Adding Reinforcement Learning Features to the Neural-Gas Method

M. Winter, G. Metta and G. Sandini.
LIRA-Lab, DIST, University of Genoa, viale Causa 13, I-16145 Genoa, ITALY.
sandini@lira.dist.unige.it

We propose a new neural approach for approximating function using a reinforcement-type learning: each time the network generates an output, the environment responds with the scalar distance between the delivered output and the expected one. Thus, this distance is the only information the network can use to modify the estimation of the multi-dimensional output. This reinforcement feature is embedded in a neural-gas method, taking advantages of the different facilities it offers. We detail the global algorithm and we present some simulation results in order to show the behaviour of the developed method.

1. Introduction

The original objective of this study is to develop a biologically inspired method for the control of a robotic head while performing simple tasks such as tracking a moving target. We want the robotic head to learn how to track without any *a priori* knowledge. The only information that is to be given to the head is the distance between the target and the centre of the image.

In other words, we want the neural net to approximate iteratively a function $Y = F(X)$ where $F:\Re^m \to \Re^n$. At each step *k* of the learning procedure, the neural network generates an output corresponding to an input point X_k. If we denote Ψ_k the proposed output delivered by the network, the environment sends back a reinforcement signal ρ_k, which is the distance between the delivered and the expected value:

$$\rho_k = \left\| \psi_k - F(X_k) \right\| \tag{1}$$

We consider here an off-line algorithm: the training is realized with a finite number of samples. When the learning procedure is completed, the neural network is ready to be used as a function approximator and adaptation is no longer performed.

In the next section, we explain the principle of our approach on a very simple case, and the complete algorithm is detailed in section 3. Some simulation results are presented in section 4, and section 5 is dedicated to the conclusions and perspectives of our work.

2. Principle of the new approach

In order to make the explanation easier, we consider that we just want to approximate the value of function *F* for a given input point *X*. Let's call *Y* the desired vector *.i.e.* $Y = F(X)$, and R_X the reinforcement function defined as follows:

$$\begin{aligned} R_X: \quad & \Re^n \to \Re \\ & \psi_k \to R_X(\psi_k) = \left\| \psi_k - F(X) \right\| \end{aligned} \tag{2}$$

We will distinguish between Y_k^*, the internal approximation of *Y* made by the network at the iteration step *k*, and Ψ_k the output of the neural network. The basic idea of the method consists in making a linear modelling of the function R_X in the neighbourhood of the current value Y_k^*. In other words, it is necessary to separate Ψ_k and Y_k^*: Ψ_k will be chosen in the neighbourhood of Y_k^* so that it gives information about the slope of the function R_X.

If we note $R_{X,k}^*$ the approximation of the function R_X at iteration *k*, we defined:

$$R^*_{X,k}(\psi) = r_{X,k} + \overline{R}_{X,k}\left(\psi - Y^*_k\right) \tag{3}$$

where $\overline{R}_{X,k}$ is a vector that models the orientation of the plane that is tangent to the function R_X on the point Y^*_k. The new value Y^*_{k+1} is moved downward along the slope of the function R_X, according to the linear model:

$$Y^*_{k+1} := Y^*_k + \gamma \overline{R}_{X,k} \tag{4}$$

where γ scales the size of the adaptation step. The approximation of the function R_X can be updated by applying a gradient descent method [1,2]:

$$r_{X,k+1} := r_{X,k} + \lambda_r \left[R_X(\psi_k) - R^*_{X,k}(\psi_k)\right] \tag{5}$$

$$\overline{R}_{X,k+1} := \overline{R}_{X,k} + \lambda_R \left[R_X(\psi_k) - R^*_{X,k}(\psi_k) - \overline{R}_{X,k}\left(\Psi_k - Y^*_k\right)\right]\left(\Psi_k - Y^*_k\right) \tag{6}$$

where λ_r and λ_R are parameters that allow to tune the adaptation step.

Since the input vectors Ψ_k are used during the learning phase in order to estimate the function R_X in the neighbourhood of the current value Y^*_k, it seems reasonable to choose these vectors in the neighbourhood of Y^*_k:

$$\Psi_k = Y^*_k + \sigma_n \Phi \tag{7}$$

where Φ is a noise term that is generated according to an uniform distribution between –1 and 1, and σ_n is a parameter that defines the size of the neighbourhood.

3. The Off-Line Algorithm

We described in the previous section the algorithm to estimate the value of the function *F* for a particular input *X*. The extension for modeling the complete function is then straightforward with the help of the neural-gas *'way of thinking'*: the input space of the function *F* is split dynamically into small regions during the learning. Since each of these regions is managed by a cell, we can estimate separately for each of them the output of the function, using the method described in the previous section. The precision of the estimation can be increased as much as necessary by increasing the number of input cells. Moreover, some interpolation mechanism can be added to smooth the estimated function between two regions. Each cell *i* is defined by the following quantities:

W_i : the vector that defines the position of the cell in the input space, with $dim(W_i)=m$

r_i and $\overline{R}_i$: the parameters of the model of the reinforcement signal, where we changed the subscript symbol *X* with the index *i* of the cell. We also omit the index *k* that refers to the iteration number in order to simplify the notation, although the learning algorithm is still iterative.

Following [3,4], the connections between each cells are managed with the help of variables a_{ij} which represents the *age* of the connection between the *i*-th and the *j*-th cell. When no connection is established we set $a_{ij} = -1$, otherwise the value represents the number of iterations since the connection has been created or refreshed. The learning algorithm is as follows:

{1} initialise randomly Y^*_i, A_i, r_i, $\overline{R}_i$ and W_i for all neurons. Initialise the age of all the connections: $a_{ij} = -1 \;\; \forall i, j \in D$, where D is the set of the indices of all the neurons.

{2} when an input X is received, find the indices of the two closest neurons, *i.e.*:

$$i^* = Arg \min_{i \in D} \left\| X - W_i \right\| \tag{8}$$

$$i^{**} = Arg \min_{i \in D / \{i^*\}} \left\| X - W_i \right\| \tag{9}$$

{3} generate an approximated output:

$$\Psi = Y_{i^*}^* + \sigma_n \Phi \tag{10}$$

{4} receive the reinforcement signal ρ from the environment.

{5} update of the position of the neurons in the input space:

- the closest neuron is moved towards the input X by a fraction ε_w :

$$\Delta W_{i^*} = \varepsilon_w (X - W_{i^*}) \tag{11}$$

- the cells in the neighbourhood N_{i^*} of the closest cell are also moved towards the input by a fraction ε_n :

$$\Delta W_i = \varepsilon_n (X - W_i) \qquad \forall i \in N_{i^*} \tag{12}$$

where the neighbourhood is defined as:

$$N_i = \{ j / a_{ij} \geq 0, j \neq i \} \tag{13}$$

{6} update the model of the function *R* using equations derived from (5) and (6):

$$\Delta r_{i^*} = \lambda_r \left[\rho - R_{i^*}^* (\psi) \right] \tag{14}$$

$$\Delta \overline{R}_{i^*} = \lambda_R \left[\rho - R_{i^*}^* (\psi) - \overline{R}_{i^*} \left(\Psi - Y_{i^*}^* \right) \right] \left(\Psi - Y_{i^*}^* \right) \tag{15}$$

{7} update the output of the neuron i^* :

$$\Delta Y_{i^*}^* = \gamma \overline{R}_{i^*} \tag{16}$$

{8} update the ages of the connections:

$$\begin{aligned} & a_{i^* i^*} := 0 \\ & a_{i^* j} := a_{i^* j} + 1 \qquad && \forall j \in N_{i^*} \\ & a_{i^* j} := -1 \ \text{ if } \ a_{i^* j} > a_{\max} \qquad && \forall j \in N_{i^*} \end{aligned} \tag{17}$$

where $a_{\max}$ is a threshold parameter of the method.

{9} if a stopping criterion is not fulfilled continue with step {2}.

In order to illustrate the behaviour of the modified neural net, we present in the next section some simple simulations we made using the modified neural-gas network.

4. Simulation results

We want the networks to approximate the following function:

$$F(x,y) = \begin{pmatrix} \cos\left(\frac{\pi x}{2}\right) \sin\left(\frac{\pi y}{2}\right) \\ \cos\left(\frac{\pi y}{2}\right) \sin\left(\frac{\pi x}{2}\right) \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \tag{18}$$

This function is also illustrated on figure 1. The neural networks consisted in 50 neurons, and we chose $\sigma_n = 0.5$, $\gamma = 0.01$ and $\lambda_r = \lambda_R = 0.1$. Concerning the placement of the neurons in the input space, we chose $\varepsilon_w = 0.005$, $\varepsilon_n = 0.001$ and $a_{\max} = 88$ as suggested in [4].

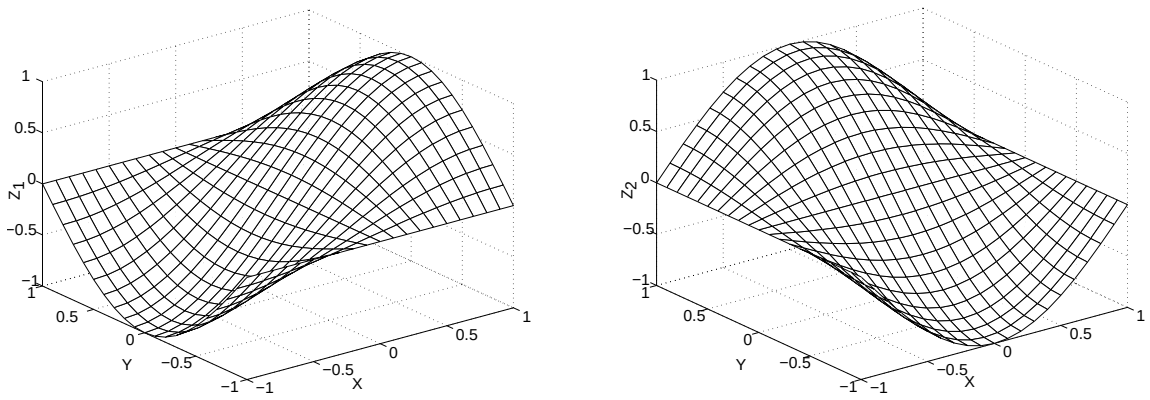

Figure 1: the function $F:\Re^2 \rightarrow \Re^2$ to be approximated

The resulting estimated function is showed in figure 2.

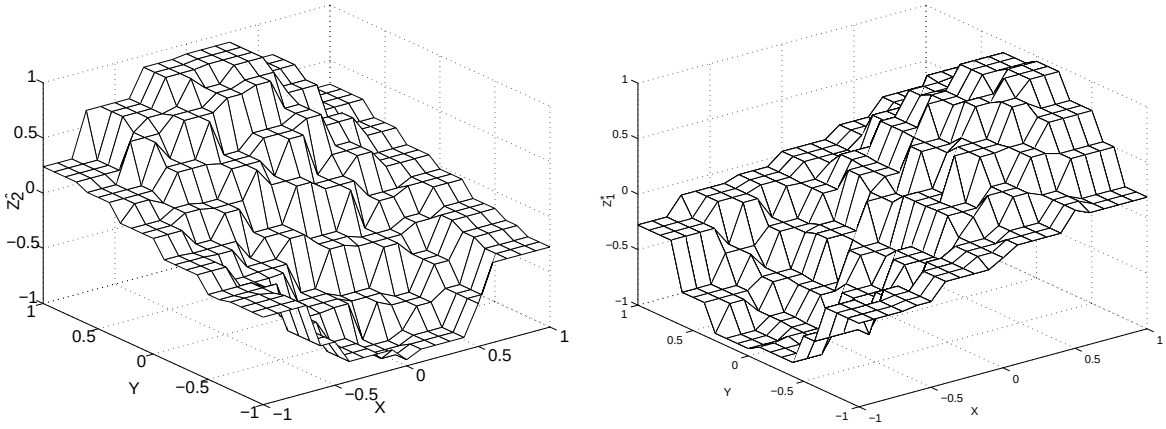

Figure 2: the approximated function

According to figure 2, we can conclude that the proposed approach behaves correctly. Of course, it is always possible to increase the precision of the method by increasing the number of neurons.

5. Conclusion

We proposed a reinforcement learning approach based on the neural-gas neural network. The simple simulations we made show that our method yields good results, and some straightforward improvements can be considered, such as smoothing the output or adding new neurons during the learning procedure. It is also easy to imagine an on-line algorithm for which the range of the noise added to the estimated vector is decreased as the reinforcement signal decreases.

[1] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten. "Neural-Gas" Network for Vector Quantization and its Application to Time-Series Prediction, *IEEE Transactions on Neural Networks*, vol. 4, No. 4, pages 558-569, July 1993.

[2] T. M. Martinetz and K. J. Schulten. A "neural-gas" network learns topologies. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 397-402. North-Holland, Amsterdam, 1991.

[3] J. A. Walter and K. J. Schulten. Implementation of self organizing neural networks for visuo-motor control of an industrial robot, *IEEE Transactions on Neural Networks*, vol. 4, No. 1, pages 86-95, 1993.

[4] B. Fritzke. Some competitive learning methods, *Technical Report from the Systems Biophysics Institute for Neural Computation*, Ruhr-Universitat Bochum, April 1997.