Unsupervised Learning of a Kinematic Arm Model

Heiko Hoffmann and Ralf Möller

Cognitive Robotics, Max Planck Institute for Psychological Research,
Amalienstr. 33, D-80799 Munich, Germany
hoffmann@psy.mpg.de, moeller@psy.mpg.de

Abstract. An abstract recurrent neural network trained by an unsupervised method is applied to the kinematic control of a robot arm. The network is a novel extension of the Neural Gas vector quantization method to local principal component analysis. It represents the manifold of the training data by a collection of local linear models. In the kinematic control task, the network learns the relationship between the 6 joint angles of a simulated robot arm, the corresponding 3 end-effector coordinates, and an additional collision variable. After training, the learned approximation of the 10-dimensional manifold of the training data can be used to compute both the forward and inverse kinematics of the arm. The inverse kinematic relationship can be recalled even though it is not a function, but a one-to-many mapping.

1 Introduction

A basic task in controlling a robot arm is to establish a relation between joint angles and end-effector coordinates. A model that determines the end-effector coordinates from the joint angles is called a forward model. On the other hand, the transformation from end-effector coordinates to joint angles is established by an inverse model. While forward models are one-to-one or many-to-one mappings, inverse models can be one-to-many. Feedforward neural networks are function approximators and therefore fail to learn a one-to-many mapping [3].

Steinkühler and Cruse [10] therefore suggested to use a recurrent neural network (RNN) to solve the redundant problem. In a RNN, all variables are combined in a single pattern, and there is no distinction between input and output at training time. Output values are retrieved by completion of a partially defined pattern, the input. Steinkühler and Cruse used a hard-wired RNN with predefined connections and weights. Here, we present an abstract RNN modeling the manifold of stored patterns by unsupervised learning from the training examples.

We apply this network for learning and recall to the kinematic model of a 6 degree-of-freedom robot arm. Each collected training pattern contains 6 joint angles, 3 end-effector coordinates, and the collision state. In the training phase, a local principal component analysis (local PCA) method is used to determine a model of the manifold of the training data. In geometrical terms, it is approximated by a collection of hyper-ellipsoids. It is assumed that the training patterns lie on a lower-dimensional manifold embedded in the pattern space. In the application phase, a pattern completion is performed. The input part of a test pattern defines the offset of a constraint in the pattern space. The output is determined from the intersection of the constraint space and the approximated data manifold. The presented network has better generalization abilities than a Hopfield network [2], which only restores isolated patterns under the condition that they are uncorrelated or orthogonal.

Section 2 describes training and recall in the abstract recurrent neural network. Section 3 explains the application of the model to a kinematic arm model and presents the results, which are discussed in Sec. 4.

2 Unsupervised learning and recall in an abstract RNN

2.1 Training

The training algorithm is an extension of Neural Gas to local PCA. Neural Gas [4] is a robust vector quantization technique with soft competition between the units. It is an online method, where the model is updated after each presented pattern. A network contains N units, with unit index $k = 1, ..., N$. In Neural Gas, each unit contains a center vector $\mathbf{c}_k$. For each presentation of a training pattern $\mathbf{x}$, all centers are ranked from 0 to $N - 1$ in ascending order of their distance d_k (Euclidean in Neural Gas) to $\mathbf{x}$. The rank r_k of a center determines its individual weight $\alpha_k = \varepsilon \cdot \exp(-r_k/\varrho)$. The centers are updated with

$$\mathbf{c}_k \leftarrow \mathbf{c}_k + \alpha_k \cdot (\mathbf{x} - \mathbf{c}_k) \ . \tag{1}$$

The learning rate ε and the neighborhood range ϱ decrease exponentially during training, so that the algorithm converges and descends from soft to hard competition. The unit centers are initialized by randomly chosen examples from the training set.

We extend the units to hyper-ellipsoids defined by a center $\mathbf{c}_k$, m principal axes $\mathbf{w}_{ki}$ with half axis length $\sqrt{\lambda_{ki}}$, and a spherical complement with radius $\sqrt{\lambda_k^*}$ in the $n - m$ minor dimensions. The $\mathbf{w}_{ki}$ and λ_{ki} are the estimates of the eigenvectors and eigenvalues of the local covariance matrix of the training data. The distance measure is extended from a Euclidean distance to a normalized Mahalanobis distance plus reconstruction error [1],

$$d_k(\mathbf{x}) = \mathbf{y}_k^T \mathbf{\Lambda}_k^{-1} \mathbf{y}_k + \frac{1}{\lambda_k^*}(\boldsymbol{\xi}_k^T \boldsymbol{\xi}_k - \mathbf{y}_k^T \mathbf{y}_k) + \ln \det \mathbf{\Lambda}_k + (n - m) \ln \lambda_k^* \ . \tag{2}$$

$\boldsymbol{\xi}_k = \mathbf{x} - \mathbf{c}_k$ is the deviation between the input vector $\mathbf{x}$ and the center of the unit $\mathbf{c}_k$, and the vector $\mathbf{y}_k = \mathbf{W}_k^T \boldsymbol{\xi}_k$ contains the coordinates of $\boldsymbol{\xi}_k$ in the system of the first m vectors $\mathbf{w}_{ki}$, which are the columns of $\mathbf{W}_k$. The eigenvalues are comprised in the diagonal matrix $\mathbf{\Lambda}_k$. The second term in (2) is the reconstruction error divided by λ_k^*. The logarithmic terms take care of the normalization. λ_k^* depends on the estimate of the residual variance σ_k^2 which is updated according to

$$\sigma_k^2 \leftarrow \sigma_k^2 + \alpha_k \cdot (\boldsymbol{\xi}_k^T \boldsymbol{\xi}_k - \mathbf{y}_k^T \mathbf{y}_k - \sigma_k^2) \ . \tag{3}$$

The residual variance is evenly distributed among all $n-m$ minor dimensions by

$$\lambda_k^* = \frac{\sigma_k^2}{n-m} \ . \tag{4}$$

To adjust the principal axes and their lengths, one step of an online PCA method is performed:

$$\mathbf{W}_k, \mathbf{\Lambda}_k \leftarrow \mathrm{PCA}\{\mathbf{W}_k, \mathbf{\Lambda}_k, \boldsymbol{\xi}_k, \alpha_k\} \ . \tag{5}$$

For simplicity, we omit the index k in the rest of this section. We use a PCA algorithm similar to RRLSA [6]. RRLSA is a sequential network of single-neuron principal component analyzers based on deflation of the input vector [9, 8]. While the $\mathbf{w}_i$ are normalized to unit length, internally the algorithm works with unnormalized $\tilde{\mathbf{w}}_i$,

$$\tilde{\mathbf{w}}_i \leftarrow \tilde{\mathbf{w}}_i + \alpha \cdot (\boldsymbol{\xi}^{(i)} y_i - \tilde{\mathbf{w}}_i), \quad i = 1, \ldots, m \ , \tag{6}$$

where a neuron i (with the weight vector $\tilde{\mathbf{w}}_i$) sees the deflated input vector $\boldsymbol{\xi}^{(i)}$,

$$\boldsymbol{\xi}^{(i+1)} = \boldsymbol{\xi}^{(i)} - \mathbf{w}_i y_i \quad \text{with} \quad \boldsymbol{\xi}^{(1)} = \boldsymbol{\xi} \ . \tag{7}$$

After each online step, the eigenvalue and eigenvector estimates are obtained from

$$\lambda_i = \|\tilde{\mathbf{w}}_i\|, \quad \mathbf{w}_i = \frac{\tilde{\mathbf{w}}_i}{\|\tilde{\mathbf{w}}_i\|}, \quad i = 1, \ldots, m \ . \tag{8}$$

The eigenvector estimates are initialized with random orthogonal vectors. The eigenvalues λ_i and the variance σ^2 are initialized with the value 1.

Since the orthogonality of $\mathbf{W}$ is not preserved for each step, the algorithm has to be combined with an orthogonalization method, here we used Gram-Schmidt [5]. Orthogonality is essential for the computation of the distance (2).

2.2 Recall

After learning, the manifold of training patterns is represented by a collection of hyper-ellipsoids with centers $\mathbf{c}_k$, direction vectors $\mathbf{w}_{ki}$, lengths $\sqrt{\lambda_{ki}}$ of the principal axes, and the complement λ_k^*. An input to the network (one part of the components of $\mathbf{p} \in \mathbb{R}^n$) defines the offset of a constraint space $\mathbf{x}(\boldsymbol{\eta})$ spanning over all possible output values:

$$\mathbf{x}(\boldsymbol{\eta}) = \mathbf{M}\boldsymbol{\eta} + \mathbf{p} \ . \tag{9}$$

$\boldsymbol{\eta}$ is a collection of q free parameters (q being the dimension of the network output) in the subspace. $\mathbf{M}$ is a $n \times q$ matrix.

Recall of the complete pattern takes place in two steps. First, for each unit k determine the point $\hat{\mathbf{x}}_k \in \mathbb{R}^n$ on the constraint subspace with smallest distance (2) to $\mathbf{c}_k$. Second, choose the unit k^* resulting in the smallest distance $d_{k^*}(\hat{\mathbf{x}}_{k^*})$. The corresponding $\hat{\mathbf{x}}_{k^*}$ yields the desired output values (see Fig. 1 A).

The distance d_k as a function of the free parameters $\boldsymbol{\eta}$ can be written as:

$$\begin{aligned} d_k(\mathbf{x}(\boldsymbol{\eta})) = {} & (\mathbf{M}\boldsymbol{\eta} + \boldsymbol{\pi}_k)^T(\mathbf{W}_k\boldsymbol{\Lambda}_k^{-1}\mathbf{W}_k^T + \frac{1}{\lambda_k^*}\{\mathbf{I} - \mathbf{W}_k\mathbf{W}_k^T\})(\mathbf{M}\boldsymbol{\eta} + \boldsymbol{\pi}_k) \\ & + \ln\det\boldsymbol{\Lambda}_k + (n-m)\ln\lambda_k^* \ , \end{aligned} \tag{10}$$

with $\boldsymbol{\pi}_k = \mathbf{p} - \mathbf{c}_k$. We derive with respect to $\boldsymbol{\eta}$:

$$\frac{\partial d_k}{\partial \boldsymbol{\eta}} = 2\,\mathbf{M}^T\mathbf{D}_k\mathbf{M}\boldsymbol{\eta} + 2\,\mathbf{M}^T\mathbf{D}_k\boldsymbol{\pi}_k \tag{11}$$

with

$$\mathbf{D}_k = \mathbf{W}_k\boldsymbol{\Lambda}_k^{-1}\mathbf{W}_k^T + \frac{1}{\lambda_k^*}\{\mathbf{I} - \mathbf{W}_k\mathbf{W}_k^T\} \ . \tag{12}$$

Setting the derivative equal to zero yields,

$$\hat{\boldsymbol{\eta}}_k = -(\mathbf{M}^T\mathbf{D}_k\mathbf{M})^{-1}\mathbf{M}^T\mathbf{D}_k(\mathbf{p} - \mathbf{c}_k) \ . \tag{13}$$

The function d is convex. Therefore, $\hat{\boldsymbol{\eta}}_k$ is closest to the center. Thus, $\hat{\mathbf{x}}_k = \mathbf{M}\hat{\boldsymbol{\eta}}_k + \mathbf{p}$. The presented algorithm always gives a unique output for a given input. This approach has the advantage over a recall mechanism using gradient descent relaxation that it does only recall the global minimum on the constraint. Local minima on a constraint may not relate to the data manifold.

Equation (9) defines a general linear contraint. In the special case of constraint planes parallel to the coordinate axes, arbitrary components can be assigned the role of input and output variables. This is exploited for the application to a kinematic robot arm control task, where the same network is used as an inverse model and as a forward model without relearning, as described in the following.

3 Kinematic Arm Model

A robot arm with 6 rotatory degrees of freedom is simulated. It corresponds to a real robot arm in our lab. Figure 1 B shows the setup of the model. An arm model with the geometry of the arm and its environment can determine a collision between different parts of the arm and between the arm and the environment.

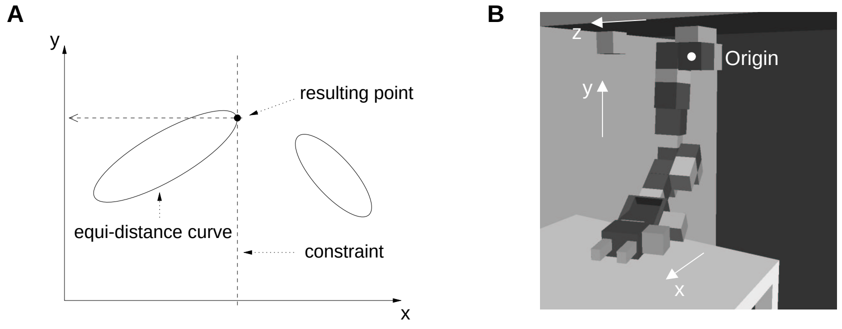

Fig. 1. *A.* Pattern recall. The input x defines the offset of a constraint space from zero. In this space the point closest to a unit center is chosen. Its y-value is the desired output. The ellipses describe the points having same distance d_k to unit k. *B.* Simulated robot arm. Location of the origin and axes of the end-effector coordinate system are shown

The training set was generated by randomly choosing 50 000 joint angle sets. Angles were chosen from a uniform interval of ±120 degrees centered at a predefined zero position. For each joint angle set, the end-effector position was determined from the arm model. It was also calculated if the angle set resulted in a collision. Thus, each training pattern is 10-dimensional and contains 6 joint angles, 3 end-effector coordinates, and one collision variable. Only training patterns with an end-effector position inside a workspace of $500 \times 500 \times 500$ mm above the table were included in the training set. Further more, the patterns were chosen such that half of the set were collision trials and half no-collision trials. All values were scaled such that they fit in a 10-dimensional cube with side length 1. Collision was encoded by a binary value 0 or 1. That means the subsets of collision patterns and no-collision patterns had zero variance in the collision variable, which in the training could lead to undefined values of the distance measure. Therefore, random noise was added to all pattern values. This noise was uniformly distributed in the interval [-0.0001,0.0001] and added whenever a pattern was drawn from the pattern set.

Three networks with different number of units ($N = 50, 100, 200$) were trained and tested. All networks were trained using the parameter set $T = 400\,000$ (the number of training steps), $m = 6$, $\varrho(0) = 10$, $\varrho(T) = 0.0001$, $\varepsilon(0) = 0.5$, $\varepsilon(T) = 0.001$. The number of principal eigenvectors m was chosen after inspecting the eigenvalues of a test run without dimension reduction ($m = 10$). Only the first 6 eigenvalues noticeably differed from zero. The training of a network with 200 units took about 33 minutes on an Athlon XP 2200+ with 1 GB RAM. After learning, the number of training patterns were approximately evenly assigned to the N different units of a network. For $N = 200$, the number of assigned patterns to a unit ranged between 32 and 403, with a mean of 250 and a standard deviation of 73. The distribution was nearly Gaussian. Table 1 shows the performance of the network with $N = 200$.

Table 1. Position and collision errors for an abstract RNN with $N = 200$ units, compared with a multilayer perceptron (MLP). Results are shown for different directions of recall, forward and inverse. The inverse model takes the desired collision state as an additional input variable (third column). Position errors are averaged over all test patterns, and are given with standard deviations. In the inverse case, the collision error is the percentage of trials deviating from the collision input value. In the forward case, it is the erroneous number of collision state predictions

Network	Direction	Input	Position error (mm)	Collision error (%)
RNN	Inverse	No collision	27 ± 15	5.1
RNN	Inverse	Collision	23 ± 13	7.7
RNN	Forward	-	44 ± 27	11.4
MLP	Inverse	No collision	310 ± 111	30.1
MLP	Forward	-	93 ± 48	13.4

For the inverse direction, the constraint space specified the end-effector coordinates and the collision state, and the network had to find the joint angles. Position errors were calculated between the desired end-effector coordinates and the ones obtained by feeding the joint angles produced by the network into the analytical geometric arm model. Collision errors were obtained in a similar way. Desired end-effector coordinates were taken from a $11 \times 11 \times 11$ grid inside the working space.

In the forward direction, the 6 joint angles were constrained, and the network had to find the end-effector coordinates and the collision state. The position error and collision prediction error were computed by directly comparing the network output with the result from the geometrical model. The test pattern set used here was randomly generated in the same way as the training set. It contained 1331 patterns (the same number as for the inverse direction).

Table 2 shows the dependence of the network performance on the number of units. Performance increases with increasing network size. Obviously, the manifold is better approximated the more units are used.

The distribution of the position error in the space of test pattern can be seen in Fig. 2. This data is taken from the inverse direction test with no-collision as input. Higher errors can be seen at the border of the workspace (e.g. right bottom corner in both images). The error flow field is not continuous. Different regions are visible.

The abstract RNN results were compared with the performance of a multilayer perceptron (MLP). We used a simple structure with one hidden layer containing 200 neurons (smaller or higher numbers did not improve the performance) and trained 2000 epochs of resilient propagation [7]. Training and test sets were the same as for our network. As can be seen in Tab. 1, the MLP cannot

Table 2. Performance of an abstract RNN for different number of units

Direction	Input	Error	$N = 50$	$N = 100$	$N = 200$
Inverse	No collision	Position (mm)	48	38	27
Inverse	No collision	Collision (%)	4.8	4.9	5.1
Inverse	Collision	Position (mm)	47	35	23
Inverse	Collision	Collision (%)	8.2	9.1	7.7
Forward	-	Position (mm)	74	56	44
Forward	-	Collision (%)	16.3	13.7	11.4

cope with the redundant inverse problem, and performs worse on the forward problem.

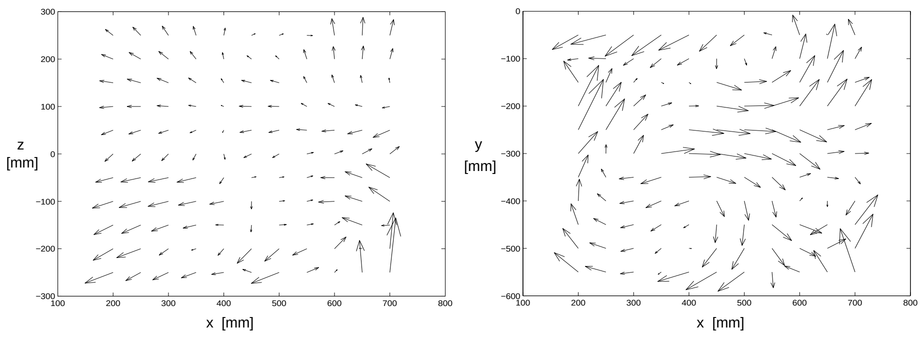

Fig. 2. The position errors of the inverse model with input 'collision' (here $N = 200$). *Left:* horizontal plane (approximately 70 mm above the table). *Right*: vertical plane through the origin ($z = 0$)

4 Discussion

We presented an abstract recurrent neural network model with unsupervised learning. The model is applied to a kinematic arm control task and could learn the direct and the inverse kinematics with one and the same network, coping with the redundancy of the inverse direction. In contrast to Steinkühler and Cruse [10], the network is not restricted to learning geometric relationships, but can include additional variables such as a collision state.

The discrete collision variable splits the training data into two parallel hyper-planes, one including all collision trials and the other all no-collision trials. The distance (2) between the two hyper-planes is much bigger than any other distance between data points inside one hyper-plane. As a result, most of the hyper-ellipsoids stay within one of the hyper-planes (for $N = 200$ all but 9 units had mid-points and eigenvectors within one of the hyper-planes).

The discontinuity of the error (as seen in Fig. 2) results from the change to the next best fitting unit (the closest hyper-ellipsoid). Different regions in the error flow field correspond to different hyper-ellipsoids. As can be seen from (9) and (13), the relation between input and output is locally linear. The local linear models do not necessarily join continuously. So far, no better solution was found to avoid these discontinuities. The vortex like flow fields in Fig. 2 probably result from the error arising from approximating a trigonometric function with a locally linear model.

Recent work focuses on learning a visuo-motor model using a real robot arm. There, the image of an object and the joint angles required to grasp the object are associated.

5 Acknowledgments

The authors would like to thank Wolfram Schenck for providing an implementation of the MLP network used in this work, Henryk Milewski for implementing the geometric arm model, and Bruno Lara for comments on the manuscript.

References

1. Hinton, G. E., Dayan, P., Revow, M.: Modeling the Manifolds of Images of Handwritten Digits. IEEE Transactions on Neural Networks **8** (1997) 65–74
2. Hopfield, J. J.: Neural Networks and Physical Systems with Emergent Collective Computational Abilities. Proceedings of the National Academy of Sciences, USA **79** (1982) 2554–2558
3. Jordan, M. I., Rumelhart, D. E.: Forward Models: Supervised Learning with a Distal Teacher. Cognitive Science **16** (1992) 307–354
4. Martinetz, T. M., Berkovich, S. G., Schulten, K. J.: "Neural-Gas" Network for Vector Quantization and its Application to Time-Series Prediction. IEEE Transactions on Neural Networks **4** (1993) 558–569
5. Möller, R.: Interlocking of Learning and Orthonormalization in RRLSA. Neurocomputing **49** (2002) 429–433
6. Ouyang, S., Bao, Z., Liao, G.-S.: Robust Recursive Least Squares Learning Algorithm for Principal Component Analysis. IEEE Transactions on Neural Networks **11** (2000) 215–221
7. Riedmiller, M., Braun, H.: A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. Proceedings of the IEEE International Conference on Neural Networks (1993) 586–591
8. Rubner, J., Tavan, P.: A Self-Organizing Network for Principal-Component Analysis. Europhys. Lett. **10** (1989) 693–698
9. Sanger, T. D.: Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network. Neural Networks **2** (1989) 459–473
10. Steinkühler, U., Cruse, H.: A Holistic Model for an Internal Representation to Control the Movement of a Manipulator with Redundant Degrees of Freedom. Biological Cybernetics **79** (1998) 457–466