Proc. Int. Joint Conf. On Neural Networks, (1):587–592, San Diego, June 1990

Non-linear Prediction with Self-organizing Maps

Jörg Walter, Helge Ritter and Klaus Schulten

Beckman-Institute and Department of Physics
University of Illinois at Urbana-Champaign, Urbana, IL 61801

Abstract:

We consider the problem of predicting highly non-linear time sequence data, where the usual approach of adaptive, linear regressive models has difficulty. For this case, we suggest the use of an adaptive covering of the state space of the process with a set of linear regressive models each of which is only locally used. We show that such an adaptive covering, together with learning of the appropriate prediction coefficients, can be realized using Kohonen's algorithm of self-organizing maps. To illustrate the method, we present simulation results for a set of benchmarking problems.

1. Introduction

Many tasks involve time sequences of data for which predictions are desired. Often information about the underlying process generating the data is only partial and incomplete, so that predictions cannot be based on a known analytical model. In such cases, one has to resort to adaptive methods, which in effect construct some approximate model of the unknown process from a sufficiently long sample of sequence data.

One of the most widely used method is based on adaptive linear regression [4]. The predicted value $\tilde{x}(t+1)$ at the next time step is given in terms of a linear combination of a fixed number $m+1$ of previous values, i.e.

$$\tilde{x}(t+1) = \sum_{i=0}^{m} a_i x(t-i) \tag{1}$$

and the initially unknown coefficients $a_i, i = 0 \ldots m$ are recursively estimated by a suitable stochastic minimization procedure for the mean square error of the predicted values [4]. If the actual process can be satisfactorily described by a linear model, the coefficients a_i will gradually converge to a set of values providing a good approximation of the process.

However, in many cases of interest, the linearity condition is not well obeyed and an accurate prediction based on a fixed set of coefficients a_i is not possible. One of the major sucessful approaches for this situation was suggested in [8,9] and has nowadays become known as the "Back-Propagation" algorithm.

In this contribution, we suggest to discretize the state space of the system in such cases, and to use a separate set of prediction coefficients a_i within each discretization cell. For sufficiently small discretization cells, the linearity assumption will then be a good approximation within each cell, and a good set of prediction coefficients, locally valid for each cell, can be found.

The necessary phase space discretization and the finding of appropriate values a_i can most conveniently be achieved using Kohonen's algorithm of self-organizing maps [1,2,5]. This approach can be viewed as an extension of previous work [3,6], where the task was to learn nonlinear kinematic and dynamic transformations for a robot by generating an adaptive discretization of its state space and finding for each discretization cell a locally valid linear approximation to the nonlinear, true transformation. Here we investigate this method for the time domain and cover the state space of the unknown system by a set of locally valid linear process models of the form (1).

In the following we report results obtained with this method for a particular set of "benchmark problems": to predict the 3d-motion of an object in a set of nonlinear potentials given by

$$V(x,y,z) = \Omega_x x^c + \Omega_y y^c + \Omega_z z^c, \quad \Omega_i = const. \quad c > 0, \text{ even integer.} \tag{2}$$

This choice includes two interesting limiting cases: $c = 2$ gives rise to an exactly solvable problem, for which (2) becomes an exact solution with a single coefficient set a_i. For $c = \infty$, free motion in a cube $-1 \leq x, y, z \leq 1$, with ideally reflecting walls results. All other choices for c interpolate between these two extremes.

In the next section, we will give a description of the algorithm. In section 3 we will present simulation results for the instructive cases $c = 2$ (i.e. exactly solvable case), $c = 4$, $c = 8$ (i.e. medium nonlinearity) and $c = 16$ (i.e. very strong nonlinearity).

2. Algorithm

For the potential (2) the motions along each of the coordinate axes are independent of each other and can, therefore, be predicted by three independent modules M_x, M_y and M_z of the same kind. The trajectory of the motion is sampled at discrete time steps, and for each coordinate direction the $m+1$ last, consecutive samples are combined into a "history vector" $\vec{x}_{hist}$, $\vec{y}_{hist}$ and $\vec{z}_{hist}$, respectively. In the following, we will restrict the discussion to the $x-$direction only, the remaining two other directions are treated in the same manner.

Each module uses "its" history vector to make the prediction for the next time step. For the $x-$module, this prediction is given by

$$\tilde{x}_{t+1} = \sum_{i=0}^{m} a_{i,\mathbf{s}} x_{t-i} = \vec{a}_{\mathbf{s}} \cdot \vec{x}_{hist}, \qquad \text{with} \qquad \vec{x}_{hist} = \left(x_t, x_{t-1}, \ldots, x_{t-m}\right)^T \tag{3}$$

As explained in the introduction, the essential aspect of the method is that the coefficient vector $\vec{a}_{\mathbf{s}}$ need not be the same for each prediction, but instead can depend on the current state (location, history) of the moving particle and is chosen each time from a discrete set labeled by the additional index $\mathbf{s}$.

Both this choice and the adaptive learning of a suitable set of vectors $\vec{a}_{\mathbf{s}}$ is implemented with a neural network algorithm based on Kohonen's "self-organizing maps" [1,2,5]: The network consists of a planar lattice [1] of "neurons", labeled by their lattice positions $\mathbf{r} \in M_x$. Each formal neuron stores one of the coefficient vectors $\vec{a}_{\mathbf{r}}$ together with a "reference vector" $\vec{w}_{\mathbf{r}}$ of dimension $n+1$. The lattice acts as a "winner-takes-all" network, whose "winner" $\mathbf{s}$provides the next prediction vector $\vec{a}_{\mathbf{s}}$ and is selected by choosing the one whose reference vector $\vec{w}_{\mathbf{s}}$ has the minimal distance to the input $\vec{x}_{stim}$:

$$||\vec{w}_{\mathbf{s}} - \vec{x}_{stim}|| = \min_{\mathbf{r} \in M} ||\vec{w}_{\mathbf{r}} - \vec{x}_{stim}|| \tag{4}$$

This partitions the input space into discretization cells or "receptive fields", each of which is given by the set of inputs selecting the same unit. The reference vector $\vec{w}_{\mathbf{r}}$ is the centroid of the cell $\mathbf{r}$. The vector $\vec{x}_{stim}$ constitutes the input to the network, and describes the current state of the object. In principle, $\vec{x}_{hist}$ can be used for this task (then $n = m$), but a slightly more convenient choice [2] is the vector

$$\vec{x}_{stim} = \left(x_t, \alpha_1 x_{t1}, \ldots, \alpha_n x_{tn}\right)^T, \qquad \alpha_i = const. \tag{5}$$

which contains x_t and the differences $x_{t\mu} := x_t - x_{t-\mu}$ as its components (the α_i are auxiliary scaling constants to bring all components roughly into the same range).

Longer trajectory estimates can be achieved by using the procedure recursively with estimated "history" values instead of $\vec{x}_{stim}$ and $\vec{x}_{hist}$.

However, for successful operation of the system suitable values for $\vec{a}_{\mathbf{s}}$ and $\vec{w}_{\mathbf{s}}$ must be determined. This can be done adaptively during a "learning phase" of the system. At each prediction step, when neuron $\mathbf{s}$ has been selected, all vectors $\vec{a}_{\mathbf{r}}$ and $\vec{w}_{\mathbf{r}}$ are adjusted, according to::

$$\vec{w}_{\mathbf{r}}^{(new)} = \vec{w}_{\mathbf{r}}^{(old)} + \varepsilon h_{\mathbf{rs}} \left(\vec{x}_{stim} - \vec{w}_{\mathbf{r}}^{(old)}\right). \tag{6}$$

[1] The two-dimensionality is suggested from the situation in the cortex, where the neurons are essentially arranged in a sheet-like fashion. For technical applications, however, more elaborate topologies (e.g. a torus) may offer advantages.

[2] motivated by the physical phase vector.

$$\vec{a}_{\mathbf{r}}^{(new)} = \vec{a}_{\mathbf{r}}^{(old)} + \varepsilon' h'_{\mathbf{rs}} \Delta\vec{a}_{\mathbf{r}}, \tag{7}$$

Here ε and ε' scale the overall size of the adaption step, and $h_{\mathbf{rs}}$ and $h'_{\mathbf{rs}}$ are "bell-shaped" functions of the lattice distance $||\mathbf{r}-\mathbf{s}||$ from the selected neuron. Both functions attain their maximum value of unity if $d = ||\mathbf{r}-\mathbf{s}||$ vanishes, and decrease to zero as d increases. Their effect is to include a whole subset of neurons in each adaptation step, thus increasing the rate of convergence, but confining adjustments to a "neighborhood region" of the selected neuron, containing only neurons that have to learn very similar parameter values and that can, therefore, profitably participate in the same adjustment step. For a more detailed discussion of this strategy, see e.g. [6,7]. A convenient choice for $h_{\mathbf{rs}}$ is a Gaussian

$$h_{\mathbf{rs}} = \exp\left(-||\mathbf{r}-\mathbf{s}||^2/\sigma^2\right) \tag{8}$$

and $h'_{\mathbf{rs}}$ likewise. Here σ determines the radius of the neighborhood region. Initially, its value is chosen fairly large for rapid, but coarse adaptation of many neurons at each step, and it is gradually decreased as the system becomes more and more fine-tuned.

To complete the description of the adaptation equations (6) and (7), we still have to specify the quantity $\Delta\vec{a}_{\mathbf{r}}$. It is chosen to minimize the quadratic prediction error

$$\sum_{t'} \left(x_{t'+1} - \vec{a}_{\mathbf{s}} \cdot \vec{x}_{hi,t'}\right)^2 = \sum_{t'} \left(x_{t'+1} - \tilde{x}_{t'+1}\right)^2, \tag{9}$$

and is given by an error correction rule of Widrow-Hoff typ [10]:

$$\Delta\vec{a}_{\mathbf{r}} = \left(x_{t+1} - \vec{a}_{\mathbf{r}} \cdot \vec{x}_{hi}\right) \frac{\vec{x}_{hi}}{||\vec{x}_{hi}||^2}. \tag{10}$$

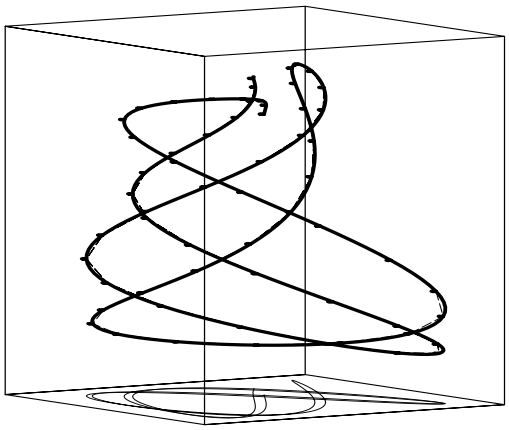
This completes our description of the algorithm.

3. Simulation Results

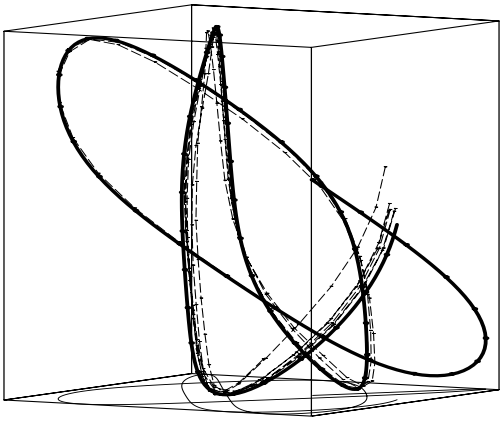
In this section, we will present some simulation results of applying the method to the 3d trajectory prediction task for potentials of the order $c = 2$, 4, 8 and 16 given by equation (2) . For comparison reasons we chose the same parameter values in all four cases : a 12×12 (=: $L\times L$) lattice of neurons with a two-dimensional task space (i.e. $n = 1$, $\alpha = 1$) and a linear prediction procedure using the last three trajectory values (i.e. $m = 2$). Parameters $\sigma = \sigma(t)$ and $\sigma' = \sigma'(t)$ are, during the first 500 learning steps, linearly decreasing from initial values of $0.7L$ and $0.8L$ to intermediate values of $0.2L$ and $0.1L$, respectively, and then more slowly to their final values of $0.05L$ and $0.02L$ during 4,500 further learning steps. Parameter $\varepsilon = \varepsilon(t)$ is changed in the same fashion, initial, intermediate and final values being given by 0.5, 0.09 and 0.001, respectively. The parameter $\varepsilon' = 0.7$ is held constant.

The vectors $\vec{w}$ are randomly initialized, and the output vectors $\vec{a}$ are initially zero (these choices are rather uncritical; the task space discretization develops for a broad parameter range rather quickly to an ordered structure allowing meaningful neighborhood cooperation).

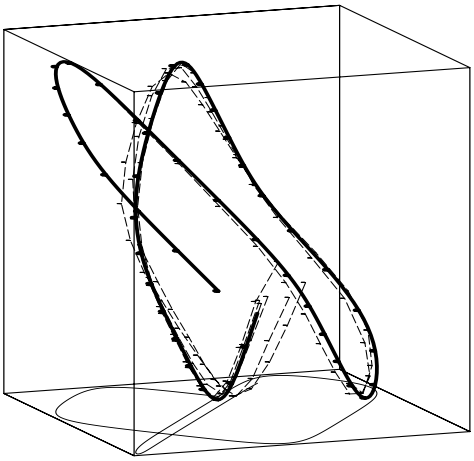
The network is trained with a series of trajectories with randomly chosen starting values. For each trial, the kinetic energy of the motion is chosen in such a way, that the trajectory remains within the cube $-1 < x, y, z < 1$. A few examples of such trajectories are shown in Fig. 1 (bold lines) together with their projections onto the base plane (shadow, thin lines). Despite their fairly intricate shape, the network can make quite accurate predictions after it has been exposed to a sufficient number of samples. For the depicted examples, the dashed lines show such predictions made by the network, and the tick marks indicate the discrete timesteps used. In the case $c = 2$, the predicted trajectory is almost indistinguishable from the true trajectory. It can be shown that *a single* coefficient vector $\vec{a}_{\mathbf{r}}$ is adequate to solve the prediction task *exactly* in this case. Therefore, all units should learn the same coefficient set, which is indeed the case. In all other cases, a single vector $\vec{a}_{\mathbf{r}}$ would not suffice for accurate predictions. Accordingly, for these cases different neurons learn different values $\vec{a}_{\mathbf{r}}$. This is shown in Figure 3 for the case c=16. Each mesh surface represents one set of coefficients $a_{i\mathbf{r}}$ ($i = 0, 1, 2$) as $\mathbf{r}$ varies over the 12×12-lattice of the x-module. Evidently, for different parts of the trajectory very different coefficient values are needed and are learnt by the network.



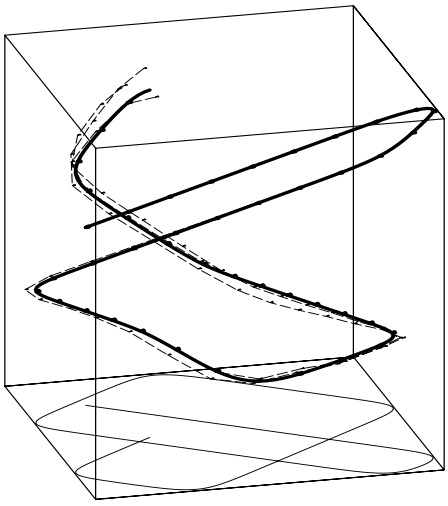
Potential c = 2



Potential c = 4



Potential c = 8



Potential c = 16

Fig. 1 a,b,c,d: Trajectory Examples (bold lines) with vertical projection onto the bottom plane (thin lines), and predictions by the network (dashed lines).

To evaluate the prediction accuracy more quantitatively, we consider the euclidean error E_μ for prediction of μ time steps ahead:

$$E_\mu = \sqrt{(x_{t+\mu} - \tilde{x}_{t+\mu})^2 + (y_{t+\mu} - \tilde{y}_{t+\mu})^2 + (z_{t+\mu} - \tilde{z}_{t+\mu})^2}. \tag{11}$$

Figure 2 a shows E_{10} (averaged over several trials) for the four cases $c = 2$, $c = 4$, $c = 8$ and $c = 16$ as learning proceeds. As can be seen, in all cases the error decays rapidly from large initial values to much smaller final values and, as would be expected, the asymptotic performance improves as one goes to smaller values of c.

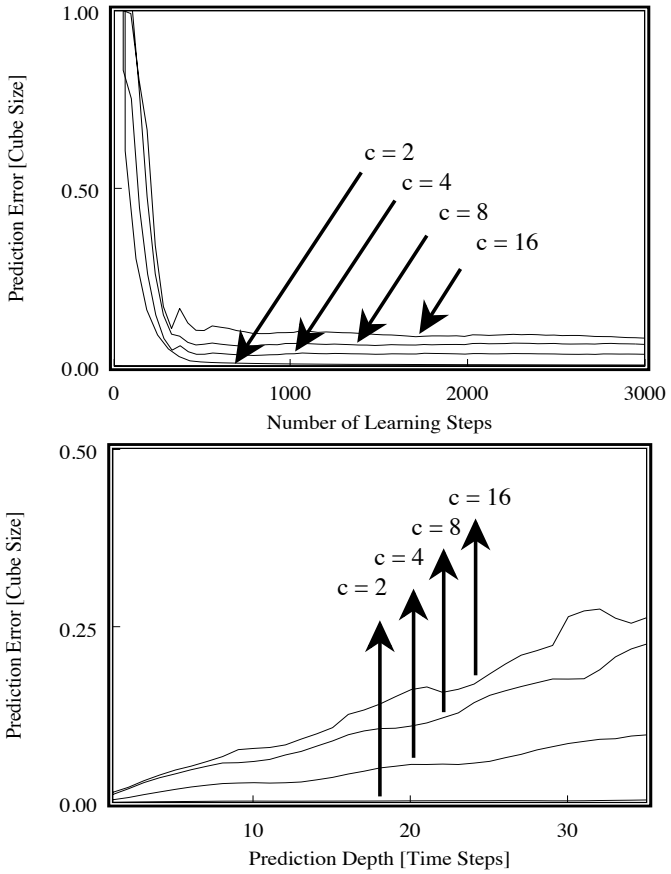


Fig. 2: (a, top) $E_{\mu=10}$ vs. number of adaption steps. (b, bottom) Error E vs. μ after learning.

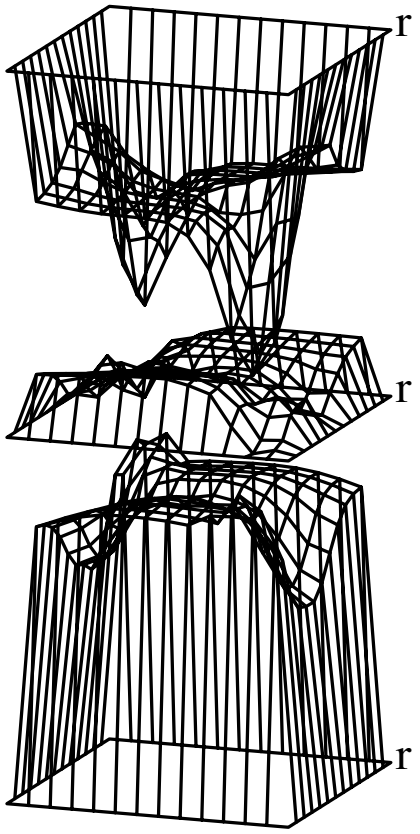


Fig. 3 a, b, c: Prediction coefficients $a_0(\mathbf{r})$, $a_1(\mathbf{r})$ and $a_2(\mathbf{r})$ of the x-module ($c = 16$) after learning.

Another way to evaluate the performance of the network is to monitor the error as a function of the prediction depth μ after learning (Figure 2 b). As can be seen, prediction is very accurate in any of the four cases, as long as μ does not grow too large. As μ increases, only the predictions for the lower order potentials remain accurate.

Finally, Figures 4 a,b show the discretization of the two-dimensional task space by the network M_x for the case $c = 16$. In both diagrams, the centers $\vec{w}_\mathbf{r}$ of the discretization cells found by the network have been "projected" into the task space. In addition, centers belonging to nearest lattice neighbors are connected by lines and a representative subset of the task space points visited by the motion is indicated by dots. Figure 4a shows the initial "random" state, Fig. 4b the state after learning. Fig.4b illustrates:

(i) Discretization centers cluster in regions with high density of dots reflecting an economical and demand-driven allocation of computational resources.

(ii) The topology conserving properties of the mapping: points close in the task space correspond to neurons close in the lattice, providing the basis for their profitable cooperation. Because similar

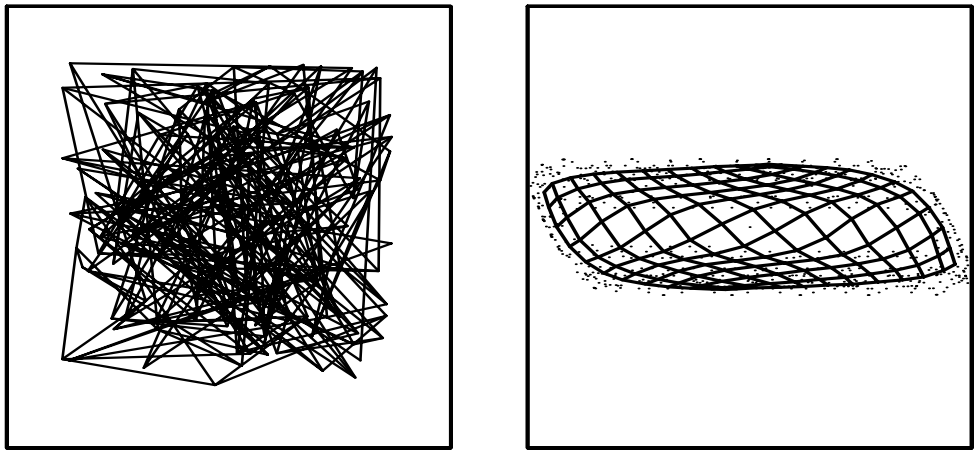


Fig. 4 a,b: Correspondence between task space and neural lattice before (left) and after learning (right).

situations require tuning to similar prediction vectors $\vec{a}_{\mathbf{r}}$, these neurons will have to learn similar vectors $\vec{a}_{\mathbf{r}}$, and can profitably participate in the same adjustment steps, which is also mirrored in the smooth surface of Fig. 3 . We found that this cooperation greatly enhances the whole system performance in respect of robustness as well as the rate of convergence.

4. Conclusion

We have investigated a new approach for the prediction of very non-linear time sequence data. The main idea is to use a Kohonen network to adaptively discretize the set of the input data, and to estimate in each discretization cell a separate set of linear prediction coefficients. We tested the method on kinetic trajectories in a set of potentials and we found reasonable performance, even with a fairly small network. In view of the generality and flexibility of the approach, an extension and investigation for more demanding tasks seems to be very promising.

5. References:

[1] Kohonen T. (1982) Self-organized formation of topographically correct feature maps. Biol. Cybern. 43:59-69.

[2] Kohonen T. (1984) Self-organization and Associative Memory. Springer Series in Information Science, Vol.8, Springer, Berlin, Heidelberg, New York.

[3] Martinetz T., Ritter H., Schulten K. (1990) Three-dimensional Neural Net for Learning Visuo-motor Coordination of a Robot Arm. IEEE Transactions on Neural Networks 1

[4] Rhodes I.B. (1971) A Tutorial Introduction to Estimation and Filtering. IEEE Transactions on Automatic Control 6:688-706

[5] Ritter H., Schulten K. (1986) Topology Conserving Mappings for Learning Motor Tasks. In: Denker J.S. (Ed.), Neural Networks for Computing, AIP Conf. Proceedings 151, pp. 376-380, Snowbird, Utah

[6] Ritter H., Martinetz T., Schulten K. (1989) Topology Conserving Maps for Learning Visuomotor-Coordination, Neural Networks 2, pp. 159-168.

[7] Ritter H., Schulten K. (1989) Convergence Properties of Kohonen's Topology Conserving Maps: Fluctuations, Stability and Dimension Selection. Biol. Cybern. 60:59-71.

[8] Rumelhart D.E., Hinton G.E., Williams R.J. (1986) Learning Representation by Back-Propagating Errors. Nature 323:533-536

[9] Werbos P. (1974) New Tools For Predictions ans Analysis in the Behavioral Sciences (Ph.D. thesis) Cambridge, Mass.: Harvard U. Committee on Applied Mathematics.

[10] Widrow B, Hoff M.E.(1960) Adaptive Switching Circuits, WESCON Conv Record, part IV, pp. 96-104.