

CNN: a Neural Architecture that Learns Multiple Transformations of Spatial Representations

J.W.M. van Dam*, B.J.A. Kröse, F.C.A. Groen

Faculty of Mathematics and Computer Science

University of Amsterdam

Kruislaan 403

NL - 1098 SJ Amsterdam

1 Introduction

In this article we introduce a network architecture that learns transformations of spatial representations as a function of the parameters of the transformation. Whereas some biologically plausible models have recently been defined in literature, our network directly minimises the squared error between the estimated and the optimal transformed representation by calculating *correlations* between the original and the transformed representations.

In many occasions it is useful if spatial representations can be rotated and translated, e.g. when a body-centered representation of the environment is to be maintained. The environment is represented in a “neural map”, a collection of neurons organised in a lattice. Many solutions to the problem of the transformation of such neural maps have been described if the maps contain a single peaking unit (see, e.g., [Bullock et al., (1993)]). The peaking unit can, e.g., be the target for a saccadic eye movement. Much less attention has been paid to the same problem when all neurons in the map have independent activation values. These activation values can, among others, be grey values, motion values, or probabilities that the corresponding spatial positions are occupied by some obstacle (occupancy grids, see [Van Dam et al., (1993)]). Recently, [Hartman (1993)] and [Groh and Sparks (1992)] have introduced models to perform transformations of such neural maps. They both adopt the principle that activation values are gated through lateral connections. Although the models are biologically plausible, they have the obvious drawback of severe blurring of the spatial representation. In this article, we follow the approach that transformed representations can best be estimated by calculating the *correlation* between original and transformed neural maps. We introduce a Cascaded Neural Network (CNN) that learns these correlations for a restricted set of transforms. This restriction serves to limit the size of the network; it is a very plausible restriction since autonomous systems allow only restricted transformations between their internal representations.

*The investigations were supported by the Foundation for Computer Science in the Netherlands (SION) with financial support from the Netherlands Organisation for Scientific Research (NWO).

First, we discuss a network that can learn the correlation between neural maps for a single transformation. This network is then extended with *neuron networks*, each of which gives a set of these correlations as a function of the transformation parameters; a CNN is thus defined that can represent multiple transformations. Also, the learning algorithm to train the CNN is described. We conclude this text with some test results and a discussion of future work.

2 Learning the correlation between transformed neural maps

To learn the *correlation* between a neural map $\mathbf{x} = (x_1, \dots, x_n)^T$ and its transformation $\mathbf{y}^* = (y_1^*, \dots, y_m^*)^T$, define a single layer feed-forward network with input units x_1, \dots, x_n and output units y_1^*, \dots, y_m^* . Each output unit y_i^* is connected to all input units x_j with weights w_{ij} . The optimal value w_{ij}^* for these weights is given by:

$$w_{ij}^* = E[y_i^* x_j] = \int \int y_i^* x_j p(y_i^*, x_j) dy_i^* dx_j \quad (1)$$

the correlation between neurons y_i^* and x_j . Traditionally, Hebbian learning is used to train such a network. However, in [Van Dam et al., (1994)] it is shown, that the δ -rule is preferable, if the activation function of the output neurons is chosen to suit the learning samples. In our application, we have binary learning samples. This means, that a number of x_j is set to 1 and that for each $x_j = 1$, the desired output value y_i^* is set to 1 with probability w_{ij}^* . All neurons not set to 1 are set to 0. Note that these learning samples obey $E[y_i^* x_j] = w_{ij}^*$. In [Van Dam et al., (1994)] it is shown, that for binary samples we must use the activation function:

$$y_i = 1 - \prod_j (1 - w_{ij} x_j) \quad (2)$$

This activation function is given by combinatorial probability theory; it computes the probability that y_i^* is set to 1 by *at least one* x_j , which corresponds to the definition of our learning samples.

The δ -rule attempts to minimise the squared error between desired output and actual output by following the negative gradient of this error measure. With the activation function given by (2), it follows the weights should be updated according to:

$$\Delta w_{ij} = -\gamma \frac{\partial}{\partial w_{ij}} (y_i^* - y_i)^2 = \gamma (y_i^* - y_i) \cdot x_j \cdot \prod_{k \neq j} (1 - w_{ik} x_k) \quad (3)$$

More details on this method of estimating correlations are given in [Van Dam et al., (1993)].

3 Learning multiple transformations of neural maps

In the previous section it was discussed how weights w_{ij}^* can be learned for a single transformation of neural maps. Obviously, for a different transformation,

we will have different values w_{ij}^* . In this section, the network is extended to train the weights w_{ij} as a function of the transform ϕ .

The CNN network is defined as follows: with each neuron x_j , associate a neuron network \mathcal{N}_j . The inputs to this network are the parameters of the transformation, ϕ . The outputs a_i of this network are the weight values w_{ij} , estimations of the correlation between y_i^* and x_j .

The learning algorithm to train the CNN is also an extension to the learning algorithm presented in the previous section. Learning samples consist of triples $(\phi, \mathbf{y}^*, \mathbf{x})$, where the transformation of \mathbf{y}^* with respect to \mathbf{x} is given by ϕ . First, ϕ is presented to all the neuron networks in parallel. The outputs a_i of networks \mathcal{N}_j are the weight values w_{ij} . An update to these weight values is then calculated using \mathbf{y}^* and \mathbf{x} , as given in (3). The update Δw_{ij} gives the error, i.e. the difference between desired output and actual output, of neuron a_i in network \mathcal{N}_j . Given these errors, the weights of the neuron networks can then be updated in parallel¹. This learning algorithm is highly parallel by nature and therefore it is very well suited to be implemented on a parallel machine.

With the use of neuron networks, the total number of weights to be trained can quickly get out of hand. However, in the introduction it was stated that only a *restricted set* of transformations was to be learned. Therefore, it can be expected that there are many units y_i and x_j that are uncorrelated for all values of ϕ . Consequently, the corresponding outputs a_i may be pruned from the neuron network \mathcal{N}_j . Therefore, if for many transformations t we have:

$$\sum_t a_i(t) < \varepsilon_1 \quad (4)$$

unit a_i can be pruned. Naturally, if units are pruned, also a possibility should be offered to add output units to neuron networks, e.g., when a different set of transformations is to be learned. An output a_i is added to neuron network \mathcal{N}_j if it does not exist yet and if

$$\exists t \quad \delta a_i(t) > \varepsilon_2 \quad (5)$$

where δa_i gives the error in the output unit.

4 Implementation, results and future work

The CNN was tested on the following problem. Both \mathbf{y}^* and \mathbf{x} are 2D 4×4 neural maps. Map \mathbf{y}^* is rotated and translated with respect to map \mathbf{x} by a transformation ϕ that is chosen from a restricted set that allows rotations between $\frac{1}{16}\pi$ and $-\frac{1}{16}\pi$ and translations by $\frac{1}{2}$ neuron in each direction. Learning samples $(\phi, \mathbf{y}^*, \mathbf{x})$ are constructed as described in sections 3 and 2. Neuron networks \mathcal{N}_j are feed forward networks with 3 inputs for the rotational and translational components of ϕ , one layer of 20 hidden units and initially 16 output units a_i for all neurons y_i . The networks are trained with back-propagation and the δ -rule. Results are shown in figure 1.

Currently, we are investigating how we can choose an optimal architecture for the neuron networks in our CNN based on a-priori knowledge of the problem

¹This algorithm is valid, regardless of what network architecture or learning algorithm is used for the neuron network themselves.

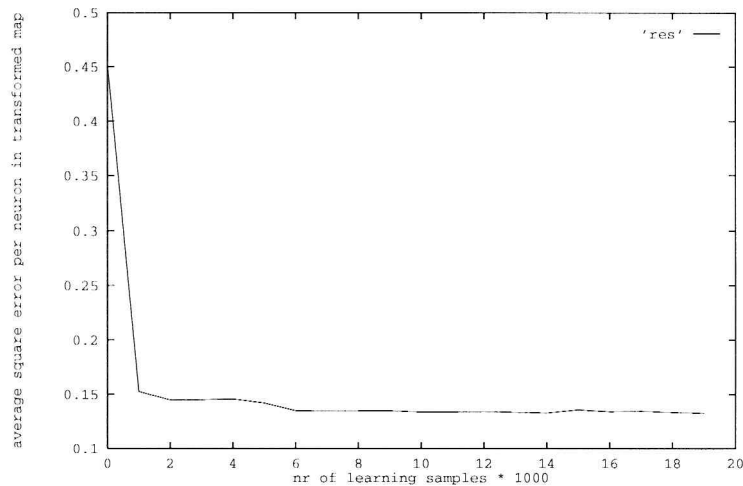


Figure 1: Results of the implementation of a CNN. The average squared error $(y_i^* - y_i)^2$ between desired and actual neuronal activation in the transformed neural map is shown as a function of the number of learning samples presented. The error is calculated over 1000 transformations. Note, that by the way samples are constructed (see section 2), the error can never decrease to 0.

at hand. Furthermore, if we start with small neuron networks and expand them using strategy 5, we can test the CNN on larger problems.

References

- [Bullock et al., (1993)] D. Bullock, D. Greve, S. Grossberg, (1993), "A Self-organizing Neural Network for Learning A Body-centered Invariant Representation of 3-D Target Position". Proceedings of the International Conference on Artificial Neural Networks, Amsterdam, 91-95.
- [Groh and Sparks (1992)] J.M. Groh, D.L. Sparks, (1992), "Two models for transforming auditory signals from head-centered to eye-centered coordinates". Biological Cybernetics, 67, 291-302
- [Hartman (1993)] Georg Hartmann, (1993), "Architectural Consequences of Mapping 3D Space Representations onto 2D", Proceedings of the International Conference on Artificial Neural Networks, Amsterdam, 898-902.
- [Van Dam et al., (1993)] J.W.M. van Dam. B.J.A. Kröse, F.C.A. Groen (1993), "Transforming Occupancy Grids Under Robot Motion", Proceedings of the International Conference on Artificial Neural Networks, Amsterdam, 318.
- [Van Dam et al., (1994)] J.W.M. van Dam. B.J.A. Kröse, F.C.A. Groen (1994), "Optimising Local Hebbian Learning: Use the δ -rule". Submitted to International Conference on Artificial Neural Networks, Sorrento.