

MailMind

A Connectionist E-Mail Sorting Agent

Peter Gorniak

December 2, 1999

Abstract

MailMind is an application of backpropagation networks to the real world problem of sorting e-mail messages based upon the sender's address. This paper presents the strategies employed in the project to adapt to the greatly varying properties of datasets in this domain, such as scale and time dependent change of the problem as well as extremes in noisiness and lack of variation. These strategies include a number of encoding algorithms for e-mail addresses, which can speed up the learning task and support generalization, and an object oriented implementation approach, which supports dynamical changes to the network structure. One possible concretization of such changes is given through an algorithm that utilizes the object oriented structure to extract subnets and create a network hierarchy. Included empirical results reveal the success of the employed strategies in solving the posed problem.

1 Introduction

Due to the fast rise of the popularity of e-mail as a means of personal communication in the last few years many e-mail correspondents are faced with a large and growing bulk of messages to keep organized. Most e-mail clients now offer tools to support the manual organization of messages into mailboxes and some provide functionality for automatic sorting based on specified match criteria. However, the interfaces to these sorting engines require extensive setup and testing to function correctly. Most people do not spend the time to learn to set up a filtering system for their messages, or if they do it becomes out-of-date and requires maintenance quickly as their correspondents change.

This problem is a natural candidate for an approach using a learning algorithm. In its most common occurrence, sufficient hand-sorted examples will already be present to provide data for the algorithm to learn the appropriate classifications from. There are relatively small and naturally discrete number of possible classifications. A significant part of the data based upon which the classification is performed consists of e-mail addresses, which conform to a known and rigid structure that can be exploited before being presented to the learning engine. In addition to already present examples messages can be used as learning examples for the algorithm as they arrive, providing an automated adaptation of the sorting scheme over time.

A learning algorithm that can solve this problem must be able to gen-

eralize based on statistical features of the data, such as significant parts of e-mail addresses. Natural access to already classified data allows for supervised learning. One approach that fulfills all these requirements is that of learning using supervised connectionist networks (Rosenblatt, 1958). In particular, the backpropagation algorithm (Rumelhart *et al.*, 1986) has all the properties sought for and in addition scales well to different numbers of possible classifications.

One of the main problems with connectionist network learning algorithms is the derivation of an appropriate network structure in terms of connectivity and number of nodes. In general, a greater number of nodes slows down learning using backpropagation, but allows for better minimization of errors during the learning phase. As a tradeoff it can lead to overfitting to the learning data and thus poor generalization (Gallant, 1993). There are some statistical approaches to this problem (Murata *et al.*, 1992), but in this paper a suggestion is made along the lines of structure-modifying algorithms. Possibilities for these include generation and elimination of nodes during the learning process (Honavar & Uhr, 1991). More recently, the human estimates of when generation or elimination should occur are being replaced by genetic algorithms performing the same task (Honavar & Balakrishnan, 1995). In section 2.4 an algorithm is suggested that looks to biological neural organization for inspiration. It uses human estimates for the criteria determining generation and elimination, but supports the generation of a new type of nodes that themselves contain neural nets extracted from significant parts of

the existing net, thus creating a network hierarchy.

MailMind, the application presented here employs a connectionist network to automate the sorting of e-mail messages by learning from already sorted examples. It uses backpropagation as a learning algorithm with many considerations for the implementation drawn from (Gallant, 1993). The domain presents many interesting problems. E-mail addresses must be encoded in a manner that scales well to large datasets, but still provides a pattern unique enough for the net to learn to recognize. Real world datasets of e-mail messages sorted into mailboxes range in their nature from excessive noise due to personal considerations in sorting to lack of variation due to mailboxes containing only one address. Thus, both learning and generalization can prove hard for given data. Connected to this issue is that of an optimal structure and size of the network. This is made more problematic by the fact that the number of outputs depends on the number of mailboxes, which varies from dataset to dataset and may even vary over time in a real world setting. The described solution is thus designed with an easily examinable and mutable network structure in mind.

2 MailMind Features

2.1 Problem Domain and Approach

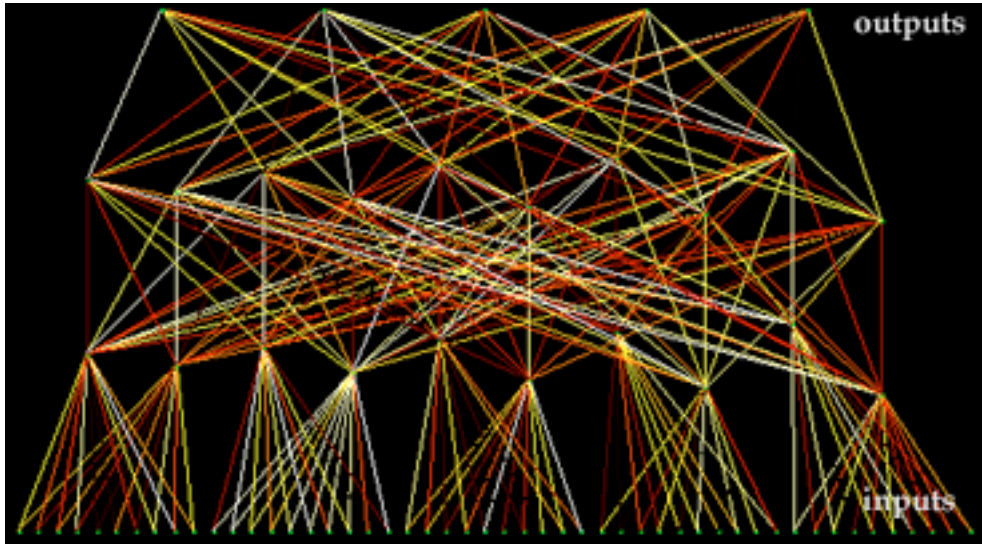


Figure 1: An Example MailMind Network Configuration

Currently, MailMind only uses one address in the 'From:' header to classify an e-mail message. Each address of the form 'userID@Host.Domain.Country' is split up into its hierarchical lowercase parts, 'userid', 'host', 'domain' and 'country'. Each of these parts is then encoded using one of the encoding schemas presented in 2.2 and the result is presented to a subset of the input nodes. These features detectors are visible as groups of inputs connected to the same intermediate nodes at the bottom of figure 1. MailMind is trained to produce an output of 1 for the mailbox number the message should be assigned and zeros for all other outputs. During sorting MailMind will put

a message into the mailbox corresponding to the output that achieves the highest value.

2.2 Encoding Email Addresses

There are two main demands of an encoding scheme for e-mail addresses to become input to MailMind's neural net:

1. Scalability
2. Pattern Distinctiveness

One possible encoding of e-mail addresses is to simply scale the ASCII characters down to real numbers between 0 and 1. Only a few characters (about 40) can actually occur in a lowercase address, so the encoding of one character differs by a minimum of $1/40$ from all others. The advantages of this encoding are that it requires no extra memory or maintenance of data structures and that it provides a unique encoding for any number of addresses encountered. The disadvantages are that the patterns created lack distinctiveness and that their similarities and differences stand in no relation to actual features of the addresses. The lack of distinctiveness arises due to the $1/40$ difference between characters. One can always find strings that involve small differences between characters. For example, in a simple version of this encoding the string 'abcd' involves only a range of $3/40$ across the whole string, making it hard to recognize as a distinct pattern. Furthermore, the encoding of the string 'abcd' will be very similar to that of 'bcde', creating

a topology amongst the input patterns that does not reflect any meaningful feature of the actual input data. The overall result is that MailMind will learn to classify messages using this encoding scheme, and will also generalize to the extent that identical patterns in any part of the address are detected as consistent features. For example, MailMind does learn to recognize 'ubc' in an address as an indicator of a message likely originating here at UBC. However, generalization exhibits sporadic failures with this method, because addresses that happen to have a string in one of their parts, that after encoding 'looks very much like' a pattern the feature detector responsible for that part has learned to recognize, are easily misclassified. Furthermore, the small differences between the activation strengths of individual input nodes slow down overall convergence because large differences in weights must be achieved to arrive at small errors in the output.

Another encoding implemented for MailMind maintains lists of parts of addresses encountered previously, together with a unique ID for each. To encode a part of an address it is replaced either by the its ID as found in the corresponding list or by a new ID, which is then associated with this new string in the list. MailMind's net is presented with the binary encodings of these ID's by matching each bit with an input node. For example, say the address 'gorniau@unixg.ubc.ca' is encountered. As for all encodings, it is split up into its four parts, and these four parts then looked up in four lists. If 'ca' is found with matching ID 3 the net will be presented with '11' on the feature detector responsible for that part of the address. The remaining inputs of

that detector are set to 0. If 'unixg' has never been encountered before, it is assigned a new ID, say 7, and paired with that ID in the corresponding list so that it will be encoded identically the next time it is encountered. The feature detector for that address part thus receives '111' and enough zeros to fill its width. This encoding achieves better distinctiveness within an address part than the previously discussed one because it does not rely on the recognition of fractional differences between inputs. However, it does not achieve full distinctiveness between patterns as any two patterns created from ID's with a difference of a power of two between them have all the same input activations except for one and thus appear similar. For example, a part of an address assigned the ID 17 and presented to the net as '10001' bears a similarity to one assigned ID 25 and presented as '11001', without that similarity corresponding to any features of the actual input data. This does not hinder learning, and is comparatively less severe than in the previous encoding presented, but it does lead to faulty generalization. Furthermore, the number of unique encodings of address parts is limited to 2^n per feature detector with this schema, where n is the width of the feature detectors. For this reason, the associative lists are actually maintained as priority queues sorted by a function of number of uses and the time last used for each entry. When they fill up a new address part encountered can replace an old one if the old one has not been used for some time and does not see an overall high usage. MailMind learns faster using this encoding scheme as opposed to the previous one, due to the more extreme nature and higher distinctiveness

of the inputs, but errors in generalization can still occur in light of to the mentioned similarities amongst unrelated address parts.

A final encoding used by MailMind is a slight modification of the above where the ID of an address part is not encoded in binary, but rather with zeros except for a one in the position corresponding to the ID. This encoding creates very distinct patterns that do not exhibit any unwanted topological features, but allows for only as many unique encodings as the feature detector has input nodes. In practice MailMind uses this encoding for the last part of the address, because the number of country codes encountered is usually small and distinctiveness of their encoding often of prime importance for both learning speed and generalization.

2.3 Network Implementation

MailMind's network graph is represented using an adjacency matrix representation rather than list structures (as described in (Corman *et al.*, 1990), for example). On the one hand this means that the memory requirements increase as the square of the number of nodes in the network, but on the other hand it is only a matter of extracting a column or row from this matrix to obtain a list of all nodes connected to a given node or all nodes a given node is connected to, respectively, including the weights of these connections. This implementation is geared towards easy examination and modification of network topology as discussed in 2.4.

MailMind follows the object oriented paradigm, and at the lowest level

its network implementation consists of a class that solely maintains the net in the shape of its adjacency matrix. Each row and column of weights in the matrix corresponds to a site, which is maintained separately. A site is a class that provides an abstract interface to the functionality expected from every node in the network, for example the evaluation of a site specific function or its derivative. In its most straightforward interpretation it simply contains a unit, for example a 'DiffUnit', which evaluates the standard sigmoid function and its derivative for use in backpropagation. The point of providing the layer of abstraction given by a site is that through it one unit can dynamically be replaced by another or even by a whole subnet. The contracting algorithm presented in 2.4 utilizes this.

A subclass of the network foundation class implements a backpropagation network. The central elements added are the concepts of input and output nodes, simply maintained as lists of node ID's, and the actual backpropagation algorithm. This class requires no concept of layers, but rather relies on non-zero entries in the adjacency matrix to establish a list of nodes being fired upon for each iteration. The following is the implementation of the resulting forward step. The backward error adjustment step is similar in nature. MailMind uses the C++ STL libraries as well as TNT by Roldan Pozo for its matrix manipulations (Pozo, 1996).

```
1 void BackPropNet::forward(  
2 const deque<int>& nodes,  
   const Vector<NumType>& incoming)  
4 {
```

```

//vector of weights going out
6 Vector<NumType> nextWeights(weightMatrix.dim(1),0.0);
//list of nodes being fired on
8 deque<int> nextNodes;

10 //step through nodes coming in
deque<int>::iterator i = nodes.begin();
12 int node = *i;

14 while(i!=nodes.end())
{
16 //extract out-connection vector
NumType *weights =
18 weightMatrix[node - 1];

20 for(int j=0;j<weightMatrix.dim(1);j++)
{
22 if(weights[j] != 0)
{
24 //mark this node as fired on
//if necessary
26 if((find(nextNodes.begin(),
nextNodes.end(),j+1) ==
28 nextNodes.end()))
{
30 nextNodes.push_back(j+1);
}
32 //add to the weighted sum
nextWeights[j] += weights[j] *
34 incoming[node - 1];
}
36 }
//update counts
38 i++;
node = *i;
40 }

42 //check that there still are nodes being

```

```

//fired upon
44 if(nextNodes.size() == 0)
    return;
46
//now apply the function to the accumulated
48 //weighted sums
i = nextNodes.begin();
50 while(i!=nextNodes.end())
{
52     node = *i;
    nextWeights(node) =
54     siteMatrix[node - 1]->eval(nextWeights(node));
    i++;
56 }

58 //recursive call for next 'layer'
forward(nextNodes, nextWeights);
60 }

```

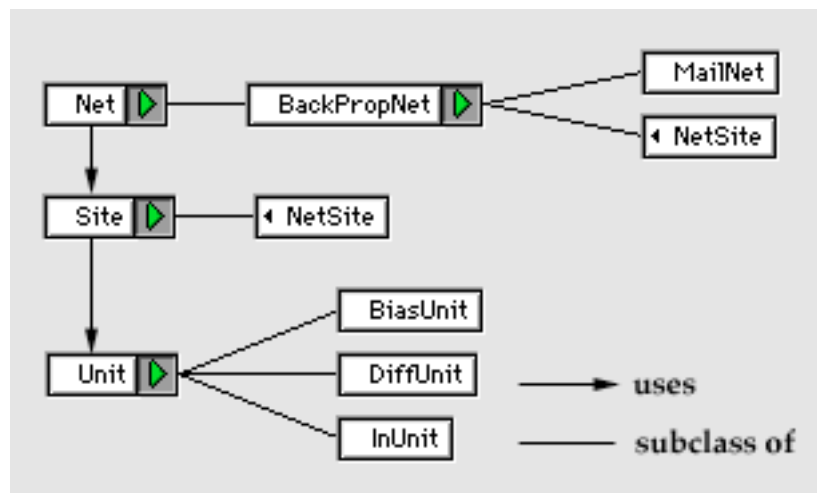


Figure 2: The MailMind Class Hierarchy

Finally, in a second subclassing, this backpropagation algorithm is augmented by methods to build the actual network structure, for example the

one shown in figure 1, and to interface with classes for reading mailbox files and for implementing the encodings from section 2.2 to make MailMind complete. Figure 2 shows the resulting object oriented class hierarchy, including the subclass 'NetSite' of a site discussed in section 2.4. As visible, MailMind also assigns biases to sites, implemented through a special 'BiasUnit' whose site is connected to every other site in the net with a constant activation of 1.

2.4 Structure Modification

2.4.1 Biological Motivation

Connectionists network researchers look to Neurobiology for inspiration (Feldman & Ballard, 1982), (Reeke & Edelman, 1988). However, where much has been gained from the general concept of networks of simple units, little progress has been made towards a general solution for the optimal structure of such a net. Recently, there has been growing interest in dynamic adjustment of the network structure to match a learning task. Honavar and Uhr summarize the common approaches to generative networks in (Honavar & Uhr, 1991). All the approaches suggested involve the generation or elimination of nodes of the type common to the network, perhaps with altered unit functions. The approach suggested here differs in that it allows for the generation of nodes that are themselves networks, thus creating a hierarchy of connectionist networks. In essence, a subnet with strong connections is

frozen, extraneous lower weight connections are cut off and the subnet is then replaced by a single node not containing the frozen substructure. Motivation for the algorithm can be found in neurobiology, where neurons grow in functional bundles and recurring substructures of neurons evolve for many specific tasks. Examples of this are the retinotopical organization of neurons in the visual cortex and the somatotopic one involved in the analysis of tactile sensations (Schwartz, 1988).

2.4.2 A Hierarchical Subnet Algorithm

A full implementation, test and optimization of this algorithm is beyond the scope of this project. Rather, the results presented here are meant to show that the idea behind the algorithm is worth pursuing and that even in its raw form as implemented in MailMind it allows for a drastic reduction in number of network nodes without significant loss of performance. The algorithm proceeds as follows:

- For each intermediate node, it attempts to grow a subnet.
- To grow a subnet, it follows connections stronger than a specified threshold weight and collects the nodes connected in this way up to a maximum number of nodes.
- If the number of nodes collected is greater than a specified minimum the growth is successful.

- The nodes in the subnet are then removed from the original net and replaced by a new node, that contains the extracted subnet minus any connections to nodes not in the subnet.
- This new node is connected to all the nodes its constituent members were connected to, with weights given by the averages of their connection strengths to a given node.

Figure 3 demonstrates the algorithm.

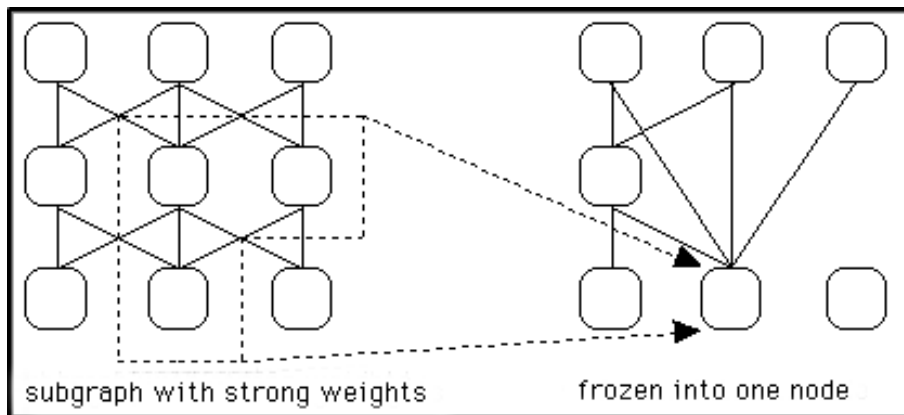


Figure 3: A Hierarchical Subnet Algorithm

In the MailMind version the algorithm uses the node the subnet was grown from as a single input node and adds a new single output node to which all connections previously leaving the subnet are reconnected. Also, the figure intentionally omits the reconnection of connections entering the subnet. It is possible to connect these to the newly created node if a check against the creation of cycles is performed for each, but in practice this does not produce good results. MailMind currently reconnects these nodes to ones

the new node is connected to, thus bypassing the new node. In general it is probably necessary to minimize the overall structural impact further to have this algorithm improve learning performance. The concept is, that with less nodes but more meaningful node functions a better balance can be achieved between a large network that can approximate more complex functions but converges more slowly and a smaller network that converges quickly but fails in more difficult cases. To allow for use in backpropagation, the subnet node's derivative is currently approximated by an average of the derivatives of the node functions it contains.

3 Results

3.1 Performance

MailMind was tested on real e-mail files to evaluate real world performance. The training sets were intentionally not chosen to be statistically representative samples, but rather the first 100 message of a mailbox files were used for training. This approximates the realistic case in which MailMind is asked to learn from already present and sorted e-mail messages and then asked to sort more e-mail as it arrives. Thus, there are a number of cases in which the training set contained more erroneous examples compared with the total number of examples than the test set did. In these cases it is likely for MailMind to achieve a higher percentage of correct classifications in the test than in the training set. The given statistics should therefore be viewed as a mea-

sure of the overall performance and applicability of MailMind as an e-mail sorting agent and as an evaluation of the two presented encoding schemes.

ASCII Encoding	% correct training	% correct testing
UBC, Glenayre	99.5	99.3
UBC, Germany	92.1	94.9
UBC, Canada	90.7	88.2
5 mailboxes	89.2	94.2
5 mailboxes, 2 layers, learned two times	62	72

Table 1: Results Using the ASCII Encoding Scheme

ID Encoding	% correct training	% correct testing
UBC, Glenayre	99.5	99.9
UBC, Germany	100.0	100.0
UBC, Canada	96.6	93.9
5 mailboxes	97.8	94.2
5 mailboxes, 2 layers, learned two times	91.8	91.1

Table 2: Results Using the Message Part ID Encoding Scheme

Tables 1 and 2 summarize the results of two of the encoding schemes from section 2.2, namely the encoding using ASCII character scaling and that assigning ID's to address parts and encoding these ID's. The latter encoding was used in the version which uses a single 1 to encode the country part of an address as explained at the end of section 2.2. Except where mentioned in the tables all tests ran with an error backpropagation coefficient of 0.06 in a network with 1 layer of hidden units and a structure similar to the one shown in 1. Each training example was given 5 times in a row and all of the examples were repeated 6 times, except for the network with two layers of hidden units, for which they were repeated 12 times.

The nature of the test mailboxes used breaks down as follows:

UBC This is a mailbox of mail the author received from within the University of British Columbia. Most examples contained in it end in 'ubc.ca'.

Many end in 'unixg.ubc.ca'.

Glenayre This box contains messages the author received from Glenayre Electronics. All examples are of the form '<userid>@<localhost>.glenayre.com'.

Germany These are messages from Germany, mostly ending in '.de' and '.org'. E-mail from one address, namely the author's mother's, by far dominates the contents.

Canada Classified as 'Canada' are messages from within Canada, but outside of UBC, thus mostly ending in '.ca'. Also contained are a number of messages ending in '.de' from Canadians temporarily visiting Germany.

Zorn This mailbox is the fifth mailbox used in the case with five output nodes. It contains only three distinct addresses as all messages in it are taken from the John Zorn Digest mailing list.

The mailbox pairings in tables 1 and 2 are chosen to test MailMind's performance in distinguishing between various types of data sets and to compare the performance effect of the two encoding algorithms in these classification tasks. Analyzing the results, we find the following:

UBC, Glenayre These mailboxes contain only addresses resulting in very distinct and consistent encodings using both schemes with little or no noisy examples, thus both lead to fast and correct learning and generalization.

UBC, Germany All address parts change in these mailboxes except for the country codes. It is clear that the first encoding scheme performs worse at learning this classification. This is due to the fact that in the encoding of 'ca' the two non-zero entries are only $1/40$ and $1/10$ different from that of 'de', making the development of an accurate feature detector for this part slower. In the second scheme, at least two entries change by 1 in the encoding, making the difference easier to detect for the net.

UBC, Canada Due to the fact that the UBC mailbox contains a number of message that do not contain 'ubc' these example are very hard to distinguish categorically from those in the Canada box, and therefore MailMind performs worse with both encoding schemes. However, the second encoding scheme again yields better performance by producing the more easily distinguishable encodings.

Five Boxes, One Layer In this larger test the performance differences of the two encoding schemes are more significant. Whereas the second one shows no performance degradation in either learning or generalizing, the first clearly causes MailMind to learn more slowly and generalize

more poorly. As explained in section 2.2 this is expected due to the fact that it produces encodings not easily distinguishable from each other and not very distinct in their internal values.

Five Boxes, Two Layers Learning slows down for both schemes when a second hidden layer is added to the network, but the performance penalty, due to the close values in the encoding using ASCII values, slows learning down immensely when the error adjustments are spread over several layers, compared with a slowdown of only a factor of about two using the message part ID scheme.

3.2 Subnet Extraction Results

Even in its raw form the algorithm shows promise when applied to network containing two hidden layers. In general, the network tends to recover very fast from the structural disturbance created by subnet extraction. For example, if in the case of five outputs and two layers MailMind uses this algorithm with a minimum of two and a maximum of four nodes per subnet, a number of nodes equivalent to a whole layer (usually about 10 for MailMind) can be eliminated from the topmost hierarchical level of the net. This only causes a short period of increased errors due to structural changes, and then convergence continues quickly. All in all this is still slower than learning in a net with one hidden layer, because the initial period of coercing the two layered net into convergence far enough to apply the algorithm is almost as long

as making the other net complete its learning task. The most interesting case to test this algorithm on, which could not be performed in the scope of this project, would be one in which a net with a single hidden layer shows poorer performance than a two layered net. In this case the algorithm may well speed up the learning task without there being an alternative that is intrinsically faster.

4 Conclusion

The success of MailMind at classifying real-world sets of e-mail messages to a high correctness percentage demonstrates the applicability of backpropagation networks to solving everyday problems for a large user base. Its speedy convergence, good generalization properties and adaptability make MailMind ideally suited for the proposed problem task.

As is usually the case with backpropagation networks, many parameters play a role in determining overall performance of MailMind in classification of messages. Of prime importance are a scalable encoding scheme for e-mail addresses, that creates patterns distinct both in their constituent values and from other patterns, and a suitable network structure, that contains enough nodes to allow for error minimization, but not too many to discourage overfitting. MailMind addresses the first with a scheme that maintains ID's for known parts of e-mail addresses and encodes these ID's so as to create input patterns with the required distinctness. Empirical results show that an enhanced version of this scheme supports fast learning and good generalization, especially as compared to other possible approaches. An approach to solve the problem of an optimal structure for a given learning task is implemented in MailMind in form of a hierarchical subnet extraction algorithm.

There are many questions still to be answered concerning this algorithm. What is the tradeoff between initial size of the network and usefulness of such an algorithm? How much has a network be allowed to learn before

extracting subnets in the proposed manner makes sense? What are good values for minimal and maximal size of such subnets in a given situation? How can structural impact be minimized in reconnecting external nodes? A mathematical examination as well as an empirical one on more complex classification problems should yield answers.

References

- Corman, Thomas H., Leiserson, Charles E., & Rivest, Ronald L. 1990. *Introduction to Algorithms*. Cambridge, Massachusetts: MIT Press.
- Feldman, J.A., & Ballard, D.H. 1982. Connectionist models and their properties. *Cognitive Science*, **6**, 205–264.
- Gallant, Stephen I. 1993. *Neural Network Learning and Expert Systems*. Cambridge, Massachusetts: MIT Press.
- Honavar, Vasant, & Balakrishnan, Karthik. 1995. *Evolutionary Design of Neural Architectures - A Preliminary Taxonomy and Guide to Literature*. Tech. rept. Department of Computer Science, Iowa State University, Cambridge, Massachusetts.
- Honavar, Vasant, & Uhr, Leonard. 1991. *Generative Learning Structures and Processes for Generalized Connectionist Networks*. Tech. rept. Department of Computer Science, Iowa State University.
- Murata, Noboru, Yoshizawa, Shuji, & Amari, Shun-ichi. 1992. *Network Information Criterion - Determining the Number of Hidden Units for an Artificial Neural Network Model*. Tech. rept. University of Tokyo.
- Pozo, Rodan. 1996. *TNT - A Template Numerical Toolkit for Linear Algebra*. Tech. rept. National Institute of Standards and Technology. Presentation at the ETPSC III.

- Reeke, George N., Jr., & Edelman, Gerald M.:1988. 1988. Real Brains and Artificial Intelligence. *In: The Artificial Intelligence Debate - False Starts, Real Foundations*. MIT Press.
- Rosenblatt, F. 1958. The Perceptron: a probabilistic model for information storage and retrieval in the brain. *Psychological Review*, **65**, 386–408.
- Rumelhart, D., Hinton, G.E., & Williams, R.J. 1986. Learning internal representations by error propagation. *Chap. 8, pages 318–362 of: Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press.
- Schwartz, Jacob T. 1988. The New Connectionism: Developing Relationships Between Neuroscience and Artificial Intelligence. *In: The Artificial Intelligence Debate - False Starts, Real Foundations*. MIT Press.