning function is created to cluster the outputs into 0s and 1s. A part of the discussion is focused on the choice of the threshold value in this function. Also, a typical subspace learning algorithm is compared to the classical method in terms of run-time.

VI. DISCUSSION
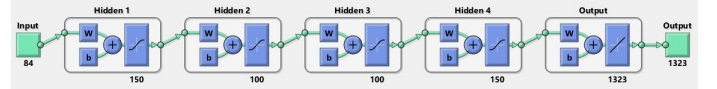
A. Analysis of a particular Subspace

Fig. 4. Feedforward neural network

The first subspace is constructed to approximate the constant orientation workspace mapping of a class of 6-DOF serial-link spherical wrist manipulators. The x,y,z-range are all on the interval $[-1,1]$, with $\Delta x = \Delta y = \Delta z = 0.1$. The orientation for the constant orientation workspace is $[0,0,0]$, which translates to $\mathbf{I}_3$ in terms of $SO(3)$. The embedding takes place both in the feature space and the output space, where for the latter only 5293-th to 6615-th elements are selected to be learnt by the deep neural network. This represents the middle section of the original cuboid partitions. A four-layer fully connected feedforward network is constructed in MATLAB®, with hyperbolic tangent function as activation function and an additional affine layer before output. Figure (4) shows the architecture of the neural network. Training of the parameters is done by a batch descent of 1×10^5 samples, and the implementation of [26]'s resilient backpropagation. The batch is then divided randomly into training set, validation set and testing set, with the ratio of $0.7 : 0.15 : 0.15$. Figure (5) presents the change in gradient during the first 200 epochs.
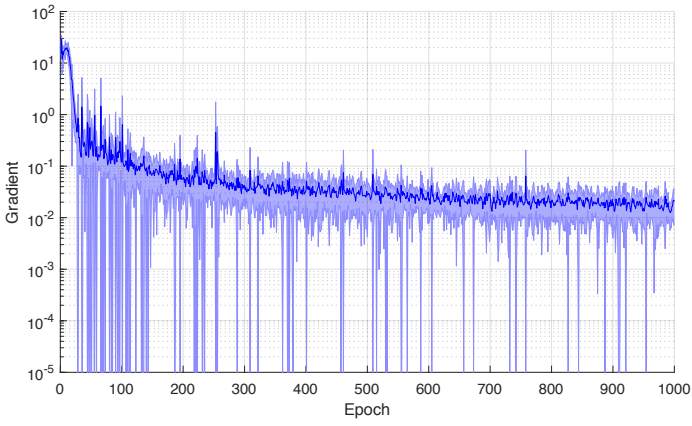
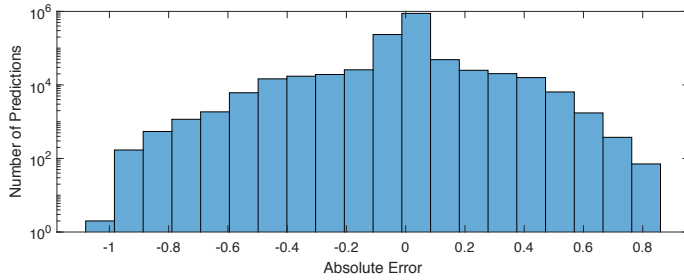Fig. 5. Average gradient ($\pm\sigma$) of 10 training sessions

Fig. 6. Average error of predictions on a test set of 1000 examples

Fig. 7. DNN-generated workspace section of 25 random manipulators in the test set

The subspace embedding reaches high accuracy during the first 200 gradient updates and continues to converge. The result is repeated for 10 training sessions, each with 10^3 epochs done on the training set. Figure (6) presents the prediction error of an independent test set of 10^3 samples evaluated on the best performing session after training. With a filter function of threshold value 0.5, the network is able to predict over 99% of the samples correctly. On average, the network has a F-measure of 0.9665 with a standard deviation of 0.0045.

B. Different subspaces with the same dimension

To verify the presumption that subspaces with the same size are similarly easy to train, seven subspace models are created with equal dimensionality according to section 6.1, except that each subspace specialize in the mapping of a particular portion in $\mathbb{R}^3$. The network still has a depth of 4, except that the number of neurons in the first and last hidden layer is reduced to 125. Figure (8) shows that some subspaces suits better to the optimization method than others when trained on a batch of 6×10^4 samples, which highlights the need of embedding to apply subspace-specific methods. Furthermore, Figure (9) suggests that with the same gradient updates, the middle subspaces has a higher precision, possibly due to the sparse nature of positives in nodes far away from the origin of the manipulator. Figure (10) then confirms that the computer run time of trained network does not deviate significantly between subspaces.
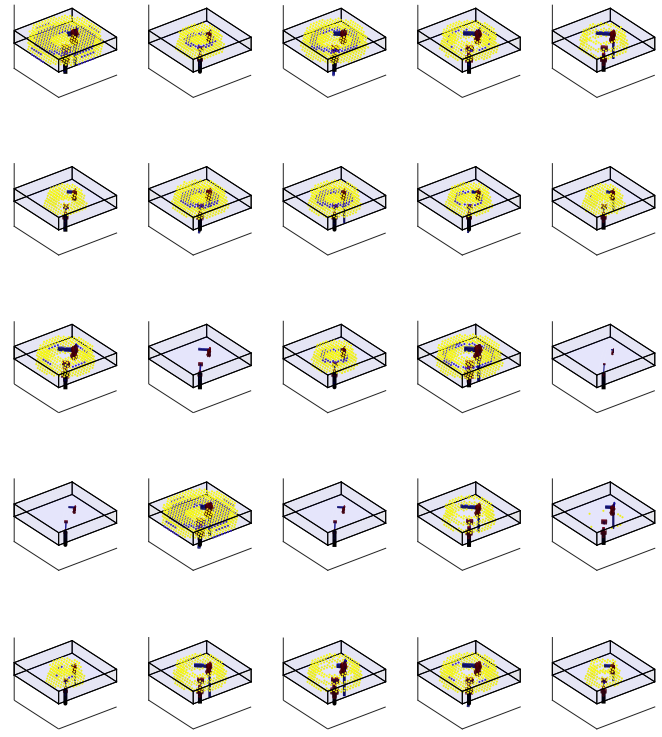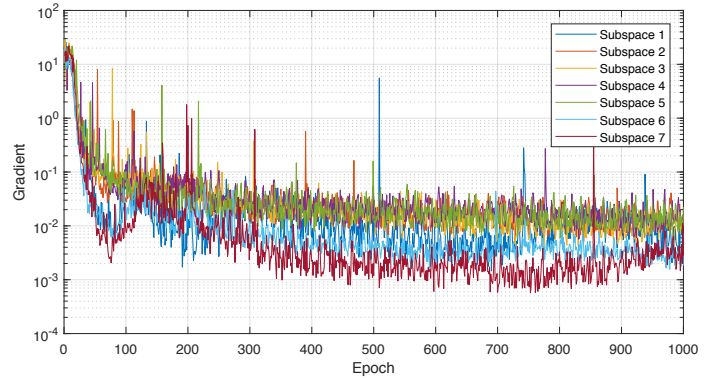

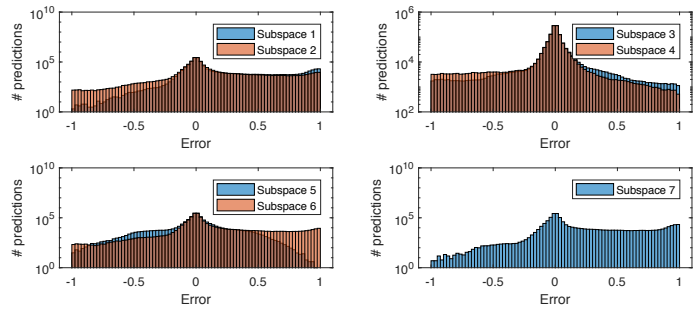Fig. 8. Subspace training of equal dimension

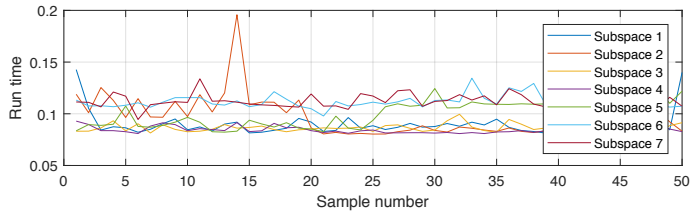Fig. 9. Histogram of prediction errors

Fig. 10. Run time of subspace models

Subspace learning enables models with small hypothesis space to mimic higher-dimensional distributions with several sets of parameters. In this case, it is possible to generate the full cuboid constant orientation workspace of the manipulator from the collaboration of the 7 subspaces that were trained separately. Figure (11) presents the deep-neural-network-generated result of PUMA560 workspace.
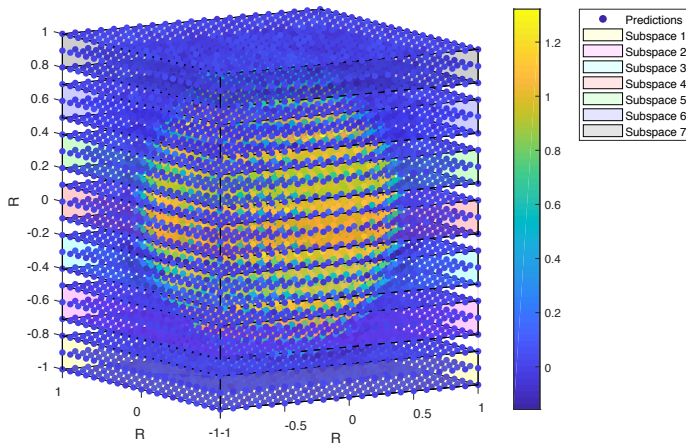

Fig. 11. Workspace of PUMA560 robot generated by 7 subspace models

C. Other Workspaces

TABLE I
F-MEASURES OF OTHER SUBSAPCE MODELS

Workspace	Δ	Size of $y \in \mathbb{B}^n$	F-measure ($\pm\sigma_f$)	Batch Size
$\mathbb{R}^3$	1	27	1.000 ± 0.00	6×10^4
$\mathbb{R}^3$	0.5	125	0.9967 ± 0.03	6×10^4
$\mathbb{R}^3$	0.1	1323	0.9221 ± 0.14	6×10^4
$\mathbb{R}^3$	0.1	3086	0.8493 ± 0.19	3×10^4
$\mathbb{R}^3$	0.1	9261	0.9038 ± 0.15	1.5×10^4
$SO(3)$	$\frac{\pi}{3}$	343	0.9981 ± 0.01	6×10^4
$SO(3)$	$\frac{\pi}{8}$	4913	0.9779 ± 0.13	1.5×10^4
$SO(3)$	$\frac{\pi}{10}$	9261	0.9920 ± 0.03	1.5×10^4

According to Table 1, for both constant orientation workspace and orientation workspace, the model reaches relatively high precision in predicting the outputs. Though beyond the scope of this paper, it is believed that a deep neural network is also capable of predicting the constant orientation workspace of an underactuated serial-link manipulator.

D. Computer Run-time

When compared to traditional discretization methods, subspace learning algorithms show exceptionally low running time. The results from Figure (12) is recorded by the *tic* / *toc* functions in MATLAB® R2017b, for implementations of both algorithms. The platform is Windows 10, and the calculations are done by a MEX worker on Intel® Xeon® CPU E3-1505M v5 @ 2.80GHz.
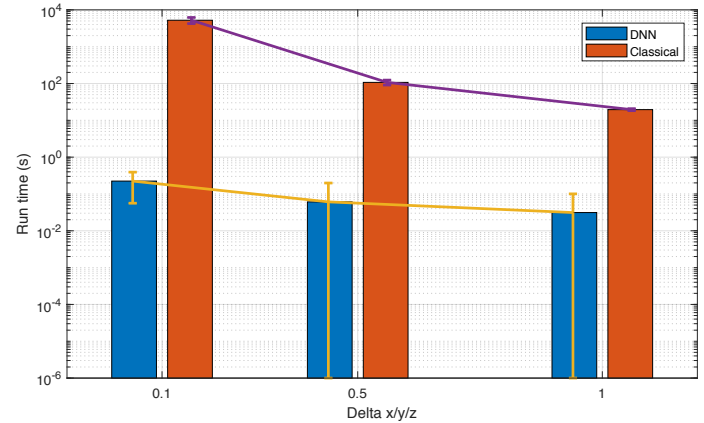

Fig. 12. Run time ($\pm\sigma$) of SL model (6-DOF, $\mathbb{R}^3$) compared to the classical algorithm

E. Optimization Methods and Performance Measure

The network in section 6.1 with layer size of $[125, 100, 100, 125]$ is retrained with varied optimization techniques and loss measure, and their F-measures are taken from the precision and recall of the final test set (10^3 samples). Table (2) presents the comparison when the network is trained on a batch of 6×10^4 samples. In general, resilient backpropagation ensures better convergence given the same number of gradient updates.

TABLE II
EFFECTS OF OPTIMIZATION METHODS AND PERFORMANCE MEASURES ON MODEL'S ACCURACY

Optimization Method	Performance	F-measure ($\pm\sigma$)
Conj. Gradient w. P/B Restarts	Cross entropy	0.3661 ± 0.2031
Variable Learning Rate G.D.	Cross entropy	0.3547 ± 0.2323
One Step Secant	Cross entropy	0.3574 ± 0.2337
Resilient Backpropagation	Cross entropy	0.3573 ± 0.2337
Scaled Conj. Gradient	Cross entropy	0.3186 ± 0.1802
Conj. Gradient w. P/B Restarts	MSE	0.6686 ± 0.3068
Variable Learning Rate G.D.	MSE	0.2398 ± 0.1081
One Step Secant	MSE	0.6228 ± 0.2624
Resilient Backpropagation	**MSE**	$\mathbf{0.9298 \pm 0.1258}$
Scaled Conj. Gradient	MSE	0.5938 ± 0.3131
Conj. Gradient w. P/B Restarts	MAE	0.3658 ± 0.1629
Variable Learning Rate G.D.	MAE	0.2256 ± 0.1011
One Step Secant	MAE	0.2778 ± 0.1178
Resilient Backpropagation	MAE	0.6573 ± 0.2388
Scaled Conj. Gradient	MAE	0.3543 ± 0.1366

F. Subspace Learning

Finally, separate models are trained with the same architecture but various subspace size/no subspace to determine the effect of subspace embedding on convergence speed and accuracy. With a three-layer fully connected feedforward network of 80 neurons at each layer, three models were created where the first model predicts the mapping in section 6.1, the second predicts the constant orientation workspace of manipulator at an arbitray orientation in $SO(3)$, and the third prrdicts an arbitrary 6-DOF manipulator's constant orientaiton workspace at $[0.312, 5.6719, 5.936]$. The last model is equivalent to no subspace embedding. All three models are tested on all test sets for subspace embedding, workspace embedding and no subspace. The results can be found in Table (III).

TABLE III
PRECISION AND RECALL OF MODELS ($\pm\sigma$)

Embedding	Test	Precision	Recall	F-measure
No subspace	n.s.	NaN	NaN	NaN
No subspace	w.e.	0.22 ± 0.18	0.35 ± 0.13	0.24 ± 0.14
No subspace	s.e.	0.01 ± 0.23	0.00 ± 0.00	0.01 ± 0.01
Workspace	w.e.	0.01 ± 0.23	0.00 ± 0.00	0.01 ± 0.01
Workspace	s.e.	0.85 ± 0.17	0.58 ± 0.42	0.81 ± 0.21
Workspace	n.s.	NaN	NaN	NaN
Subspace	**s.e.**	**0.93 ± 0.13**	**0.76 ± 0.37**	**0.91 ± 0.17**
Subspace	w.e.	0.81 ± 0.17	0.72 ± 0.35	0.82 ± 0.18
Subspace	n.s.	NaN	NaN	NaN

Interestingly, the subspace embedding algorithm also generalizes to arbitrary orientations in $SO(3)$. And when the task is focused on the specific subspace, subspace embedding outperforms the larger workspace embedding as well as the no subspace method.

G. Future Works

The threshold function presented above is a naive binary classification based on one scalar value, which could potentially limit the ability of the network to generalize. Future works can thus focus on turning the classification problem into a machine learning problem, or see the workspace mapping itself as a classification rather than the regression case introduced in this work. The creation of subspace learning is mostly motivated by pragmatic concerns in the robot kinematics field, and more theoretical proofs are needed to back up the claims of improved precision and faster convergence. The mapping can also be extended to other types of workspaces, other types of robots (parallel/nonholonomic) or even be improved to include manipulability measure.

VII. CONCLUSION

In this work, a modified version of subspace embedding algorithm is presented where it learns the workspace mapping of manipulators from D-H and scope parameters to a binary map. While the usage of subspace facilitates convergence, the general deep learning approach greatly reduces the computational cost by turning complex operations into basic arithmetic

that took places between adjacent neurons. Subspace learning explores the feasibility of approximating inverse kinematics of robots at a three-dimensional level, comparing to the conventional attempts of converging towards a single solution; and, in a broader picture, this work opens up possibilities of reducing industrial task computational cost through methods in artificial intelligence.

NOMENCLATURE

$\mathbf{A}$	A tensor
$\mathbf{A}^T$	Transpose of matrix $\mathbf{A}$
$\det(\mathbf{A})$	Determinant of $\mathbf{A}$
$p(\mathbf{x})$	A probability distribution over a continuous variable
$\mathbf{x} \sim p$	Random variable x has distribution p
$\mathbb{E}_{x\sim p}[f(x)]$	Expectation of $f(x)$ with respect to $P(x)$
$\mathcal{N}(\boldsymbol{x}; \mu, \sigma)$	Gaussian distribution over $\boldsymbol{x}$ with mean μ and covariance σ
$\{a, b, c\}$ or $\mathbb{F}$	A set
$\mathbb{B}$	Boolean domain
$SO(n)$	Special orthogonal group, the set of all orientations in n dimensions
$\boldsymbol{R}$	Orthogonal rotation matrix, $\boldsymbol{R} \in SO(3)$
$\mathcal{G}$	A Graph
$Pa_{\mathcal{G}}(x_i)$	The parents of x_i in $\mathcal{G}$
$f(\boldsymbol{x}; \theta)$	A function of $\boldsymbol{x}$ parametrized by θ
$\mathbf{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$
ξ	Abstract representation of 3-dimensional Cartesian pose
$\xi(\boldsymbol{a}, \boldsymbol{R})$	Pose construction function that takes in a position vector and an orientation matrix and returns a pose in $SE(3)$
$\mathcal{K}(\mathbf{q})$	Forward kinematic function of a serial-link robot
$\mathcal{K}^{-1}(\xi)$	Inverse kinematics
$\boldsymbol{x}^{(i)}$	The i-th example from a dataset
$\boldsymbol{y}^{(i)}$	The target associated with $\boldsymbol{x}^{(i)}$ for supervised learning
$\delta(\mathbf{x}; \mu)$ or $\delta(\mu)$	Dirac delta distribution of a scalar random variable centered at μ
c	Constant with significance indicated by context
$\mathcal{U}(a_{min}, a_{max})$	Uniform distribution of a random variable on the interval $[a_{min}, a_{max}]$

REFERENCES

[1] M. Ceccarelli and E. Ottaviano, "A workspace evaluation of an eclipse robot," *Robotica*, vol. 20, no. 3, pp. 299–313, may 2002.

[2] T. W. Lee and D. C. H. Yang, "On the evaluation of manipulator workspace," *Journal of Mechanisms Transmissions and Automation in Design*, vol. 105, no. 1, p. 70, 1983. [Online]. Available: https://doi.org/10.1115/1.3267350

[3] G. Castelli, E. Ottaviano, and M. Ceccarelli, "A fairly general algorithm to evaluate workspace characteristics of serial and parallel manipulators," *Mechanics Based Design of Structures and Machines*, vol. 36, no. 1, pp. 14–33, 2008. [Online]. Available: https://doi.org/10.1080/15397730701729478

[4] Y. Jin, I.-M. Chen, and G. Yang, "Workspace evaluation of manipulators through finite-partition of se(3)," *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 4, pp. 850 – 859, 2011, conference papers of Flexible Automation and Intelligent Manufacturing.

[5] A. Peidró, Óscar Reinoso, A. Gil, J. M. Marín, and L. Payá, "An improved monte carlo method based on gaussian growth to calculate the workspace of robots," *Engineering Applications of Artificial Intelligence*, vol. 64, no. Supplement C, pp. 197 – 207, 2017.

[6] I. Tanase, T. Itul, E. Campean, and A. Pisla, *Workspace Identification Using Neural Network for an Optimal Designed 2-DOF Orientation Parallel Device*. Dordrecht: Springer Netherlands, 2013, pp. 159–167.

[7] Z. E. Kuzeci, V. E. Omurlu, H. Alp, and I. Ozkol, "Workspace analysis of parallel mechanisms through neural networks and genetic algorithms," in *2012 12th IEEE International Workshop on Advanced Motion Control (AMC)*, March 2012, pp. 1–6.

[8] E. Campean, T. P. Itul, I. Tanase, and A. Pisla, "Workspace generation for a 2 - DOF parallel mechanism using neural networks," *Applied Mechanics and Materials*, vol. 162, pp. 121–130, mar 2012.

[9] H. Alp, E. Anli, and İ. Özkol, "Neural network algorithm for workspace analysis of a parallel mechanism," *Aircraft Engineering and Aerospace Technology*, vol. 79, no. 1, pp. 35–44, jan 2007.

[10] Z. Wang, S. Ji, Y. Li, and Y. Wan, "A unified algorithm to determine the reachable and dexterous workspace of parallel manipulators," *Robotics and Computer-Integrated Manufacturing*, vol. 26, no. 5, pp. 454–460, oct 2010.

[11] Y. Cao, K. Lu, X. Li, and Y. Zang, "Accurate numerical methods for computing 2d and 3d robot workspace," *International Journal of Advanced Robotic Systems*, vol. 8, no. 6, p. 76, 2011. [Online]. Available: https://doi.org/10.5772/45686

[12] S. Herrero, T. Mannheim, I. Prause, C. Pinto, B. Corves, and O. Altuzarra, "Enhancing the useful workspace of a reconfigurable parallel manipulator by grasp point optimization," *Robotics and Computer-Integrated Manufacturing*, vol. 31, pp. 51–60, feb 2015.

[13] E. Macho, O. Altuzarra, E. Amezua, and A. Hernandez, "Obtaining configuration space and singularity maps for parallel manipulators," *Mechanism and Machine Theory*, vol. 44, no. 11, pp. 2110–2125, 2009.

[14] D. Pisla, A. Szilaghyi, C. Vaida, and N. Plitea, "Kinematics and workspace modeling of a new hybrid robot used in minimally invasive surgery," *Robotics and Computer-Integrated Manufacturing*, vol. 29, no. 2, pp. 463–474, apr 2013.

[15] A. Hay and J. Snyman, "The chord method for the determination of nonconvex workspaces of planar parallel manipulators," *Computers & Mathematics with Applications*, vol. 43, no. 8-9, pp. 1135–1151, apr 2002.

[16] R. Hartenberg, *Kinematic synthesis of linkages*. New York: McGraw-Hill, 1964.

[17] P. I. Corke, *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*, 2nd ed. Springer, 2017, iSBN 978-3-319-54413-7.

[18] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, jan 1989.

[19] ——, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks*, vol. 3, no. 5, pp. 551–560, jan 1990.

[20] Z. E. Kuzeci, V. E. Omurlu, H. Alp, and I. Ozkol, "Workspace analysis of parallel mechanisms through neural networks and genetic algorithms," in *2012 12th IEEE International Workshop on Advanced Motion Control (AMC)*. IEEE, mar 2012. [Online]. Available: https://doi.org/10.1109/amc.2012.6197147

[21] Y. Feng, W. Yao-nan, and Y. Yi-min, "Inverse kinematics solution for robot manipulator based on neural network under joint subspace," *International Journal of Computers Communications & Control*, vol. 7, no. 3, p. 459, sep 2014. [Online]. Available: https://doi.org/10.15837/ijccc.2012.3.1387

[22] S. Chiaverini, L. Sciavicco, and B. Siciliano, *Control of robotic systems through singularities*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 285–295.

[23] B. Siciliano and L. Sciavicco, *Robotics: Modelling, Planning and Control (Advanced Textbooks in Control and Signal Processing)*. Springer, 2008.

[24] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, Apr 1997.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[26] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: the rprop algorithm," in *IEEE International Conference on Neural Networks*, 1993, pp. 586–591 vol.1.

VIII. Appendix

A. Hint of Analogy of Loss Expectation between Subspace Average and Original Space for a Specific Case

Consider first a deep neural network with any squashing activation function and an affine layer parameterized by corresponding weights and biases. The network can then be written in terms of composition of functions

$$\boldsymbol{y} = \underbrace{f \ldots (f(\boldsymbol{x}; \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}); \ldots; \boldsymbol{W}^{(n)}, \boldsymbol{b}^{(n)})}_{\text{n functions}} \tag{23}$$
$$= a(\boldsymbol{x}; \boldsymbol{\theta})$$

This model can then be used to approximate a mapping of $\mathbb{R}^n \to \mathbb{R}^m$. Referring to the maximum log likelihood estimation with m training samples $L(a(\boldsymbol{x}^{(n)}; \boldsymbol{\theta}), \boldsymbol{y}^{(n)})$ denotes the loss of a single example,

$$\mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}) \sim p_{model}}[L(a(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y})]$$
$$= \frac{1}{m} \sum_{n=1}^{m} L(a(\boldsymbol{x}^{(n)}; \boldsymbol{\theta}), \boldsymbol{y}^{(n)}) \tag{24}$$
$$= \frac{1}{m} \sum_{n=1}^{m} -\log p_{model}(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})$$

Assuming a multivariate Gaussian distribution for $p_{model}(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})$ with identity covariance matrix,

$$-\log p_{model}(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})$$
$$= -\log\left(\sqrt{\frac{1}{(2\pi)^m}} e^{-\frac{1}{2}\|\boldsymbol{y} - a(\boldsymbol{x}; \boldsymbol{\theta})\|^2}\right) \tag{25}$$
$$= -\log\left(\sqrt{\frac{1}{(2\pi)^m}}\right) + \frac{1}{2}\|\boldsymbol{y} - a(\boldsymbol{x}; \boldsymbol{\theta})\|^2$$

Now, given $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}; 0_{n \times 1}, \mathbf{I}_n)$, one can define a subspace as maintaining the normal nature of other elements while changing the first element to a Dirac delta distribution:

$$p(x_1) = \delta(\beta) \tag{26}$$

, where β is sampled from a Gaussian distribution $p(\beta) = \mathcal{N}(\beta; 0, 1)$. Let $L(a(\boldsymbol{x}'; \boldsymbol{\theta}'), \boldsymbol{y})$ denote the loss of an arbitrary

subspace created by making an observation of the distribution $p(\beta)$, the analogy can be drawn where

$$
\begin{aligned}
&\mathbb{E}_{\beta \sim p(\beta)}[L(a(\boldsymbol{x}'; \boldsymbol{\theta}'), \boldsymbol{y})] \\
&= -\log(\sqrt{\frac{1}{(2\pi)^m}}) + \frac{1}{2}\mathbb{E}_{\beta \sim p(\beta)}[\|\boldsymbol{y} - a([\beta, x_2', x_3' \ldots x_n']^T; \boldsymbol{\theta})\|^2] \\
&= -\log(\sqrt{\frac{1}{(2\pi)^m}}) + \frac{1}{2}\mathbb{E}_{x_1 \sim p(x_1)}[\|\boldsymbol{y} - a(\boldsymbol{x}'; \boldsymbol{\theta}')\|^2] \\
&= \mathbb{E}_{x_1 \sim p(x_1)}[L(a(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y})]
\end{aligned}
\tag{27}
$$

This suggests a possible connection between the expectation of the loss with respect to the model/empirical distribution and the expectation of all subspace model/empirical distribution expectations.

B. Time Complexity of a Classical Workspace Algorithm

According to Algorithm 2, the time complexity of $c(n)$ implemented in MATLAB® R2017b by [17] can be approximated as below:

$$
\begin{aligned}
&O(c(n)) \\
&\approx O(d(f(n))) + O(no(m(n))) \\
&\quad + O(j(n)) + O(i(m(n)) \times m^2(n)) \\
&= O(d(m(n) \times n)) + O(no(n^3)) \\
&\quad + O(m(n) \times n) + O(i(n^3) \times n^6) \\
&= O(d(n^4)) + O(n^3) + O(n^4) + O(n^6 \times n^6) \\
&= O(n^{12})
\end{aligned}
\tag{28}
$$

In the approximation, $O(m(n))$ indicates the time complexity of matrix multiplication, $O(i(n))$ indicates the time complexity of matrix inversion, $O(no(n))$ indicates the time complexity of computing the Euclidean norm of a vector, $O(f(n))$ indicates the time complexity of forward kinematic equation, $O(d(x))$ indicates the time complexity of the differential motion function.

The term $O(i(m(n)) \times m^2(n))$ refers to the $(\boldsymbol{J}^\top \boldsymbol{J} + \lambda^2 \boldsymbol{I})^{-1} \boldsymbol{J}^\top \boldsymbol{e}$ term in Algorithm 2.

Assuming elementary methods are used for matrix multiplication and L_2 norm calculation, while Gauss $-$ Jordan elimination method is used for matrix inversion,

$$
m(n) = O(n^3), no(n) = O(n), i(n) = O(n^2)
\tag{29}
$$

Thus, referring to Equation (28), the complexity of the algorithm is

$$
O((c(n))^3) + O(n^3) = O(n^{36})
\tag{30}
$$

C. Numerical Inverse Kinematics

Require: A manipulator object, starting joint configuration $\boldsymbol{q}_0$, toleration t, starting lambda λ, maximum rejection steps r_{max}, and maximum iteration steps i_{max}

1: **function** $\mathcal{K}^{-1}(\xi)$
2: $\quad \lambda \leftarrow 0.1$
3: $\quad \boldsymbol{e} \leftarrow$ differential motion corresponding to ξ and $\mathcal{K}(\boldsymbol{q}_0)$
4: $\quad \boldsymbol{q} \leftarrow q_i$
5: $\quad i \leftarrow 0$
6: $\quad r \leftarrow 0$
7: $\quad$ **while** $i \leq i_{max}$ **do**
8: $\quad\quad i \leftarrow i + 1$
9: $\quad\quad$ **if** $\|\boldsymbol{e}\| \leq t$ **then**
10: $\quad\quad\quad$ **return** $\boldsymbol{q}$
11: $\quad\quad$ **end if**
12: $\quad\quad \boldsymbol{J} \leftarrow$ manipulator Jacobian with joint angles of $\boldsymbol{q}$
13: $\quad\quad \boldsymbol{dq} \leftarrow (\boldsymbol{J}^\top \boldsymbol{J} + \lambda^2 \boldsymbol{I})^{-1} \boldsymbol{J}^\top \boldsymbol{e}$
14: $\quad\quad \boldsymbol{q}_n \leftarrow \boldsymbol{q} + \boldsymbol{dq}^\top$
15: $\quad\quad \boldsymbol{e}_n \leftarrow$ differential motion from $\mathcal{K}(\boldsymbol{q})$ to ξ
16: $\quad\quad$ **if** $\|\boldsymbol{e}_n\| < \|\boldsymbol{e}\|$ **then**
17: $\quad\quad\quad \boldsymbol{q} \leftarrow \boldsymbol{q}_n$
18: $\quad\quad\quad \boldsymbol{e} \leftarrow \boldsymbol{e}_n$
19: $\quad\quad\quad \lambda \leftarrow \frac{\lambda}{2}$ or other methods to decrease λ
20: $\quad\quad\quad r \leftarrow 0$
21: $\quad\quad$ **else**
22: $\quad\quad\quad \lambda \leftarrow 2\lambda$ or other methods to increase λ
23: $\quad\quad\quad r \leftarrow r + 1$
24: $\quad\quad\quad$ **if** $r > r_{max}$ **then return** $\emptyset$
25: $\quad\quad\quad$ **end if**
26: $\quad\quad$ **end if**
27: $\quad$ **end while**
28: $\quad$ **return** $\emptyset$
29: **end function**

D. Algorithm for Experience Generation

Require: Number of training samples intended i_f , $p(\mathbf{R})$, $p(\mathbf{N})$, where $\mathbf{R} = \begin{bmatrix} \mathbf{r}_d \\ \mathbf{r}_a \\ \mathbf{r}_\alpha \end{bmatrix}$ and $\mathbf{N} = \begin{bmatrix} \mathbf{n}_x \\ \mathbf{n}_y \\ \mathbf{n}_z \\ \mathbf{n}_i \end{bmatrix}$

Ensure: $\{\mathbb{X}, \mathbb{Y}\}$ = the training samples generated from the designed distributions

1: $i \leftarrow 1$
2: $\mathbb{X} \leftarrow \emptyset$
3: **for** $i = 1, 2, \ldots i_f$ **do**
4: $\quad$ (Re) initialize a manipulator object constructed from a random state of $\begin{bmatrix} \mathbf{r}_d \\ \mathbf{r}_a \\ \mathbf{r}_\alpha \end{bmatrix}$
5: $\quad$ (Re) initialize x_{max} , y_{max} , z_{max} , $\mathcal{K}^{-1}(\xi)$ and $\mathcal{K}(\boldsymbol{q})$ according to Algorithm 1 (Re) initialize and populate $\mathbf{P}$ with steps from Algorithm 1
6: $\quad \boldsymbol{x} \leftarrow \begin{bmatrix} \mathbf{r}_d & \mathbf{r}_a & \mathbf{r}_\alpha & \mathbf{n}_x & \mathbf{n}_y & \mathbf{n}_z & \mathbf{n}_i \end{bmatrix}^\top$
7: $\quad \boldsymbol{y} \leftarrow$ all-zero vector with size of $x_{max} \times y_{max} \times z_{max}$
8: $\quad c \leftarrow 0$
9: $\quad$ **for** $i = 1, 2, \ldots x_{max}$ **do**

```
10:            for j = 1, 2, . . . y_max do
11:                for k = 1, 2, . . . z_max do
12:                    y_c ← P_{i,j,k}
13:                    c ← c + 1
14:                end for
15:            end for
16:        end for
17:        X ← X ∪ x^{(i)}
18:        Y ← Y ∪ y^{(i)}
19: end for
```

本参赛团队声明所提交的论文是在指导老师指导下进行的研究工作和取得的研究成果。尽本团队所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果。若有不实之处，本人愿意承担一切相关责任。

The participating team stated that the submitted paper was the research work and the research results obtained under the guidance of the instructor. As far as the team is aware, in addition to the content listed in the text and the contents of the acknowledgment, the paper does not contain research results that others have published or written. If there is anything wrong, I am willing to bear all relevant responsibilities.

参赛队员：　__廖培元__　　　指导老师：　__Matthew Austin__

2018 年　　**9** 月　　**11** 日