Mining over loosely coupled data sources using neural experts

Sergio A. Alvarez
Computer Science Dept.
Boston College
Chestnut Hill, MA 02467 USA
alvarez@cs.bc.edu

Takeshi Kawato
Computer Science Dept.
Worcester Polytechnic Institute
Worcester, MA 01609 USA
takeshi@wpi.edu

Carolina Ruiz
Computer Science Dept.
Worcester Polytechnic Institute
Worcester, MA 01609 USA
ruiz@cs.wpi.edu

ABSTRACT

Artificial neural networks (ANN) are capable of extracting patterns from multidimensional data and are natural candidates for use in multimedia data mining. However, error backpropagation training of a standard fully connected ANN can be slow. In this paper we present an ANN architecture that enables faster backpropagation training in the presence of multiple loosely coupled data sources. We show that this architecture can achieve classification performance similar to that of a standard fully connected feedforward ANN while speeding up training by a significant factor.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.5.1 [**Pattern Recognition**]: Models—*Neural nets*; I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation*; I.5.5 [**Pattern Recognition**]: Implementation—*Special architectures*

General Terms

Design, experimentation, performance

Keywords

Data mining, machine learning, neural networks, experts

1. INTRODUCTION

In the present paper we are interested in Artificial Neural Networks (ANN) as a technology for data mining over data from multiple sources. Such data abound within multimedia data mining [11]. ANN have been applied in related contexts with success. For example, in [2], an ANN classifier is applied to the problem of detecting tumors in digital mammography images; in [3], ANN are used for text mining, in order to extract "typical messages" from e-mail discussions in a computer–supported collaborative work environment. A recent collection of papers on ANN techniques for multimedia data processing is available in [5].

MDM/KDD'03, August 27, 2003, Washington, DC, USA.

We are concerned in particular with reducing the complexity of training ANN for use in classification and regression tasks involving multiple data sources. Toward this end, we employ an ANN architecture that is adapted to multisource data. In this architecture, which we call a "mixture of attribute experts", the set of input attributes (or a set of features extracted from the input attributes) is partitioned into disjoint subsets corresponding to data sources. Each of these subsets is fed into a dedicated "expert" consisting of a single layer of processing units, and the outputs of the different experts are then combined in a separate output layer. As we will show, this reduces the number of network connections and the time required for error backpropagation training while providing classification performance comparable to that of a standard fully connected feedforward ANN. The input data sources need not be selected to correspond with distinct data types (e.g. images, speech, video) that might be present in a multimedia context. The main requirement on the partition of the set of input attributes into sources is that only "loose" interaction be needed across sources in order to predict the target attribute.

Related work

Jacobs et al. [6] proposed a technique known as *hierarchical mixtures of experts* (HME). In HME as presented in [6], the experts are feedforward ANN. Each expert operates on the full set of input attributes. Separate gating networks, also feedforward ANN that receive all of the attributes as inputs, are used to implement a "soft partition" of the input space into regions corresponding to experts; the outputs of a layer of experts are weighted as dictated by the gating networks and the combined output is fed into the next layer. [6] evaluated HME for a vowel recognition task, with good results. An HME approach based on [6] has been applied to text categorization [9]. In contrast with HME, in the approach of the present paper it is the set of data attributes that is partitioned, not the input space that has the set of all attributes as coordinates. In our approach no two experts share any input attributes. This results in a significant reduction in the total number of network connections that emanate from the input layer. This reduction has a beneficial effect on training time but may lead to a loss of representational power if the target task requires strong interactions among attributes in different cells of the input partition.

An interesting application using an approach related to that presented in this paper is considered in [8]; their approach involves feeding the outputs of several ANN as inputs to

another ANN. Each expert in [8] is trained to recognize a specific value of the target (class) attribute, while in the present paper we associate experts with partitions of the set of input attributes. Moreover, [8] is concerned mainly with the classification performance of various approaches in the target domain of emotion recognition in speech and, in terms of the architecture described, focuses on classification accuracy. In contrast, we will in the present paper explicitly address also issues of efficiency and representational limitations associated with the system architecture.

Outline of the paper

Our paper begins with a description of ANN in general and the mixture of attribute experts topology in particular. We discuss representational and complexity issues for this topology and contrast it with the standard fully connected feedforward topology. We provide proof of concept for our ANN approach in the context of mining over multiple data sources by applying a mixture of attribute experts ANN to the problem of detecting advertisments in images embedded in web documents, using the Internet Advertisements dataset from the UCI Machine Learning Repository [4]. We conclude with a discussion of our results and suggestions for future work.

2. ARTIFICIAL NEURAL NETWORKS

Artificial neural networks (ANN) are models of distributed computing by a network of very simple processing units. The concept of an ANN is inspired by the view of the brain as a system of interconnected neurons. Formally, a typical feedforward ANN can be defined by a weighted directed acyclic graph (G, E, w). The nodes of G are the *processing units*. The state of each processing unit i is described by its *activation value*, usually assumed to be a real-valued function of time. The weights $w_{i,j}$ attached to the edges $E(j,i)$ (from unit j to unit i) measure the relative importance of the activations of various units j in determining the activation of unit i. We will assume a memoryless model in which the activation y_i of node i at a given time is an instantaneous function of the activation values y_j at the same time of all nodes j for which the weight $w_{i,j}$ is nonzero. Specifically, we assume the activation y_i to be the result of applying a nonlinear *activation function* f to a linear combination of the activations y_j:

$$y_i = f\left(\sum_j w_{i,j} y_j\right) \tag{1}$$

The activation function f is assumed to be a logistic, or sigmoid function:

$$f(x) = \frac{1}{1 + e^{-\sigma x}}, \tag{2}$$

where σ is a "steepness" parameter.

2.1 Network training

Multilayer neural networks may be trained by the method of error backpropagation (see e.g. [10]). This supervised learning algorithm requires that a set of training pairs $(I_k, \hat{O}_k)$ be presented to the network, where I_k is an input vector and $\hat{O}_k$ is the desired output vector corresponding to I_k. The network weights are iteratively adjusted during training so as to reduce the output error, that is, the mean square difference between the desired outputs $\hat{O}_k$ and the actual outputs O_k produced by the system on input I_k. The manner in which error backpropagation adjusts the network weights corresponds to a gradient search in weight space.

Error backpropagation pseudocode

For each training instance pair $(I_k, \hat{O}_k)$ consisting of inputs I_k and target outputs $\hat{O}_k$:

1. Pseudo–randomly initialize the network weights.
2. Repeat until the termination condition is satisfied {
   (a) Propagate the inputs forward to the outputs by repeatedly applying Eq. 1.
   (b) Compute δ values for all output nodes and hidden nodes, as follows. For each output node k and each hidden node h:
   $$\begin{aligned} e_k &= \hat{O}_k - O_k \\ \delta_k &= O_k(1 - O_k)e_k \\ \delta_h &= y_h(1 - y_h) \sum_{\text{outputs } k} w_{k \leftarrow h} \delta_k \end{aligned} \tag{3}$$
   Here, y_h is the activation level of node h, and $w_{k \leftarrow h}$ is the current value of the connection weight from node h to node k. [1]
   (c) Update the weights for all pairs of nodes using the δ values as follows:
   $$\text{new } w_{j \leftarrow i} = \text{old } w_{j \leftarrow i} + \eta \delta_j y_i, \tag{4}$$
   where η is a so–called learning rate.
}

Various termination criteria may be used. For example, training can be terminated when the error over some reserved validation set of instances has not decreased for a certain number of iterations. If the initial weights happen to be chosen sufficiently close to a particular local minimum of the output error, and if the learning rate η is sufficiently small, then error backpropagation is guaranteed to produce a sequence of weights that converges to this local minimum. Non–global optimality may be addressed by comparing the results of error backpropagation for different pseudorandom choices of the initial weights. For example, a variant of error backpropagation that incorporates an automatic random restarting mechanism may be considered [1].

2.2 Network architectures

We consider feedforward ANN with two layers of processing units. The inputs feed directly into the units of the first, or "hidden" layer. The hidden units feed into the units of the second, or "output" layer. The standard network topology in this context is the "fully connected" (FC) network, in which all pairs of (input, hidden) units are connected, as are all (hidden, output) pairs. See Fig. 1. We consider also a second network topology which we call a *mixture of attribute experts* (MAE). In this topology, the set of inputs

[1] The name of the error back–propagation algorithm is derived from the recursive form of the δ equations: the errors e_k at the output layer are "propagated back" through previous layers by Eq. 3.

Figure 1: Fully-connected network.

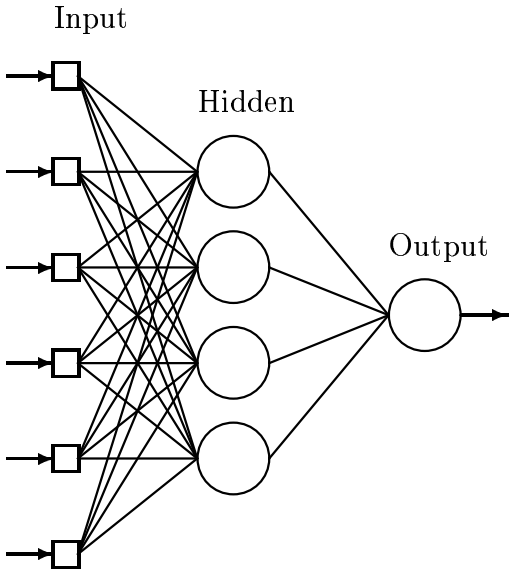
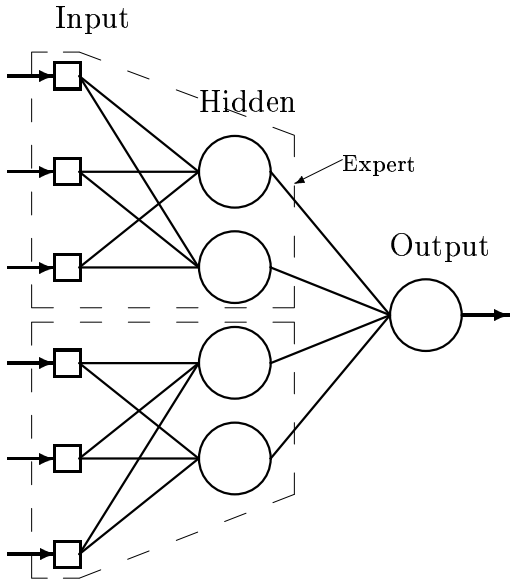


Figure 2: Mixture of experts network.



and the units of the hidden layer are partitioned into k disjoint groups. The groups in the hidden layer are called *experts*. Each of the experts will process data from a different source. Each expert is fully connected to its corresponding set of inputs, but is disconnected from the remaining inputs. The hidden layer is fully connected to the output layer. Fig. 2 shows a mixture of attribute experts network with two experts. As discussed in the Introduction, our mixture of attribute experts architecture differs from the hierarchical mixture of experts architecture described in [6]; in the latter, all inputs are fed into each expert, and separate gating networks are used to effect the mixture of the experts' outputs.

3. EFFICIENCY AND EXPRESSIVENESS

There is a difference in the intrinsic complexities of the learning tasks for FC and MAE which may be understood in terms of the spaces in which learning takes place. For either architecture, the target of learning is a function from the n–dimensional Euclidean space $\mathbb{R}^n$ to the y–dimensional Euclidean space $\mathbb{R}^y$, where n is the number of inputs and y is the number of outputs. However, each of the two architectures explores a different portion of this function space as we point out below.

Consider a MAE architecture with k experts, in which the i–th expert has n_i inputs and h_i hidden nodes, and a FC architecture with the same total number of inputs $n = \sum_{i=1}^{k} n_i$ and the same total number of hidden nodes $h = \sum_{i=1}^{k} h_i$. FC targets a general function from $\mathbb{R}^n$ to $\mathbb{R}^y$ expressed as a composition

$$f(g(x_1, \cdots, x_n))$$

of a function $g : \mathbb{R}^n \rightarrow \mathbb{R}^h$ and a function $f : \mathbb{R}^h \rightarrow \mathbb{R}^y$. On the other hand, because of the grouping of inputs and hidden nodes into k mutually noninteracting experts, MAE effectively adopts a factorization approach that targets a function of the more restricted form

$$f\left(g_1(x_1^{(1)} \cdots x_{n_1}^{(1)}), \cdots, g_k(x_1^{(h)} \cdots x_{n_h}^{(h)})\right) \qquad (5)$$

Here, each g_i is a function $g_i : \mathbb{R}^{n_i} \rightarrow \mathbb{R}^{h_i}$, where h_i is the number of hidden nodes in expert i. Notice that the space of the "outer" functions f is the same in both approaches, since the total number of hidden nodes h and the total number of outputs y is the same in both cases. However, while FC searches for a single g in the nh–dimensional space of all functions from $\mathbb{R}^n$ to $\mathbb{R}^h$, MAE instead performs k searches for g_1 through g_k, with the i–th search taking place in the $n_i h_i$–dimensional space of functions from $\mathbb{R}^{n_i}$ to $\mathbb{R}^{h_i}$. Since the former (FC) space has higher dimensionality, the FC architecture, and indeed any method that will potentially consider the full function space, has a more complex task to consider than does MAE or any other factorization method. One consequence of this is that MAE trains more quickly than FC. Another consequence, however, is a reduction in the expressive power of MAE. We discuss these issues below.

3.1 Time complexity of training

The total training time in the error back–propagation algorithm (see section 2.1) is proportional to the number of weight updates; the number of weight updates in turn equals the product of the number of network connections and the number of training iterations required for convergence.

It is straightforward to calculate the number of network connections in each of the two architectures. The number of inputs is generally much larger than the number of hidden and output nodes of the network, so connections between the input and hidden layers dominate the overall count. Since in the FC architecture there is one weight for each pair consisting of an input attribute and a hidden node, the total number of weights between the input and hidden layers in the FC architecture is the product nh of the number of input attributes n and the number of hidden nodes h. For the MAE architecture with k experts of roughly equal sizes,

each input is connected to about h/k hidden nodes, so the number of weights from the input layer to the hidden layer is roughly nh/k. Therefore, *the MAE architecture can reduce the number of network connections by a factor roughly equal to the number of experts.*

It is more difficult to provide precise *a priori* estimates of the number of iterations. This is because of the complex nature of the error surface, that is, the surface described by the function that maps a vector of connection weight values to the mean square error attained by the corresponding network with respect to a given set of training data. Indeed, the very space of weights over which the error surface is defined is different for FC and MAE. Some of our experiments have shown a distribution of training times for the mixture of attribute experts architecture that has a heavier tail than that of the fully connected architecture. This has led us to propose a random restarting technique in [1] that brings the average number of training iterations closer for the two architectures, thereby allowing the reduction in the number of network connections associated with the mixture of attribute experts architecture to be reflected in a similar reduction in the overall time complexity of training.

3.2 Expressive power

Intuitively, the fact that there are groups of inputs that feed into different experts makes it difficult for a MAE network to model relationships across such input groups. We will show that this is indeed true, by showing that there are functions of $m + n$ variables that cannot be expressed in MAE form with two experts that have, respectively, m and n inputs.

We consider the specific case $m = 2$, $n = 2$, and assume that the partition of the input attributes into experts is predetermined, with (x_1, x_2) and (x_1', x_2') as the two halves. We focus on functions of the form $\phi(x_1x_1', x_2x_2')$, where ϕ is a non-constant function and the arguments are products of one variable from each of two "halves" of the input vector. Specifically, take the example in which ϕ is the sum operator, so that the target function is $(x_1, x_2, x_1', x_2') \mapsto x_1x_1' + x_2x_2'$.

In order for the target function to equal the MAE composition $f(g(x_1, x_2), g'(x_1', x_2'))$, the following must hold:

$$x_1 = f(g(x_1, x_2), g'(1, 0)) \Rightarrow g(x_1, x_2) \text{ independent of } x_2$$
$$x_2 = f(g(x_1, x_2), g'(0, 1)) \Rightarrow g(x_1, x_2) \text{ independent of } x_1$$
$$x_1' = f(g(1, 0), g'(x_1', x_2')) \Rightarrow g'(x_1', x_2') \text{ independent of } x_2'$$
$$x_2' = f(g(0, 1), g'(x_1', x_2')) \Rightarrow g'(x_1', x_2') \text{ independent of } x_1'$$

which implies that $x_1x_1' + x_2x_2'$ must be constant, but of course it's not. This contradiction shows that the target function in question is not expressible by a MAE network.

Loosely interacting attributes

Because of the above phenomenon, the success of a MAE approach in a particular context will depend on the possibility of partitioning the set of input attributes in a way that requires only "loose" interactions among attributes in different groups of the partition. One domain in which such a data partition may occur naturally is that of information filtering or recommendation based on a combination of social (collaborative) and content information; we have obtained promising results in this domain using the approach of the present paper [1]. The meaning of "loose" is somewhat loose here, but may be made more precise by noting that the space of attainable target functions should be as described by Eq. 5. Thus, if the terms in Eq. 5 are standard sigmoids operating on linear combinations as in Eq. 1 and Eq. 2, then we see that "loose" interactions will include those that depend on linear combinations of input attributes in different cells of the partition.

4. EXPERIMENTAL EVALUATION

4.1 Data

We used the Internet Advertisements dataset [7], available through the UCI Machine Learning Repository [4]. The 3279 instances of this dataset represent images embedded in web pages; roughly 14% of these images contain advertisements and the rest do not. There are missing values in approximately 28% of the instances. The proposed task is to determine which instances contain advertisements based on 1557 other attributes related to image dimensions, phrases in the URL of the document or the image, and text occurring in or near the image's anchor tag in the document.

4.1.1 Attributes

A description of the attributes for the Internet Advertisements dataset appears below as presented in the dataset's summary page at the UCI Machine Learning Repository [4]. The first three attributes encode the image's geometry; *aratio* refers to the aspect ratio (ratio of width to height). The binary *local* feature indicates whether the image URL points to a server in the same Internet domain as the document URL. The remaining features are based on phrases in various parts of the document; the terms origurl, ancurl, alt refer respectively to the document URL, anchor (image) URL, and alt text in the anchor tag for the image. See [7].

1. height: continuous. — possibly missing
2. width: continuous. — possibly missing
3. aratio: continuous. — possibly missing
4. local: 0,1.
5. 457 features from url terms,
   each of the form "url*term1+term2...";
   for example: url*images+buttons: 0,1.
6. 495 features from origurl terms, in same form;
   for example: origurl*labyrinth: 0,1.
7. 472 features from ancurl terms, in same form;
   for example: ancurl*search+direct: 0,1.
8. 111 features from alt terms, in same form;
   for example: alt*your: 0,1.
9. 19 features from caption terms, in same form;
   for example: caption*and: 0,1.
10. class attribute: ad/nonad

4.1.2 Input partition

Applying a mixture of attribute experts network to the above data requires that the set of non–class attributes be split into disjoint subsets to be used as inputs for the respective experts. We will adopt the simplest possible approach to this task here by using a natural grouping present in the list above. The first four attributes in the enumeration will

constitute the first group. Each of the other items in the enumeration except for the target attribute will be a group also. Thus, we will have six experts in all, corresponding to the following groups of input attributes:

1. Image geometry
2. Phrases in image's URL
3. Phrases in base URL
4. Phrases in anchor URL
5. Phrases in alt text
6. Phrases in caption

4.1.3 Feature extraction

Although the Internet Advertisements dataset does not have a particularly high occurrence of missing values, we chose to reduce the dimensionality of the input attribute vectors using the singular value decomposition (SVD) in order to allow training to be completed more quickly. We apply the SVD to each expert's group of input attributes separately so that both the fully connected and the mixture of attribute experts architectures operate on the same input data. Only groups 2–5 in the list above were processed using SVD. We are able to halve the number of attributes by keeping only the largest singular values after applying the SVD, discarding those that contain a total of 1% or less of the "energy". That is, if the singular values are $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_N$, then we keep $\sigma_1 \cdots \sigma_n$, where n is the smallest integer such that

$$\frac{\sum_{j=n+1}^{N} \sigma_j^2}{\sum_{j=1}^{N} \sigma_j^2} < 0.01$$

The effect of SVD on the size of the input attribute groups is summarized in Table 1.

Table 1: Attribute count before and after SVD.

	Geom	URL	OrURL	AncURL	Alt	Capt
Before	4	457	495	472	111	19
After	4	265	196	279	107	19

4.2 Networks

4.2.1 Mixture of experts module

We used the implementation of error backpropagation for feedforward neural networks provided in the Weka 3 system [12]. We implemented the mixture of attribute experts architecture as a new module, ExpertNetwork, which we added to the Weka neural network module. The new module was included in the weka.classifiers.neural directory together with the standard Weka modules and may be accessed through the Weka GUI. The new module allows one to set the structure of the network by assigning only specific inputs to the hidden neurons. As an example, one can assign the first 2000 inputs to one neuron and the following 3000 inputs to two other neurons by typing [2000,1][3000,2] in the expertString field. [2] We also added the capability of tracking the precision and the accuracy of a network output every certain number of iterations as specified by the user. One can specify the file to write this information to in the precAccFile field, and the number of iterations between writings in the precAccPerIter field. [3] Finally, we added the capability of writing the final network outputs for each test instance to a file. The module uses the first input as an ID, so that one can track whether there are any trends in the network outputs. One trick for giving a reasonable ID to the module is to embed the ID into the data as the first attribute, and then assign 0 hidden neurons ([1,0]...) to the first input so that it does not affect the network training.

4.2.2 Network configurations

We consider two basic architectures: a standard fully connected feedforward network as depicted in Fig. 3, and a mixture of attribute experts network as shown in Fig. 4.

Figure 3: Fully–connected network.

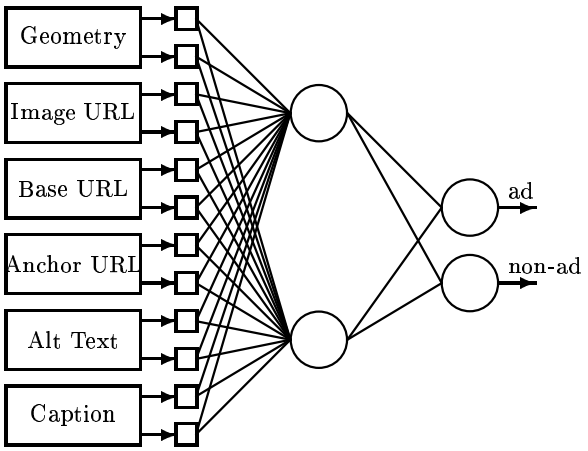
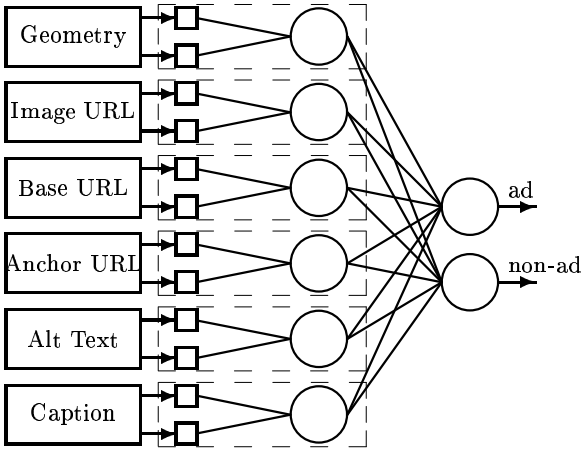


Figure 4: Mixture of experts network.



In both cases, two output nodes are used, each corresponding to one of the two possible values of the class attribute (ad, nonad). We consider fully connected networks with 2, 3, and 6 hidden nodes, and mixture of attribute experts net-

[2] The number of hidden neurons specified in the expertString field must be lower than or equal to the number of hidden neurons specified in hiddenLayers field.

[3] One must note that specifying this field will affect the training time considerably, and that one should leave this field as default to obtain a reliable training time.

works with either 1 or 2 nodes per expert. [4] The number of experts is fixed at 6. Network inputs are obtained directly as the output of SVD preprocessing of the data attributes as explained in section 4.1.3 above.

4.3 Performance metrics

Several different metrics were used to evaluate the classification performance and time efficiency of the various networks, including classification accuracy (fraction of labeled test instances for which the system's predicted classification matches the class label), the F–measure, defined in terms of the information retrieval metrics of precision (specificity, fraction of true positives among true and false positives) and recall (coverage, fraction of true positives among true positives and false negatives) by the formula:

$$F = \frac{2 * precision * recall}{precision + recall},$$

and the total training time needed for convergence of the error backpropagation algorithm.

4.4 Evaluation protocol

We employed n-fold cross-validation throughout. The number of folds, n, was either 4 or 10 depending on the experiment. For each experiment, we randomly partitioned the data into n parts and proceeded to carry out n training and testing trials. For each of these trials, one of the n parts of the data set was reserved for testing and the union of the other nine parts was randomly split into a 70% portion used for training and a 30% portion used as a validation set to determine when to stop training. The networks were trained using the error backpropagation algorithm with a learning rate of 0.3 and a momentum coefficient of 0.2. Training continued until 20 consecutive iterations resulted in no increase in the precision as measured over the validation set, or until a maximum of 2000 training iterations had been carried out. After training was completed, performance measures were evaluated based on the networks' performance on the reserved testing part of the data for that trial. The values of the performance measures were then averaged over the n trials; these averages are the values that we report here.

5. RESULTS

5.1 Classification performance

Table 2: Classification accuracy.

Architecture	Mean	Median
Fully–connected (2 hidden nodes)	0.94	0.96
Fully–connected (4 hidden nodes)	0.96	0.96
Fully–connected (6 hidden nodes)	0.94	0.96
Experts (1 node per expert)	0.96	0.96
Experts (2 nodes per expert)	0.96	0.96

Table 2 shows the classification accuracy obtained for the different system architectures. The evaluation protocol used in this experiment was four-fold cross-validation. As the figure shows, the accuracy values appear very similar for all architectures tested.

[4]The "two nodes per expert" MAE architecture has 10 hidden nodes: one for each of the image geometry and caption terms experts, and two for each of the other four experts.

Nonetheless, the similarity in the accuracy values hides a significant difference in the classification performance of the different networks. This difference becomes apparent when we examine the observed values of the F–measure in Table 3 (computed using four-fold cross-validation).

Table 3: F–measure.

Architecture	Mean	Median
Fully–connected (2 hidden nodes)	0.65	0.85
Fully–connected (4 hidden nodes)	0.87	0.86
Fully–connected (6 hidden nodes)	0.65	0.86
Experts (1 node per expert)	0.85	0.87
Experts (2 nodes per expert)	0.85	0.84

Although Table 3 shows little difference across architectures in the median values, a significant drop is observed in the mean value of the F–measure for the fully connected architecture with 2 or 6 hidden nodes. Here, this was associated with convergence of the error backpropagation training algorithm for these networks to non–global minima of the output error landscape in some of the runs; the resulting classifiers predict the same class (for example, nonad) for all test instances. A random restarting version of error backpropagation [1] might reduce this phenomenon somewhat, but we have not yet carried out such an experiment.

5.2 Time efficiency

The total time required to train each network over 75% of the set of 3279 instances is reported in Table 4. This experiment was performed in the Weka 3 system with our neural experts module, using JVM version 1.4.1 on a Pentium–based system with 2.0 GHz clock rate and 256 MB RAM.

Table 4: Training times (seconds).

Architecture	Mean	Median
Fully–connected (2 hidden nodes)	1532.5	925.3
Fully–connected (4 hidden nodes)	2578.8	2167.6
Fully–connected (6 hidden nodes)	4243.7	4631.3
Experts (1 node per expert)	1977.2	2324.7
Experts (2 nodes per expert)	890.4	830.5

Table 4 (four–fold cross–validation) shows that the mixture of attribute experts architecture with two nodes per expert trained faster than the fully connected architectures. Qualitatively, this phenomenon appears to be stable. However, we note that the precise training times depend significantly on the stopping criteria used for training.

5.3 Discussion

The results in Table 4 show that the fully connected architecture with 4 and 6 hidden nodes took much longer to train than either of the mixture of attribute experts architectures, each of which has at least 6 hidden nodes. Thus, the mixture of attribute experts architecture provides a more efficient way of making use of the available computational resources. As Tables 2 and 3 above show, this time advantage is achieved without sacrificing classification performance. This supports the implicit assumption that the chosen assignment of input attributes to experts yields a "loosely interacting" partition as described in section 3.2.
The only version of the fully connected architecture that trained in a time comparable to the slower of the two mixture of attribute experts networks is that with 2 hidden nodes. However, Table 3 shows that this particular fully

connected network displayed inferior classification performance in the sense of the F–measure on some of the runs. The fastest training was achieved by the mixture of attribute experts architecture with 2 nodes per expert. Remarkably, this architecture trained even faster than the mixture of attribute experts architecture with only one node per expert. This may point to differences in the output error landscape that make it more difficult to find the minimum points in weight space when the experts contain only one hidden node. We carried out an additional cross-validation experiment, increasing the number of folds from 4 to 10 in order to better assess the statistical stability of the above results. We increased the number of hidden nodes from 10 to 12 in the larger of the MAE architectures and compared the results with those for a fully connected network with 3 hidden nodes. Again, error backpropagation for the fully connected architecture fails to converge on some runs. This is reflected in a significantly lower mean F–value for the fully connected architecture with 3 hidden nodes in Table 5, just as previously observed for the fully connected architecture with 2 or 6 nodes in Table 3. These results provide further support for the finding that the mixture of attribute experts architecture has better classification performance in this context.

Table 5: F–measure (10-fold cross-validation).

Architecture	Mean	Median
Fully–connected (3 hidden nodes)	0.79	0.87
Experts (1 node per expert)	0.85	0.86
Experts (2 nodes per expert)	0.87	0.87

Table 6: Training times (sec.), 10-fold cross-valid.

Architecture	Mean	Median
Fully–connected (3 hidden nodes)	2760.8	2086.9
Experts (1 node per expert)	2676.8	2779.0
Experts (2 nodes per expert)	1568.8	1131.8

Table 6 shows an increase in the mixture of attribute experts training time relative to the results in Table 4, which is consistent with the increased number of hidden nodes in the larger of the MAE networks. Nonetheless, the computational superiority of the mixture of attribute experts architecture with two nodes per expert is still evident in Table 6.

6. CONCLUSIONS

We have shown that in situations involving multiple data sources, a mixture of attribute experts neural network architecture provides a natural way to reduce the number of network connections and, accordingly, the time needed for network training. We have applied this idea to a particular problem involving detecting advertisements in images embedded in web pages. In this context, we were able to reduce the training time while maintaining the same level of classification performance as a fully connected network operating on the same input data. Convergence of the error backpropagation algorithm to non–global local minima of the error function was observed in the case of a fully connected network architecture in some of our experiments. Experimental evaluation of the effect of a random restarting variant of error backpropagation [1] in this context would be desirable, as would an assessment of how training times depend on the stopping criteria for training. Based on our results to date, we believe that our approach holds promise as a technique for multimedia data mining. As we pointed out, the success of this technique depends on the availability of a set of input attributes that can be partitioned into subsets with relatively little mutual interaction in determining the value of the target attribute. It will be beneficial to identify domains in which such a situation may occur naturally, as well as feature selection techniques that contribute to this goal in other domains.

7. ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their helpful comments.

8. REFERENCES

[1] S. Alvarez, C. Ruiz, T. Kawato, and W. Kogel. Faster neural networks for combined collaborative and content-based recommendation. *Preprint*, 2003.

[2] M.-L. Antonie, O. Zaïane, and A. Coman. Application of data mining techniques for medical image classification. In *Proc. of Second Intl. Workshop on Multimedia Data Mining (MDM/KDD'2001)*, pages 94–101, San Francisco, CA, August 2001.

[3] M. Berthold, F. Sudweeks, S. Newton, and R. Coyne. Clustering on the net: Applying an autoassociative neural network to computer mediated discussions. *J. Computer Mediated Communication*, 2(4), 1997.

[4] C. Blake and C. Merz. UCI repository of machine learning databases. [http://www.ics.uci.edu/∼mlearn/MLRepository.html], 1999. Dept. of Information and Computer Science, University of California. Irvine, CA.

[5] L. Guan, T. Adali, S. Katagiri, J. Larsen, and J. Principe. Guest editorial, special issue on intelligent multimedia processing. *IEEE Transactions on Neural Networks*, 13(4):789–792, July 2002.

[6] R. Jacobs, M. Jordan, S. Nowlan, and G. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

[7] N. Kushmerick. Learning to remove internet advertisements. In *Proc. of the Third Annual Conference on Autonomous Agents (AGENTS99)*, pages 175–181, Seattle, WA, USA, 1999. ACM.

[8] V. Petrushin. Emotion recognition agents in the real world. In *Socially Intelligent Agents: the Human in the Loop, Papers from the 2000 AAAI Fall Symposium (K. Dautenhahn, Chair)*. Technical Report FS-00-04, AAAI Press, 2000.

[9] M. Ruiz and P. Srinivasan. Hierarchical text categorization using neural networks. *Information Retrieval*, 5(1):87–118, 2002.

[10] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In D. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing, Vol. 1*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[11] S. Simoff, C. Djeraba, and O. Zaïane. MDM/KDD2002: Multimedia data mining between promises and problems. *SIGKDD Explorations*, 4(2):118–121, 2003.

[12] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers, 1999.