March 1991

LU TP 91-4

Self-organizing Networks for Extracting Jet Features

Leif Lönnblad[1], Carsten Peterson[2], Hong Pi[3] and Thorsteinn Rögnvaldsson[4]

Department of Theoretical Physics, University of Lund
Sölvegatan 14A, S-22362 Lund, Sweden

Computer Physics Communications **67**, 193 (1991)

Abstract:

Self-organizing neural networks are briefly reviewed and compared with supervised learning algorithms like back-propagation. The power of self-organization networks is in their capability of displaying typical features in a transparent manner. This is successfully demonstrated with two applications from hadronic jet physics; hadronization model discrimination and separation of b,c and light quarks.

[1] thepll@seldc52 (bitnet) leif@thep.lu.se (internet)
[2] thepcap@seldc52 (bitnet) carsten@thep.lu.se (internet)
[3] thephp@seldc52 (bitnet) pihong@thep.lu.se (internet)
[4] thepdr@seldc52 (bitnet) denni@thep.lu.se (internet)

1 Introduction

High energy physics contain many challenging feature recognition problems ranging from low level triggers to off-line data analysis. When extracting information from experimental data the standard procedure is to make various cuts based on human inspection of histograms. A specific choice of cuts corresponds to a particular set of feature functions $o_i = F_i(x_k)$ in terms of the kinematical variables x_k. One would like to have an optimal choice of this function with respect to expected distributions. This is exactly what artificial neural networks (ANN) aim at. In refs. [1][2][3] this approach was successfully applied to the problem of identifying the origin of jets in e^+e^-annihilation and hadron-induced large p_T reactions using feed-forward networks with the back-propagation learning algorithm [4]. This approach is an example of **supervised learning** where the features to be recognized are known beforehand. In many situations this is the information one wants to extract and it would be advantageous to have a network that is based on **self-organization**, which means that the network forms its own classification of the input patterns. The purpose of this work is to explore such self-organizing networks in the context of jet identification.

The most commonly used supervised learning algorithm is back-propagation [4] in multi-layered feed-forward neural networks. Here a particular form of F_i is chosen

$$F_i(x_k) = g\left(\sum_i \omega_{ij} g(\sum_j \omega_{jk} x_k)\right) \tag{1}$$

where the connection weights ω_{ij} and ω_{jk} are the parameters to be fitted to the expected distributions and $g(x)$ is a sigmoid function, typically $g(x) = 0.5[1 + \tanh(x)]$. Eq. (1) corresponds to the one hidden layer feed-forward architecture of fig. 1. The bottom layer (input) corresponds to the measured kinematical variables x_k and the top layer to the features o_i. The mission of the so-called hidden layer is to build up an internal representation of the observed data. Each unit or neuron has the threshold behaviour given by $g(x)$. Fitting to a given data set (or learning) takes place with gradient descent on e.g. a summed square error for each pattern

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

with respect to the weights ω_{ij} and ω_{jk}, where t_i are the desired feature values. This is done by presenting all the **training patterns** over and over again with successive adjustments of the weights. Once this iterative learning has reached an acceptable level in terms of a low error E the weights are frozen and the ANN is ready to be used on patterns it has never seen before. The capability of the network to correctly characterize these **test patterns** is called **generalization** performance.

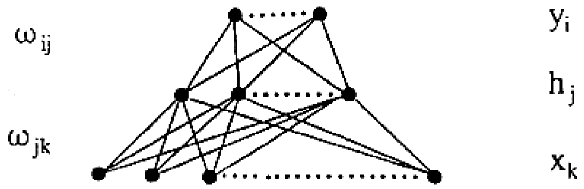

Figure 1: A one hidden layer feed-forward neural network architecture.

This procedure has turned out to be extremely powerful for a wide variety of feature recognition problems, among those jet identification. In particular one has the following virtues:

- Very good generalization performance. In some low-dimensional jet identification applications, where it has been computationally feasable to compute overlaps between distributions, one reaches values close to the theoretical **Bayesian limit** [5], given by the minimum overlap of the multidimensional distributions.
- The method is fairly insensitive to choice of parameters.
- Limited experimental acceptance and noise problems are easily handled since the network is forced to deal with and to ignore these things respectively.

These are algorithmic issues. In addition one of course has the common advantage of all neural modeling, that of inherent parallelism which facilitates real-time hardware implementations. There are, however, a few limitations with the back-propagation learning algorithm:

- The number of features (o_i) has to be known in advance - the feature space is divided once and for all by one or more hypersurfaces.
- It is somewhat cumbersome to analyze what features the hidden nodes focus on.

Self-organizing networks function in a different way. The desired feature values are not known beforehand and the network must organize itself according to the natural structure of the data. For each presented pattern, a winner h_m is picked out among the hidden (feature) nodes h_j (see fig. 2)

$$h_m = \max_j(h_j) \tag{3}$$

In a simple version of self-organization, winner-takes-all, the connections from the input nodes to h_m are then updated in such a way that h_m will become more sensitive

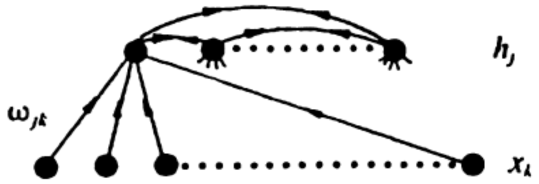

Figure 2: A one-layer self-organizing network. Note that possible interactions among the feature nodes are of feedback nature.

to the presented pattern, similar to standard clustering algorithms. This will result in each feature node being sensitive to clusters in the input data. If the updating is extended to regions around the winner, a topology among the feature nodes will be defined and the network can be used as a map of the input space.

This paper is organized as follows: Section 2 contains a brief review and discussion on competitive learning schemes. In Section 3 we apply the self-organization approach to the problems of distinguishing different hadronization models and to quark identification The results are compared with the ones obtained using the back-propagation algorithm. Finally in Section 4 we give a brief summary and outlook.

2 Self-organization

2.1 Clustering in a competitive network

In a **competitive network** a pattern x_k is presented and the responses of the feature nodes h_j are computed,

$$h_j = g(\sum_k \omega_{jk} x_k / T) \tag{4}$$

where the temperature T sets the gain for the sigmoid function $g(x)$. For each node j the the argument of g in eq. (4) can be written as a scalar product between the **weight vector** $\vec{\omega}_j$ and the **input vector** $\vec{x}$.

$$\sum_k \omega_{jk} x_k = \vec{\omega}_j^\dagger \vec{x} = |\vec{\omega}_j||\vec{x}| \cos\theta_j \tag{5}$$

Thus finding the winner according to eq. (3) amounts to finding the $\vec{\omega}_j$ which is closest in profile to the input vector $\vec{x}$ - the different feature units h_j compete in being representative for the pattern in question. This assumes that the $\vec{\omega}_j$:s are normalized. If we want to purify the role of each node j we need to modify the weight vectors such that each $\vec{\omega}_j$ picks up a certain "feature direction" of the data (see fig. 3).

An error for the winner node h_m can be defined as

$$E_m = \frac{1}{2} \sum_{\vec{x} \in \mathcal{M}} (\vec{x} - \vec{\omega}_m)^2 \tag{6}$$

where $\mathcal{M}$ is the set of all input vectors $\vec{x}$ that give node h_m as a winner. Minimizing this error, using gradient descent, gives us the updating rule for weight ω_{mk}

$$\Delta\omega_{mk} = -\eta \frac{\partial E}{\partial \omega_{mk}} = \eta \sum_{\vec{x} \in \mathcal{M}} (x_k - \omega_{mk}) \tag{7}$$

where η is the learning rate parameter. In practice, however, the weight vector is updated every time a pattern $\vec{x} \in \mathcal{M}$ is presented, giving the updating rule

$$\Delta\omega_{mk} = \eta (x_k - \omega_{mk})_{\vec{x} \in \mathcal{M}} \tag{8}$$

The weight vector $\vec{\omega}_m$ will consequantly be adjusted such that it points towards the center of all data points that activate unit h_m. This procedure is identical to the "adaptive k-means" clustering algorithm [6].

Eq. (8) is followed by a renormalization of the weight vector:

$$\omega_{mk} \rightarrow \frac{\omega_{mk}}{\sqrt{\sum_{k'} \omega_{mk'}^2}} \tag{9}$$

Following this training procedure the final result will be extremely easy to analyze; one simply inspects the weight vectors $\vec{\omega}_j$ and they will mimic the specific features that the h_j's are most sensitive to. This is the key virtue of this method.

2.2 Dynamical winner-takes-all

Picking out the winner node can be done dynamically by introducing inhibitory connections $T_{jj'}$ between the h_j's and after each pattern presentation let the **feedback** network of h_j's settle according to

$$h_j = g(\sum_{j'} T_{jj'} h_{j'} + \sum_k \omega_{jk} x_k) \tag{10}$$

Eq. (10) corresponds to minimizing the energy

$$E = -\frac{1}{2} \sum_{jj'} T_{jj'} H_j H_{j'} - \sum_{jk} \omega_{jk} x_k \tag{11}$$

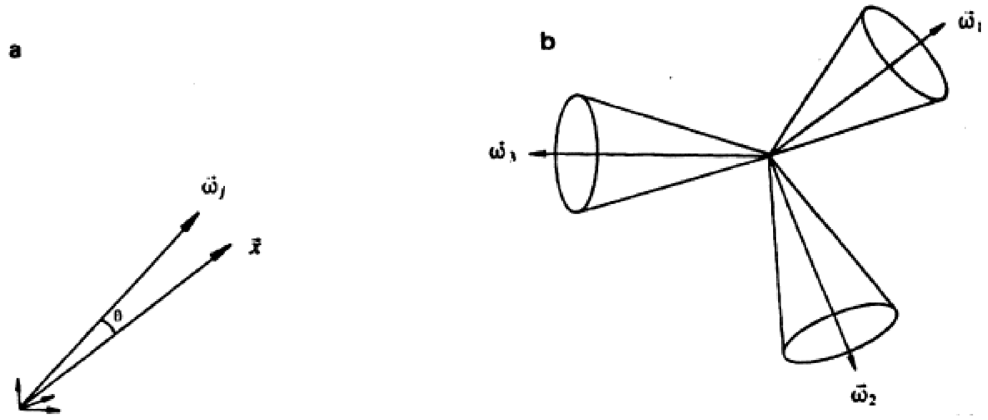

Figure 3: **(a)** A weight vector $\vec{\omega}_j$ lining up with an input vector $\vec{x}$. **(b)** Three feature vectors averaging different clusters of input categories.

using the mean field theory approximation [13] on binary decision elements $H_j \in \{0,1\}$. A straight-forward choice of inhibitory $T_{jj'}$ is

$$T_{jj'} = -\alpha(1 - \delta_{jj'}) \tag{12}$$

where α is a parameter governing the relative strength between the inhibition and the external input.

These inhibitory connections constitute a "soft" way of implementing a winner-takes-all mechanism. A "hard" way of doing the same thing would be to replace the different N_h sigmoidal nodes with a single **Potts** unit [2] with N_h components, each updated according to

$$h_j = \frac{e^{a_j}}{\sum_{j'} e^{a_{j'}}} \tag{13}$$

with

$$a_j = \sum_k \omega_k x_k / T \tag{14}$$

where T is a "temperature" setting the gain of the generalized sigmoid of eq. (13).

2.3 Topological ordering

It would be desirable to have a topological order among the feature units, where neighbouring units will respond to adjacent points in the input space. A commonly

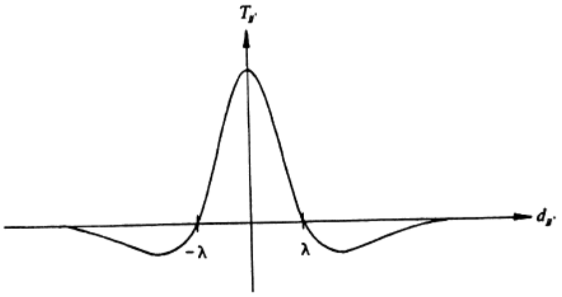

Figure 4: A "mexican hat" potential between h_j and $h_{j'}$.

used technique to achieve such a **topological map** is by updating units close to the winner node in the same way as the winner node [7]. In this way neighbouring units will end up with similar weight vectors. This can be accomplished dynamically by using a "**mexican hat**" potential (see fig. 4 for $T_{jj'}$ in eqs. (10,11))

$$T_{jj'} \propto e^{-d_{jj'}^2/\lambda^2} \cdot (1 - d_{jj'}^2/\lambda^2) \tag{15}$$

where $d_{jj'}^2$ is the squared distance between node j and j' in the network and λ is a width parameter. The "distance" $d_{jj'}$ between the feature units defines the topology of the net. The "mexican hat" potential enforces signals to neighbouring units and inhibits signals to units some distance apart. Annealing the network using this potential will create a group of winners instead of one single winner. The "hat" width λ is decreased as learning proceeds. This dynamic method, and other similar versions [8], is very sensitive to the choice of parameters and there exists a robust **short-cut** [7] version of this procedure where the entire neighbourhood defined by λ is updated. This modified algorithm is used throughout this paper and is implemented in `JETMAP 1.0` [9] (see Appendix B). Suitable parameters and some tips for implementing this short-cut version for our applications are found in appendix A.

2.4 Unnormalized data

So far our competitive network has only organized itself in terms of pattern profile, i.e. the network only cares about the angle between the input vector $\vec{x}$ and the weight vector $\vec{\omega}_j$. The input patterns could just as well be normalized to unit length. In many cases, like jet classification, the magnitude of the input vector is a significant feature and one would prefer the network to work with unnormalized input. One simple way to do this is by introducing a Gaussian response function for the feature units [11]

$$h_j = e^{-(\vec{\omega}_j - \vec{x})^2/T} \tag{16}$$

where the updating again is done according to eq. (8), with the difference that the weight vectors $\vec{\omega}$ are not normalized after each updating. The temperature T sets the width of the function (see appendix A). In this procedure only the distance between $\vec{\omega}_j$ and $\vec{x}$ is of relevance and it hence handles all kinds of data. We have subsequently used it for all the applications studied in this paper.

2.5 Learning vector quantization

Although the self-organizing map is an unsupervised learning algorithm, for classification purpose one can augment it with supervised learning for fine tuning the units specific to certain features [7]. This is called **learning vector quantization** (LVQ) and amounts to learning correct answers and unlearning incorrect answers, *i.e.*, the learning rate parameter in Eq. (7) supplemented with a minus sign whenever an input data activates a unit that has been assigned to the other types of data. In this way it is sometimes possible to achieve as good results for pattern classification as with a feed-forward network like back-propagation (BP) and it has successfully been used for quark/gluon separation in ref. [12].

The relative performance of LVQ and BP strongly depends upon the dimensionality of the problem and the number of feature nodes. For most practical purposes one uses Gaussian units in feature maps (eq. (16)). Hence the network has fo fill the space with "localized bumps" in order to represent the data well. For high-dimensional problems this requires many nodes. For a BP feed-forward network, on the other hand, feature space is divided by hyperplanes corresponding to the different sigmoidal hidden nodes. It is evident that this is a very economical way of dividing up a high dimensional space. Indeed, from the "benchmark" studies in refs. [10][13] it is clear that LVQ had difficulties in reaching the Bayesian limit for synthetic problems consisting of overlapping high dimensional Gaussians in contrast to BP and other supervised learning algorithms based on sigmoidal units like the Boltzmann machine [14] and Mean Field Theory learning [13][15][5]. Even with a number of feature nodes that increased with the dimensionality of the problem (q) the performance was not impressive [6].

[5]The conclusions with respect to performance of BP in [10] are in error since the architecture used for BP was inappropriate for the problem. We refer to ref. [13] for a proper discussion of this issue.

[6]LVQ would presumably have reached the Bayesian limit if the authors of ref. [10] would have allowed for a number of feature nodes increasing like e.g. 2^q.

3 Jet Feature Applications

There exists today a number of QCD Monte Carlo (MC) programs which describe experimental data from e.g. LEP very well [16]. Most of them are structured in the same way; one part dealing with the hard interaction (e.g. $e^+e^- \rightarrow Z^0 \rightarrow q\bar{q}$) and the perturbative evolution into multi-parton states. Another part performs the non-perturbative hadronization of partons into observable hadrons and finally there is a part administrating the decays of the hadrons. Whereas the perturbative part is fairly well understood theoretically, the non-perturbative hadronization can only be described by phenomenological models.

Comparisons between MC models and experimental data as in ref. [16] are all based on averaged features such as inclusive distributions, event shape parameters, multiplicities etc.. These comparisons have given no hint as to which MC model gives the optimal description of the data and it would therefore be desirable to confront more detailed features of the different MC models with experimental data. In this section we demonstrate how self-organizing networks can be used to accomplish this for two problems: **jet flavor separation** and **hadron model discrimination**.

Heavy quark jets were separated from those stemming from light quarks in ref. [2] using the supervised BP algorithm with very good results. Small but significant differences between data generated by the different MC programs were observed, especially in the case of bottom jets. Armed with the self-organizing network we here examine these differences in a more detailed and transparent way by mapping out the underlying discriminating features. For both applications studied in this section we also compare the classification powers of self-organizing networks with those of supervised BP.

3.1 The Monte Carlo programs

We use two different MC programs to generate jets; `ARIADNE 3.1` [17] and `HERWIG 3.4` [18]. `ARIADNE` is a program implementing the colour dipole approximation of perturbative QCD cascades [19] which uses the Lund string fragmentation [20] as implemented in `JETSET 7.3` [21]. It also uses `JETSET` to perform particle decays. Throughout this section we only generate 2-jet events where the perturbative QCD shower is not very important, hence `ARIADNE` can be considered essentially equivalent to `JETSET`.

`HERWIG` differs from `ARIADNE/JETSET` mainly in the hadronization part where it uses a cluster fragmentation [18] procedure. Here the gluons formed in the perturbative

QCD shower are forced to split into $q\bar{q}$ pairs, which are then combined to form colorless clusters. If the mass of such a cluster is above a certain limit it will decay anisotropically into two smaller clusters. If, on the other hand, the mass is below this limit the cluster will instead decay isotropically into one or two hadrons. This is very different from string fragmentation where a string is stretched from a quark via the gluons to an anti-quark. This string is then fragmented into hadrons by forming $q\bar{q}$ pairs along the string. For a more detailed description of how these models are implemented in the programs we refer to refs. [17] [18] [21].

All jets used in our analysis are taken from two-jet events generated in e^+e^- annihilation at the Z^0 peak. The clustering into jets is performed using the LUCLUS algorithm found in the JETSET program, with the value of the "d-join" parameter set to 5 GeV. All other parameters in the clustering and in the event generation are set to their default values.

3.2 Jet flavor separation

The main difference between light and heavy quark jets is that the leading particles in the light quark jet have a large probability of stemming from the hadronization process, whereas in the heavy quark case they usually originate from the weak decay of the heavy meson. For string fragmentation we consequently expect the leading particles to be strongly ordered in energy in the case of a light quark jet and, since the decay of the heavy hadron is isotropic, to be more evenly distributed in energy for the heavy quark jets.

In this analysis we use a data set consisting of 9000 quark jets, with equal amounts of the three flavor classes. The data set is divided into two parts; one training set with 6000 jets and one test set with 3000 jets. The test set is used to measure the networks ability to generalize. In the unsupervised case, when no classification is done, we use all 9000 jets.

With the **supervised BP** algorithm, as implemented in the subroutine package `JETNET 2.0` [22], we first make a "benchmark" study to establish the existence of a difference between jets of different flavors. We use a feed-forward network with 8 input units, encoding the p_z and p_T of the four leading particles ordered in decreasing rapidity w.r.t. the jet-axis and two hidden layers with 20 units in the first hidden layer and 10 units in the second. The output layer has 3 output units encoding the flavor (100, 010 and 001 for uds, c and b respectively). Winner-takes-all criterium [2] is used for establishing the output class. The resulting overall generalization performance is 51%.

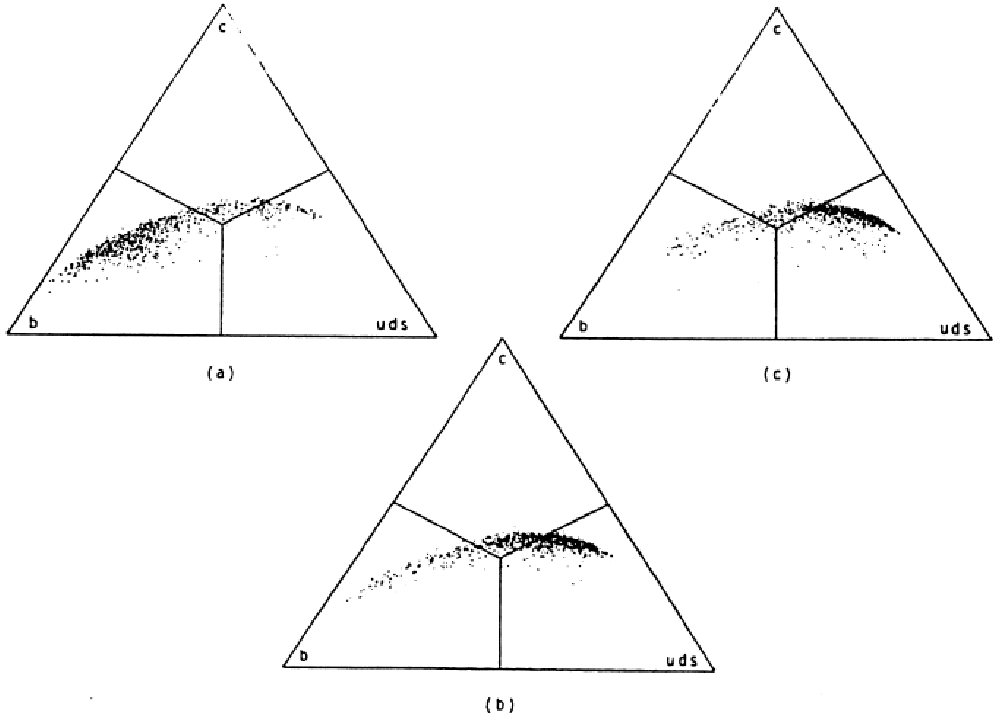

Figure 5: Output distributions for a BP network with 2 hidden layers and 3 output nodes for **(a)** b-quark jets, **(b)** c-quark jets and **(c)** uds-quark jets. The lines separate the decision regions obtained by using the winner-takes-all criterion.

To visualize the output space of the BP network we exploit the fact that all possible output vectors fall inside the cube defined by the unit vectors (1,0,0,) (0,1,0), (0,0,1). We project all output vectors onto the equilateral triangle-plane with corners (1,0,0), (0,1,0) and (0,0,1). These output distributions are shown in fig. 5. It is clear that the network has found some typical features for the jets. One interesting thing that immediately emerges is that the c-quark jet distribution looks very much like the uds-quark jet distribution, indicating that c-jets are more similar to uds-jets than b-jets. Unfortunately, there is no simple way with the BP network to visualize what the input space looks like for those jets and what features the network has found.

We then proceed by presenting the data to a two-dimensional **self-organizing** network with 9×9 feature nodes. The number of nodes can be crucial for good performance when classifying, but is not very important for mapping. We have found 9×9 to be optimal for this classification problem. The generalization performance is just below 50%, which is close to the BP result.

For mapping we shrink the network to 7×7. The total number of nodes should not be too large, in order to get good statistics for "tagging" the features. After 300 epochs (an **epoch** is a presentation of the entire training set), corresponding to $9 \cdot 10^5$ updatings, the distributions of the different jet flavors acquire the shapes shown in fig. 6. In fig. 7 the resulting map is shown in terms of dominant flavors. The network detects the significant features of the jets and is able to group the different flavors topologically. It is now straightforward to understand what these features are, simply by inspecting the weight vectors associated with the different fetaure nodes (see fig. 8). Apparently the network arranges itself such that the feature nodes in the upper right corner (see figs. 6,7 and 8) tunes to jets containing one hadron with large momentum and the nodes in the upper left corner tunes to jets with two large momentum hadrons. Jets with an even distribution of momentum among the hadrons turn up in the lower right corner. As expected, the nodes that pick up jets with with one or two large momentum hadrons are dominated by uds-jets (see fig. 6). In contrast, the nodes that are sensitive to jets with an even momentum distribution are dominated by heavy quarks (mainly b-jets). Typically the jets with large momentum leading particles have $\sim 90\%$ of the total jet energy gathered in the four leading particles, and the jets with an even distribution of the momentum only contain $\sim 50\%$ of the total jet energy in the four leading particles.

3.3 Hadronization model discrimination

In ref. [2], when using BP learning on a network to separate out b-jets from others, we noticed a small but significant difference between b-jets produced by `ARIADNE` and b-jets produced by `HERWIG`. Although the network seemed to pick up the same feature of b-jets in both cases, this feature was more pronounced in the `HERWIG` b-jets. This facilitated the separation of b-jets from other jets more in `HERWIG` than in `ARIADNE`. In the following we present results from analysis using an unsupervised self-organized network that exhibits this difference in a more lucid way. The data set consists of 3000 b-quark jets, with equal parts from `ARIADNE` and `HERWIG`, again obtained from two-jet events at the Z^0 peak. We use 2000 jets for training and 1000 jets for testing generalization.

We first use the **BP algorithm** on a feed-forward network with one hidden layer to establish a "benchmark" for the self-organized network, with respect to classification. The input layer has 12 nodes, corresponding to p_z and p_T for the 6 leading particles in the jet. The hidden layer has 10 nodes and the output layer one node encoding the MC origin of the jet (1 for `HERWIG` and 0 for `ARIADNE`). The distribution in the output node for `ARIADNE` and `HERWIG` b-jets from the test set are shown in fig. 9. When applying a cut of 0.5 we are able to separate between the two MC programs on a confidence level of 62%.

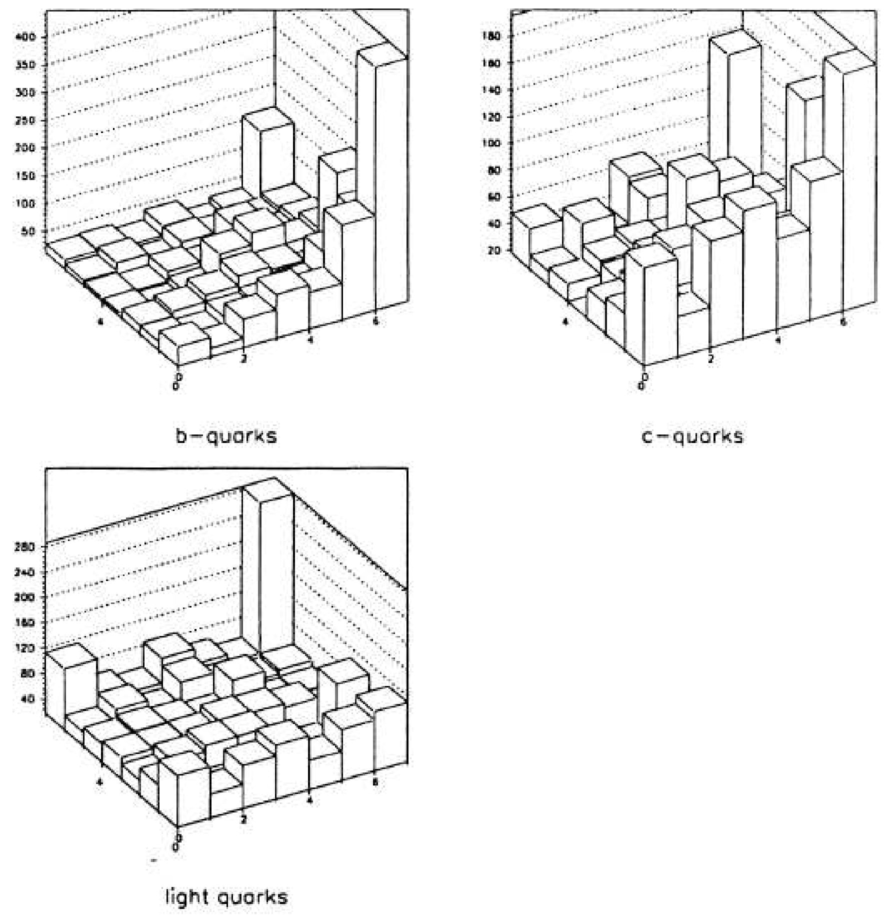

Figure 6: Distribution of b-quarks **(a)**, c-quarks **(b)** and uds-quarks **(c)** over the self-organized 7×7 feature nodes.

We then use our two-dimensional **self-organizing** network on the same data. The number of nodes is 7×7, which seems to be the optimum for this problem[7]. Increasing the number of nodes decreases the performance. The networks generalization is shown in fig. 10 as a function of training epochs. The corresponding separation for the test set in this case is the same as the 62% obtained from BP. The training is done for a total of 170 epochs, with unsupervised learning during the first 150 epochs and LVQ for the last 20 epochs. With the implementation of LVQ, the efficiency is boosted to 69% for the training set. However, the performance of the network when generalized to the test set shows no sign of improvement with LVQ. The short-cut learning is doing almost as good as the BP. We also note that for both cases the `ARIADNE` b-jets are more evenly distributed in the output space, so that there are very few `HERWIG`

[7]The optimal number of nodes depends both on the problem and the size of the data set.

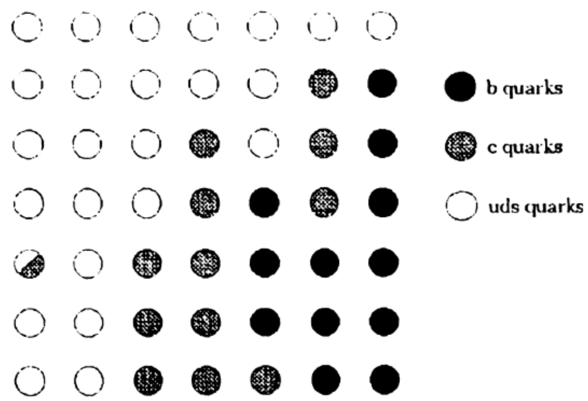

Figure 7: The resulting map for uds- c- and b-quarks. The shading indicates the dominant flavor for the units. The units are numbered as in fig. 6.

b-jets in the ARIADNE dominated regions as compared with the number of ARIADNE b-jets in the HERWIG dominated regions. The distributions among the feature nodes of the self-organized network are shown in fig.11. In the node where ARIADNE dominates the most we find more than 6% of the ARIADNE b-jets but only 0.6% of the HERWIG b-jets; there is clearly a very strong leading particle effect (see fig. 12). In the corresponding HERWIG-dominated node, where we find 8% of the HERWIG jets and 5% of the ARIADNE jets; the p_z of the leading particles are more evenly distributed. As we have seen in the previous subsection this leading particle effect is exactly what the network uses for distinguishing light quark jets from b-jets. Hence it is clear why this separation was more difficult in the ARIADNE case than in the HERWIG case in ref. [2].

This leading particle effect does not come primarily from the hadronization process but rather from the weak decay of the produced B-meson which typically gives rise to $5-10$ hadrons. There are however differences in the B-decay in the two programs. For example; it has been observed that in HERWIG the multiplicity in B-decays is somewhat larger than in JETSET [23] (which is used by ARIADNE). Since we are here mainly interested in differences in the hadronization scheme we generated one set of b-jets where the hadronization was made by HERWIG but all hadronic decays were performed by JETSET. Comparing these jets with the ones from ARIADNE using the same network architecture as in fig. 11, we find that the difference between the two samples has decreased (the separation efficiency is dropped to 57%). We still get an ARIADNE dominated node (containing $> 6\%$ of the ARIADNE b-jets) with a strong leading particle effect, but now also $\approx 3\%$ of the HERWIG b-jets are associated with this node. The resulting map is found in fig. 13, from which it is clear that using the JETSET decay has shuffled the HERWIG jets towards the previously ARIADNE-dominated

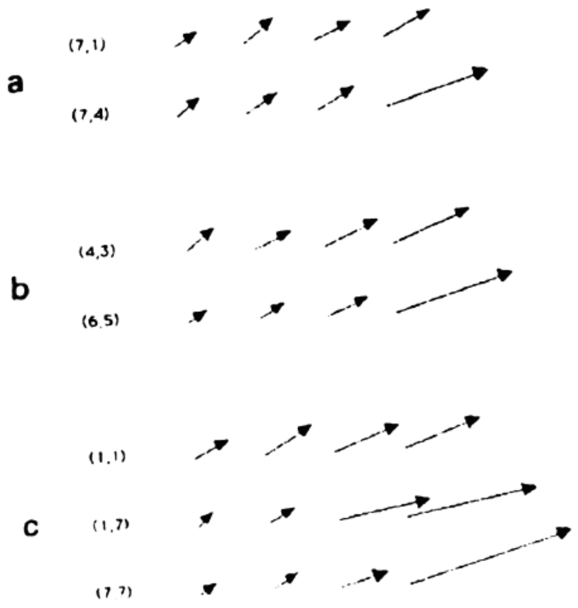

Figure 8: The weight vectors corresponding to the 4 leading hadrons for some of the units in the self-organizing network; b-jets (**a**), c-jets (**b**) and uds-jets (**c**). The p_T component has been multiplied by a factor of 5 relative to the p_z component. Coordinates refer to the position in the map (cf. figs. 6,7).

region of the input space (cf. fig. 11 c). We conclude that this leading particle effect has its origin in the JETSET decay of B-mesons. The reason why this effect is less pronounced in the HERWIG case is found in the hadronization. In the JETSET string fragmentation the B-meson is always the leading particle in the jet whereas in the HERWIG cluster fragmentation the b-flavored cluster decays isotropically. The B-meson will in the latter case not always be the leading particle in the jet and the leading particle effect in the decay is therefore "screened".

This analysis could fairly easily be compared with data from e.g. LEP by selecting two-jet events where one jet has been identified as a b-jet by e.g. lepton tagging. The remaining jets in these events would then make a pure sample of b-jets [27] which could be analyzed with this self-organized network, with the prospect of experimental discrimination of the two models.

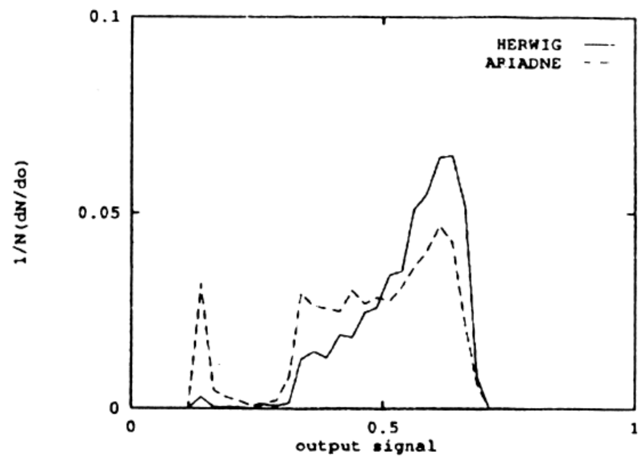

Figure 9: Output node distributions for ARIADNE and HERWIG respectively using the BP algorithm described in the text.

4 Summary

We have demonstrated how self-organizing networks can reach similar performance levels to those of supervised learning in jet classifying situations. In applications where one is only interested in obtaining a classifier the supervised BP algorithm is more efficient, at least for jet applications[8]. The real virtue of self-organizing networks is in the transparency of the internal representation of the problem - the resulting weight vectors offer a direct interpretation and visualization of what properties in the data the network singles out as being typical. Both in the case of hadronization model discrimination and jet flavor tagging such an analysis turned out to be very educating with respect to the underlying physics.

As one useful result we have identified typical signatures of the ARIADNE and HERWIG MC models that could be used to experimentally disentangle one model from the other.

One could of course also imagine hybrid architectures where the learning is supervised with a set of predetermined features (output nodes) but where one retains inhibitory or "mexican hat" interactions among the hidden nodes. Such an architecture would yield the best of two worlds - efficient feature recognition together with transparent interpretation of the results. With feed-back interactions between the hidden nodes

[8] This depends on the dimensionality of the problem and the distribution of the classes in the input space. We have not studied this for other applications.

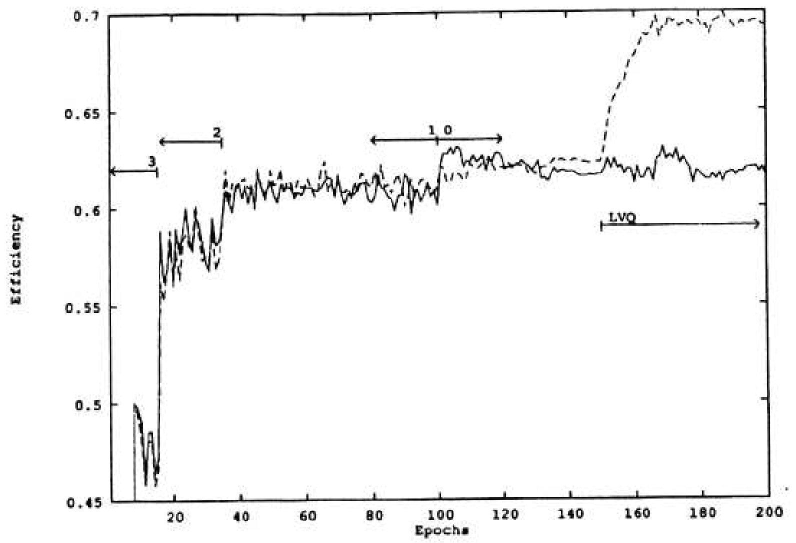

Figure 10: Efficiency for classifying ARIADNE b-jets from HERWIG b-jets versus the training epochs for a self-organizing network. The solid curve is for the test set and the dashed one for the training set. The number accompanying the arrow is the neighbourhood size λ. The arrow marked by "LVQ" indicates the duration when Learning Vector Quantization is applied.

Mean Field Theory learning [13] [15] would then be preferable over the feed-forward BP algorithm.

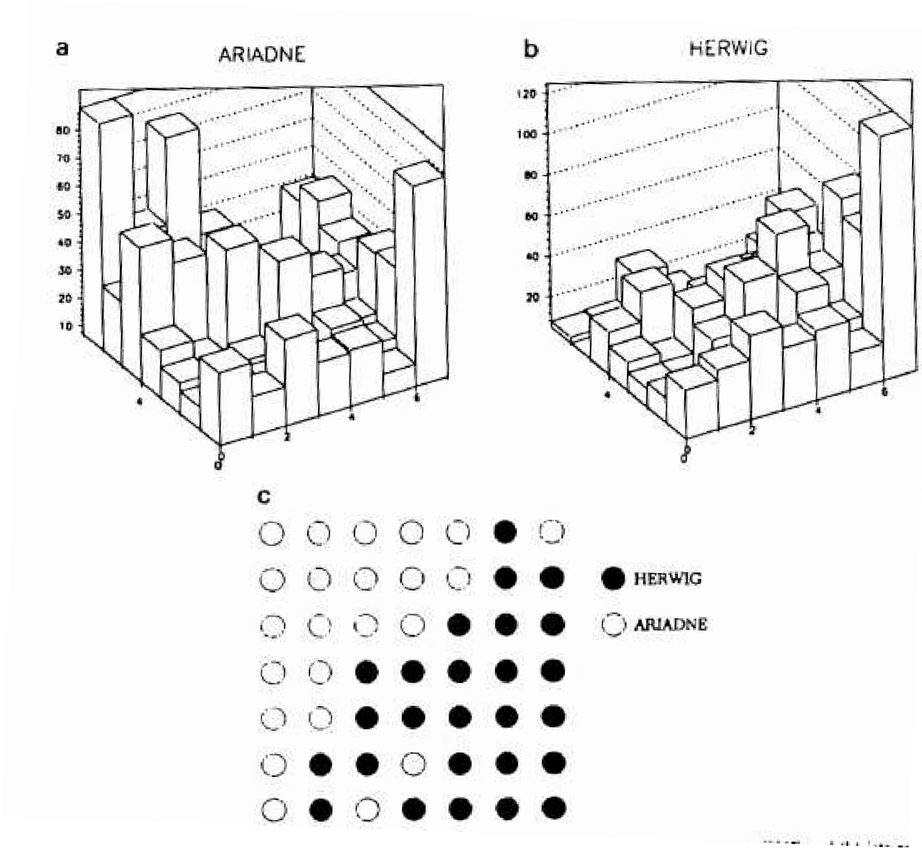

Figure 11: The distribution of b-jets over the self-organized 7×7 feature nodes. **(a)** Jets generated by ARIADNE and **(b)** jets generated by HERWIG. **(c)** The resulting map where the shading indicates the dominant model for that unit (cf. fig. 7).

Appendix A

Parameters and Hints

The training of the self-organizing network is robust with regard to all parameters except the temperature T of the Gaussian response function. The parameters and procedures we have used to achieve the results in this article are given below.

Initializing the weights. The input data typically occupies a very small subspace of the total input space and, ideally, one wants the feature nodes to focus on this subspace from the start. We have initialized the weights by giving them random values of the same sign as the input. This works well when a local neighbourhood is used in the training. If a LVQ supervised training is used without a local neigh-

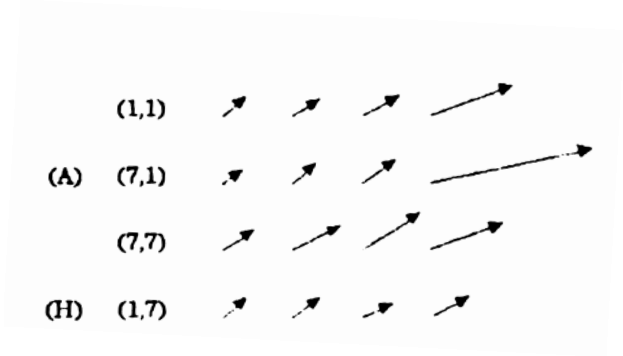

Figure 12: The weight vectors corresponding to the 4 leading hadrons for the corner units of figure 11. The p_T component is multiplied by a factor of 5 relative to the p_z component. Ariadne and Herwig dominated nodes are indicated.

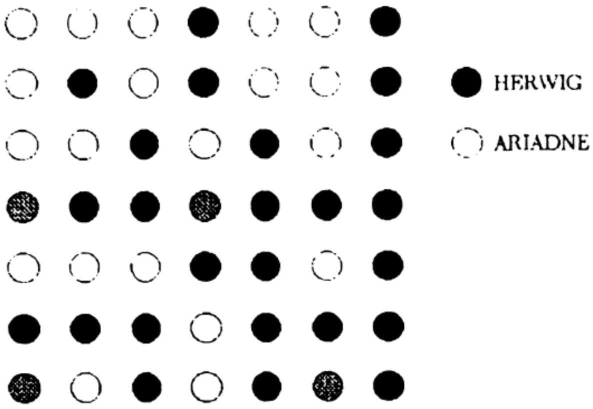

Figure 13: Resulting map for `ARIADNE` and `HERWIG` b-quarks when the weak decay effect has been corrected for (cf. fig. 11 c). The weight vectors are very similar to the non-corrected case (fig. 12).

bourhood the weights must be initialized differently. One way is to pick, at random, some input patterns and assign to the weights. Both these options are implemented in the subroutine package `JETMAP 1.0` (see appendix B).

Local neighbourhood. In order to produce a good topological map, one should start with a wide neighbourhood λ, approximately half the width of the network. This will force all units to become more or less aligned. The neighbourhood is then decreased during the training, permitting the units to focus on local regions of the input space. If λ is too small in the start, the coupling between the units is broken and no ordering will emerge. If λ is large all the time, all units in the network will end up pointing in the same direction. This is illustrated in fig. 14 where we show

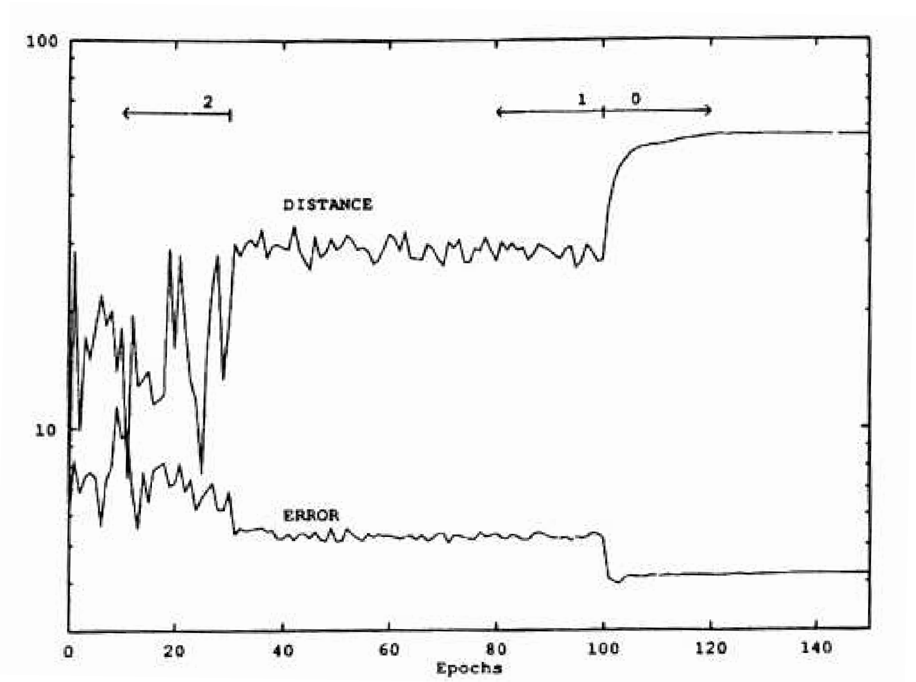

Figure 14: The average error and average distance between feature nodes as a function of training epochs. The numbers accompanying the arrows are the corresponding neighboorhood size parameters λ.

how the error in eq. (6) averaged over all units and the distance D_{jl} between weight vectors averaged over all pairs of units vary in a typical network training process. Since λ can take only integer values, the two measures behave like step functions when the neighbourhood size λ transits to a smaller value. The distance D_{jl} between weight vectors $\vec{\omega_j}$ and $\vec{\omega_l}$ is defined by

$$D_{jl} = \sum_k (\omega_{jk} - \omega_{lk})^2, \tag{A1}$$

It is clear from fig. 14 that the convergence of each unit increases with decreasing neighbourhood λ, while the coupling between the units decreases when λ decreases. In our applications we have allowed the neighbourhood to decrease linearly with the number of updatings. If a good classification result is desired, the neighbourhood must be allowed to shrink down to zero units (i.e. only the winner is updated), possibly followed by training with LVQ towards the end. On the other hand, if a good map is desired where the distance between neighbouring units should be the same all over the net, the neighbourhood should only decrease down to one unit (i.e. the nearest neighbours around the winner will always be updated), allowing some coupling between the units.

The shape of the neighbourhood has in all applications been square. In JETMAP 1.0 there is also an option for circular neighbourhoods.

Learning rate. The learning rate η is a number between 0 and 1. It should not be a constant since the weights are then likely get into an oscillation mode and make the convergence an unnecessarily slow process. So in practice η should be reduced gradually as training progresses. In the beginning of the training, the learning rate should be set large, $\eta > 0.5$, the network needs to make large changes in the weights to align them and focus them onto the desired subspace, creating a coarse mapping. Once this is achieved, a lower learning rate is used for fine tuning the units to local regions. This takes much longer time than finding the coarse mapping. We therefore let the learning rate decrease geometrically with the number of updatings, $\eta(t+1) = k \cdot \eta(t)$, where $k \leq 1$. Typically $\eta \approx 0.9$ in the beginning and $\eta \approx 0.01$ towards the end of training. An alternative way [7] is to specify a start up value and a final value for η and then let it decrease as a linear function of the updating steps.

Temperature. The temperature T sets the slope of the response function $g(x)$. To optimize the performance of the network it is necessary to choose T properly. An appropriately chosen temperature should distribute the output values in the full range of the non-linear function $g(x)$. It hence depends both on the shape of $g(x)$ and the type of input. If $g(x)$ is a $tanh(x)$ type sigmoid function and the input is normalized, the argument $x \in [0,1]$, the temperature should be T$< 1/3$. Alternatively, if $g(x)$ is a Gaussian function (16) it is important not to get overflow in the exponent, in which case the network will not respond. This means that T $\sim Na^2$, where N is the number of input nodes and a is a typical magnitude for the inputs. As learning proceeds, and the weights converge towards the inputs, the temperature can be allowed to decrease. We have used a self adaptive algorithm to adjust the value of T. The procedure is to reduce T by a factor $(1-\eta)$ if the maximum output on all units induced by an input is smaller than 0.5, and to increase it by a factor $1/(1-\eta)$ if the minimum output is larger than 0.5. We find a satisfying value of T if this procedure is activated during the first 5 to 10 epochs of training. This procedure is not included in JETMAP 1.0.

Number of nodes in network. The total number of nodes in the network depends on the application. In the case of classification one needs at least as many nodes as the number of classes. In general, more nodes give better performance, provided that the size of the data set is sufficient. If the number of nodes is $\approx$ the number of data points the network will specialize too much and generalization performance will be poor. In the case of making a map one must make sure that the number of nodes $\ll$ number of data points, in order to get good statistics for assigning classes to the units when analyzing the map. For our map applications we have used typically $\mathcal{O}(10)$ feature nodes for data sets with $\mathcal{O}(1000)$ patterns.

Dimension of the network. The ideal dimension of the network equals the inherent dimensionality of the input data, which in most cases is lower than the physical dimension of the input space. For example, if one uses input vectors of dimension 16 and it turns out that all data points lie on a 2-dimensional surface, then the inherent dimensionality of the data is 2. If the dimensionality of the network equals the inherent dimensionality, then the network will be globally ordered. If the dimensionality of the network is lower than the inherent dimensionality of the data, the network will be locally ordered but not globally [8]. In our applications we have used a two-dimensional net, which has worked well. If one class should occur in several small "islands" in the map, it is usually a sign of a too small dimensionality of the net. We have a maximum of two dimensions in `JETMAP 1.0`, mainly because the advantage of using a self-organizing network, as compared to other methods (histogramming etc.), decreases with increasing dimensionality of the net.

Learning vector quantization. Standard LVQ can be done in two different ways: One is to assign classes to the feature nodes right from the start [12], another is to allow the network to do a map first and then assign classes to the nodes. We have used the latter method in this paper but both methods are possible in `JETMAP 1.0`. Wether LVQ actually improves the classification performance in the latter case clearly depends very much on the inherent structure of the input data distribution. In our applications to jet classification problems we have seen that LVQ either does not improve the results, or it improves only on the training set but not on the test set where it really matters.

Appendix B

JETMAP 1.0

The analysis reported in this paper was performed using the FORTRAN 77 subroutine package `JETMAP 1.0` developed by the authors. It implements the self-organizing ANN as it is described in section 2 and appendix A. The package and manual is available on request via BITNET. It is designed to be similar to the subroutine package `JETNET 2.0` [22] that uses the BP algorithm.

The user interface consists of one common block and eight subroutines. All information handling between the program and the user is executed via the common block `/JMDAT1/`. Initialization, training and testing is performed by calling the subroutines `JMINIT`, `JMINWE`, `JMDUMP`, `JMREAD`, `JMSTAT`, `JMTRAL` and `JMTEST`. For bookkeeping purposes the subroutine `JMINDX` is included.

The program works, in short, as follows: The network is defined by setting the switches in `JMDAT1` and calling the subroutine `JMINIT`. To train the network one presents input patterns to the network through vector `OIN` and calls the subroutine `JMTRAL`. The trained network is then used for classifying or mapping by putting an input pattern in vector `OIN` and calling `JMTEST`.

B.1 The common block

- `COMMON /JMDAT1/ MSTJM(20),PARJM(20),OIN(100),OUT(1000),MXNDJM`

  `MSTJM` is a vector of switches used to define the network to be used:

  `MSTJM(1)` (D=1) number of dimensions in the net
  `MSTJM(2)` (D=0) symmetry of initial weight values
  $0 \rightarrow \omega_{jk} \in [-\omega_0, \omega_0]$
  $1 \rightarrow \omega_{jk} \in [0, \omega_0]$
  `MSTJM(3)` (D=1) overall response function used in the net
  $1 \rightarrow g(x) = \frac{1}{1+\exp(-2x)}$
  $2 \rightarrow g(x) = \exp(-x)$
  `MSTJM(4)` (D=0) error measure
  $0 \rightarrow$ summed square error
  `MSTJM(5)` (D=0) updating procedure
  $0 \rightarrow$ normal self-organized clustering
  $1 \rightarrow$ as above, but with winner node picked at random
  $2 \rightarrow$ learning vector quantization
  `MSTJM(6)` (D=6) file number for output statistics
  `MSTJM(7)` (D=0) magnitude of weight vector

$0 \rightarrow$ free
$1 \rightarrow$ normalized
MSTJM(8) (D=0) neighbourhood (λ) size
$> 0 \rightarrow$ square neighbourhood
$< 0 \rightarrow$ circular neighbourhood
MSTJM(9) not used
MSTJM(10+I) number of nodes in dimension I (I=0 $\rightarrow$ input nodes)
MSTJM(10) (D=8)
MSTJM(11) (D=10)
MSTJM(12) (D=1)
MSTJM(13-20) (D=0)

Switches 5,6,7 and 8 can be changed at any time by the user, the others are only active before the network is initialized with subroutine JMINIT.

PARJM is a vector of parameters determining the performance of the net:

PARJM(1) (D=0.01) learning parameter η ('learning rate')
PARJM(2) (D=0.0) not used
PARJM(3) (D=1.0) overall inverse network temperature 1/T
PARJM(4) (D=0.1) width of initial weights = ω_0
PARJM(5-20) not used

All parameters can be changed at any time by the user.

OIN is the vector used to pass the values of the input nodes to the program.

OUT is a vector used to pass the desired value of the output nodes, if learning vector quantization is used, to the program during training with JMTRAL and to pass the values of the output nodes produced by the program given an input pattern in OIN (routines JMTEST and JMTRAL).

MXNDJM is the index of the winner node in the vector OUT.

B.2 The subroutines

- `SUBROUTINE JMDUMP(NF)`

  Writes all relevant information of a network to the file `ABS(NF)`. If `NF` is negative the output is unformatted otherwise it is formatted. The user must make sure that the corresponding file is opened with write access accordingly.

- `SUBROUTINE JMINDX(INOD,I,J)`

  If the network is multi-dimensional, `JMINDX` is used to transform network coordinates `(I,J)` to and from the index `INOD` used in the vector `OUT`. If `INOD` is > 0, network coordinates `(I,J)` are returned. If `INOD` equals 0, network coordinates `(I,J)` are used to calculate `INOD`.

- `SUBROUTINE JMINIT`

  Initializes the network according to the switches set in `/JMDAT1/` and gives start values to the weights.

- `SUBROUTINE JMINWE(INOD)`

  Sets the weight vector for network node `INOD` equal to the current values in vector `OIN`.

- `SUBROUTINE JMREAD(NF)`

  Reads the information from the file `ABS(NF)` produced by `JMDUMP` and initializes the network described there. All switches and parameters in common block `/JMDAT1/` set prior to a call to `JMREAD` are lost. The comments about the file number for `JMDUMP` applies also here.

- `SUBROUTINE JMSTAT(IS)`

  Writes out information about the network on file number `MSTJM(6)`.
  `IS`=1 gives a header and number of nodes in each dimension (done automatically when the network is initialized).
  `IS`=2 gives the switches and parameters used in common block `/JMDAT1/`.

- `SUBROUTINE JMTEST`

  Takes the pattern stored in the vector `OIN` and uses the current values of the weights in the network to produce an output pattern which is stored in the vector `OUT`.

- `SUBROUTINE JMTRAL`

  Takes the pattern stored in vector `OIN` and trains the network. If learning vector quantization is used, a corresponding output must be stored in vector `OUT` before `JMTRAL` is called.

References

[1] L. Lönnblad, C. Peterson and T. Rögnvaldsson, "Finding Gluon Jets with a Neural Trigger", *Physical Review Letters* **65**, 1321 (1990).

[2] L. Lönnblad, C. Peterson and T. Rögnvaldsson, "Using Neural Networks to Identify Jets", *Nuclear Physics* **B 349**, 675 (1991).

[3] P. Bhat, L. Lönnblad, K. Meier and K. Sugano, "Using Neural Networks to Identify Jets in Hadron-Hadron Collisions", LU TP 90-13 (to appear in *Proc. of the 1990 DPF Summer Study on High Energy Physics Research Directions for the Decade, Colorado, 1990*).

[4] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning Internal Representations by Error Propagation", in D. E. Rumelhart and J. L. McClelland (eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition (Vol. 1)*, MIT Press (1986).

[5] See e.g. R. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons (1973).

[6] J. MacQueen. "Some Methods for Classification and Analysis of Multivariate Observations", in L. M. LeCam and J. Neyman (eds.), *Proc. 5th Berkeley Symposium on Math. Stat. and Prob.*, U. California Press, Berkeley (1967).

[7] T. Kohonen, "Self Organized Formation of Topologically Correct Feature Maps", *Biolological Cybernetics* **43**, 59 (1982);
T. Kohonen, *Self-organization and Associative Memory*, third edition, Springer-Verlag, Berlin, Heidelberg (1990).

[8] G. Tattersall, "Neural Map Applications", in I. Aleksander (ed.) *Neural Computing Architectures*, North Oxford Academic (1989).

[9] L. Lönnblad, C. Peterson, H. Pi and T. Rögnvaldsson, `JETMAP 1.0` program and manual [Available via BITNET request].

[10] T. Kohonen, G. Barna and R. Chrisley, "Statistical Pattern Recognition: Benchmarking Studies", *Proceedings of the IEEE Second International Conference on Neural Networks*, San Diego, California (1988).

[11] J. Moody and C. Darken, "Learning with Localized Receptive Fields", in D. Touretzky, G. Hinton and T.S. Sejnowski (eds.), *Proceedings of the 1988 Connectionists Models Summer School, Carnegie Mellon University*, San Mateo, CA: Morgan Kaufmann (1988).

[12] I. Scabai, F. Czakó and Z. Fodor, "Quark and Gluon Jet Separation using Neural Networks", ITP Budapest Report 477 (1990).

[13] C. Peterson and E. Hartman, "Explorations of the Mean Field Theory Learning Algorithm", *Neural Networks* **2** 475 (1989).

[14] D.H. Ackley, G.E. Hinton and T.J. Sejnowski, "A Learning Algorithm for Boltzmann Machines", *Cognitive Science* **9**, 147 (1985).

[15] C. Peterson and J.R. Anderson, "A Mean Field Theory Learning Algorithm for Neural Networks", *Complex Systems* **1**, 995 (1987).

[16] M. Z. Akrawy et. al., "Event Shape Distributions in the Hadronic Decays of the Z^0", *Z. Phys* **C47**, 505 (1990).

[17] L. Lönnblad, "ARIADNE-3, A Monte Carlo for QCD Cascades in the Colour Dipole Formulation", Lund preprint LU TP 89-10.

[18] G. Marchesini and B.R. Webber, "Monte Carlo Simulations of General Hard Processes with Coherent QCD Radiation", *Nuclear Physics* **B310**, 461 (1988);
I. G. Knowles, "Spin Correlations in Parton-parton Scattering", *Nuclear Physics* **B310**, 571 (1988).

[19] G. Gustafson, "Dual Description of a Confined Gluon Field", *Physics Letters* **B175**, 453 (1986);
G. Gustafson, U. Pettersson, "Dipole Formulation of QCD Cascades", *Nuclear Physics* **B306**, 746 (1988);
B. Andersson, G. Gustafson, L. Lönnblad, "Gluon Splitting in the Colour Dipole Cascades", *Nuclear Physics* **B339**, 393 (1990).

[20] B. Andersson and G. Gustafson, "Semiclassical Models for Gluon Jets and Leptoproduction Based on the Massless Relativistic String", *Z. Phys.* **C3** (1980) 223;
B. Andersson, G. Gustafson, G. Ingelman, T. Sjöstrand, "Parton Fragmentation and String Dynamics", *Phys. Rep.* **97**, 31 (1983).

[21] T. Sjöstrand, JETSET 7.2 program and manual. See B. Bambah et al., QCD Generators for LEP, CERN-TH.5466/89.

[22] L. Lönnblad, C. Peterson and T. Rögnvaldsson, JETNET 2.0 program and manual [to be released; an earlier version JETNET 1.1 is available via BITNET request].

[23] J.W. Gary, Private communication.

[24] B.A. Kniehl, J.H. Kühn, and R.G. Stuart, "Hadronic and Photonic Corrections to A_{LR}", in *Polarization at LEP (Vol. 1)*, CERN-88-06.

[25] G. Arnison et. al. (UA1 collaboration), "Analysis of the Fragmentation Properties of Quark and Gluon Jets at the CERN SPS $p(\bar{p})$ Collider", *Nuclear Physics* **B276**, 253 (1986).

[26] B. Andersson, G. Gustafson, T. Sjöstrand, "How to find the Gluon Jets in e^+e^- Annihilation", *Physics Letters* **B94**, 211 (1980).

[27] L. Bellantoni, J. Conway, J. Jacobsen, Y.B. Pan and S.L. Wu, "Identifying b-jets with a Neural Network Method", to be published.