May 1990
LU TP 90-8

Using Neural Networks to Identify Jets

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

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

Nuclear Physics **B 349**, 675 (1991)

Abstract:

A neural network method for identifying the ancestor of a hadron jet is presented. The idea is to find an efficient mapping between certain observed hadronic kinematical variables and the quark/gluon identity. This is done with a neuronic expansion in terms of a network of sigmoidal functions using a gradient descent procedure, where the errors are back-propagated through the network. With this method we are able to separate gluon from quark jets originating from Monte Carlo generated e^+e^- events with $\sim$ 85% accuracy. The result is independent on the MC model used. This approach for isolating the gluon jet is then used to study the so-called string effect.

In addition, heavy quarks (b and c) in e^+e^- reactions can be identified on the 50% level by just observing the hadrons. In particular we are able to separate b-quarks with an efficiency and purity, which is comparable with what is expected from vertex detectors. We also speculate on how the neural network method can be used to disentangle different hadronization schemes by compressing the dimensionality of the state space of hadrons.

PACS numbers: 13.65.+i, 12.38Qk, 13.87.Fh

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

1 Introduction

During the last couple of years there has been an upsurge in interest for brain-style computing in terms of artificial neural networks (NN). The origin of this enthusiasm is the power this new computational paradigm has shown for a wide variety of real-world feature recognition applications. Not only is the performance of the NN promising but the entire approach is very appealing with its adaptiveness and robustness. Another attractive feature is the inherent parallelism in neural networks and the feasablity of making custom made hardware with fast execution times and thereby facilitating real-time performance.

High energy physics contain many feature recognition problems, ranging from low-level trigger conditions in experimental setups, to extraction of theoretically relevant quantities in collected data. Needless to say, the demand for efficient feature extraction procedures will become more acute with increasing luminosity and energy. In a previous paper [1] preliminary results for gluon/quark separation in e^+e^-reactions using the neural network approach by only considering energy and momentum information from the leading particles were reported. The present paper contains, among other things, elaborations and extensions of those results with respect to the inclusion of secondary quarks, using more than one Monte Carlo model for generating the events, inclusion of experimental acceptance effects, and estimates of theoretical limits. The resulting detection efficiency ($\sim 85\%$) exceeds previous state-of-the-art in this field. The approach is also used to identify heavy quarks with very encouraging results. We are able to distinguish b-quarks with an efficiency and purity level, which is comparable with what is expected from vertex detector methods. Also, we pursue the separation of gluons to hadron-hadron reactions. We feel that our results could serve as examples for many other applications areas; we are just scratching the tip of an iceberg.

The neural network approach is nothing but functional fitting to data. In feature recognition situations one wants to construct a mapping F between a set of observable quantities x_k and feature variables y_i.

$$y_i = F(x_k) \tag{1}$$

by fitting F to a set of M known "training" patterns $(x_k^{(p)}; y_i^{(p)})$, $p = 1, ..., M$. Once the parameters in F are fixed one then uses this parametrization to interpolate and find the features of "test" patterns not included in the "training" set. The NN approach consists of a particular choice for F. It is an expansion of sigmoidal units $g(a, T)$ in a network structure (see fig. 1):

$$g(a, T) = \frac{1}{2} \tanh(a/T) \tag{2}$$

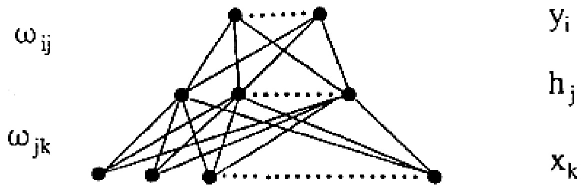

Figure 1: A feed-forward neural network with one layer of hidden units

where the "temperature" sets the gain. The adjustment of parameters , or learning, is done with a gradient descent method, e.g. back-propagation [2].

This paper is organized as follows. In Sect. 2 we briefly describe the back-propagation learning algorithm. Its derivation and details on parameter choices etc. can be found in Appendix A. Appendix B contains a brief description of a F77 software package, JETNET 1.0 [3], that implements the back-propagation learning algorithm for jet identification studies. The MC models used to generate the events and the jet clustering algorithm are described in Sect. 3. Sect. 4 contains the gluon/quark and heavy quark identification in e^+e^--reactions. Gluon/quark identification for jets originating from hadronic collisions can be found in Sect. 5. Using the gluon/quark separator to study the string effect is treated in Sect. 6. A brief Summary and outlook is found in Sect. 7.

2 The Network Learning Algorithm

The basic ingredients in a neural network are **neurons** n_i and connectivity **weights** ω_{ij}. For feature recognition problems like ours the neurons are often organized in a feed-forward layered architecture (see fig. 1) with input (x_k), hidden (h_j) and output (y_i) nodes. Each neuron performs a weighted sum of the incoming signals and thresholds this sum with a sigmoid function $g(a,T)$ (eq. (2)). For the hidden and output neurons one has

$$h_j = g(a_j/T) \tag{3}$$

$$y_i = g(a_i/T) \tag{4}$$

where the "temperature" T sets the slope of g. The weighted input sums a_j and a_i are given by $\sum_k \omega_{jk} x_k$ and $\sum_j \omega_{ij} h_j$ respectively.

The hidden nodes have the task of correlating and building up an "internal representation" of the patterns to be learned. Training the network corresponds to changing the weights ω_{ij} such that a given input pattern $\vec{x}^{(p)}$ gives rise to an output (feature) value $\vec{y}^{(p)}$ that equals the desired output or target value $\vec{t}^{(p)}$. A frequently used procedure for accomplishing this is the **back-propagation** learning rule [2] where the **least mean square** error function

$$E = \frac{1}{2}\sum_{p}\sum_{i}(y_i^{(p)} - t_i^{(p)})^2 \tag{5}$$

is minimized. Changing ω_{ij} by gradient descent corresponds to [2] (see Appendix A)

$$\Delta\omega_{ij} = -\eta\delta_i h_j + \alpha\Delta\omega_{ij}^{old} \tag{6}$$

for the hidden to the output layer, where δ_i is given by

$$\delta_i = (y_i - t_i)g'(a_i) \tag{7}$$

Correspondingly for the input to the hidden layer one has

$$\Delta\omega_{jk} = -\eta\sum_{i}\omega_{ij}\delta_i g'(a_j)x_k + \alpha\Delta\omega_{jk}^{old} \tag{8}$$

In eqs. (6,8) η is a learning strength parameter and we have also included so-called momentum terms $\alpha\Delta\omega_{ij}^{old}$ and $\alpha\Delta\omega_{jk}^{old}$ in order to damp out oscillations. This procedure is repeated for each pattern p until the network has learned all patterns to a satisfactory level. As will be discussed in connection with our applications below termination is set by performance quality on a set of patterns that are not included in the learning process. Suitable choices of parameters, η, α, T and initial weight range, are discussed in Appendix A.

The error function of eq. (5) and the corresponding updating rules (eqs.(6,8)) are by no means the only possibilities. In Appendix A we discuss two alternatives, **entropy error** and **Kullback error**, where the latter is handy in situations with more than one output unit. Also, the gradient descent can be modified into a so called **"Manhattan"** updating (see Appendix A). There is, of course, also the option of having more than one layer of hidden nodes. However, it turns out that for most applications in this paper the plain "vanilla" version of eqs. (5-8) is sufficient.

An important question is what NN architecture (number of layers, hidden nodes and degree of connectivity) to use for a particular problem. Clearly one should use as few parameters as possible, in order to have powerful generalization properties. Compare with the situation in curve fitting, where an abundance of parameters might lead to a non-smooth curve between the data points giving rise to poor interpolation. A straight-forward but costly way to get a rough feeling for the relevant architecture given a particular class of problems is of course trial-and-error. However, it would be advantageous to have a more algorithmic method available, since one could then also analyze how the network captures the different features of the data set. One such **pruning** procedure goes as follows [4]: Add to the error function (eq. (5)) a complexity term [4]

$$E \rightarrow E + \lambda \sum_{ij} \frac{\omega_{ij}^2}{1+\omega_{ij}^2} \tag{9}$$

where the sum extends over all weights. For large weights $|\omega_{ij}|$, the cost is λ, whereas for small weights it is zero. Hence the network gets pruned to only contain weights that are really needed to represent the problem. The choice of λ requires some fine-tuning. Typically λ is incremented ($\lambda \rightarrow \lambda + \epsilon$) whenever the error (eq. (5)) decreases with a very small value on ϵ. We stress that this procedure should only be done in a "research mode" and not when doing the final neural network learning.

3 The Monte Carlo Data

3.1 The Models

The Monte Carlo data used in our analysis were generated with three different generators; ARIADNE 3.1 [5], HERWIG 3.4 [6] and JETSET 7.2 [7]. All three generators are able to give a good description of event shapes, multiplicities etc. at LEP energies [8], although they differ substantially in the detailed description of e.g. heavy quark fragmentation. The main difference is that HERWIG uses a cluster fragmentation scheme whereas JETSET and ARIADNE uses the Lund string fragmentation [9] as it is implemented in JETSET. Among other things this results in softer fragmentation functions for heavy quarks in HERWIG.

The generators also differ in the treatment of the perturbative stage of the jet evolution. However they all include coherence effects due to soft and colinear gluon emission, so even if the implementation is quite different (eg. ARIADNE uses a colour dipole cascade [10] whereas HERWIG and JETSET uses partonic cascades [6, 11]) the observable differences are quite small.

All options and parameters in each generator were set to their default values when the event samples for the neural network analysis were produced.

3.2 Clustering the Data

After generating an event, the jets are defined using a clustering algorithm. We have chosen to use the `LUCLUS` algorithm contained in JETSET. The method used in LUCLUS is to form clusters of particles which are close together in phase space where closeness is defined as the relative p_T of two particles. All particles which are closer together than a certain cut are in this manner clustered together into a jet. In most cases in this paper, this cut was set to 2.5 GeV, with an additional constraint requiring at least three jets. An exception is the "forced three-jet" events, where the cut was set to infinity requiring exactly three jets.

A quark jet is the defined as the jet closest to a quark, with the closeness defined in the same way as in the clustering. All jets not assigned to a quark are defined as gluon jets. Note that we also include secondary quarks from the perturbative splitting of a gluon into a quark-anti-quark pair, and that in the case where both the quark and the anti-quark ends up in the same jet, this is taken to be a gluon jet.

We have only used events where the jets are well separated (minimum angle between two jets is 40°). In addition we have required at least four particles in each jet and that no jet has less than 5% of the total center of mass energy.

In the analysis of the string effect, slightly different cuts were used (see below).

4 e^+e^--Annihilation

4.1 Separating Quarks from Gluons

Being able to distinguish whether a jet of hadrons originates from a quark or a gluon is important from many perspectives. It can shed light on the hadronization mechanism. For example experimental studies of the so-called string effect [12] needs identification of the gluon jet. Also a fairly precise identification of the gluon jet is required for establishing the existence of the 3-gluon coupling in e^+e^--annihilation [13, 14]. To date the gluon jet identification has been done by making various cuts on the kinematic variables. The most straightforward one is to select the jet with smallest energy as the gluon [12]. This procedure, which is based on the underlying perturbative QCD matrix element, typically yields $\sim 65\%$ identification rate. More elaborate schemes have been suggested [14, 15, 16] with improved performance ($\sim 70-75\%$) as a result.

Here we will use the back-propagation learning algorithm to do quark/gluon identification. Preliminary encouraging results with this method were reported in [1] using the ARIADNE Monte Carlo. In this section we give a more complete treatment of this problem with several extensions, among others the inclusion of other different MC models, allowing for secondary quarks in the MC data, and using entire

e^+e^--events as input to the network.

Events are generated with all the three models, ARIADNE, JETSET and HERWIG, at two different energies, **29** and **92** GeV respectively. The clustering algorithm described in Sect. 3 is used to find the jets in each event. The data is then compiled according to two different options:

- **Single jets**. The data set consists of jets with no correlation to their event origin. In this case the network sees only one jet at a time and knows nothing about the total number of jets in the event or their relative spatial orientation.
- **3-jet events**. The data set consists of forced (see above) 3-jet events.

For each of the different data sets we use two different approaches of presenting the jets to the network.

- **L4** uses as inputs the four-momenta $(\vec{p}_k, E_k)$ of the four leading particles in the jet (k=1,..,4). In this way we do not reveal too much about the structure of the low momentum part of the jets. Hence e.g. the string effect might be studied in a fairly model-independent way.
- **JL1** uses as inputs the total energy and momentum of the jet together with the four-momentum of its leading particle. The model-dependency is thus reduced when studying details of the jet structure, such as asymmetries.

Each data set is divided into two parts; one that is used for training the network (**training set**) and one that is used for testing the ability of the network to classify the jets (**test set**). We thus make sure that the network is tested on patterns which it has never seen before and that we really measure its ability to generalize.

We next discuss the details of the learning for single jets and 3-jet events separately.

4.1.1 Single Jets Learning

The relative sizes of all training and test sets are 2:1. For both the training and testing phases patterns are picked at random from respective set in such a way that the network is presented with equal amounts of gluon jets and quark jets. In the L4 input mode the network consists of 16 linear input units, 10 hidden neurons and one output neuron. In the JL1 case there are 6 linear input units, 6 hidden neurons and one output neuron. The network performance is not very sensitive to the number of

Model	$\sqrt{s}$	Input mode	udscb	uds
ARIADNE	29 GeV	JL1	83% (87/81)	85% (87/83)
		L4	84% (85/83)	85% (90/81)
	92 GeV	JL1	84% (92/78)	85% (89/83)
		L4	85% (89/82)	85% (91/80)
JETSET	29 GeV	JL1	82% (89/79)	84% (86/82)
		L4	84% (79/86)	84% (86/81)
	92 GeV	JL1	84% (90/80)	85% (89/80)
		L4	84% (85/83)	84% (90/79)
HERWIG	29 GeV	JL1	84% (73/90)	82% (76/88)
		L4	83% (83/83)	84% (86/83)
	92 GeV	JL1	87% (93/82)	88% (94/81)
		L4	86% (92/82)	88% (94/81)

Table 1: Percentage correct classifications (averaged over q and g) of jets for different models, energies, input and flavour options (see text). The numbers within parentheses denote how the success rate splits up into quarks and gluons respectively. The errors are about 1% for the overall classification numbers.

hidden units (see discussion below). The output unit is used to code the jet identity; 1 for gluon and 0 for quark. We use strict middle point success condition; if the output is > 0.5 the jet is interpreted as a gluon jet and if the output is < 0.5 the jet is interpreted as a quark jet.

The weights in the network are initialized at random with values in the range [-0.1,0.1]. For every pattern $\Delta\omega_{ij}$ and $\Delta\omega_{jk}$ are computed (eqs. (6,8)) and recorded. Actual updating takes place with these numbers summed over 10 patterns. The parameters used are α=0.5, T=1 and η=0.01. The performance is remarkably insensitive to these choices, although a too high learning rate η leads to the network not being able to learn at all (see Appendix A).

In fig. 2 a typical learning curve is shown. As can be seen the network to a good approximation learns to recognize the feature even before it has seen the entire learning set once (one "epoch"). In table 1 the performance for different models, energy and input options are displayed. The results are impressive[4]. For completeness we have also included the case when heavy quarks (c and b) are not present, using this reduced set only marginally improves the performance.

In order to check the model independence of our approach we have also made "mixed" runs, where we train on one MC model and test on another (see table 2). As can be

[4] The numbers reported here are slightly lower than those in ref. [1], since we here also allow for the possibility of secondary quarks.

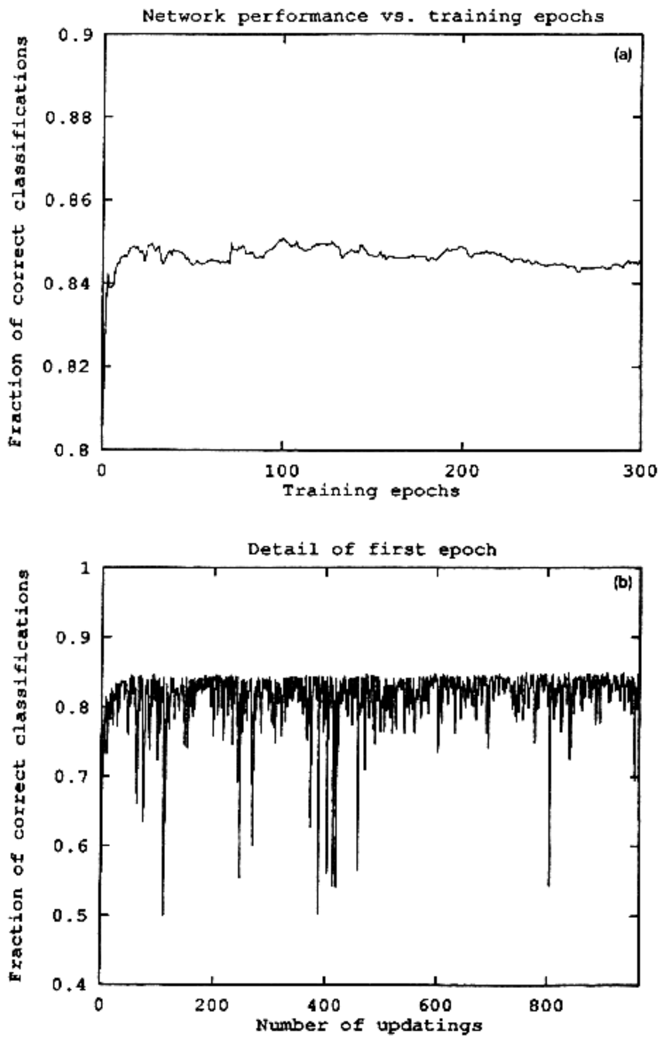

Figure 2: (a). Network performance as a function of training epochs using a 92 GeV (ARIADNE) data set with the L4 input option. (b). The first epoch displayed in detail.

seen the approach is indeed very model independent.

Two important questions need to be answered. How well can the network in principle perform given a certain data set? What characteristics in (or correlations) does the network utilize?

The Bayesian limit. The upper limit of performance is set by the Bayesian limit [17], which is given by the minimal overlap between the two multidimensional distributions. An estimate of this limit can be obtained numerically within realistic

	ARIADNE	JETSET	HERWIG
ARIADNE	85	85	86
JETSET	84	86	87
HERWIG	83	85	87

Table 2: Percentage correct classifications of 92 GeV jets for "mixed" runs. Rows indicates training set and columns test set. The values are for L4 representation and both heavy and light quarks.

	ARIADNE	JETSET	HERWIG
Theoretical limit	87	88	89
Neural Network	85	86	87

Table 3: Comparison of neural network prediction with the theoretical limit.

CPU consumption by reducing the accuracy of the kinematical variables and using the total energy and momentum of the jet. By dividing the kinematical ranges into 100×100 bins the integrations yielded the results shown in table 3. As can be seen from table 3 the neural network prediction is just a few % below the theoretical limit. We feel confident that with proper adjustments and parameter fine-tuning of the learning algorithm the theoretical limit can be reached.

We have tried using a "Manhattan" [18] updating algorithm (see Appendix A) when training the network since it has been shown to work better on inconsistent data sets but we found no improvement.

Correlations used by the network. It is clear from the sharp rise of the learning curve in fig. 2, that some distinguishing properties of the data are obvious and immediately picked up by the network. This striking property is the fact that the gluon jets often have smaller energy as predicted by the leading order QCD matrix element and utilized in ref. [12]. But the network is more subtle than just making an energy cut. The way the network solves the problem is very transparent if we look at the solution when the network is fed with a reduced 2-dimensional input, E_{jet} and p_{jet}. The performance then drops to 80% corresponding to the sharp rise in fig. 2a. In figs. 3a and b we show the distributions of quark and gluon jets in the E_{jet}-p_{jet} plane. From fig. 3 we see that there is a non-negligible region where the two distributions are not trivially separable with e.g. a cut. The network does its best by producing the separating curve. This solution is responsible for the sharp rise.

What determines the remaining few %? In order to answer that question we apply the pruning method of eq. (9), starting with a generous architecture ($O(10)$ hidden

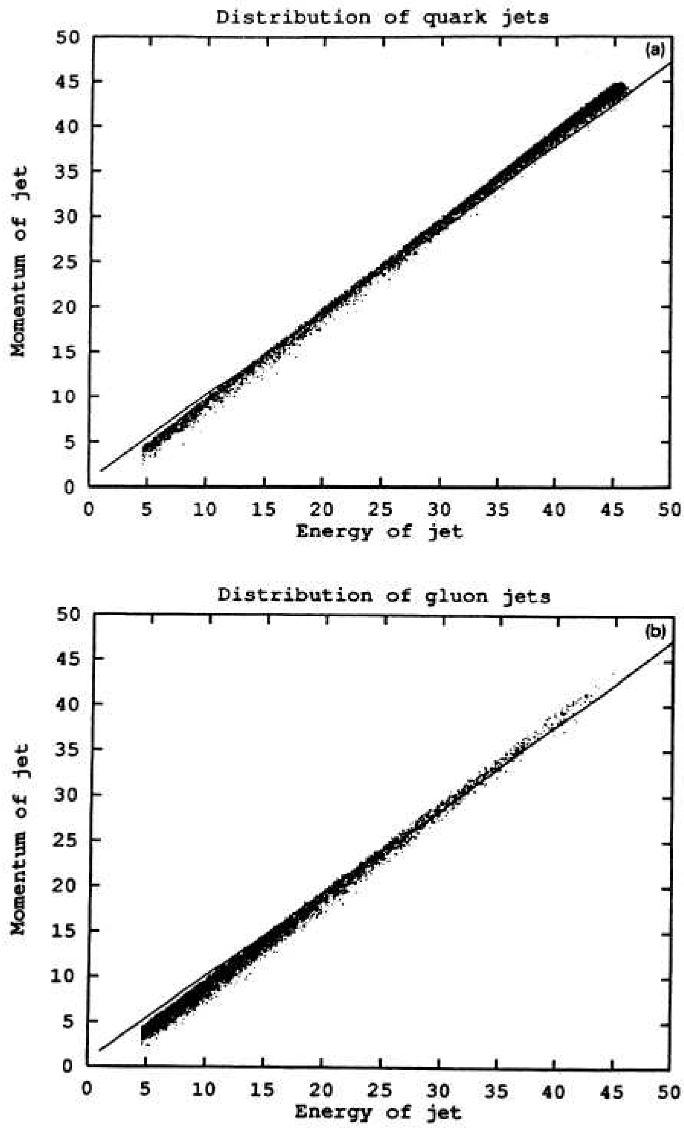

Figure 3: Distribution of quark and gluon events in the E_{jet}-p_{jet} plane. The curve (approximate line) corresponds to the network solution of the problem

units) in the L4 representation. Careful tuning of λ in eq. (9) puts most weights to zero. What remains are those weights connecting p_y, p_z and energy of the hadrons to the hidden layer and two of the hidden nodes to the output layer. The overall performance is very little affected by this procedure. This shows that the network needs at most two hidden units to achieve its goal. Representing the line in the E_{jet}-p_{jet} plane (fig. 3) needs no hidden units at all!

4.1.2 3-jet Event Learning

The analysis above was based on single jets processed through the network. In an experimental situation it is more natural to consider an entire event. The question then arises how to feed an arbitrary event into the network. Since the network needs a fixed number of input units we confine ourselves to samples with a fixed number of jets. We have chosen forced 3-jet events (see Sect. 3.2 above). An architecture of 3×16 input, 10 hidden and 3 output units is used (L4 representation), where the output units represent whether jet #1, #2 or #3 is the gluon. In other words syntactically correct answers are (100), (010) or (001). These are used for training as target values. For testing purposes we adopt a **winner-takes-all** interpretation in the sense that the output neuron with largest value is taken to be the gluon. A more compact treatment of 3-feature outputs can be achieved with so called Potts representation (see Appendix A). These are multi-valued neurons with solutions (100), (010) or (001). In this case we stick to the winner-takes-all representation for simplicity since no improvements were obtained with the Potts presentation. The situation is different in the string-effect analysis (see Sect. 5). The learning rate was $\eta = 0.001$, whereas other parameters, α, T and updating procedures were the same as in the single jet case above. The relative training set/test set size used is again 2:1. All identifications were made with the ARIADNE generator. In addition to present the jets in "raw" form to the network, two modifications were tried. One was to rotate the momenta onto a plane and the other to supplement the momentum information with two input units representing relative angles between the jets expressed in radians. As can be seen from table 4 the results are independent of these optional choices. This is not too surprising since all the information is already present in the "raw" mode.

It is very satisfying that the neural classifier is able to separate gluons from quarks in an event environment as well as when presented in the single jet mode.

All exercises so far have taken place in an ideal world with no acceptance limitations, all particles detected etc. In order to partly emulate an experimental situation we have chopped the precision of ingoing momenta and energies to ± 0.5 GeV. The corresponding performance of the network only degrades by 1%. A check on how our approach might work under real experimental conditions has been done where the

input structure	performance
"raw" data	85%
rotated data	85%
"raw" data + 2 angles	85%

Table 4: Percent correct classifications of 3-jet events. The network was trained to find which one of three jets that is the gluon jet.

MC models have been processed through the DELPHI detector simulator [19]. We find only a modest degradation in performance (~3 %), which is very encouraging.

4.2 Identifying Heavy Quarks

Identifying heavy quarks from light quarks is another important pattern recognition task. We will here pursue the possibility of detecting heavy quarks by considering hadron distributions only, as in the gluon/quark separation case above. We use two different ways to study the problem; one where the network is taught to classify the (single) jets into three different classes - gluon, light quark and heavy quark - and one where the network only distinguishes between a heavy quark jet and any other jet.

In tagging processes it is not only the **efficiency** (success rate) which is important but also the **purity** of the tagged sample. We have therefore in both cases compared the efficiency of the network trigger with the purity of the classified sample.

4.2.1 Gluon/light quark/heavy quark classification

We present single L4 jets to the network. The network architecture is 16 input units, 10 hidden neurons and 3 output neurons. The output neurons code the three classes gluon (100), light quark (010) and heavy quark (001). By "light quark" we mean u,d or s. When testing the network we use a "winner takes all" criteria for the output neurons (see sect. 3).

In table 5 we show the efficiencies (and purities of the resulting samples) for the different MC models are shown. The results do not differ much between the models, although some characteristics can be extracted; ARIADNE seems to produce the "cleanest" light quark sample while HERWIG has the highest combination of efficiency and purity for heavy quarks at 92 GeV. These are really specific signatures of

Model	$\sqrt{s}$	gluons	light quarks	heavy quarks
ARIADNE	29 GeV	82% (70%)	54% (59%)	45% (48%)
	92 GeV	85% (75%)	54% (80%)	50% (35%)
JETSET	29 GeV	84% (69%)	54% (61%)	41% (44%)
	92 GeV	85% (80%)	50% (72%)	54% (38%)
HERWIG	29 GeV	85% (67%)	55% (55%)	44% (46%)
	92 GeV	87% (81%)	55% (76%)	56% (42%)

Table 5: Efficiency for different MC generators for three output neurons. The numbers within brackets are the purities of the samples.

Training set	gluons	light quarks	heavy quarks
ARIADNE	85% (82%)	53% (72%)	51% (43%)
JETSET	85% (80%)	50% (72%)	54% (38%)
HERWIG	84% (79%)	49% (66%)	47% (34%)

Table 6: Efficiencies for testing on a data set generated by JETSET at 92 GeV for networks trained on data sets generated by the three different MC generators. The numbers within brackets are the purities of the resulting samples.

the individual MC generators as will be discussed more in the next section. To check the MC independence for the training of the network we have trained the network on a data set generated by one MC model and then tested on a data set generated by another model. The results for networks trained on different data sets at 92 GeV and then tested on a data set generated by JETSET at 92 GeV are found in table 6. The success rates do not differ much, indicating that the training is rather insensitive to which MC model that is used. On the other hand the different purity values imply that, although the available four-momentum space region for the jets is the same for the different models, the distribution of these jets in four-momentum space is somewhat different from model to model.

4.2.2 b-quark/others classification

A commonly used signature for a b- or a c-quark jet is the presence of a fast lepton. Branching ratios are $O(10\%)$ so many events are lost. On the other hand the purity levels are high. A more ambitious (and expensive) approach is to use a vertex detector to find the secondary vertex. With such a device one expects about 25% tagging efficiency for heavy quarks (c+b) with 20% background from light quarks [20]. Since there are roughly as many c as b-quarks at 92 GeV e^+e^-reactions, the background for identifying a b-jet with this method is actually 60%

Ideally one would prefer 2-jet events only so that no additional confusion is added by the presence of gluon jets, since these are often broader like the heavy quark jets. However, at 92 GeV there are very few clean 2-jet events so we have chosen to only consider the jet with the largest energy from every multijet event as input to the network. In this way we reduce the gluon background and improve the results considerably. As input variables we use the total jet energy and momentum along with the energy and direction for the 6 leading particles in the jet, giving a total of 20 input units, corresponding to a calorimeter with infinitely high angular resolution. The binary answer is represented by a single output neuron (b-jet=1, non-b-jet=0). The number of hidden nodes is 10.

In fig. 4 we show output value distributions for b-jets and non-b-jets. As can be seen from fig. 4 it is not clear that a decision threshold value $\theta_{out} = 0.5$ as used in the quark/gluon single jet studies above gives an optimal mixture of efficiency and purity. In fig. 5 we give efficiency-purity plots resulting from neural network learning of data generated at 92 GeV with ARIADNE and HERWIG MC as described above. The figure for JETSET looks much the same as for ARIADNE and is not included.

As can be seen from figs. 5a and b the expected result of a neural b-quark trigger depends on which MC generator is used. The HERWIG MC is generally more optimistic, suggesting the possibility of detecting b-quarks with purity/efficiency levels higher than in the vertex detector (assuming no b/c separation is made). The JETSET and ARIADNE MC are more modest, indicating purity/efficiency levels comparable with the vertex detector. A direct comparison between figs. 4a and b shows that this is really a signature for the different models. The HERWIG b-quark jets are more "typical" than the ARIADNE (and JETSET) jets. Note however that the purity/efficiency results are *independent* of the model used for the training. We see here a possibility to distinguish between different MC models!

In order to more directly compare the neural network with the vertex detector, we have trained the network to classify heavy quarks (c+b) and other jets. The results for the different MC models are compatible with the vertex detector, with JETSET and ARIADNE slightly below. We find the results to be very impressive!

One could of course also imagine powerful combinations of vertex and neural triggers where the vertex selects (c+b) and then the NN separates b from c. Also a neural network model could be employed to select b-quarks decaying semileptonically as demonstrated in [21] using calorimeter signals.

An interesting question is to what extent the identified b-quark jets contain μ-mesons. We found that the muon content in the whole b-jet sample is about 11-13%, which

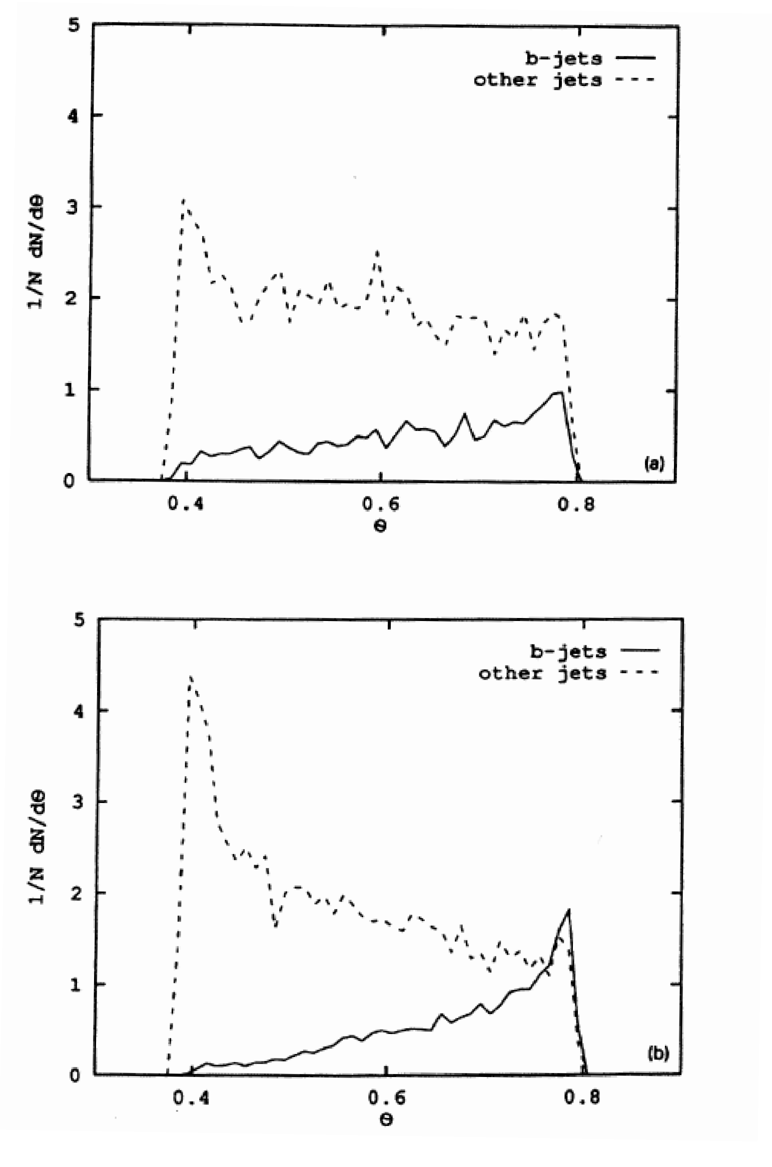

Figure 4: Distribution of output node values for b-jets and non-b-jets respectively using a network trained on ARIADNE data. (**a**) Testing on ARIADNE data. (**b**) Testing on HERWIG data.

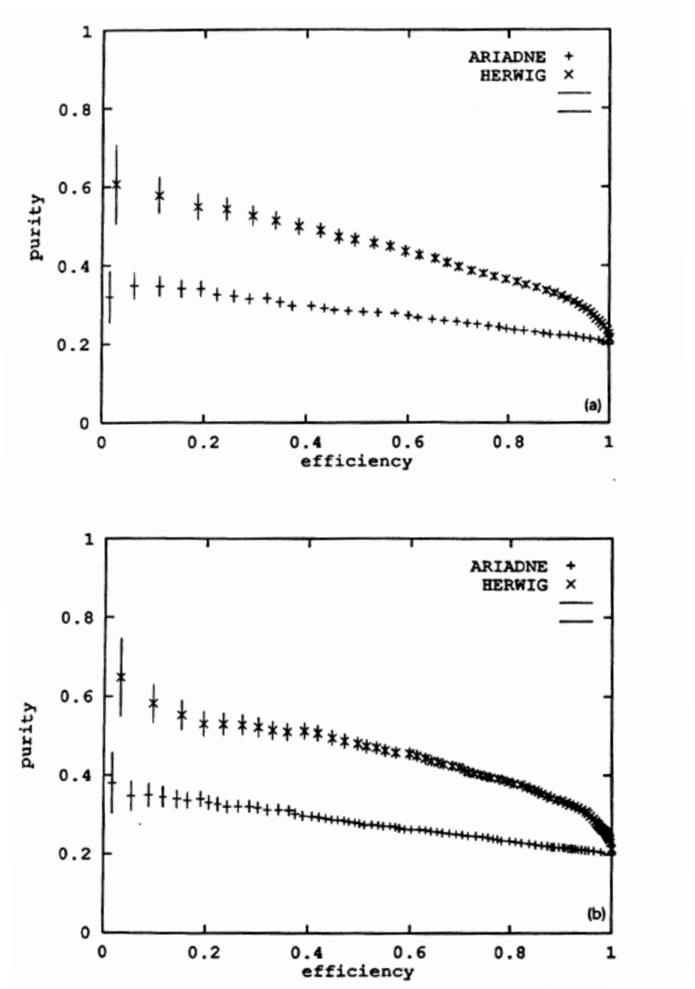

Figure 5: Purity-efficiency plot for b-quark identification on ARIADNE and HERWIG data using a network trained on (**a**) ARIADNE and (**b**) HERWIG data.

remains constant for the enhanced samples. This shows that the network has not learned to recognize muonic decay but has found some other signature of the b-quark jets. In other words, the NN method supplements leptonic triggering. This also means that leptonic triggering can be used to calibrate the NN trigger in experiments.

All the results for b-quark tagging reported so far has been done in the single jets mode. Similar runs could of course be done for 3-jet events.

If the objective of b-tagging is to measure $b\bar{b}$ mixing then one needs to know the charge of the $b(\bar{b})$ jets. Without leptonic information this can be done fairly reliably by considering various weighted charge measures [22], i.e.

$$Q_4 = \sum_i^4 z_i^{1/3} Q_i \tag{10}$$

where i runs over the L4 set and $z = E_i/E_{jet}$.

5 Large p_T Production

In hadron-hadron collisions the situation is different due to the lack of knowledge about the event structure. The gluon jet is not very likely to have least energy as in the e^+e^-case. Consequently we expect a smaller identification rate. Also the experimental setup is normally different from e^+e^-experiments. An event is recorded as a calorimeter signal and not as reconstructed energy-momentum pairs. For that reason we will feed our network with transverse energies only. A square of cells are defined around a jet according to two different options:

- 4×4 cells around the energy c.m.s. of the jet
- 7×7 cells centered around the calorimeter cell with the largest energy (the "tower").

In both cases the network performance never exceeds 70% in performance.

We have here tried using selective input fields for the hidden units (allowing one hidden unit to see only the central calorimeter cell and another to see only the closest surrounding cells etc..) hoping that this would make it easier for the network to see an energy profile for the jet but we have found no improvement in network performance for these architectures. Nevertheless we have included selective fields as an option in JETNET 1.0.

6 The String Effect Revisited

Using the NN approach to identify gluon jets when measuring the string effect [12, 23] is potentially very dangerous. It is very likely that the network has already used the string effect when learning to recognize the gluon, in which case the measurements will be very model dependent. In the following analysis we have tried to teach the

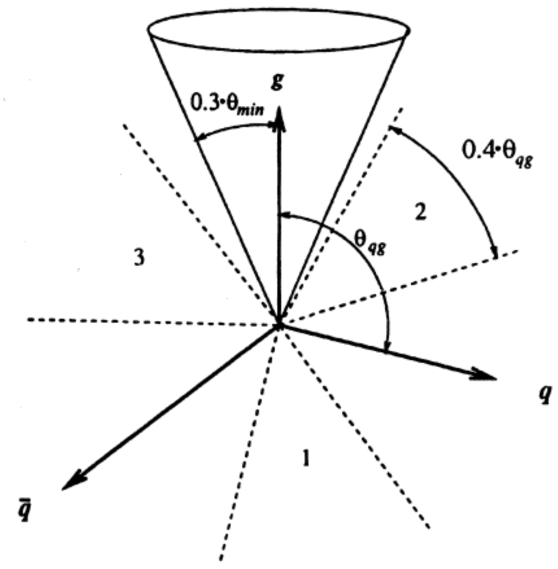

Figure 6: The sectors in the plane of an three-jet event used for the measurements of the string effect

network as little as possible about the string effect and in particular show it nothing about the regions where we perform the measurements in an event.

We have chosen to take as a measure of the string effect the ratio $r = < n_2 > / < n_1 >$ where $< n_2 >$ and $< n_1 >$ are the multiplicities in the sectors between the quark and the gluon jet and the quark and the anti-quark jet respectively as defined in fig. 6. The event sample was generated using "forced three-jet" clustering, requiring a minimum angle θ_{min} of 80° between the jets, more than 10% of the total c.m. energy in the smallest jet and at least four particles in each jet.

An architecture with 48 input nodes, 12 hidden and 3 output nodes was used. The input was the energy and direction (encoded as E, E_x, E_y, E_z) of the four fastest particles in each jet, and we trained the network to give the output $(0,0,1)$, $(0,1,0)$ or $(1,0,0)$, where $(0,0,1)$ corresponds to the gluon jet being the least energetic one etc. To be sure that we do not give the network any information about what we later intend to measure, we only used particles which lay inside a cone of opening angle $0.3 \cdot \theta_{min}$ around each jet (see fig. 6). If there are only three particles in this cone, the nodes corresponding to the fourth particle are set to zero and so on.

Since we do not give the network as much information as in section 4.1 the performance of the network is much lower. In this case it turns out that the performance can be increased slightly if we use Potts neurons (see appendix A). It seems that

	MC truth	Neural network	smallest jet
success rate	100%	74%	67%
$r = < n_2 > / < n_1 >$	2.16 ± 0.03	2.00 ± 0.02	1.60 ± 0.02
$r\ (m_{had} \geq m_K)$	3.0 ± 0.1	2.7 ± 0.1	1.9 ± 0.1

Table 7: Results for the $r = < n_2 > / < n_1 >$ measured on a sample of 10000 MC events (ARIADNE) for different methods of identifying the gluon jet; using the true gluon jet of the MC, the NN approach and the "smallest jet" approach.

the associated asymmetric error measure makes it easier for the network to correctly classify events where the gluon jet is the most energetic one. Using the "winner-takes-all" approach of section 4.1 the network chooses to disregard these events since there are so few of them ($\approx 15\%$).

We used 10000 training patterns and 100 training epochs ($\eta = 0.001, \alpha = 0.5, T = 1$) which yielded a fraction of correctly classified events of 74%. This should be compared with the method of assuming that the least energetic jet is the gluon one, which gives 67%. This does not seem to be much of an improvement, but it does make a big difference in the measurements.

The results for the measurement of r, using MC data generated by ARIADNE, are presented in table 7. The difference between the NN approach and the "smallest jet" approach in measuring r is much larger than what is expected from the difference in gluon tagging performance. This indicates that the network still uses the little information it gets about the string effect when identifying the gluon, and hence generally prefers to make the identification in a manner that increases the string effect. But even if the NN approach in this way is not a very "objective" way of studying the string effect we still feel that it may be an important way of studying the consistency of the model used to compare with data.

In table 7 we also present results for the ratio r when looking only at K-mesons (and heavier hadrons). In ref. [12] it was shown that the string effect is strongly dependent on the particle masses, hence the ratio r is larger for heavy particles than for lighter. At 29 GeV where the string effect is dominated by the hadronization process, this is easily explained within the Lund model [9] simply by looking at the Lorentz boost back to the events c.m. after the string has been fragmented in its c.m. At higher energies this effect is reduced as the string effect becomes more dominated by the perturbative phase of the jet evolution [24]. A recent study [25] indicates that this becomes important already at 92 GeV.

Since the network has not been shown the masses of the particles one should be able to use it in a fairly model independent way, when measuring this dependence of the string effect on the masses.

7 Summary and Outlook

We have used a simple neural network learning algorithm to parametrize various jet features in terms of energy and momentum of the leading particles in the jet. Our results can be summarized as follows:

- Gluons can be separated from quarks in e^+e^--reactions on a 85% level in a Monte Carlo independent way. Only modest degradation is obtained when processing the data through a detector simulator (provided the detector is of high quality).
- The corresponding numbers are lower for hadron induced large p_T jets ($\sim$ 70%).
- b-quarks can be separated from other quarks and gluons on an purity/efficiency level comparable with what is expected from a vertex detector.

These results were obtained with the simplest possible feed-forward network using the back-propagation learning rule. Very little parameter fine-tuning were needed. (In Appendix A and B we have nevertheless included more elaborate options as potential tools for studies of other and more difficult problems in the future.) What determines the success of the approach is the information contained in the observed hadron kinematics. Indeed, when computing the *Bayesian* limit we find that the network almost hits this information-theoretical limit.

It is remarkable that we are able to tag b-quarks on such a high level of efficiency and purity with no leptonic triggers. It seems like a general purpose detector with reasonable neutral detection capability is what is needed.

We have used the NN algorithms to separate different features. Another possible use of NN emerged from the varying θ_{out} in connection with b-quark separation. The NN *compresses* a multidimensional description of a MC data, which is difficult to analyze, into a compact encoding of a few feature variables (*one* in the b-quark case). It might be easier to understand the data produced in different MC models in this way. This is very different from looking at integrated distributions, where information has been averaged away.

We also used the method to study the string-effect. The results here are less "clean" in the sense that the neural network has picked up some of the effect already in the learning phase. However, the approach in this case could be very useful for verifying model consistency.

The NN approach is in general very noise- and damage-resistant. Hence it is suitable for high energy experiments where various parts of a detector might malfunction. Also with its inherent concurrency and simple structure fast execution custom-made hardware could be a real asset for on-line triggering.

Needless to say, the success rate of the NN method will vary with different detectors. Much detailed work with various detectors needs to be done.

One should mention that neural networks have also shown great promise for solving difficult optimization problems [26, 27]. Also in this application area the NN approach could be fruitful for high energy physics in real time track-finding. [28, 29].

Acknowledgements:

We have benefitted from discussions with O. Barring, T. Sjöstrand and P. M. Zerwas.

Appendix A

A.1 Error Measures

In this appendix we derive the back propagation learning rule for three different cases, the summed square error measure of eq. (5), an entropy error measure and the asymmetric error (Kullback) measure for the Potts representation. Very little in this appendix is original; it is included for completeness.

Summed Square Error

This is the case described in Sect. 2 and used throughout most of the paper. We limit ourselves to the case of one hidden layer. The notation follows fig. 1. Gradient descent with respect to ω_{ij} and ω_{jk} for each pattern p corresponds to

$$\Delta\omega_{ij} = -\eta \frac{\partial E}{\partial \omega_{ij}} \tag{A1}$$

and

$$\Delta\omega_{jk} = -\eta \frac{\partial E}{\partial \omega_{jk}} \tag{A2}$$

The chain rule for the partial derivatives with respect to E gives

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial a_i}\frac{\partial a_i}{\partial \omega_{ij}} = \delta_i g'(a_i) h_j \tag{A3}$$

where δ_i is given in eq. (7).

$$\frac{\partial E}{\partial \omega_{jk}} = \sum_i \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial a_i}\frac{\partial a_i}{\partial h_j}\frac{\partial h_j}{\partial a_j}\frac{\partial a_j}{\partial \omega_{jk}} = \sum_i \delta_i \omega_{ij} g'(a_j) x_k \tag{A4}$$

Eqs. (A3,A4) trivially generalize to architectures with more than one hidden layer; the δ's are just back-propagated further down according to equations identical to eq. (8).

Entropy Error

An alternative error measure to eq. (5) is the log-likelihood or entropy measure [30]

$$E = -\sum_p \sum_i [t_i^{(p)} \log y_i + (1 - t_i^{(p)}) \log(1 - y_i)] \tag{A5}$$

For $\partial E/\partial \omega_{ij}$ one then gets for each pattern p

$$\frac{\partial E}{\partial \omega_{ij}} = \hat{\delta}_i h_j \tag{A6}$$

with

$$\hat{\delta}_i = y_i - t_i \tag{A7}$$

The only difference is the absence of $g'(a_i)$ in eq. (A7). For the updating of the input to hidden layer the expression is of course identical to the summed square error case (eqs. A4,8).

Potts Neurons - Asymmetric Error

As discussed in connection with 3-jet events and heavy flavour tagging when the output consists of more than two exclusive alternatives there are two options at our disposal. One is to use 3 normal binary neurons together with a "winner-takes-all" interpretation in the testing phase. The other option is to use *one* multi-state output neuron $\vec{y}$ corresponding to a **Potts** spin in physics. The sigmoid updating rule of eq. (2) is then replaced by

$$y_i = y(a_1, a_2, ..., a_n, T) = \frac{e^{a_i/T}}{\sum_l e^{a_l/T}} \tag{A8}$$

satisfying

$$\sum_i^n y_i = 1 \tag{A9}$$

A suitable error function for this representation is the Kullback measure [31]

$$E = \sum_p \sum_i t_i^{(p)} \log \frac{t_i^{(p)}}{y_i} \tag{A10}$$

For $\partial E / \partial \omega_{ij}$ one gets for each pattern p

$$\frac{\partial E}{\partial \omega_{ij}} = - \sum_k t_k \frac{1}{y_k} \frac{\partial y_k}{\partial a_i} h_j \tag{A11}$$

which with

$$\frac{\partial y_k}{\partial a_i} = \delta_{ik} y_k - y_k y_i \tag{A12}$$

together with $\sum_k t_k = 1$ gives

$$\frac{\partial E}{\partial \omega_{ij}} = \hat{\delta}_i h_j \tag{A13}$$

as in the entropy error case above. Again, the input to the hidden layer updatings are unaffected and identical to eqs. (A4,8).

A.2. Parameters and Options

The training of the network is quite insensitive to the choice of learning parameters. The parameters we have used to achieve the results described in this article are are given below.

Learning rate. Ideally, the learning rate should decrease as learning proceeds. We have used a fixed learning rate in the range $\eta = [0.01, 0.001]$. A too high learning rate results in the network not being able to learn.

Momentum term. We have used $\alpha = 0.5$. The network is very insensitive to this choice. The momentum term is used for damping out oscillations and can be increased with learning epochs.

Temperature. The temperature has throughout all applications been T = 1.0. A too low temperature obstructs the network from learning, since $g'(a_i)$ becomes too narrow. One would expect that a high temperature should be used in the beginning of learning and then lowered as learning proceeds. We found no increase in the network performance if we allowed the temperature to vary in this way.

Initial weights. The weights are initiated at random in the range [-0.1,0.1]. This is important, since a too "wide" initiation is unfavourable for the network when the input signals are very different in magnitude (as they are in our applications). It is preferable to start at low weight value and work up those weights that are supposed to be large instead of starting at high values and decrease the other weights, since the updating step increases with the size of the inputs.

Updating frequency. Ideally, each step in weight space reflects the influence of the entire training set. In most cases positive and negative contributions result in moderately sized weight changes, but in instances with a large number of weights and training passes some contributions might fail to balance out resulting in inappropriately large changes for some of the weights. It is then advisable to have more than one training pass between the updates. In all our applications we have chosen this updating frequency to be 10 patterns/update.

Manhattan updating. In cases where there are many patterns/update it might be beneficial to replace the standard gradient descent with "Manhattan" updating [18]

$$\Delta\omega_{ij} = -\eta \cdot sign[\frac{\partial E}{\partial \omega_{ij}}] \tag{A14}$$

In this case the learning is bounded and it is easier to find an appropriate value for η, which should decrease with increasing learning. In all applications dealt with in this paper there was no need for this alternative procedure.

In most cases, when training, one should present equal amounts of patterns from the classes one wants the network to distinguish between. This is because the back propagation strives to minimize the total error. If there is an excess of one type of patterns, the network will concentrate on them, leading to a poorer result for the other classes. After training it is possible to shift the cut between the classes by manipulating the output threshold θ_{out}, as we have done in the case with b-quark jets.

Number of hidden layers. We have tried using two hidden layers for the quark/gluon separation. No improvement of performance was found.

In theory, any pattern classification task is realizable with at most two hidden layers [32]. This is because any reasonable function can be represented by a superposition of Gaussian-like "bumps" (similar to Fourier analysis). Combining two sigmoids, by using two hidden layers, produces such bumps. This does not mean that two hidden layers necessarily is the optimal for every classification problem. More than two hidden layers may well lead to a solution with fewer units in all, or speed up learning. The option of several hidden layers is included in JETNET 1.0 [3].

Appendix B

All analysis in this paper were performed using the FORTRAN 77 subroutine package JETNET 1.0 developed by the authors. It implements all features described in the previous section. It is available on request via BITNET. The package includes a manual and examples of programs using JETNET. For completeness we here include a brief description of its user interface.

The user interface basically consists of two common blocks and seven subroutines. All passing of variables to and from the program is handled through the common blocks `/JNDAT1/` and `/JNDAT2/` and all actions the program takes are invoked by calling the subroutines `JNINIT`, `JNTRAL`, `JNTEST`, `JNDUMP`, `JNREAD`, `JNSEFI`, and `JNSTAT`.

The program works as follows: First the network is defined by setting the switches in common block `/JNDAT1/` and calling the subroutine `JNINIT`. Then, to train the net, simply put the input pattern in the vector `OIN` and the desired output pattern in the vector `OUT` in common block `/JNDAT1/` and call the subroutine `JNTRAL`. After training, the network is used by again placing an input pattern in the vector `OIN` and calling `JNTEST`. The network then uses the weights obtained in the training to produce an output pattern which is stored in the vector `OUT`.

B.1 The common blocks

- `COMMON /JNDAT1/ MSTJN(20),PARJN(20),OIN(100),OUT(100)`

  `/JNDAT1/` is the main common block used to communicate with JETNET:

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

  `MSTJN(1)` (D=3) number of layers in the net
  `MSTJN(2)` (D=10) number of patterns per update in `JNTRAL`
  `MSTJN(3)` (D=1) overall sigmoid function used in the net
  $1 \rightarrow g(x) = \frac{1}{1+\exp(-2x)}$
  $2 \rightarrow g(x) = \tanh x$
  $3 \rightarrow g(x) = \exp(x)$ (only used internally for Potts-nodes)
  $4 \rightarrow g(x) = x$
  `MSTJN(4)` (D=0) error measure
  $0 \rightarrow$ summed square error
  $1 \rightarrow$ entropy error
  $>2 \rightarrow$ Kullback error with Potts nodes of dimension `MSTJN(4)`
  `MSTJN(5)` (D=0) "Manhattan" updating
  $0 \rightarrow$ off
  $1 \rightarrow$ on
  `MSTJN(6)` (D=6) file number for output statistics

MSTJN(7) (I) number of calls to JNTRAL
MSTJN(8) (I) initialization done
MSTJN(9) not used
MSTJN(10+I) number of nodes in layer I (I=0 $\rightarrow$ input layer)
MSTJN(10) (D=16)
MSTJN(11) (D=8)
MSTJN(12) (D=1)
MSTJN(13-20) (D=0)

Switches 2,5 and 6 can be changed at any time by the user, the others are only active before the network is initialized with subroutine JNINIT.

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

PARJN(1) (D=0.01) learning parameter η ('learning rate')
PARJN(2) (D=0.5) momentum term α
PARJN(3) (D=1.0) overall inverse network temperature $\beta = 1/T$
PARJN(4) (D=0.1) width of initial weights
PARJN(5) (D=-1.0) pruning parameter λ ($\leq 0 \rightarrow$ pruning is turned off)
PARJN(6-20) 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 both to pass the desired value of the output nodes to the program during training with JNTRAL and to pass the values of the output nodes produced by the program given an input pattern in OIN (routines JNTEST and JNTRAL).

- COMMON /JNDAT2/ TINV(10),IGFN(10)

  The common block /JNDAT2/ is intended for the "advanced" user:

  TINV(I) (D=0.0) if greater than 0, this value is used for the inverse temperature in the sigmoid function for layer I, else the overall temperature PARJN(3) is used. These parameters can be changed at any time by the user.

  IGFN(I) (D=0) if greater than 0, this switch determines the sigmoid function to be used in layer I, else the overall function determined by MSTJN(3) is used. These switches are only active before the network is initialized with subroutine JNINIT.

B.2 The subroutines

- `SUBROUTINE JNINIT`

  Initializes the network according to the switches set in `/JNDAT1/` and `/JNDAT2/` and gives random start values to the weights and thresholds according to `PARJN(4)`.

- `SUBROUTINE JNTRAL`

  Takes the pattern stored in the vector `OIN` and uses the current values of the weights in the network to produce an output pattern, This is compared with the given pattern in the vector `OUT` to produce a value of the error function from which the the change of the weights is determined. The produced output pattern is stored in the vector `OUT`.

  The actual updating of the weights is performed every `MSTJN(2)` call to `JNTRAL`.

- `SUBROUTINE JNTEST`

  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 JNDUMP(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 JNREAD(NF)`

  Reads the information from the file `ABS(NF)` produced by `JNDUMP` and initializes the network described there. All switches and parameters in common blocks `/JNDAT1/` and `/JNDAT2/` set prior to a call to `JNREAD` are lost. The comments about the file number for `JNDUMP` applies also here.

- `SUBROUTINE JNSEFI(IL,I1,I2,J1,J2,NO)`

  Enables (`NO`=1) or disables (`NO`=0) the weight between nodes `I1` to `I2` in layer `IL` and nodes `J1` to `J2` in layer `IL-1`. When a weight is disabled it is set to 0 and is prevented from being updated by calls to `JNTRAL`. When a weight is enabled it is given a random value according to `PARJN(4)`. This choice of enabling/disabling is used for selective input fields (see Sect. 5).

- `SUBROUTINE JNSTAT(IS)`

  Writes out information about the network on file number `MSTJN(6)`. `IS`=1 gives a header and number of nodes in each layer (done automatically when the network is initialized). `IS`=2 gives the switches and parameters used in common block `/JNDAT1/` and `/JNDAT2/`.

References

[1] L. Lönnblad, C. Peterson and T. Rögnvaldsson, "Finding Gluon Jets with a Neural Trigger", LU-TP 90-3, submitted to *Physical Review Letters.*

[2] 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).

[3] L. Lönnblad, C. Peterson and T. Rögnvaldsson, JETNET 1.0 program and manual [Available via BITNET request].

[4] D. E. Rumelhart (unpublished).

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

[6] G. Marchesini and B. R. Webber, *Nucl. Phys.* **B310** (1988) 461;
I. G. Knowles, *Nucl. Phys.* **B310** (1988) 571.

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

[8] OPAL Collaboration, M. Z. Akrawy et. al., "A Measurement of Global Event Shape Distributions in the Hadronic Decays of the Z^0", CERN Preprint, CERN-EP/90-48.

[9] B. Andersson and G. Gustafson, *Z. Phys.* **C3** (1980) 223;
B. Andersson, G. Gustafson, G. Ingelman, T. Sjöstrand,
Phys. Rep. **97** (1983) 31.

[10] G. Gustafson, *Phys. Lett.* **B175** (1986) 453;
G. Gustafson, U. Pettersson, *Nucl. Phys.* **B306** (1988) 746;
B. Andersson, G. Gustafson, L. Lönnblad, "Gluon Splitting in the Colour Dipole Formulation", Lund Preprint LU TP 89-9 (to be published in *Nucl. Phys.***B**).

[11] M. Bengtsson and T. Sjöstrand, *Phys. Lett.* **B185** (1987) 453; *Nucl. Phys.* **B289** (1987) 810.

[12] JADE Collaboration, W. Bartel et. al., *Z. Phys.* **C21** (1983) 37.

[13] M. Bengtsson and P. Zerwas, *Phys. Lett.* **B208** (1988) 306.

[14] Z. Fodor, *Phys. Rev.* **D40**, (1989) 3590.

[15] Y. K. Kim et. al.,*Phys. Rev. Lett.* **63** (1989) 1772.

[16] L. Jones, "Towards A Systematic Jet Classification", University of Illinios preprint ILL-(TH)-90-01.

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

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

[19] O. Barring (The DELPHI Collaboration), private communication.

[20] B. A. Kniehl, J. H. Kühn, and R. G. Stuart, in *Polarization at LEP (Vol. 1)*, CERN-88-06.

[21] B. Denby et. al., "Neural Networks for Triggering", FERMILAB-CONF-90/20.

[22] G. Arnison et. al. (UA1 collaboration), "Analysis of the Fragmentation Properties of Quark and Gluon Jets at the CERN SPS $p(\bar{p})$ Collider", *Nucl. Phys.* **B276** (1986) 253-271.

[23] B. Andersson, G. Gustafson, T. Sjöstrand, *Phys. Lett.* **B94** (1980) 211.

[24] Ya. I. Azimov, Yu. L. Dokshitzer, S. I. Troyan, V. A. Khoze, *Sovj. J. Nucl. Phys.* **43** (1986) 91.

[25] V. A. Khoze and L. Lönnblad, *Phys. Lett.* **B241** (1990) 123.

[26] J. J. Hopfield and D. W. Tank, "Neural Computation of Decisions in Optimization Problems", *Biological Cybernetics* **52** (1985) 141.

[27] C. Peterson and B. Söderberg, "A New Method for Mapping Optimization Problems onto Neural Networks", *International Journal of Neural Systems* **1** (1989) 3.

[28] B. Denby,"Neural Networks and Cellular Automata in Experimental High Energy Physics" *Comp. Phys. Comm.* **49** (1988) 429.

[29] C. Peterson, "Track Finding with Neural Networks", *Nuclear Instruments and Methods* **A279** (1989) 537.

[30] E. B. Baum and F. Wilczek, "Supervised Learning of Probability Distributions by Neural Networks" in D. Z. Andersson (Ed.) *Neural Information Processing Systems*, American Institute of Physics, N.Y. (1988).

[31] S. Kullback, *Information Theory and Statistics*, Wiley, N.Y. (1959).

[32] R. P. Lippman, "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine* April (1987) 4-22.