INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center Street • Suite 600 • Berkeley, California 94704 • 1-510-642-4274 • FAX 1-510-643-7684

Learning Topology-Preserving Maps Using Self-Supervised Backpropagation on a Parallel Machine

Arnfried Ossen*

TR-92-059

September 1992

Abstract

Self-supervised backpropagation is an unsupervised learning procedure for feedforward networks, where the desired output vector is identical with the input vector. For backpropagation, we are able to use powerful simulators running on parallel machines. Topology-preserving maps, on the other hand, can be developed by a variant of the competitive learning procedure. However, in a degenerate case, self-supervised backpropagation is a version of competitive learning. A simple extension of the cost function of backpropagation leads to a competitive version of self-supervised backpropagation, which can be used to produce topographic maps. We demonstrate the approach applied to the Traveling Salesman Problem (TSP). The algorithm was implemented using the backpropagation simulator (CLONES) on a parallel machine (RAP).

*International Computer Science Institute, Berkeley <ossen@icsi.berkeley.edu>, and Technische Universität Berlin, Germany <ao@cs.tu-berlin.de>.

1. Introduction

Topology-preserving maps can be developed by a variant of the competitive learning procedure. Self-supervised backpropagation is an unsupervised learning procedure for feedforward networks, in which the desired output vector is identical with the input vector. For backpropagation we are able to use powerful simulators running on parallel machines. In a degenerate case, self-supervised backpropagation is a version of competitive learning. This relationship between competitive learning and self-supervised backpropagation has been pointed out by Hinton [1987].

In the simplest version of competitive learning, an input vector is represented by the weight vector of a single winning unit. The learning procedure then moves the weight vector in the direction of the input vector. The amount of change is proportional to the distance between weight and input vector. For small enough changes, this is equivalent to performing gradient descent in the sum-squared difference of weight and input vectors.

To see the relationship, we describe a degenerate case of self-supervised backpropagation. Suppose we add a layer of linear units with the same number of units as the input layer on top of the layer of competing units. We use the input vector activations as the desired output activations. We constrain the weights from the input units to the competing units to be identical to the corresponding weights from the competing units to the output layer. We keep the "winner-take-all" nonlinearity for the competing units and use backpropagation to determine the weight changes for the weights from the output layer to the competing layer. Then all error derivatives for the weights of non-winning units are zero and the derivatives of the winning unit will be proportional to the difference between the input/output and weight vectors. Thus, the above degenerate case of self-supervised backpropagation is equivalent to the simple competitive learning procedure.

2. Topology-Preserving Self-Supervised Backpropagation

One important version of competitive learning is concerned with learning topographic maps or feature mapping [Kohonen, 1989]. In this variant, there is some geometrical arrangement of the competing units. Usually they are placed on a line or in a plane. We then call a mapping between inputs and competitive units a *feature mapping* if it preserves neighborhood relations, i.e. if nearby input vectors are mapped to competitive units with nearby locations on the line (or in the plane).

A feature mapping can be learned by a competitive learning procedure if the winner-take-all nonlinearity is changed to a "winner-take-more" rule. In this case, not only does the winning unit get its weights changed, but so do units close to the winner. The amount of change is usually defined as a Gaussian function of the distance between the winning unit and the unit in question.

The winner-take-more rule cannot be realized using a self-supervised backpropagation with a winner-take-all nonlinearity at the hidden units. Since there is still a single winning unit, the error derivatives for the weights of non-winning units remain zero and non-winning units will never get an update of their weights. The neighborhood preserving feature mapping cannot develop. Instead, a smoothed version of the winner-take-all rule is needed, in which the winning patterns of activation will have a single peak at the winning unit and decreasing activation at neighboring units. We would also like to lift the equality constraints on the weights. One way to accomplish this is to view the winner-take-all nonlinearity as an extension of the usual backpropagation cost function.

For standard backpropagation we have:

$$E = \sum_i (t_i - o_i)^2 \tag{1}$$

where t_i is the desired activation (target) for output unit i and o_i the activation of output unit i.[1]

Instead of using a winner-take-all nonlinearity at the hidden units, we simply define the needed winner-take-more patterns as targets t_j for the *hidden units*. We can now use standard backpropagation, but get an extended error function:

$$E = \frac{1}{2}\sum_i (t_i - o_i)^2 + \frac{1}{2}\sum_j (t_j - o_j)^2 \tag{2}$$

The resulting error derivatives (assuming on-line mode and a sigmoidal nonlinearity) are then:

$$\frac{\partial w_{ij}}{\partial t} \propto -\frac{\partial E}{\partial w_{ij}} = o_i(1-o_i)(t_i - o_i)o_j \equiv \delta_i o_j \tag{3}$$

$$\frac{\partial w_{jk}}{\partial t} \propto -\frac{\partial E}{\partial w_{jk}} = o_j(1-o_j)[(\sum_i \delta_i w_{ij}) + (t_j - o_j)]o_k \tag{4}$$

Now, we can run the system on a standard backpropagation simulator. The only extension necessary is to find an appropriate winner-take-more pattern that can be used as a target pattern at the hidden units. Following the competitive learning procedure, we choose the closest (in squared error) winner-take-more pattern from a set of possible target patterns.

2.1. Teacher Forcing

A potential shortcut in the forward propagation phase is to use *teacher forcing*. Williams and Zipser [1989] have shown its value for recurrent networks. Since a non-recurrent net is just a special case of a recurrent net, teacher forcing can be applied here, too.

[1] For notational convenience, index i is always used for output units, j for hidden units and k for input units

The idea in teacher forcing is to replace the activation value of any unit by its target value if the target value is available. This must be done *after* the error derivatives for the units in question have been calculated. In the backpropagation phase, the partial derivatives for any forced unit have to be set to zero. In the above case, the equations simplify considerably.

$$\frac{\partial w_{ij}}{\partial t} \propto -\frac{\partial E}{\partial w_{ij}} = o_i(1-o_i)(t_i-o_i)t_j \tag{5}$$

$$\frac{\partial w_{jk}}{\partial t} \propto -\frac{\partial E}{\partial w_{jk}} = o_j(1-o_j)(t_j-o_j)o_k \tag{6}$$

The resulting equations are exactly those of two one-layer feedforward nets, where the targets of the first are the inputs of the second net. However, the targets of the first net are a result of a competition: the closest winner-take-more pattern is chosen.

3. Application to the Traveling Salesman Problem

The Traveling Salesman Problem is an *NP-hard* problem, and even very restricted versions of the TSP are *hard* problems. One way to cope with the intractability of the general TSP is to approximate the solution. Many global or local heuristics are known [Johnson, 1990]. More recently, neural network approaches were suggested for the TSP. A summary can be found in [Peterson, 1990]. We compare our approach to these algorithms.

In order to apply the self-supervised backpropagation feature maps to the TSP, we use a strategy known from research on the Kohonen feature map. We restrict the problem to a random uniform distribution of city coordinates in a unit square and place the competing units on a ring. For N cities, the Kohonen approach needs far more competing units than there are cities. In our approach, the number of hidden units can actually be less than the number of cities. Only the number of competing patterns must be large enough. Empirically, it is sufficient to use only $N/2$ hidden units. We generate $4N$ winner-take-more patterns using a bell-shaped function. We then use backpropagation to minimize the error function.

Figures 1 to 4 demonstrate a typical run. In a first phase the patterns organize into a ring. In a second phase, the local irregularities become integrated. After about 100 epochs, a good approximation has developed.

In general, it is difficult to make precise statements about the quality of a TSP approximation. Since the solution is usually not known we have to compute a lower bound on the tour length. The goodness of an algorithm is then its average behavior in comparison to the lower bound. Our approach has not yet been tested against a lower bound. Instead, we compare it to other Neural Network algorithms for the TSP. A detailed description of a benchmark study for the TSP can be found in [Peterson, 1990]. This testbed consists of 50-, 100- and 200-city TSPs in the unit square. Table 1 shows the average tour lengths for *elas-*

tic nets (EN), *genetic algorithms (GA)* and *simulated annealing (SA)* from the benchmark study, and the results for the self-supervised backpropagation feature map (FM).

N	EN	GA	SA	FM
50	5.62	5.58	6.80	5.80
100	7.69	7.43	8.68	8.05
200	11.14	10.49	12.79	11.60

Table 1: Average tour length for several neural net algorithms

4. CLONES Implementation

CLONES (Connectionist Layered Object-oriented Network Simulator) provides the user with an efficient and flexible object-oriented library. The underlying assumption is that a "modern object-oriented language (such as C++) has all the functionality needed to build and train [Artificial Neural Nets] ANNs". Resulting programs for the simulation of Neural Nets should be shorter and easier to read, write and modify [Kohn *et al.*, 1992]. CLONES runs on UNIX platforms and on the Ring Array Processor (RAP). A RAP consists of a number of digital signal processors (nodes) which are added to a host workstation. The theoretical peak performance is 128 MFLOPS for a four node RAP. For practical reasons, the number of nodes is limited to 64. For classification tasks using backpropagation, all of the necessary objects are already defined. These tasks run between 20 and 200 times faster on a RAP than on a SUN Sparc-2 workstation.

The CLONES library functions are small and well-localized. Non-standard simulations only require the creation of specialized new objects which inherit most of their functionality from library objects. E.g. for the general backpropagation task not restricted to the 1-out-of-N classification output, it is sufficient to redefine the function *set-error* in a derived object.

```
class Auto_assoc_BP_net : public BP_net
{
 public:
    Auto_assoc_BP_net(Param* param, Database* train_db, Database* test_db) :
        BP_net(param, train_db, test_db)
    {}

    void set_error()
    {
        // error = -(target-output) == -(input-output)
        sub_vv_v(n_output, output, input, error);

        // set error for units in output layer objects
```

```
        put_error(error);
    }
};
```

This redefinition of *set-error* is one part of the necessary modifications for the self-supervised feature mapping system. However, we have not yet determined the winning pattern of activation for the hidden units. Since CLONES is based on vector and matrix operations, it is not surprising that the required functionality is already there. We put the set of target (row-)vectors for the hidden units into a matrix and call the function *closest-row* with the actual activations at the hidden units. The function returns the index of the closest row, that is, the index of the winning pattern. This pattern is subsequently used as the target pattern for the hidden units. In the teacher forcing variant, the pattern is also used as a substitute for the actual hidden unit activations after the error has been calculated. For a CLONES implementation, we derive a competing layer object from the standard BP-layer object. The program fragment below contains the essential extensions for the teacher forcing version.

```
class Comp_layer : public BP_sigmoid_mse_layer
{
 public:
    // create new competing layer
    Comp_layer(int n_unit);

    // contains all available bell-shaped patterns
    Fmat *bell_matrix;

    float *hidden_act, *target_act;

    void best_bell()
    {
        // put_row, get_row, closest_row: Fmat methods
        // put, get: Layer methods

        // get current hidden layer activations and put them into
        // top slot of bell_matrix
        get(hidden_act);
        bell_matrix -> put_row(max+1, hidden_act);

        // find closest row to row[max+1],
        // remember index of best_bell for this pattern in global array
        bell_matrix -> closest_row(max+1, &best_bell_list[pattern_num]);

        // get closest row and put it into target_act;
        // force activations for hidden layer to target activations
```

```
        bell_matrix -> get_row(best_bell_list[pattern_num], target_act);
        put(target_act);
    }

    void
    forw_post_propagate(Net *)
    {
        transfer(output); // as in BP_sigmoid_mse_layer

        best_bell();
    }
};
```

5. Discussion

The search for "good" internal representations is an important element for neural network learning procedures. Usually it is an indirect search, in which internal representations are defined by connection weights. However, it is possible to make this search more explicit. Krogh et al. [1990] have suggested an extended error function which is an explicit function of the internal representations. The learning procedure then uses gradient descent for the two layers of connections *and* the hidden unit activations. The hidden unit activations "relax" to their optimal values. This approach achieves improved results on simple mapping problems.

We have shown how the search for internal representations can be used to realize a variant of competitive learning called feature mapping. Our approach also uses an extended cost function. But in contrast, we define targets for the hidden unit activations. An appropriate target is found using *competition.*

The use of backpropagation as learning procedure made it possible to take full advantage of existing parallel simulators in terms of execution time and implementation effort.

6. Acknowledgements

I would like to thank Jerome Feldman, Phil Kohn and Steve Omohundro for helpful comments.

References

[Hinton, 1987] Geoffrey E. Hinton. Connectionist learning procedures. Technical Report CMU-CS-87-115, Carnegie-Mellon University, Pittsburgh PA 15213, 1987.

[Johnson, 1990] D. S. Johnson. Local optimization and the traveling salesman problem. In *Proceedings 17th Colloquium on Automata, Languages and Programming*, pages 446–461, 1990.

[Kohn *et al.*, 1992] Phil Kohn, Jeff Bilmes, Nelson Morgan, and James Beck. Software for ANN training on a ring array processor. In John E. Moody, Steven J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems IV*, pages 781–788. Morgan Kaufmann Publishers, San Mateo, California, 1992.

[Kohonen, 1989] Teuvo Kohonen. *Self-Organization and Associative Memory.* Springer Verlag, 1989.

[Krogh *et al.*, 1990] Anders Krogh, G. I. Thorbergsson, and John A. Hertz. A cost function for internal representations. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems II*, pages 733–740. Morgan Kaufmann Publishers, San Mateo, California, 1990.

[Peterson, 1990] Carsten Peterson. Parallel distributed approaches to combinatorial optimization: Benchmark studies on traveling salesman problem. *Neural Computation*, 2(3):261–269, 1990.

[Williams and Zipser, 1989] Ronald J. Williams and David Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Neural Computation*, 1(1):87–111, 1989.

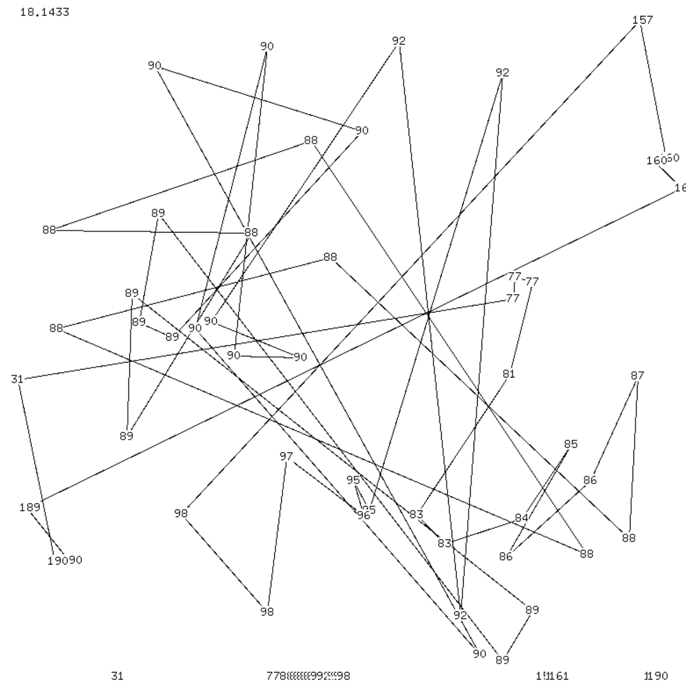

Figure 1: 50 city TSP, after 0 epochs

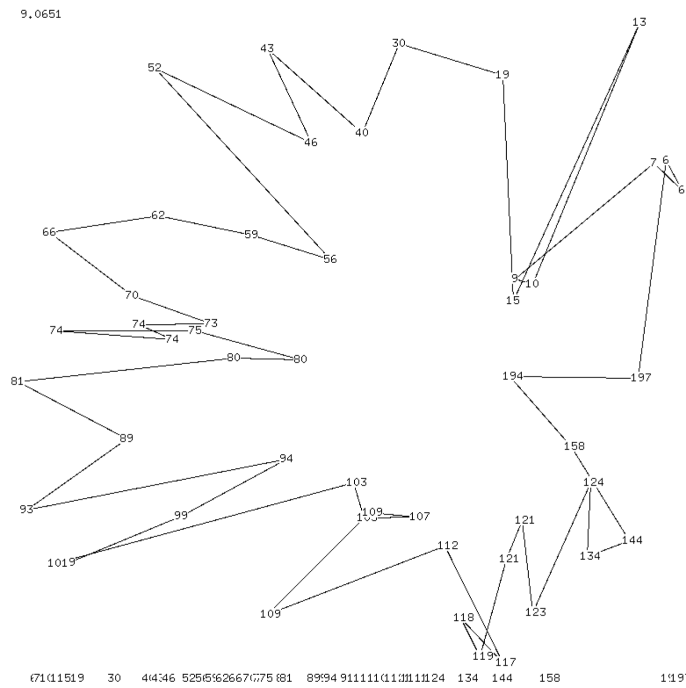

Figure 2: 50 city TSP, after 10 epochs

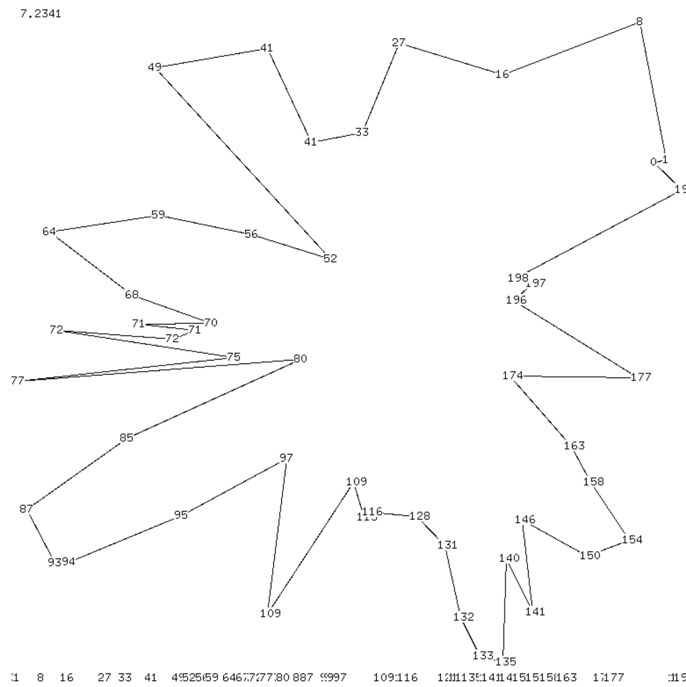

Figure 3: 50 city TSP, after 20 epochs

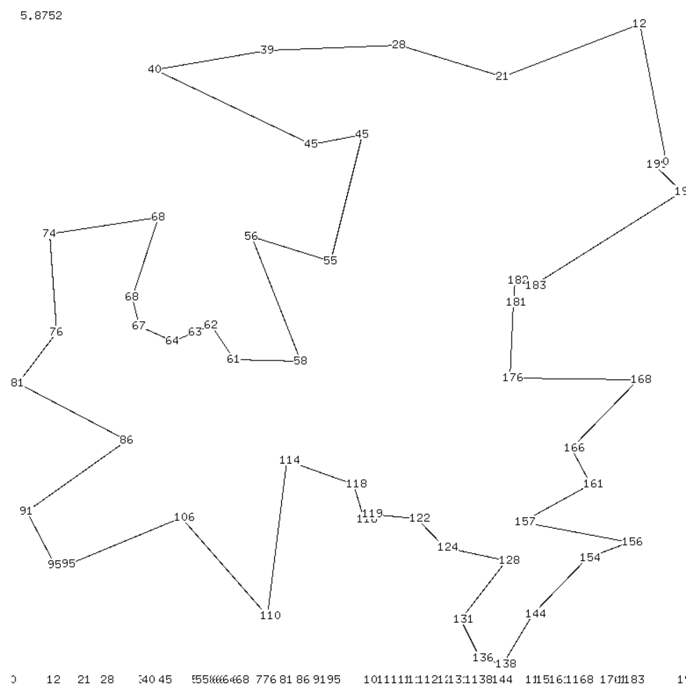

Figure 4: 50 city TSP, after 100 epochs