Reflective Modular Neural Network Systems

F.J. Śmieja
smieja@gmdzi.uucp

H. Mühlenbein
muehlen@gmdzi.uucp

German National Research Centre for Computer Science (GMD),
Schloß Birlinghoven, 5205 St. Augustin 1, Germany.

March 13, 1992

Abstract

Many of the current artificial neural network systems have serious limitations, concerning accessibility, flexibility, scaling and reliability. In order to go some way to removing these we suggest a *reflective neural network architecture*. In such an architecture, the modular structure is the most important element. The building-block elements are called "MINOS" modules. They perform *self-observation* and inform on the current level of development, or scope of expertise, within the module. A *Pandemonium* system integrates such submodules so that they work together to handle mapping tasks. Network complexity limitations are attacked in this way with the Pandemonium problem decomposition paradigm, and both static and dynamic unreliability of the whole Pandemonium system is effectively eliminated through the generation and interpretation of *confidence* and *ambiguity* measures at every moment during the development of the system.

Two problem domains are used to test and demonstrate various aspects of our architecture. *Reliability* and *quality* measures are defined for systems that only answer part of the time. Our system achieves better quality values than single networks of larger size for a handwritten digit problem. When both second and third best answers are accepted, our system is left with only 5% error on the test set, 2.1% better than the best single net. It is also shown how the system can elegantly learn to handle garbage patterns. With the parity problem it is demonstrated how complexity of problems may be decomposed automatically by the system, through solving it with networks of size smaller than a single net is required to be. Even when the system does not find a solution to the parity problem, because networks of too small a size are used, the reliability remains around 99–100%.

Our Pandemonium architecture gives more power and flexibility to the higher levels of a large hybrid system than a single net system can, offering useful information for higher-level feedback loops, through which reliability of answers may be intelligently traded for less reliable but important "intuitional" answers. In providing weighted alternatives and possible generalizations, this architecture gives the best possible service to the larger system of which it will form part.

Keywords: Reflective architecture, Pandemonium, task decomposition, confidence, reliability.

1 Introduction

The aim of this work is to define and demonstrate a new type of neural network architecture. The architecture is a *reflective* one, and it is the themes of *reflection* and *self-assessment* that are uppermost in its design criteria. The architecture provides an environment for neural networks that allows them to be used practically in hierarchical, hybrid integrated systems, which may be applied to real problems in a changing world. Such systems need to be *robust* and *fault-tolerant* in order to stand any chance of having practical real-world applicability. However, current neural networks are characterized by their *non*-reliability, and this is indeed indirectly linked to their prime advantage: enabling *generalization* to occur. Using our MINOS module prescription for an intrinsic self-assessment ability, and the Pandemonium reflective system idea for modular integration, we show how the disadvantages of neural networks may be overcome, and thus open the way to their exploitation in workable hybrid systems.

It may seem logical that one should strive to create modular systems of networks, but relatively few neural network programs actually actively pursue this goal. It is maybe because the message received from the first wave of the Parallel Distributed Processing (PDP) movement was one of "distributedness" of information [23]. The idea was to pull as far away as possible from standard symbol-pushing methods of information processing, that tended to concentrate on local representation of knowledge. The drawback of symbolic methods was that they required explicit specifications of interrelatedness among concepts. The PDP contribution was to concentrate explicitly on the representations, and indeed to evolve them in a distributed manner through adaptive establishment of underlying correlations between concepts, in one giant interconnected network. The first wave of PDP, was thus, naturally, to focus on the single, monolithic network idea, and not encourage interaction between smaller networks.

The current situation can be characterized as follows. At the symbolic extreme one finds little prospect of emergent knowledge through representational generalization, but the networks can be scaled relatively well with volume of information. Whereas at the PDP extreme one is offered good generalization possibilities but, unfortunately, disappointing scaling prospects. Scaling problems are one broad difficulty one is confronted with when attempting to employ a *general, unstructured, monolithic* network on a general problem domain. Furthermore, depending on the size of the association/classification problem, the network used is always going to be limited by its storage *capacity* determined by the number of free parameters it has. There is, however, no gentle way of avoiding slapping into the capacity upper-bounds. Additionally, depending on the difficulty of an association/classification problem, there is always going to be a limitation in the *capability* of the particular network chosen to do the job. If a very general network is used, that can perhaps perform any relevant mapping, it would nevertheless have the drawback of being too cumbersome and will require interminable adaptation times. By definition, flexible learning systems have no notion of task difficulty before adaptation begins, and so premeditated network architecture choice is not always possible. Fundamentally-speaking, for egoistic (single-net) learning, enforcement of distributed learning on the large-scale is rewarded by complexity nightmares.

The larger and more horrendously complicated a net becomes, the less we understand and can control what is or may be occurring in it. Even apart from complexity and capacity difficulties, which might be dealt with through some particular "divide–and–conquer" technique [16], there are other problems related to most neural network models. One is confronted with them when it is desired that the system really does operate in an adaptable and "malleable" way. This belongs to the general notion of *progressive learning*. Unless our system can cope with a changing world, we may as well employ existing static, and vastly more efficient and thorough, mathematical optimization and partitioning techniques instead of neural network models, to build a purely functional system operating in a highly restricted world of possibilities. The basic neural network architecture and learning algorithm generally do not of themselves allow for robust progressive learning (learning knowledge progressively, as it is experienced). They tend to suffer from the so-called "recency effect", whereby the *order* of mappings/memories experienced in the world determines their degree of acquisition by the net. Thus, the most recent association that is learned will be so well learned in relation to all the others previously learned, that one finds a "swinging" of network specialization between the currently learned patterns.

A further problem is that a network possesses what we name the *dog-paw problem*. This is exemplified by the reaction of trained networks to so-called "garbage", or meaningless, patterns. It may be possible for a few arbitrary blobs at the input to be "recognized" by the network as a letter "C", or as one angle of view of a familiar face. Such responses by the network are only to be expected, when one considers what in fact the network learns during its training session [11].

When it is desired to learn new patterns the progressive learning property may allow it to be done fairly easily, but it may also be found that the generalization the association represents is at odds with a number of previous associations. Thus without some fixed point on the optimization of the current association, the subtleties of learning this generalization among the previous ones will be lost, and the previous ones unavoidably "catastrophically unlearned" [5] in order to accommodate the current one most easily. Such drawbacks must also be dealt with, before a neural network system can be claimed to be robust to changing environments.

With our reflective architecture we propose to deal with the above points in the following way. First the basic net is bound into a *module* that can provide some weighted idea of its progress, as it adapts. The module consists of two parts: the *Monitor* tries to assess the capability of its *Worker*. The adaptation is looked upon as a continuous process. Then many of these modules are employed in parallel, within a *Pandemonium* system, that consists of *specialized modules*. The method of attacking the complexity and scaling issue is then through *automatic* decomposition of the problem domain. The Pandemonium system is allowed to operate through dynamically allocating tasks to be dealt with, or learned, by the appropriate sub-module, and accounting for environment alterations by internal shift of the reliability parameters. Non-dynamicism problems of new learning are tackled by the constantly high value of reliability, even as new problems are being tackled. Then using the set of reliability parameters provided by the comprising sub-modules, and inherited by the encompassing system, and also external knowledge of the possible acceptable form of the answers (positive and/or negative feedback), an *acceptance filter* is constructed, that gates and qualifies the utterances of the system. With this development the region of the fuzzy answer may be probed.

The paper is organized in the following way. In section 2 we discuss other neural network modularization methods, and look at the modular idea in general. In section 3 the basic Pandemonium idea is introduced, and our modular extension to it is described. This extension raises the important technical issue of the *modular decomposition–recomposition problem*, which we discuss in section 4. Our solution to this problem comprises, firstly, the definition of the MINOS module environment for individual networks (section 5), and, secondly, our technique for evolving expert modules within a Pandemonium in a sensible and directed manner. This is described in section 6. In section 7 the early stages of relaxation in the system are considered. A higher-level answer acceptance filter mechanism that uses the self-observation property of the MINOS modules is described in section 8. Finally, in section 10 a further sub-module system is described, that is to be employed by the Monitor component of a MINOS, for problems of higher complexity. The reflective architecture is then tested on a handwritten digit recognition problem (section 11), and the parity problem (section 12). We demonstrate that *reliable* and *usable* behaviour is always achieved. In section 13 we discuss the issues of complexity and scaling and in section 14 consider future developments of our architecture. In section 15 conclusions are formed.

2 Modularization

There are a number of techniques currently used for simplifying neural network learning through modular methods. The general conjecture assumed in all such methods is that the scaling difficulties unavoidably present for single nets are somehow removed when modular structure is imposed, due to the following factors. Firstly, to deal with higher complexities, the size of the networks themselves need not increase explosively, since we may add more modules and divide the problem some more. Secondly, the networks themselves are always of a handleable size, and thus do not scale badly in terms of training times, as do the single nets.

The methods currently considered fall into two main categories in the literature: problem decomposition and hierarchical sub-tasking.

In problem decomposition it is recognized that a large-scale problem is too tough to be learned by a single network, and so it is broken up over a number of sub-networks. With such techniques there always exists the "integration problem": having broken up the tasks over a number of parallel experts, how do we recombine the results to form a full answer? One may solve this using adaptive sub-module weighting methods [21, 8], individual sub-module reference vectors [9], pre-determined sub-module functionality followed by "connectionist gluing" of the functionalities together [36, 19], or dynamically determined and tabulated sub-module specialities [1].

In hierarchical sub-tasking it is observed that the global task can be usefully split up in a vertical manner, as a number of quasi-independent sub-tasks, that may simultaneously, or sequentially, influence the overall output. Simultaneous sub-tasking is demonstrated in [25, 17, 4], while strictly hierarchical decomposition has been tried in [3, 4] and numerous hybrid neural methods, where the problem is separated into a number of distinct and consecutive sub-tasks. Indeed hierarchical systematic decomposition is always borne in mind when one intends (as we do) to develop general flexible and autonomous systems. The initial priorities for us must be, nevertheless, to develop reliable building blocks for this purpose, such as those presented in this paper. Therefore we restrict ourselves for the moment to the lower levels of modular hierarchies.

The modular thesis is that we may "get away with" manageable scaling at the modular level even though the use of a single net gives exponential scaling. One reason for this is precisely because the decreased *number* of patterns to be accommodated by each sub-module reduces the *time* needed in learning. Another reason is that, by the same token, the *complexity* of task tackled by each individual network becomes lower than the total task complexity.

Now, in the same way that problem complexity explodes with the size of the input space, it may be argued that dividing this space linearly (addition of modules) may cause the individual module problem complexity to *diminish* likewise *super*–linearly. In order to explain how this might be possible, we define a simple mathematical measure of mapping complexity, for the special case of a 2–class mapping problem. This complexity measure, M_c, was suggested in [33, 32]. If the problem is defined as the separation of two sets of binary patterns $\mathcal{A}$ and $\mathcal{B}$ in a N_I dimensional space, then

$$M_c := \frac{\min(|\mathcal{A}|, |\mathcal{B}|)}{2^{N_I - 1}} \sigma_{\mathcal{AB}} \qquad 0 \leq M_c \leq 1 \tag{1}$$

where $\sigma_{\mathcal{AB}}$ is the overlap between the two sets of patterns, defined by the average (over the members of, say set $\mathcal{A}$) fraction of nearest neighbour points in the input space that are foreign (belonging to the other set).

With this measure a purely random mapping has a complexity of $M_c \approx 0.5$, since about half the nearest neighbour points are foreign, the parity problem has the maximum complexity $M_c = 1$ (all nearest neighbour points are foreign), and for the "=1" problem [15], the maximum value, for $N_I = 3$, is $M_c = 0.56$. Easy problems such as separating a single pattern from all others, has intrinsic M_c that diminishes to zero with N_I very quickly, actually given by $M_c = 2^{-(N_I - 1)}$. This is because only one of the points has all foreign nearest neighbours.

In decomposing the problem over N_M modules, in fact we are sharing out the sets $\mathcal{A}$ and $\mathcal{B}$ into N_M pairs of sub-sets $(\mathcal{A}^1, \mathcal{B}^1), (\mathcal{A}^2, \mathcal{B}^2), \ldots, (\mathcal{A}^{N_M} \mathcal{B}^{N_M})$. The complexity of the problem for the (parallel) modular system is now given by the maximum sub-tasks complexity:

$$M_c^{\text{sys}} := \max_p \{M_c^p\}, \qquad p \in \{1, 2, \ldots, N_M\} \tag{2}$$

Now, since the individual sub-tasks have less patterns to separate, it may reasonably be claimed that their individual complexities derived from equation (1) are all less than that of the original problem. The interesting question is, however, how low the individual complexity can be made. When one views the simplicity gained through decomposition as merely a matter of reduction of the *quantity* of patterns to be mapped, then equation (1) tells us that minimal complexity M_c^{sys} will be obtained when each sub-module receives an equal number $\frac{|\mathcal{A} \cup \mathcal{B}|}{N_M}$ of patterns. The minimal system complexity is then given by

$$M_c^{\mathrm{sys}} = \frac{M_c}{N_M}. \tag{3}$$

This is the case when only the decomposition of the first term in equation (1) is considered. The modular hypothesis, however, goes further, and argues that it is not only the sets $\mathcal{A}^p$ and $\mathcal{B}^p$ that are reduced in size but also their average overlap $\sigma_{\mathcal{A}^p\mathcal{B}^p}$. Thus,

$$\sigma_{\mathcal{A}^p\mathcal{B}^p} \leq \sigma_{\mathcal{A}\mathcal{B}} \qquad \forall p \in \{1, 2, \ldots, N_M\} \tag{4}$$

This is only possible when the sub-modules have overlapping regions of specialization, such that

$$(\mathcal{A}^p \cup \mathcal{B}^p) \cap (\mathcal{A}^q \cup \mathcal{B}^q) \neq \emptyset, \qquad p \neq q \tag{5}$$

for only then will the nearest neighbour points for sub-module p be possibly different from those for the original problem.

The question is, *how* might the sets be decomposed in order to achieve such useful problem breakdown? And what are the side-effects. The first question was tackled using external and static knowledge of the pattern classification problem domain, using reference vectors in [9], phoneme sets in [36, 19], and a Kohonen net clustering method in [18]. Our method of reflective modular neural network division is different to the methods used by other researchers mainly because we try to retain the independence and self-sufficiency properties of the sub-modules. This will become clearer later when we describe the modules themselves (section 5). The result is to allow greater flexibility in what is learned by the experts, that is automatically chosen by the experts themselves, and may dynamically change with the environment. The question of side-effects of the overlapping regions of specialization results in the modular decomposition–recomposition problem (section 4. We return to it once more with respect to the decomposition afforded in the Pandemonium system in section 13.

3 Pandemonium

The Pandemonium architecture was originally proposed in 1958 by Selfridge [27]. The idea is basically one of dividing and conquering a complex problem domain, through use of a number of specialized agents working in parallel. All these agents, or *demons*, process the same signal, or question, in parallel, and each provides a possible answer. The demon that "shouts the loudest" is taken to be that which is most believable because the question posed lies in its region of speciality. Thus, for a letter classification problem we might have 26 demons, each of which is specialized on recognizing a particular letter. Each demon processes using a slightly different filtering technique on the incoming pattern, and thus recognizes characteristic features belonging to its area of expertise. The learning procedure used by Selfridge was gradient descent for adapting weights determining

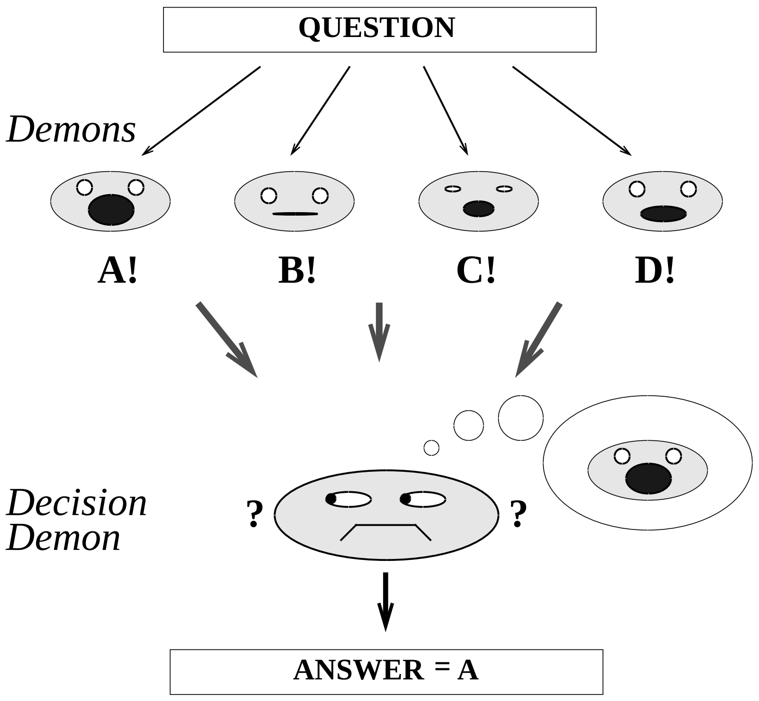

Figure 1: Selfridge's Pandemonium

what types of filters each demon used. A "decision demon" at the top of the hierarchy had the job of choosing the demon that shouted the loudest, and passing on his output. After learning there results the scenario illustrated in figure 1. The first demon shouts the loudest for the input shown, conveying his claim that it contains a lot of "`A`–ness". The other demons may see aspects of their particular letter specialities in the input, but do not shout as loud as the first demon, who sees a great many aspects of "`A`". Thus the decision demon chooses the `A`–demon.

It is clear that we can translate the demons used by Selfridge into our present-day notion of "grandmother cells" (GC), and indeed the entire Pandemonium system as a GC neural network. However we prefer to take the *idea* employed in Pandemonium, that led to a very successful pattern recognition system, and extend it to specifying a modular system of neural networks. We replace each of the demons in figure 1 with a neural network module, and run the system using the divide-and-conquer principle [28, 31]. Each module will become specialized to a greater or lesser degree to a particular aspect of the task. Equivalently, if the neural network modules are concerned with storing memories, each will be concerned with (specialized on) memorizing a certain set of memories.

With this extension we retain the divide-and-conquer specialization problem-solving strategy,

Figure 2: Illustration of the modular recomposition confusion problem.

but where Selfridge's division resulted in *local* knowledge distribution (GC), or in his later applications [26] distributions of a number of non-adaptive filters, our method has knowledge partly distributed, in the neural network modules themselves. What is more, adaptability is at the core of the system.

4 The modular decomposition–recomposition problem

Adoption of the Pandemonium module architecture may help us with the complexity issue, but at the same time it immediately generates a very significant problem that also affects other modular neural network systems. We call this the *modular decomposition–recomposition problem.* It can be expressed in the two questions:

1. How can the modules adaptively specialize into sub-tasks?
2. Which of the specialized modules is likely to give the correct answer, at any time (figure 2)?

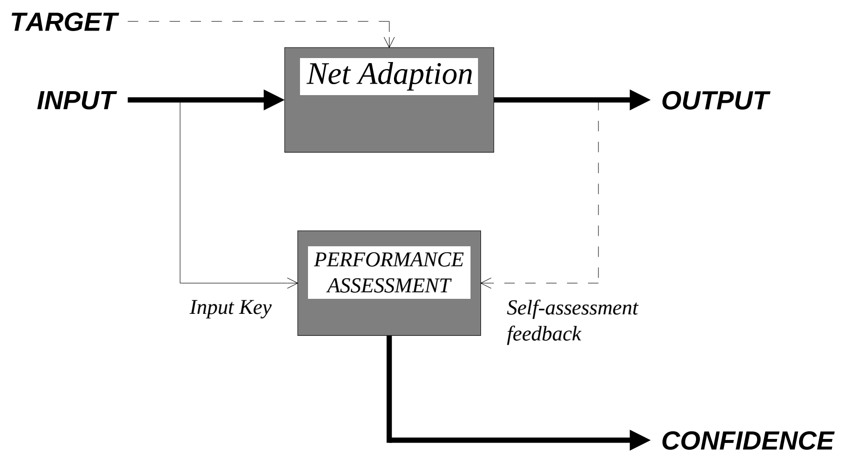

Figure 3: Self-assessment

Other modular techniques get around the first question either by explicitly deciding how the problem will be decomposed, or deciding at a certain point in learning to form a new expert to resolve some conflict [21, 1]. Some approaches to the second question were discussed in section 2. The problem was considered in depth in [9], in which it was eventually decided to use a reference linear vector quantizer (LVQ) method to label regions of expertness. The problem results directly from the fact that we wish to have modular disassociation in terms of a system of self-contained agents, but at the same time need to retain enough global knowledge somewhere in the system to be able to *choose* in the allocation phase and *discriminate* in the recall phase.

Now, Selfridge's Pandemonium was not bothered by such questions, because each demon had merely a YES/NO output function, modified by the degree of shout associated with it. But neural network outputs are generally of a vector form, thus the question is an important one for our system. How might we construct a "shout" measure from the output vector? How might we the experts be coaxed into particular specializations at all? We avoid the first question through the design of our MINOS modules, each of which generates a confidence measure. The second question is dealt with through our dynamic module allocation technique (section 6).

5 Reflective MINOS modules

5.1 The principle of self-assessment

The principle to be followed in our design of the MINOS module is illustrated in figure 3. The original neural network system associates an input pattern via some mapping that has been learned, and produces the result vector of its processing as an output. There may also be a target input to the network (shown with a dashed line above the network). This is the case for supervised learning problems, to which we confine ourselves in this paper.

Such a "dumb" network system may be given a voice of sorts by adding on the "performance assessment" block. This produces the sought-after *confidence* measure. It is to take as input the input to the processing network and possibly (shown dashed) the current output of the same network. During normal (non-learning) operation of the system, the processing net produces an output, and the assessment, working on the way the input "looks" and (possibly) the way the output "looks", produces a measure of the confidence that this module has in itself, that the output produced is correct. During learning, the whole system adapts, so that the confidence changes dynamically as well as the output. This idea of using one network to monitor the progress of another has also been employed in exploration [34] and control [24]. In these approaches it was attempted to reproduce the current error of the other network with respect to its target mapping (so that robot configurations of high error might be explored and better learned). Our performance assessor has a different, simpler, form, that has better learning prospects (the assessor must also cope with its *own* mapping problem!) and allows for an obvious decomposition interpretation (section 6).

Self-assessment is a most important notion, and one which we claim cancels out the problems of imperfection and unreliability with which one is confronted, when neural networks are employed in a larger system. Plain neural networks are intrinsically limited. We do not believe that with any number of minor improvements to the algorithms used, or the structure employed, or the training examples used, really significant advances can be made, or that perfect performance can be obtained. However, if we decide to *accept* that a network may not learn everything perfectly, or may not always be able to generalize reliably, or may have a too limited structure to be able to cope with all aspects of a particular problem domain, then we begin to make steps in the right direction. This direction is one of training modules not only to learn what they can, but also to *learn to know what they can.* Thus, if a module can self-observe so that it is able correctly to couple its response of "I know" with a correct answer, then it will be performing perfectly—for it only gives correct answers!

The quest then, is no longer for perfection within a single network that always answers, but for perfection in modules that only answer sometimes. The latter modules optimize their *reliability*, which is defined (see section 8) as the measure of performance when only questions that are actually answered by the system are considered.

5.2 Architecture of the MINOS module

A MINOS module is shown in figure 4. From the outside (i.e., as far as the rest of the system is concerned) the module has three input sites and two output sites. The pattern to be associated by this module is communicated during learning through the site "input" and "target". In a MINOS module both the net adaptation and performance assessment blocks of figure 3 are neural networks;

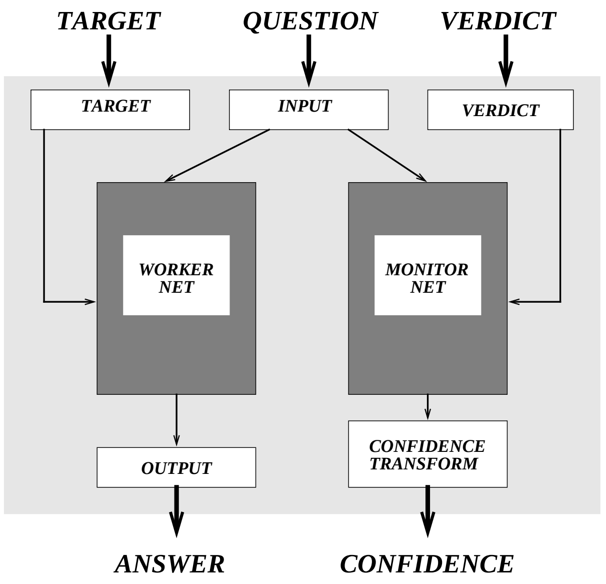

Figure 4: The MINOS module architecture.

the adaptation block is called the *Worker net* and the assessment block the *Monitor net*.

The choice for the Monitor network should depend on the type of Worker network chosen, since part of our philosophy is also to have similar positive and negative aspects of learning in both the networks in a MINOS. We choose the Worker net to be a feed-forward neural network, trained using the Back-Propagation learning procedure [22].

With the Monitor network it is wished that the module may comment on its ability to provide a reliable answer to the question being posed. We noted in our forerunner of this paper [31] that there are various possibilities for this, ranging from table-lookup to error-estimation techniques, such as mentioned above. However, because of complexity and adaptability considerations, and the desire that the two nets have similar memory-loss and learning time characteristics, so that the Monitor can inform well about knowledge decay in the Worker through gradual pattern infrequency or environment alterations, we went for the following simple choice.

The Monitor net is a single output unit classifier network, with one layer of hidden units. Each time a MINOS module is chosen for positive learning of the input, the output of its Monitor is trained to become closer to 1, and for negative learning (or simply no learning at all), closer to 0. The output during a recall phase indicates the confidence the MINOS module has in its ability to say anything about the input using its Worker net. Thus the "confidence transform" box shown in figure 4 is in this implementation version simply an identity function.

The verdict site informs the module whether the association currently being performed is of a positive or a negative nature. This information is used by the Monitor to bring its output with respect to this input closer to a higher or lower confidence, respectively.

As suggested in the previous section, the Monitor might also receive an input from the output of the Worker net itself (not shown), which could be used to base the confidence measure also on the characteristic form of the Worker output. This possible extension is not yet implemented in our current versions. So what the Monitor learns is to recognize the input pattern as one that has been processed (learned) by the Worker, and not to recognize (or classify oppositely) a pattern that has not.

5.3 Operating phases in a MINOS

During a learning phase the input pattern and target pattern are applied to the module input sites and adaptation is performed by the Worker net. It learns the current association until it reaches the learn tolerance, *tol* (= generally 0.2). So, unlike most implementations, which perform either batch learning (update weights only after all patterns in the training set have been seen) or online learning (perform one or a few update epochs per pattern then move on to next one), we demand that the association be *completely* learned every time it is seen by the system. This is important: it means at every stage in the system's development, that it has complete knowledge of that which has just happened, and does not require to see everything hundreds of times before any event is perfectly learned. The requirement for this property should be clear: we want the system to work in a real-world environment, and not in a static one in which the training set is always defined and may be seen an indefinite number of times. Furthermore, the environment may be changing with time, and the system should expect that its past knowledge may decay and become less important with time, and that other associations may become more important. The system should have the capability of organizing its knowledge dynamically, and being sure about what has most recently happened.

So the Worker of the MINOS that has been chosen to learn the association, learns it. The other Workers may either perform negative learning (we avoid this in our implementations) by which is meant responding in an unsharp way to the input, or simply do nothing (as in our experiments). From the verdict site the Monitor divines the form of the learning. If it is positive learning it learns to associate this input pattern with a higher confidence (mapping its output to 1 within tolerance), and if it is negative (or no) learning, with a lower confidence (mapping its output to 0 within tolerance). The details of how a MINOS is chosen is explained in section 6.

During a recall phase the Question represents the only input to the MINOS, and the outputs are the Confidence from the Monitor and the Output from the Worker. Thus our Pandemonium "picture" of recall has a slightly different form to figure 1, and is illustrated in figure 5.

Through the set of self-assessing confidences obtained from each MINOS for a particular pattern, the controlling system can decide how further to process and deduce from the answers given. This

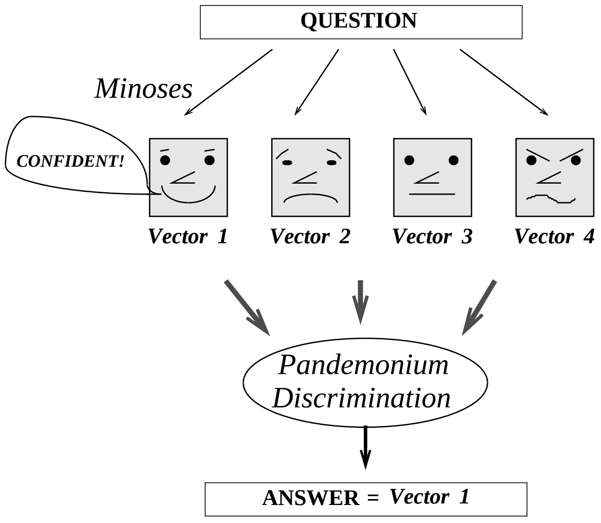

Figure 5: Illustration of our version of Pandemonium. Unlike Selfridge's shouting demons, that based believability on the strength of the actual answer given, our sub-modules accompany each answer, however strong, with a *confidence* measure.

is expounded in section 8.

The name MINOS is an acronym for "**M**INOS **I**ntrospects, **N**urturing **O**uter **S**peculations", which is intended to describe the co-existence of its (and the system of which it forms a part) functional and of its self-assessing nature, and of the recursive nature of a hierarchically modular system. One may also like to think of MINOS modules as roaming their own private trails in a giant labyrinth, to which the solution may only be seen at a higher level (which may also be in another labyrinth).

6 Evolving specialized nets

With our MINOS specification and the notion of confidence we have solved the second aspect of the modular decomposition–recomposition problem, through providing information about the self-assessed abilities of each expert. But what of the first: How do we decide which module to choose for positively learning an input association?

Our solution involves choosing the most appropriate *form* of all the output vectors from the Worker networks. We illustrate our choice criterion in figure 6. Say the target vector consists of a single strongly activated unit, with all the others at low values. The *form* of this target may be described as "fairly uniform, with a blip at unit 8". The Worker that best reproduces this in its

Figure 6: Evolution of specialized nets. The target to be learned will be allocated to that module producing the closest *renormalized* output.

output is the third one shown in figure 6. The first one has most of the units correct (at zero) but the blip in the wrong place. If this were chosen, the amount of learning required would be great and also probably very destructive for other learned patterns. The second one has a larger blip than the third, in the right place. But this blip is also too broad, and whatismore the relative form of the other units is wrong, so using this MINOS would also tend to be destructive to its past history of learning that produced this effect. The fourth Worker has little blips, but in the wrong place.

We may write the choice method mathematically in the following way. If $\vec{o}$ is the output from a Worker, then first we *amplify* it:

$$\vec{o} \mapsto \vec{o}' = \frac{\vec{o} - \vec{I} \cdot \min\{\vec{o}\}}{\max\{\vec{o}\} - \min\{\vec{o}\}} \tag{6}$$

then we subtract from the resulting vector $\vec{o}'$ the target vector $\vec{t}$:

$$\vec{s} = \vec{o}' - \vec{t}. \tag{7}$$

The appropriateness of this module is given by the *flatness* of $\vec{s}$. So let us define the "shout" for a module in this Pandemonium as

$$1 - (\max\{\vec{s}\} - \min\{\vec{s}\}). \tag{8}$$

Our reasons for choosing this method are the following. What we need to prevent in decomposing the problem domain, is that we lose hope of generalization, which is one of the most important aspects of using neural networks at all [29, 15]. Random allocation of tasks is unlikely to provide us with generalization benefits from the Pandemonium system. Such a method will obviously work on the set-size factor in equation (1) to reduce the complexity (for the Worker) a certain amount, but the σ_{AB} overlap factor will not have been optimally reduced. As for the Monitor complexity, that will be lower not through the set-size factor (which remains the same), but directly through the σ_{AB} factor. This is because patterns that were previously nearest-neighbours may be defined as belonging to the same set in a Monitor. The other methods of decomposition, sketched in section 2 are not appealing for our purposes either. They either restrict themselves to particular, clear-to-separate, pattern classification tasks, or they employ a single adaptive component to adjust the allocation of all the sub-modules. We believe the latter method will suffer drawbacks in the complexity scaling of the weighting–component with sub-module number, and may not be easily applicable to dynamic environments and one-shot learning. The theoretical scaling properties of our system are discussed in section 13.

One of our tenets in the Pandemonium system is the idea of *module independence*. We desire that a module possesses self-assessment properties (as discussed above), and require too, that it is not tied to a particular Pandemonium *system*, or indeed, requires to be an expert for only one limited application. Thus, we envisage the final whole system more as a pool of self-contained modules, any one of which may be used in sets of (overlapping) Pandemoniums. The realization of this is a future development (section 14, but we require the lower-levels to permit it. Therefore we always need to look ahead to such expansions, and make sure the modular structure of our system allows it.

In our allocation procedure it is emphasized that we *use* the very generalization promised by the network itself. Before a pattern has been learned, the form of the output can be observed, and it is decided how much effort will be required to change the current output to the desired output (target). Now our procedure uses the information of how great an effort will be required in order to shift the required hyperplanes, in order to perform the mapping. It is held that the network that best generalizes into this region of learning is also that network that can most easily be forced into learning this pattern.

By dealing with the *shape* of the output, rather than the simple squared-error measure, we can more effectively find the network that can learn the pattern with the least amount of *destructive learning* [30, 5, 2]. Theoretically, the argument is that the generalization properties of a network are determined by its set of hyperplanes in pattern–space (input $\rightarrow$ hidden) and in hidden–space (hidden $\rightarrow$ output). These hyperplanes are constructed during the learning, through the mapping of the training set patterns. If any regularities exist in the task domain [29], then the hyperplane configurations set up in the network will be sufficient to give good estimates for the mappings of unseen patterns from the same domain. Our choice method allows us to probe the current set of relative hyperplane biases for a particular pattern, in the hidden–space. If the *relative* set of output mappings looks promising then it is assumed that the network need disturb less of its configuration in order to learn them fully, than a net that looks less promising. Furthermore, since the net was already biased more *in this direction*, it is likely to grab more of the training set patterns that generalize in this direction. In this way a module may become an expert automatically.

It would be even better if the input–space hyperplanes were used as well, in the determination of module appropriateness, and ways of doing this are being investigated. In [30] it was suggested that the maximum effort required to move an input–space hyperplane could be used directly to discover the "destructiveness" of a pattern. Note that this method does not confine itself to a particular form of the target (in figure 6 a simple classification was chosen for clarity). Regardless of the pattern output coding required, the appropriate module will be selected. New experts evolve naturally from this technique, since sharp, but wrong, outputs produced by highly-trained nets are not chosen in preference to uncommitted outputs from untrained nets.

The expert evolution procedure is a *dynamic* allocation process, since the appropriateness of modules for particular inputs may change as the whole set adapts. An important consequence of this is that expertness may without much problem be altered as environments change and new experiences become more significant.

In summary, if a module has something correct to say about a generalizable input, it would most probably have been the module chosen to learn such associations, because the form of the output was best suited. Changing environments and changing emphases in training sets are accounted for in this allocation system through its constant appropriateness checking, which allows for forgetting, recency, and mild alterations of the task. Catastrophic forgetting of desired mappings may be avoided, since the most appropriate module will be chosen to learn significantly different patterns.

The final Pandemonium system is illustrated in figure 7. A number of MINOS modules containing Worker networks of various weight initializations, connectivity, or hidden layer sizes, are made available to the Pandemonium in solving the dynamically changing problem. The output from the Pandemonium consists of its answer and a confidence measure associated with it.

7 Module specialization and Worker-Monitor conflicts

A Pandemonium system is constructed of many *plastic* modules, and as a result is plastic itself, in the overall specializations it forms, and in the shifting of the expertise of its component MINOS modules. It is therefore natural to expect the system to "evolve" as a whole through time. Indeed the very ability to evolve to cope with varying environments is perhaps the most important concept of our Pandemonium. One would expect the system, given a number of available MINOS modules over which to divide the labour, to require a certain amount of time before it settles down, with the expertise of the individual sub-modules stabilizing (given also that the environment is unchanging, or changing slowly relative to the relaxation time).

The MINOS modules start off with their networks randomly initialized. The form of the initialization is as described in [30], with the "mass" of the hyperplanes somewhat larger than usual, so that the nets may start off with a certain degree of "direction" in their specialization, given by the

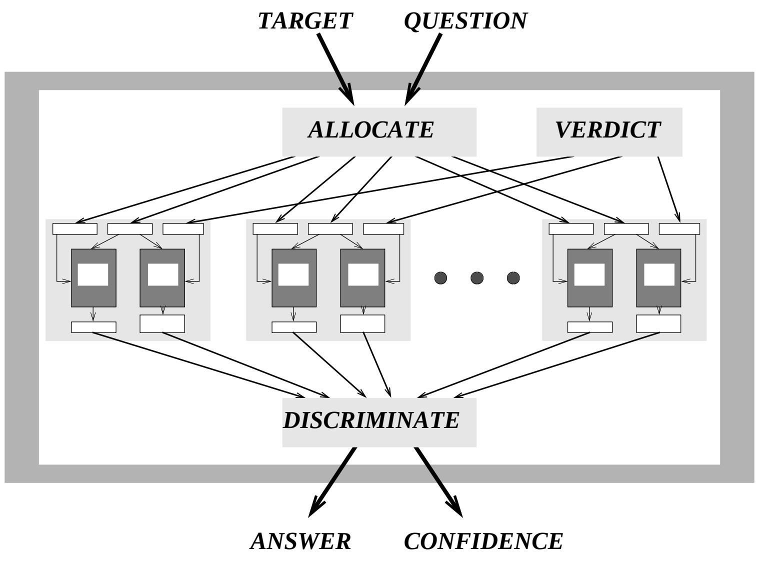

Figure 7: Our Pandemonium system in component form. The allocation and discrimination procedures described in the text are used to adapt the set of constituent MINOS modules.

initial random hyperplane divisions. The first N_M patterns to be learned, where N_M is the number of MINOS modules available to the Pandemonium, will be most probably distributed equally over the MINOS modules. The reason for this is that, once a MINOS Worker has learned a pattern, it is likely to be worse than all the other randomly initialized MINOS modules with respect to learning the next pattern. And so an unused MINOS will always have a higher shout for the class to be learned, than one just used to learn a different class. However, when all MINOS modules have been used up the specialization may proceed in a manner dependent on the generalization abilities of the Worker networks, as argued in the last section.

The obvious next question is whether this evolution process is at all controllable, so that, instead of automatically using all available modules straightaway, the Pandemonium drafts in fresh ones only when it is deemed necessary. Before discussing how this might be done, we consider the worst-case scenario: overspecialization and development of grandmother modules (GM).

As suggested above, if a particular MINOS in the system has been chosen early on to do a single mapping, then it is possible that it will always produce the answer it has been specialized to produced, whatever the input question is. This may be likened to a crude form of an overregularization problem in language learning by children: add "-ed" to every verb to form the past tense [12], since the great majority of cases require it. The problem then of forming hyperplanes between possibilities, and sectioning off this kind of answer, is completely up to the system's discriminator (the Monitor networks). Clearly, there would have been no benefit in using such a MINOS—a single node could replace it. We have the modular system equivalent of the GC in single networks, although in this case no problem has in fact been solved by such an allocation, that couldn't have been solved in an easier way. This occurrence is inevitable because in the relaxation process that is being carried out in the allocation of patterns (section 6) the only aspect of the MINOS modules that is being considered is the progress of their Worker networks. Now this is all well and good if the various forms of the possible answers are numerous enough to exclude the GM possibility. If there are N_c distinct classes to be distinguished, then a rule-of-thumb requirement to avoid a GM situation is:

$$N_c \gg N_M \tag{9}$$

This is clearly unsatisfactory: How is one supposed to *know* how many possibilities there are going to be before actually experiencing the problem? And what about future furthering of knowledge in larger environments? Surely it must be possible to use an arbitrary number of MINOS modules, and not necessary to know N_M beforehand.

We see this problem as one that may be dealt with at a later stage. It is a "meta" problem—to be solved at a higher level than the MINOS modules. For our first experiments this aspect (pools of MINOS modules and mixing of Pandemonium systems) of a large-scale system is not being considered, and we confine ourselves to investigating particular details of the decomposition and confidence ideas. The GM scenario is prevented in these experiments by restricting N_M. For example, in the handwritten digit experiments (section 11) the number of possible classes, $N_c = 10$, and most experiments were performed with $N_M = 2$ or 3.

One may derive more insight into the progress of the problem decomposition within the Pandemonium by considering Worker–Monitor conflicts. A conflict is defined to be the non-agreement of maximum shout and maximum confidence in the Pandemonium. Thus, if, during learning, a MINOS has the highest shout value, but the confidence is not the highest, then it is obviously not yet considered by the discrimination system to be the expert it claims it is. During normal processing another MINOS has a higher confidence, but presumably its Worker does not have the correct output. Reliability is likely not to be terribly affected by this (section 11), since ambiguity will be high (see next section). But the effect does inform one of the progress being made. Eventually, when no more conflicts exist, the system will always be choosing the correct expert to answer the questions posed the net.

8 Selective answering

In the normal processing mode of our modular Pandemonium system we obtain a set of confidences from each module. The output we choose to use may be simply that from the module with the largest confidence. However, we cannot escape the fact that there may be some wrong answers. For example, the modules may have been trained for only a (relatively) short time, or the environment is changing quite rapidly. This problem, of the system's *reliability*, is dealt with to a certain extent by the production of a confidence: for changing environments the confidences will generally be low. But recent patterns will tend to produce reasonably high confidences at the early stages of learning. We may use the information however in a subtler way too, so that we may, even for badly trained nets, nevertheless design a 100% reliable module.

The method involves answering only a subset of the questions posed to the system. In order for a system to decide whether it is going to answer or not, it must have an idea of whether it is confident about an output. This depends on the problem domain being tackled, and on the system's output possibilities. As we shall see in sections 11 and 12, one may impose output vector restrictions, so that only those outputs are accepted that have a particularly boolean form, or a boolean form that belongs to one of a set of possible boolean answers. In the case of a Pandemonium system, however, it is always possible, regardless of the form the output should take, or the number of answers the system is allowed to offer, to generate a confidence value.[1] Therefore a Pandemonium may always collect a set of confidences from the MINOS modules. This set may then be used to give information so that *filters* may be constructed, to sift out the foggy answers that are inevitably obtained.

The filter we use takes as parameters the maximum confidence from the Pandemonium, C_{max}, and the *ambiguity*, α, of the answer.

$$\alpha = 1 - (C_{max} - C_{nextmax}) \qquad 0 \leq \alpha \leq 1 \tag{10}$$

where $C_{nextmax}$ is the next highest MINOS confidence after C_{max}. So an answer is ambiguous when the two highest confidences are similar, and not ambiguous when the top confidence is high and all the others are near zero.

The filter (figure 8) only lets through answers from the Pandemonium when $(C_{max} \geq \kappa)$ `AND` $(\alpha \leq \lambda)$.

Given a filter of any sort, it is possible now to define some measures that relate to the system's characteristic response, given a particular problem domain.

Let $P(a)$ be the probability that the system will answer a question posed it, then for a particular test set $\mathcal{T}$ of size $N_{\mathcal{T}}$ as,

$$P(a) := \frac{N_a}{N(\mathcal{T})} \tag{11}$$

where N_a is the number of questions answered (a stands for "answered").

Let $P(c|a)$ be the probability the system will answer a question *correctly*, given that it is answering it at all:

$$P(c|a) := \frac{N_{a\&c}}{N_a} \tag{12}$$

where $N_{a\&c}$ is the number of test set questions that are answered correctly.

[1]We consider this important for the mappings we envisage performed in our intended robot system, that will involve self-generated (continuous) coding of outputs.

Figure 8: The filter mechanism for answers from a Pandemonium.

We call $P(c|a)$ the *reliability* of the system. A low reliability implies a system that is not to be trusted, but used preferably in conjunction with other back-up systems that may corroborate the accurate answers from the system and eliminate the inaccurate ones. A high reliability implies a very trustworthy system. Such trustworthy systems may, however, due to their own self-criticism, turn out to have a very restricted area of expertise. That is, the value $P(a)$ may turn out to be very low. The problem of reliability in real-world applications is not a new one, and has been considered in depth in relation to expert systems [20], where the issue of usefulness vs. trustworthyness is relevant.

It becomes clear that in order fairly to estimate the use of any system, both the quantities $P(a)$ and $P(c|a)$ need to be taken account of. A general definition of a measure incorporating the two, the *quality* Q of a system, can be defined:

$$Q_{SM}(\beta) := 2P(a)[(1-\beta)P(c|a) - \beta P(w|a)] \qquad -1 \leq Q_{SM} \leq 1 \tag{13}$$

where $P(w|a) := 1 - P(c|a)$ and β $(0 \leq \beta \leq 1)$ represents the relative importance associated with getting an answer correct and not getting it wrong (w stands for "wrong"). Such relative weights must be determined for the application in question, as in the quantities used in decision theory. Thus, if for every correct answer a score of $+1$ is received, and for every wrong answer a score of -2 is received, and for every question not answered no score is received, it would be equivalent to defining Q_{SM} with $\beta = 2/3$. Equation (13) simplifies to

$$Q_{SM}(\beta) := 2P(a)[P(c|a) - \beta] \qquad -1 \leq Q_{SM} \leq 1, \quad 0 \leq \beta \leq 1. \tag{14}$$

For our experiments we define the quality of a system to be

$$Q_{SM}(0.5) = P(a)(2P(c|a) - 1) \tag{15}$$

so that correctly answering and not incorrectly answering have the same weight.

Other types of quality functions may be defined. For example, *Nestor Applied Systems* (personal communication, Feb. 1992) use the following definition:

$$Q_{Nestor} := P(a) - 10P(w|a) \qquad -10 \leq Q_{Nestor} \leq 1 \tag{16}$$

for their *NestorReader* character and digit recognizer. Le Cun *et al* [10] also considered the question of quality ("accuracy" and rejection), but instead of defining a general quantity, they focused on the rejection rate required when the reliability (performance on the answered questions) was to be at least 99% (i.e. the fraction of patterns that need to be rejected in order to have an error of only 1%).

For the questions to which a system does not provide an answer, the problem lies either in the *ambiguity* (more than one answer realistically possible) or in the genuine *unfamiliarity* of the input (in the Pandemonium case, all the confidences are low). The first case signifies generalization will be of the *interpolation* variety, and the second that it will be more of an *extrapolation* to outer reaches of the workspace. This generalization may be used in other parts of the complete system, provided reasonable weights are associated with them. Pandemonium provides these weights automatically, as the confidence values, regardless of the set of possible output codings. Other systems may only provide such weights when they are inherent in the form of the output, as for example in a digit classification problem. For general output codings there is no obvious method in such systems. In section 11 we will be comparing the various alternatives offered by a single net system and a Pandemonium system, in the digit recognition task.

9 The "dog-paw" test and negative learning

For a system to be considered viable as a real-world, useful, device, it must pass at least one further test. This we call the *dog-paw test.* Imagine you are presented with a fully trained, apparently 100% effective and successful digit recognition system, where you are required merely to manipulate a pen on a pressure-sensitive plate in order to draw a character. The system would then process it, identify it and recode it into a standard machine character. Absent-mindedly you leave this wonderful machine lying around on the floor, and your dog comes by, places, equally unintentionally, its paw upon the pressure-sensitive plate for a second, and walks off. Is it *possible* that the system will reply as if it had recognized a digit?

It would clearly be nonsense to believe that the dog either accidentally or very cleverly actually did imprint a recognizable digit upon the machine. But such a positive response from the machine is unfortunately all too possible. The cause of it is that the system has only normally ever been exposed to *positive* examples, and not negative ones. The task of the machine was always to *distinguish* between classification possibilities, and not to determine whether or not the input question was a member of the super-set of all possible classifications for which it is responsible. It was shown in [11] how the essential elements of a class—its "template" pattern—that the network itself learns, may be very different from what we consider to be a typical exemplar. The template pattern generated by the network depends completely on the contents of the training set: how they are correlated, which pixels are most often on, pixels that are maximally uncorrelated with other classes, and so on. This may *distinguish* an input pattern wonderfully among the possible classes, but it does not distinguish it from a nonsense pattern, like, for example, a paw-print.

In other words, when a pattern completely uncorrelated with any other learned pattern is sent through the system, there is no way of knowing how this pattern is going to be classified, unless the system knows something about negative examples.

The problem has been (relatively seldom) mentioned in the neural network literature, and attacked principally through the introduction of a "garbage" or "don't care" node at the output of the network, in order to introduce an additional class. Then sample nonsense patterns are shown during the learning and trained to be mapped to this new class. Another possibility would be to "level-off" (or reduce to nil) the outputs when such patterns are presented, so that the net does not give any sure answers. The second technique will, however, most probably degrade the performance on the former classes. The first will also scale badly, because the network, however large it is, will be a network that is responsible for the positive identification of all possible input patterns. It appears, however, that with a single network one cannot but accept these compromises.

One may always consider, at a higher level, an input filtering system, that decides first whether the input looks *anything like* the classes dealt with, by, for instance, comparing it with class templates. This is however at a level above where we would like to attack the problem. We would like to be able to cope with it to a certain extent right at the level of the Pandemonium, although such higher filtering levels are always possible.

With our Pandemonium system, it is possible to deal with the dog-paw problem in quite an elegant manner. It is not wished adversely to affect the "real" learning, but an extra garbage node may still be introduced, when there are several MINOS modules around. One would envisage a MINOS automatically specialized solely to distinguishing garbage from "digit-like" inputs (or, equivalently, recognizing digit-like inputs!). The learning of digits might not be degraded, because it may be dealt with undisturbed by other MINOS modules. This is tested in section 11.3.

Another way is to perform a type of *negative learning.* The method is, during the learning, to process a nonsense pattern, and to observe if the filter system *would* allow an answer through. If this is the case, then all the Monitors (not the Workers) are trained to reduce their confidences when they see this pattern. So the Monitors perform negative learning. In fact what the Monitors do now, that they did not do before, is to separate the parts of input space handled by their MINOS not only from those handled by other MINOS modules, but also from any other adjacent regions that are handled by no MINOS, and are considered to belong to none of the classes. We expect such limited negative learning to be not so destructive as that caused when an extra node is used, or when the output is levelled. This is also tested in section 11.3.

10 The Monitor sub-module system

It may conceivably happen that the task for the Monitor is inordinately difficult: it is the Worker that essentially chooses the patterns it wishes to learn, and not the Monitor. The Monitor has to adapt to the choice of the Worker. That the Monitor requires only to perform a 2–class mapping does not mean this mapping is necessarily easy (although it will not be harder than the entire complexity of the problem). Our solution to this is to accept the decomposition performed by the Workers, but also to employ it further at the Monitor level, but this time *within the monitors themselves*. Thus we abandon trying to cause alternative task-sharing over the MINOS modules, and allow the Monitor component of a MINOS to employ more than one *sub-monitor* (which may also, incidentally, form part of a future "network pool") in order to solve its own mapping problem.

The Monitor system is not itself a mini-Pandemonium. A sub-monitor is not a "mini-MINOS". We do not allow the modular structure to continue *ad absurdum*: this is the final level. A Monitor part of a MINOS has simply become cleverer, insofar that it, like the Pandemonium system may choose sub-modules from a network pool, and privately employ more than one network from the pool to help. This follows in the spirit of complexity decomposition, and network size restriction, through spreading a task that has not been simplified at the Worker level over more than one Monitor network. The principle itself is quite simple, and is illustrated in figure 9.

The inputs to the Monitor system are, as before, the pattern itself (with the hidden layer activations from the Worker net, depending on the implementation) and the "verdict"—a target of 1 or 0 that indicates whether the Worker has been allocated this exemplar or not. The target is *always* either 1 or 0, and so the nets may have an optimized form. During learning, the control mechanism chooses the output that is largest (equivalent to the more general "shout" of the Workers, and actually practically the same as Selfridge's shouts) and allocates the target of 1 to that sub-monitor. All the others that are taking part[2] learn to map the input to 0. During recall, all the outputs from the sub-modules in use are collected, and the maximum value passed on as the confidence. The sub-modules do not *need* to be like MINOS modules themselves, because there are only ever two classes that need to be separated within the Monitor.

For the digit application domain we did not need more than one sub-monitor. For the parity

[2]One envisages here, as in section 7, some mechanism for switching available modules on and off, so that not all available ones are employed right at the beginning. In our experiments we avoided this issue for the moment, by restricting the number of available sub-modules (see the parity experiments).

Figure 9: The Monitor sub-network system.

problem the Monitor system was employed with varying number of sub-monitors; the results are described in section 12.2. The reason sub-monitors are required is because parity is basically a 2–class mapping problem. The most likely way that a Pandemonium system would split up a two–class mapping problem is, as equation (9) is always going to be satisfied, the GM way. Such a scenario would of course result in no problem simplification occurring.

11 Handwritten digits learning

11.1 Description of the problem domain

The problem chosen was the classification of handwritten digits produced by three different people. The task described here is fairly lightweight, in that the actual task of learning all the mappings can be performed by a single network with 10 hidden units, without any problem. The possible generalization performance was also limited by the number of exemplars available. Such things were however not of utmost importance at this stage, since the objective is to study the relative performances of the single net and a Pandemonium. Further experiments are currently being performed, with larger digits (pixel size 16x24) and with 3000 original exemplars [7]), and these results will be published elsewhere.

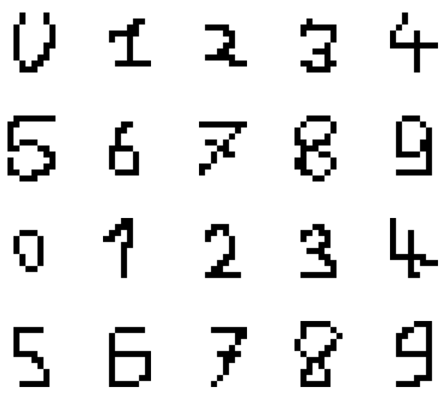

Figure 10: Examples of the digits to be classified.

280 digits were written by 2 people (14 exemplars of the 10 digit classes each), and 250 were written by the third person (a further 25 exemplars for each class). They were generated using the Sun–3[†] icon editor, and as a result were somewhat shakier than digits that would be produced free-hand using a pen. Nevertheless they could, in their final coded form of size 8×11 (88 dimensional input space), all be correctly identified by human observers. A random selection of 490 exemplars were used as the training set, and the remaining 140 exemplars served as the test set.

In order to compare the performance and potential of the Pandemonium network system, a single net system was used as control. The minimal aim is to achieve at least comparable performance from the Pandemonium systems, although they divide up the training set. With this it should be shown how significant information is not lost through forgoing a total connectivity.

The complexity of the problem domain can be demonstrated using a simple nearest-neighbour classifying system. In this procedure, a pattern in the test set is compared with *each* of the patterns in the training set, and classified as that class to which it is nearest, using the direction cosine between patterns as the metric (kNN with $k = 1$). This method involves storing *every* training pattern; it is not feasible as a real-world system since it scales very badly with the training set size it receives. We use it in order to compare the best possible estimates that can be made on a test set, given the training set, regardless of the time taken to produce an answer. The answer is an interpolation within the training set. With this method we obtained the result that 87.1% of the test set could be correctly identified, given the information in the training set, when all are answered. That is, for the classifier, $Q_{SM}(0.5) = 0.742$. Examples of the patterns are shown in figure 10.

A single net can solve this problem with a hidden layer containing only 10 hidden units. That is,

[†]Sun–3 is a trademark of Sun Microsystems Inc.

it can learn to reproduce the training set mappings. We compare two single net systems: one of 10–hidden nodes, and one of 5–hidden nodes. The second is to demonstrate the behaviour of the single net system under "complexity stress"—when the training set cannot be fully learned. A system with 5 hidden nodes per network in each MINOS is taken as the standard size for the Pandemonium. A system with 3 hidden nodes for each network was also used for the Pandemonium "complexity stress" examination.

Details of the experiments are as follows. Step-size in the back-propagation learning was set to the value 0.1, and momentum was switched off (= 0). The value of the step-size parameter makes practically no difference to results. The only effect it can really have in our method of one-shot learning is to reduce the number of repetitions required in order to learn a single pattern. One-shot learning was also used for the single-net experiments. The training set was presented to the nets in the *same order* (for this application) each epoch. An epoch is defined as the period in which the system sees the whole training set once. All runs were halted after 33 epochs. All nets in the Pandemonium were initialized differently, and all nets were initialized after the method described in [30]. Tolerance for all learning was set at $tol = 0.2$. The Pandemonium systems consisted of 3 MINOS modules.

A "clean-peak" filter was also used at the output of the net systems. This is commonly used (in varying forms) in digit and character recognition experiments [10, 6]. It disallowed answers that do not consist of a clear peak at just one node. Our criterion was that the peak was to be ≥ 0.7, AND the other positions ≤ 0.3. This is reasonable, when one considers the learn tolerance was set at 0.2. In comparison with the other commonly used criteria, however, it is somewhat strict. For the Pandemonium filter the confidence threshold was set at $\kappa = 0.6$ and the ambiguity threshold at $\lambda = 0.4$.

11.2 Performance comparison

Typical runs for the systems are shown in figure 11 with the quality measure Q_{SM} for the training set and the test set, throughout the 33 epochs of learning. For the training set the quality increases most quickly for the single net of 10 hidden units, although the Pandemonium 5 hidden unit system also has a fast curve. Surprisingly the smaller Pandemonium system is always significantly better than the single net 5 hidden unit system. On the test set the Pandemonium system rapidly overtakes the corresponding single net system.

It is best to compare learning performance of these systems by comparing the value of Q_{SM} and Q_{Nestor} at two values: when the filters are used and when $P(a) = 1$ (all questions are answered). In the following table best values for Q_{SM} are shown. This was decided instead of using averages with error values, because we wish to compare the best solutions that could be obtained, for each system size. On average 10 runs were made for each system to get these values.

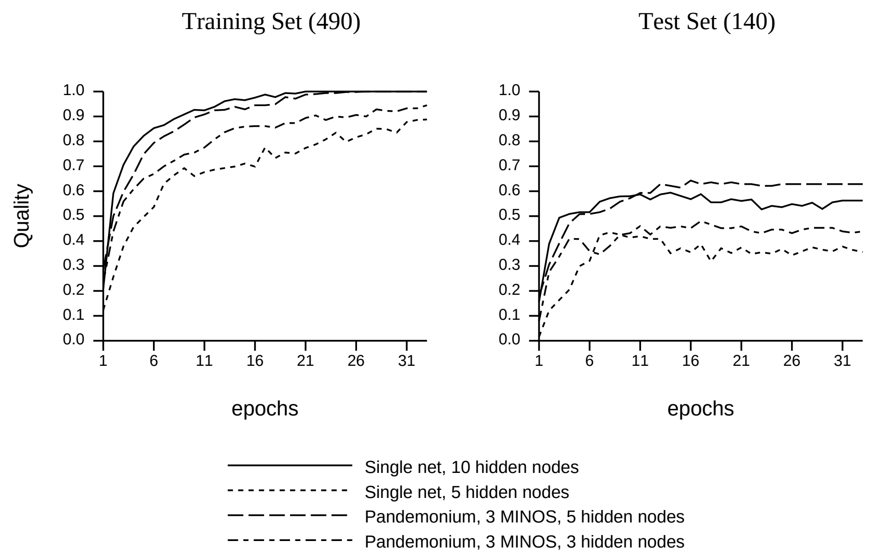

Figure 11: Comparison of the 4 types of system on the digit problem

System		with filters		$P(a) = 1$	
		Q_{SM}	Q_{Nestor}	Q_{SM}	Q_{Nestor}
Single-net 10 hidden	train	1.0	1.0	1.0	1.0
	test	0.686	0.434	0.758	-0.21
Pandemonium 5 hidden	train	1.0	1.0	1.0	1.0
	test	0.721	0.721	0.714	-0.43
Single-net 5 hidden	train	0.893	0.836	0.964	0.82
	test	0.443	-1.23	0.442	-1.79
Pandemonium 3 hidden	train	0.945	0.945	0.996	0.98
	test	0.464	-0.029	0.486	-1.57

One notices that the quality of the Pandemonium 5–hidden unit system is significantly better than that of the single net with 10 hidden units, when it is *not* required that all questions are answered (first column). Then the reliability is very high (for the Pandemonium 100%). The single net has better quality when it is demanded that all questions are answered. The similar case applies when comparing Pandemonium with 3–hidden units with a single net of 5 hidden units, although this time the second test is also better for the Pandemonium system.

11.3 Learning to pass the dog-paw test

The two methods of learning to distinguish garbage patterns from valid digits, described in section 9, were tested using the Pandemonium 3–MINOS system. All the nets had 5 hidden units. The experiments were performed with $ac = 1.0$ and ran for 33 epochs. A random pattern was presented to the system as frequently as a positive training set pattern, during the training of the net. The testing was over another set of 140 random patterns. Every random pattern was randomly generated every time (i.e. no random patterns were stored and presented deliberately more than once to the system). The random patterns themselves were simply uniformly distributed initialized pixels (average 50% on, 50% off).

The first way of learning to pass the dog-paw test involved adding a class "garbage". Thus the output vector of every network is now one element larger, with the extra element representing the new class garbage. Every time a garbage pattern is presented to the system to be learned, the garbage node is trained to be one, otherwise it is trained to be zero. In the Pandemonium *every* Worker network was given an extra node.

For Pandemonium another technique could be used for dealing with the random patterns. The 10 classes at the output of the Worker nets were unchanged, but this time the confidences of the Monitors were reduced (to the value *tol*) every time a garbage pattern appeared. The criterion for performing this Monitor unlearning was that the ambiguity in confidences between the MINOS with the highest shout and the most confident MINOS was *greater* than λ (= 0.4) AND the maximum confidence was *greater* than κ (= 0.6). This technique prevents a garbage pattern from being confidently answered.

In the following table the results using both techniques for the Pandemonium and the garbage node technique for the single net, are compared with those obtained using the standard systems. The systems are compared in their performance on both the training set and the test set. Compared also are the results using our Q–value and *Nestor*'s.

Architecture/Technique		$P(c\|a)$	$P(a)$	Q_{SM}	Q_{Nestor}	Q_{SM} $(P(a) = 1)$	garbage
Pandemonium/	train	1.00	1.00	1.00	1.00	1.00	–
Garbage node	test	0.950	0.707	0.636	0.202	0.586	100.0
Pandemonium/	train	1.00	0.994	0.994	0.994	1.00	–
Monitor unlearning	test	0.989	0.629	0.614	0.515	0.671	0.0
Pandemonium/	train	1.00	1.00	1.00	1.00	1.00	–
Standard	test	1.00	0.721	0.721	0.721	0.714	32.9
Single net/	train	1.00	0.984	0.984	0.984	0.968	–
Garbage node	test	0.935	0.657	0.571	0.005	0.742	100.0
Single net/	train	1.00	1.00	1.00	1.00	1.00	–
Standard	test	0.971	0.729	0.686	0.434	0.758	26.0

The best performance was chosen from 10 differently initialized runs, determined by that run producing the highest Q_{SM} for the test set. The advantage in having the Pandemonium system for dealing with patterns not belonging to any of the classes being learned, is that it can be done without disturbing the actual classifications themselves. Thus in the above table it can be seen that learning random patterns through Monitor unlearning does not result in greater numbers of erroneous classifications (i.e., in the reliability significantly dropping), in training and test sets, merely in a more critical filter. So less questions are answered: the system reacts in a robust way to new learning conditions. The same is true, although to a lesser extent, when a garbage node is used in the Pandemonium system. Evidently the monitor unlearning technique is more reliable than the garbage node technique. On the other hand, the reliability of the single net system drops noticeably when a garbage node is used. In both systems the quality when no filter is used drops when a garbage node is used. It is also to be noticed that the performance of the system *for the training set* is not altered when a Pandemonium Monitor unlearning system is used, whereas for the garbage systems the Pandemonium system is very slightly reduced in quality, while the single net has a lower quality.

All the systems that learned garbage patterns in the training set responded perfectly to garbage when it was shown as part of the test set. This can be seen in the last column of the above table. For the garbage node method the garbage patterns were always correctly identified as such. For the monitor unlearning method the value 0.0% indicates that garbage patterns were never classified by the system: that is, they were rejected. The standard systems, however, identified garbage as digits. That is, positive answers were allowed from the clean-peaks filter and ambiguity filter (for Pandemonium) for 26% of the garbage in the single net, and 33% of the garbage in the Pandemonium. The reason the standard Pandemonium classifies more garbage is the following. In order for a random pattern to be classified as positive in the Pandemonium it needs to land in the area of input space that will light up (only) one of the 3 Monitor nodes. Once it has broken through this net it has to light up (only) one of the 10 Worker nodes. This second requirement is relatively easy, because the Worker just separates the pattern into one of the classes it is responsible for. On the other hand, in order for a random pattern to be classified in the single net, it has to light up one of ten nodes. This also explains why it is a good thing to do Monitor unlearning to filter out the random patterns. The Workers are not disturbed, and the only effect is to make the whole system answer more critically.

11.4 Alternative answers

The information given by the MINOS modules can be used in many ways at higher decision levels. To give some idea of the possibilities of such "community of experts" processing, we consider the alternative answers produced by a Pandemonium trained over 33 epochs on the digit problem. We already know that a certain number of the test set may be answered confidently, and some more are correct when the highest confidence is used without reliability control. What of the others? Is it possible to provide enough information to a possible higher level so that it may, as one of the alternatives, receive the correct answer?

Details of this comparison are as follows. In the 3–MINOS Pandemonium systems a maximum number of 3 answers could be offered (one from each MINOS). These consisted of the maximum class (leaving out the clean-peak procedure now) showing in each of the 3 Worker nets. For the single net up to 3 answers could also be offered, and consisted of the classes with the top 3 activations. Additionally, all answers had to have an activation *larger than* the learning tolerance ($tol = 0.2$). The results are shown in the following table, in terms of the percentage of test patterns that would be correctly identified in the top 1, 2, or 3 answers offered by the system. Also shown are the results when *any* number of alternatives, all with activation larger than *tol*, are considered.

System	# Answers offered			
	1	2	3	> 3
Single, 10 hidden	87.9%	91.4%	92.9%	94.3%
Pandemonium, 5 hidden	85.7%	92.9%	95.0%	97.9%
Single, 5 hidden	72.1%	81.4%	81.4%	81.4%
Pandemonium, 3 hidden	74.3%	87.1%	93.6%	97.1%
1–Nearest Neighbour	87.1%	94.3%	97.9%	–

Compared also were the results using the nearest-neighbour classifier. For this method, and the single net with 10 hidden units and the Pandemonium with 5 hidden units, the greatest improvement is when one uses the top two answers, with only a couple per cent more correct guesses obtained using the third answer. For the larger networks the Pandemonium system performs significantly better than the single network, when more than one answer is allowed. This is just what we would like to see: for questionable patterns the suggestions given by the various experts can lead to correct identification, given that the higher level has other forms of classification. This proves that the interconnectedness striven for in the PDP networks to provide maximal generalization need not be lost in the decomposed learning of a Pandemonium system. Indeed, in the digit example, the alternative answers provided by the components of the Pandemonium are not only *stronger* (higher activation) but more favourable than those from the single networks.

The nearest-neighbour classifier performs better than all the others, when alternative answers are allowed. There is however no way of filtering the response of such a system in order to make it reliable. It is an example of a "number-crunching" way requiring high memory capability, of solving the digit problem. In comparing with the network systems, however, it can be noticed that they are not so far behind in performance, when no filter is used. Indeed, in its 3 answers score the Pandemonium with 5 hidden nodes comes within reach of the maximum score of the nearest-neighbour classifier. The result of the final column shows that the information is indeed latent somewhere in the system.

The performance of the 3 hidden unit Pandemonium is quite revealing. It indicates that the Worker networks are indeed already very good at their classification, but it is the fault of the Monitor learning that holds back the system. As soon as Monitors are ignored, the alternative answers provided by the experts are shown to be accurate. Another way of employing alternative answers from a Pandemonium is discussed in section 14.

12 The parity problem: slicing complexity through sharing

In the case of the parity problem, the output consists of only one node, and hence $N_c = 2$. Therefore, as mentioned in section 7, there is no point is splitting up the problem over more than two MINOS modules (when the standard Pandemonium decomposition method is used). The Monitors will however require the submodule systems described in section 10.

In order to compare runs it was decided to use a full training set (all 2^n patterns) every epoch in the results presented below. An additional refinement was to permute the set of (complete) patterns every epoch, such that they had a new order every presentation. In the Pandemonium system the filter was the same as in the digit experiment, with the "cleanness" detector not relevant. An answer was considered by the tester to be correct when it was the right side of either 0.48 or 0.52. This was not particularly relevant for the Pandemonium system, because Workers always give strongly boolean answers.

The statistics (for runs not resulting in complete solutions) were taken in two sets: the first set was the last 10 epochs of the individual run; the second set was the averages from the first set averaged over the number of runs, differing only in initial weights. The number of runs used was generally of order 6–10.

12.1 Solving parity in a lateral modular way

For one–output node problems of the type: $N_{\text{class} = +1} + N_{\text{class} = -1} = 2^n$ (i.e. the input pattern space comprises solely patterns belonging to one of the two possible classes) a special case may be defined for a Pandemonium system. It follows logically from the Monitor sub-monitor system described in section 10, and involves removing the Worker networks from the MINOS modules altogether. The Workers will no longer be necessary, when we allow that the whole Pandemonium becomes, say, a "class = 1 detector". Thus, when the confidence of the system is greater than 0.5, it will claim the class is +1, and when less, it will claim that it is −1. So our set of MINOS modules, with their missing Workers, look now very much like a Monitor sub-system, and will indeed be controlled in this way. A MINOS will be chosen to learn a class +1 pattern on the basis of how high its confidence is, and all will be trained to produce zero confidence when the target for the pattern is −1. The confidence of the entire system is the value of the highest confidence, C_{max}, when C_{max} is greater than 0.5, and $1 - C_{max}$ when less than 0.5. The value 0.5 determines whether the answer is class +1 (> 0.5) or class −1 (< 0.5). It is also possible to define an ambiguity:

$$\alpha = \begin{cases} 2(1 - C_{max}) & \text{when class} = +1 \\ 2C_{max} & \text{when class} = -1 \end{cases} \tag{17}$$

We use this system to prove the point that a decomposition over lateral nets is also capable of solving complex problems with, when the total number of hidden units in the *system* are the same as the minimum number of hidden units required by a single net to solve the problem. As would be expected, however, the larger the problem becomes, the less probable it is that the system will find the particular decomposition leading to the minimal solution.

With this method the following parity problems were solved: (I = # inputs, H = # hidden units—all submonitors have the same number—S = # submonitors, limiting values of H, for $S = 2, 3$ only are shown):

I	2	2	3	3	4	4	5	5	6	6
H	1	1	2	2	2	2	3	2	3	3
S	2	3	2	3	2	3	2	3	2	3

These are the minimal possible solutions that can be found by systems of the given size, when the restriction of equal number of hidden units per net is made. This can be simply shown to be the case when one appreciates that the the minimal hyperplane configurations for the parity problem consist of n non-intersecting hyperplanes. Hyperplane i ($i \in \{1, 2, 3, \ldots, n\}$) separates $\binom{n}{i-1}$ patterns of one class from $\binom{n}{i}$ patterns of the other class [33]. When $S = 2$ the minimal solution for each sub-module will involve $n/2 + (n \bmod 2)$ hyperplanes each, Then each module will separate in total, $\sum_{j=0}^{\frac{n}{2}-1} \binom{n}{j}$ patterns from the rest. But this value is less than 2^{n-1}, when n is even. This is because the remaining patterns are isolated on the one side by the outermost hyperplane of one submodule, and on the other side by the outermost hyperplane of the other submodule. When $S = 3$ the minimal solutions require $n/3 + (n \bmod 3)$ hyperplanes each. If the sub-modules were allowed to have different number of hyperplanes, it would be theoretically possible to obtain a minimal solution of in total n hyperplanes. The sub-modules must have, for any parity problem other than $n = 2$, at least 2 hyperplanes, in order theoretically to solve the problem.

12.2 Larger problems

As the parity problem to be solved increases in dimension, it becomes more unlikely that the exact solutions for 2 or 3 sub-modules will be found. This is for the same reasons that a single network finds it more improbable to solve higher dimensional parity problems [33]. It is not that it is not *possible* (as shown in [15]), when the minimal number of hidden units are present, but that it is *harder* to find the solution. For the Pandemonium case of the last section it is also more unlikely that the sub-networks arrange themselves in the ideal configuration, as S and/or H scale.

So what is it that is gained from the Pandemonium system? We maintain that solving a problem completely is *not* the aim we have in mind. The complexity limits are intrinsic and we do not claim to be able to remove them—but to deal with them reliably. The Pandemonium system admonishes the use of small networks to deal cleanly with hard problems. But the full

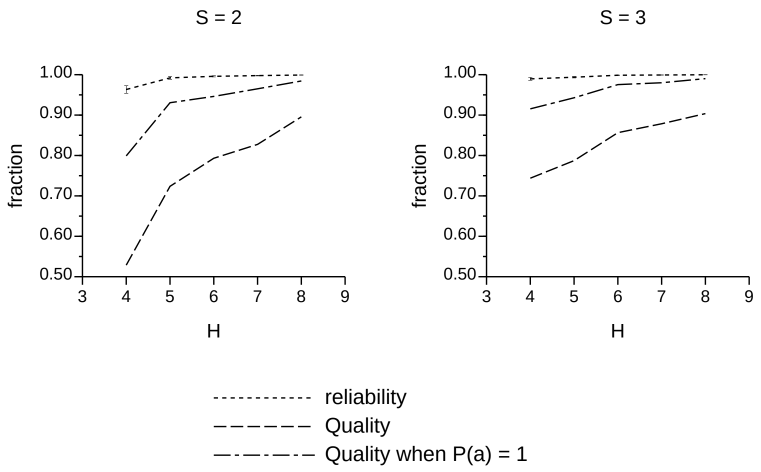

Figure 12: Performance of the Pandemonium system in learning the parity–10 problem, when the number of hidden units used were not enough to solve the problem exactly every time.

Pandemonium system must be used for this, and not the reduced one of the last section. The reason is that Pandemonium, through the independence of its MINOS modules, generates two or more interpretations of the problem space. These interpretations overlap in regions of uncertainty, and reliably model the problem space in non-overlapping regions. The above reduced MINOS is unable to produce reliable answers when it can not solve the problem, and this is because there is only ever one model of the problem being made: we duplicated it for the purposes of the ambiguity estimation (equation (17)).

In the following experments we consider the parity–10 problem. For the Pandemonium the number of hidden units for the Worker nets is unimportant, since they only give the one answer.

In figure 12 the performance for the parity–10 system is shown for various hidden units and sub-monitor sizes 2 and 3. The reliability is always steady at the 0.99–1.00 level, regardless of system limitations (apart from the $S = 2$, $H = 4$ case). The qualities are also very impressive. In general the performance improves in all respects with hidden unit number, most notably in the fraction of patterns accepted through the filter. A very slight improvement may be seen in the curves when 3 sub-modules per Monitor instead of 2 are used.

In figure 13 the same quantities are shown for the parity–10 system with all nets having 5 hidden units, but varying numbers of sub-monitors. Slight improvements in the (already very good) performance are observed with number of submodules used. Note the system cannot on average solve the parity–10 problem with only 5 hidden units, when the usual 2 or 3 sub-modules are used, but with $S = 7$ more than 0.96 is obtained for Q_{SM} when $P(a) = 1$. As usual, when the

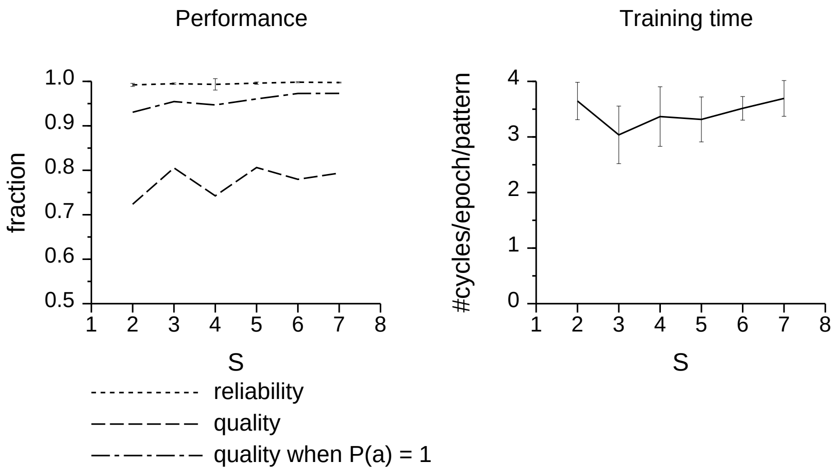

Figure 13: Scaling of the performance (left) of the Pandemonium system in learning the parity–10 problem with 5 hidden units, and varying numbers of submodules. Also shown (right) is the scaling of the average maximum number of back-propagation cycles required per pattern, per epoch.

filter is used, the reliability is constantly high, at around 0.999. The fraction of accepted patterns remains similar for each S, and this is why the quality does not improve. Even when no filtering is allowed, the quality increases very slowly. This indicates that the improvement possible through simple addition of more submodules decreases with sub-module number. This observation relates to the increasing complexity of network cooperation as the number of networks increases (like the N–body problem in physics, it scales very quickly). It is therefore probably normally sufficient to employ 2 or 3 submodules for the Monitor. Note, however, that this need not be the case for the Pandemoniums developed from pools of networks, already specialized in particular ways, where the whole system has some kind of regularity and purpose about it.

Shown also are the averaged (over last 10 epochs) maximum number of learning cycles required per pattern, per epoch. Since the modules are all to run in parallel, the number counted was that of the net requiring the most cycles each time. It is seen that the training time does not increase, within error, regardless of the *width* of the Pandemonium system, when all nets have the same size.

13 Discussion

Our experiments show that it is possible to distribute the complexity of an application over MINOS modules. They serve as benchmarks for more complex applications. But how does the way our system decomposes problems relate to the modularization issues raised in section 2? The complexity reduction is different for the Worker and Monitor components. For the Worker networks the overlap of the patterns dealt with by each sub-module is probably finite (but small) for the *complete pattern set* (equation (5) is satisfied). Indeed it should be in a Pandemonium, for this is what makes possible the set of expert estimates, used for example in the digit problem to produce alternative answers. The degree with which the overlap term in the *learning complexity* (restricting ourselves to the training set only) is reduced is dependent on the directions of specialization chosen by the individual Workers, and cannot generally be predicted. However, it is always the case that the Workers have a much easier task than the Monitors (this is observed from the number of training cycles required to learn each pattern). The overlap terms will be automatically reduced through the decomposition procedure (difficult patterns, characterized by larger values of overlap, will be less likely to be chosen by a Worker). In the Monitor, the total set size, $|\mathcal{A} \cup \mathcal{B}|$, will not have been reduced at all, since the Monitors need to map all the input patterns. However, the actual mapping itself will be different: patterns that were formally nearest-neighbours of different classes, could have been separated in the Worker, and the Monitor requires merely to separate these patterns from all the others that its Worker is not responsible for. It is not possible to prove theoretically that the maximum Monitor complexity is less than M_c^{sys}/N_M, as hypothesized in equation (3). To demonstrate the complexity decomposition for an exact case, we can consider the parity problem and the system of section 10. Here, the decomposition is carried out in such a way, for the n–even problems, that the complexity for each sub-module is exactly $1/N_M$, where the complexity for the whole problem is 1. As an easy example, consider the $n = 2$ problem (XOR). The system assigns, say pattern (0,0) as class = 1, to sub-module 1, and all the others as class = 0, to sub-module 1. Now, the set $\mathcal{A}$ has only one member, thus the first term in equation (1) will be 1/2. The second term will be $2/2 = 1$, since all the nearest-neighbours of set $\mathcal{A}$ are foreign.

The scaling of the entire system, when it is required that a problem is completely solved, will not be significantly better than any other network system. But scaling of the system performance, with respect to quality and reliability, has been shown to be excellent, especially when resources are low (only small nets available). thus the robustness of the system has good scaling behaviour. The extra complexity within the task sharing when more sub-modules are used does not disturb the reliability of the system, but allows only gradual improvement in quality.

The real goal of a Pandemonium system is to enable robust dynamic learning. Since the task allocation and learning procedure itself are completely flexible and dynamic, changing task domains and one-shot learning do not disturb the large-scale parameters of reliability of the system. The quality of answer may be reduced (because less answers are given), but reliability remains high. One envisages staged learning in such systems: first a few "standard" examples of characters might be learned, and gradually harder and more fancy examples may be learned, together with digits and so on, with the old learning not being disturbed, but built upon.

The Pandemonium itself would only form part of a full recognition system, acting as the parallel pattern processor and recognizer part, and working on a structureless input—the pixel representation. It produces either confident answers, or various alternatives that possess a set of weights (confidences) indicating how likely they are. 100% *performance* may not be achieved or expected from such a processor that only considers non-structural information. Other levels are required, that takes the structure in the input, such as line, intersection and stroke recognition and so on. Such levels offer their own suggestions as to the identity of the pattern, that may then be combined with the set of possibilities from the Pandemonium level, and a definite identification should emerge. Further higher levels could be envisaged as providing further "clues" as to a character's identity, such as context information for words, and ultimately semantic information. The point with our reflective architecture is to retain the information gleaned from every level of the processing—even from the most primitive levels, which in our case is the MINOS module. This information may always be used in higher levels, and in this way an efficient feed-back loop may be set up, so that higher levels may interact with lower levels of processing.

14 Towards a technology of brain design

The experiments presented in this paper were used mainly to investigate the basic components of our reflective neural network architecture, and to demonstrate that it indeed works and is efficient and reliable. Our intended application areas are for larger domains, such as automonous system construction, where automatic specialization of sub-modules is necessary, and where without self-observation written into the very heart of the architecture, coherent development would not be possible. A significant future development will involve implementation of the "Network pool" idea, where the networks used in Worker or Monitor of one Pandemonium system, working on a particular task, will be available for use by other Pandemoniums. In this way specialist knowledge may be transported into other problem domains and shared. Some ideas for dealing with the problem of when to use an available module, so that the development does not result in a GM scenario, are:

- Negatively weight the strength of shout from an as yet unused module, so that it will only be drafted in when all the other shouts are indeed much worse.
- Allow the Pandemonium supervisor to possess a list containing id's of all the modules it has used. Then make the probability that it uses another new one in preference to one already used a function of the number of modules that have already been used. The function will of course somehow require to be different depending on the number of categories that need to be mapped—a major disadvantage of such a method.
- Since the GM problem apparently arise from overspecialization at the beginning, of the modules just used, a solution might be to give each new module a certain relaxation time, in which it may be trained to produce not just the one output possibility, but a number of possibilities. When the development of the Worker net has reached a certain level, it is permitted to introduce new modules if necessary.

Our reflective modular neural network architecture is similar in spirit to Minsky's "Society of Mind" [13]. The notion was put forward as a way of releasing the would-be brain designer from the headaches and prohibitiveness of scaling and control complexity. He insisted that the large-scale advances could only be handled through some sort of modular paradigm, that involved employment of many experts, or little "minds", all working together to solve problems at a decomposed level. Missing however was the method and architecture with which to do this. Mühlenbein [15], and once again Minsky [14], remark that the next generation of neural network systems will require much more of a directed *evolution* in their construction, so that specialization and modular society structures can be developed. Our network pool idea, and the detailed Pandemonium mechanism, enable such a desired development to be realized in practice, to produce a "Society of MINOS".

Also in the application area of pattern recognition, a more sophisticated reflective neural network architecture proves to show much promise. In the application described in this paper we have employed only the pixel information of an image, and demanded that the system use only this information to provide an answer. But it need not be so restrictive. Indeed, commercially available systems use several recognition paradigms in order to determine the identity of an image. Imagine that, additionally to the pixel information, there is also available a further preprocessed representation of an image, for example the curvature information. We suggest three ways in which this can be processed in a Pandemonium.

- Combine pixel and curvature information into one input vector, and allow the MINOS modules of the single Pandemonium to specialize. It is plausible then that the modules so specialize such as effectively to separate the components of the input vector: one MINOS may be a particularly good "curvy letter" detector for example, while another reacts to pixels lighting up in a particular region of the image. This idea of combining two types of representation into one vector has also been employed in context-sensitive single neural network experiments. The Pandemonium architecture allows for more promising specialization on the modular, rather than hidden unit, level.

- Use two Pandemoniums. The first processes the pixel information and the second the curvature information. The Pandemoniums will provide different interpretations of the same problem domain. Particularly reliable answers will result when both Pandemoniums agree on the identity of a pattern, and also when they both are unable to provide sure answers (especially with respect to garbage patterns!). Additionally, when one Pandemonium answers reasonably sure, but the other is ambiguous and non-concurring, then a matching alternative answer from the second Pandemonium will also lead to a very strong reliability.

- Use one Pandemonium but MINOS modules of two types: the image processing type and the curvature processing type. This is the same as the last suggestion in all aspects except that only one decision demon is used, and in such a way that the confidence is also a function of the number of concurring confident shouts. Furthermore, the allocation phase would be designed so as to allow more than one MINOS to learn a pattern at once.

15 Conclusion

We have presented our reflective neural network architecture. It is intended to form part of a larger, hierarchical system of modules. Each module itself is not required to be perfect in that it is expert in every area of its input space, but is required to give reliable estimates of how confident it is, for every answer it gives. This is then the most useful information a higher level can have, for it may be either accepted and acted upon, ignored and other modules asked, or partly believed and used in conjunction with other modules and/or future learning.

It was shown how systems of networks offer advantages that one may not receive from single networks, namely in the size of networks that need to be used, the nice behaviour when the system is overloaded, the complexity decomposition, the quality of alternative answers offered in ambiguous situations, non-destructive handling of garbage patterns, and stability under dynamic learning.

This behaviour was achieved through the general concept of self-assessment. The importance of such a concept becomes apparent as soon as larger systems of semi-independent modules are built, and is equivalent to a mini-supervisor for each component of the system, that will allow the control within the system to be hierarchically distributed. Without such delegation of control, the system will scale very quickly to chaotic behaviour.

16 Acknowledgements

The system structure used in the experiments described here is complicated, and development in such a modular direction is not possible without a suitable simulator. Our simulator is called `SESAME` (**S**oftware **E**nvironment for the **S**imulation of **A**daptive **M**odular syst**E**ms) [35], and employs the `C++` object-oriented language. We want particularly to thank Jörg Kindermann, Alexander Linden and Christoph Tietz for their support and for helping with our extensions to the simulator. We have also benefited from numerous discussions within our Adaptive Systems group, and thank the researchers and students involved, especially Sabine Hubrig-Schaumburg for writing the nearest-neighbour classifier. Thanks are due also to Gerd Paaß, for helping explain the quality measure.

References

[1] L. Bochereau, P. Bourgine, and H. E. Priso. Generalist vs. specialist neural networks. Technical report, CEMAGREF, France, 1991.

[2] V. Ruiz de Angulo and C. Torras. Minimally disturbing learning. In A. Prieto, editor, *Proceedings of the IWANN 91*. Springer Verlag, 1991.

[3] S. Eberlein. Developing a decision-making network for a rover. *Neural Computation*, 1(2), 1989.

[4] G.M. Edelman. *Neural Darwinism. The theory of neuronal group selection.* Basic Books, New York, 1987.

[5] R. M. French. Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. Technical Report CRCC–51–1991, Indiana University, Bloomington, Indiana, 1991.

[6] I. Guyon. Applications of neural networks to character recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 5(1-2):353–382, 1991.

[7] S. Hubrig-Schaumburg. Handwritten character recognition using a reflective modular neural network system. Master's thesis, Bonn University, Germany, 1992.

[8] R. A. Jacobs and M. I. Jordan. Adaptive mixtures of local experts. *Neural Computation*, 3(1), 1991.

[9] K. Joe, Y. Mori, and S. Miyake. Construction of a large-scale neural network: simulation of handwritten Japanese character recognition on NCUBE. *Concurrency, Practice and Experience*, 2(2):79–107, June 1990.

[10] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[11] A. Linden and J. Kindermann. Inversion of multilayer nets. In *Proceedings of the IJCNN-89*, Washington, 1989. IEEE.

[12] G. F. Marcus, M. Ullman, S. Pinker, M. Hollander, T. J. Rosen, and F. Xu. Overregularization. Technical Report Occasional paper #41, MIT Center for Cognitive Science, November 1990.

[13] M. Minsky. *The Society of Mind.* Simon and Schuster, New York, 1985.

[14] M. Minsky and S. Papert. *Perceptrons.* MIT Press, 1988. See particularly the Epilogue.

[15] H. Mühlenbein. Limitations of multilayer perceptrons – steps towards genetic neural networks. *Parallel Computing*, 14(3):249–260, 1990.

[16] H. Mühlenbein and J. Kindermann. The dynamics of evolution and learning – towards genetic neural networks. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Amsterdam, 1989. Elsevier. Proceedings of the International Conference Connectionism in Perspective, University of Zürich, 10–13 October 1988.

[17] J. M. Murre, R. H. Pfaf, and G. Wolters. CALM networks: A modular approach to supervised and unsupervised learning. In *Proceedings of the IJCNN–89.* IEEE, 1989.

[18] Y. Nishikawa, H. Kita, and A. Kawamura. NN/I: a neural network which divides and learns environments. In *Proceedings of the IJCNN–90*, Washington D.C., 1990. IEEE.

[19] L. Y. Pratt and C. A. Kamm. Improving a phoneme classification neural network through problem decomposition. In *Proceedings of the IJCNN–91.* IEEE, 1991.

[20] T. L. D. Regulinski. On reliability of expert systems. *IEEE Transactions on Reliability*, 40(4):401, 1991. Editorial.

[21] D. L. Reilly, C. Scofield, C. Elbaum, and L. N. Cooper. Learning system archictectures composed of multiple learning modules. In *IEEE First International Conference on Neural Networks*, pages 495–503, 1987.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Nature*, 323(533), 1986.

[23] D. E. Rumelhart and J. L. McClelland, editors. *Parallel Distributed Processing. Explorations in the Microstructure of Cognition*, volume I, II. MIT Press, Cambridge, MA, 1986.

[24] J. H. Schmidhuber. Adaptive confidence and adaptive curiosity. Technical Report FKI-149-91, Institut für Informatik, Technische Universität München, Munich, Germany, 1991.

[25] M. Sekiguchi, S. Nagata, and K. Asakawa. Behaviour control for a mobile robot by multi-hierarchical neural network. In *Proceedings of the IEEE international conference on robotics and automation*, 1989.

[26] O. G. Selfridge and U. Neisser. Pattern recognition by machine. *Scientific American*, 203(2):60–68, August 1960.

[27] O.G. Selfridge. Pandemonium: a paradigm for learning. In *The Mechanisation of Thought Processes: Proceedings of a Symposium Held at the National Physical Laboratory, November 1958*, pages 511–527, London: HMSO, 1958.

[28] F. J. Śmieja. Evolution of intelligent systems in a changing environment: I. First steps with a structured brain. Technical report, Gesellschaft für Mathematik und Datenverarbeitung, St Augustin, Germany, August 1988.

[29] F. J. Śmieja. *Learning and generalization in feed-forward neural networks.* PhD thesis, Edinburgh University, Department of Physics, September 1989. Unpublished.

[30] F. J. Śmieja. Hyperplane "spin" dynamics, network plasticity and back-propagation learning. Technical report, Gesellschaft für Mathematik und Datenverarbeitung, St Augustin, Germany, November 1991.

[31] F. J. Śmieja. Multiple network systems (MINOS) modules: Task division and module discrimination. In *Proceedings of the 8th AISB conference on Artificial Intelligence, Leeds, 16–19 April, 1991*, 1991.

[32] F. J. Śmieja. Neural network constructive algorithms: Trading generalization for learning efficiency? Technical report, Gesellschaft für Mathematik und Datenverarbeitung, St Augustin, Germany, November 1991.

[33] F. J. Śmieja and H. Mühlenbein. The geometry of multilayer perceptron solutions. *Parallel Computing*, 14:261–275, 1990.

[34] S. Thrun and K. Möller. Active exploration in dynamic environments. In *Proceedings of NIPS-4*, Denver, Colorado, 1992. to appear.

[35] C. Tietz, P. Hendricks, A. Linden, and H. Mühlenbein. Object-oriented simulation of complex neural architectures on parallel computers. In T. Kohonen, editor, *Proceedings of COGNITIVA 90*, pages 387–400, Paris, 1990. AFCET.

[36] A. Waibel. Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1, 1989.