The Pandemonium System of Reflective Agents *

Frank Śmieja

October 6, 1993

Abstract

The Pandemonium system of reflective MINOS agents solves problems by automatic dynamic modularization of the input space. The agents contain feed-forward neural networks which adapt using the back-propagation algorithm. We demonstrate the performance of Pandemonium on various categories of problems. These include learning continuous functions with discontinuities, separating two spirals, learning the parity function, and optical character recognition. It is shown how strongly the advantages gained from using a modularization technique depend on the nature of the problem. The superiority of the Pandemonium method over a single net on the first two test categories is contrasted with its limited advantages for the second two categories. In the first case the system converges quicker with modularization and is seen to lead to simpler solutions. For the second case the problem is not significantly simplified through flat decomposition of the input space, although convergence is still quicker.

1 Introduction

In recent years attention has been turning ever more to the concept of modular systems and hierarchies of models. The idea is to approach processing flexibility through combining diverse specializations of the comprising modules (agents). Such diversity complements the inherent generality in the modeling process of the agents themselves, Although theory implies [1, 2, 3] that a neural network is always *capable* of *representing* the solution. It does not show how the solution may be found, whether it is the best solution, how long it may take to find, and how much data need to be seen before it is found. These are fundamental limitations [4] to a "universal modeling" philosophy.

Modular systems are in no respect a new idea. Their usefulness has long been recognized and indeed in most practical disciplines like engineering, electronics and chemistry, it goes without saying that they are vital to the successful running and understanding of the complex system that is constructed. Whether or not a universal modeler is a viable prospect, the practical application of neural networks and other approximation methodologies requires a

*This work is part of the GMD Adaptive Systems REFLEX project, supported by BMFT grant number 01IN111A/4.

modular perspective. Numerous suggestions have been made in the literature regarding modular systems of neural networks, examples are to be found in the citations [5]–[18]. In [19] these methods of modularization are briefly compared and contrasted. Two basic types of modular systems may be identified: those that divide the input space among modules and those that consist of modules each of which learns the entire set of examples. The latter type is also referred to as a *team architecture* [20]. The Pandemonium system is designed to carry out the former modularization technique. A fundamental problem to be faced by such modular systems is known as the *decomposition–recomposition problem* [19, 7]: how is the input space to be (automatically) divided and then recombined after processing through the agents? This turns out, as do so many aspects of data-driven information processing [20, 21] to be problem dependent. As a result three different methods of dynamic decomposition are developed for the Pandemonium architecture.

The paper is organized as follows. In section 2 the Pandemonium system is described, in section 3 it is applied to problems involving smooth functions with discontinuities, and in section 4 to the boolean problems of two-spirals and parity–10. Finally, in section 5 the performance of a Pandemonium system on optical character recognition (OCR) problems is discussed with reference to previous experimental results.

2 The Pandemonium system

The Pandemonium architecture was originally proposed in 1958 by Selfridge [22]. The idea is to divide and conquer a complex problem domain, through use of a number of specialized agents working in parallel. All these agents, or *daemons*, process the same signal in parallel, and each provides a possible answer. The daemon 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 daemons, each of which is specialized on recognizing a particular letter. Each daemon 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 what types of filters each daemon used. A "decision daemon" at the top of the hierarchy had the job of choosing the daemon that shouted the loudest, and passing on his output. After learning a scenario depicted in figure 1 results. The first daemon shouts the loudest for the input shown, conveying its claim that it contains a lot of "A–ness". The other daemons may see aspects of their particular letter specialities in the input, but do not shout as loud as the first daemon, which sees more likeliness with "A". Thus the decision daemon chooses the A–daemon.

It is clear that we can translate the daemons used by Selfridge into our present-day notion of "grandmother cells" (GC), and indeed the entire Pandemonium system can be seen as a GC neural network [23]. We take the general idea employed in Pandemonium, and extend it to specifying a modular system of neural networks. Each of the daemons in figure 1 is replaced by an adaptive module, and the system is run using the divide-and-conquer principle [18, 19]. Each module will become specialized to a greater or lesser degree on a particular aspect of the task.

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

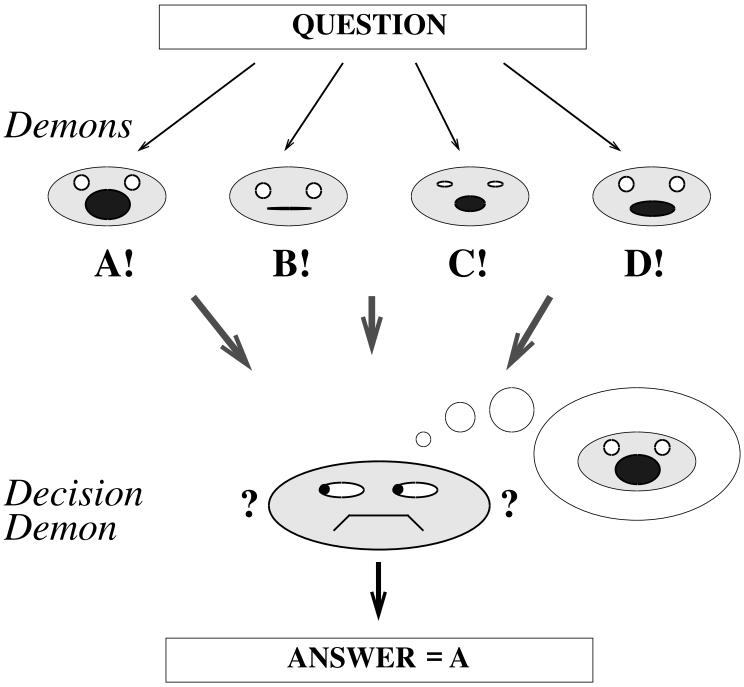

Figure 1: Selfridge's Pandemonium

strategy. However, Selfridge's division resulted in *local* knowledge distribution (GC), or in his later applications [24] distributions of a number of non-adaptive filters. Our method partly *distributes the knowledge*, in the adaptive modules themselves, through the coarser-grained nature of the processing elements.

In order that it be possible to use such coarse-grained modules in a Pandemonium architecture, a necessary requirement is that they are *reflective* [19, 20]. This means they have the property illustrated in Figure 2.

Reflection is the method by which an agent observes its own behavior and produces information relating to the quality of its output. The agent's task is to associate a set of x with corresponding y, with its internal function $f(x)$. The tuples $\{(x, y)\}$ come from the teacher and make up the learning set $\mathcal{L}$. After learning $\mathcal{L}$ the agent is to provide a y for a previously unseen x. Self-observation uses the $\{(x, y)\}$ information to generate for every x an estimate that the resulting y will be correct. With this extra reflective element the agent is transformed into a reflective agent, or *reagent*. In Figure 2 the information produced by reflection has been denoted by c. It is called **confidence** and is defined in the following way.

Def 1 (confidence) *The confidence function $c_(x)$, $0 \leq c_(x) \leq 1$, is a model of the probability that the prediction $f(x)$ is correct for an input x:*

$$c_(x) \models P(f(x) = y). \tag{1}$$

The function $c(x)$ attempts to establish the zones of competence of the agent in the

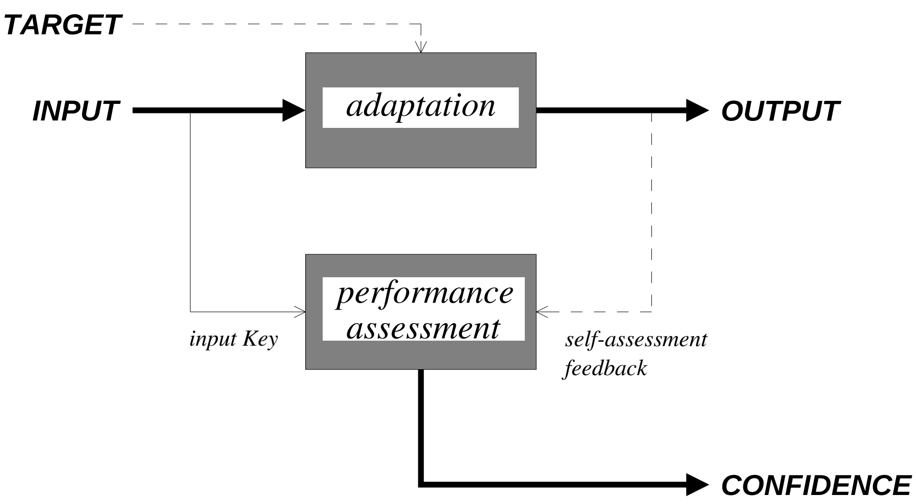

Figure 2: A reflective adaptive agent: "reagent"

x–space. The function is adaptable and is fitted to the learning set data $\mathcal{L}$. Reflection is discussed in more detail in [20]; we concentrate in this paper on the results obtained using reflection and the Pandemonium system for solving two particular problem domains.

The modules in our Pandemonium system are called MINOS, and they generate confidence in ways depending on the problem type. In this paper we focus on three problem types: continuous problems, discrete (boolean) problems and discrete (classification) problems. The latter was studied extensively with Pandemonium in [19, 20, 25], and we will be discussing the results of those experiments in this paper. The main strength of Pandemonium is shown to lie in the recognition of large-scale structure and in dynamic breakdown of a problem with non-overlapping regions.

A MINOS reagent is shown schematically in Figure 3. It is constructed from two agent modules, called the Worker and Monitor. The Worker learns the mapping $x \mapsto y$ and the Monitor's job is to generate the confidence $c(x)$. The Worker is a feed-forward neural network, trained using the back-propagation learning procedure [26].

A Pandemonium system has the following structure. A number of MINOS modules are chosen to comprise a system with a common task. They are not connected to one another, and can be trained and run independently. Any MINOS module may also form part of another collection of MINOS modules to form a further Pandemonium concerned with performing another task. In this way generic functions that are specialized on by a MINOS in one Pandemonium may be transfered to another (later) Pandemonium.

The task itself defines the Pandemonium, in terms of input and output coding, although the number of constituent MINOS modules may be altered dynamically (demonstrated in section 4.1). It is Pandemonium's job to allocate the examples to be specialized on by the individual MINOS modules—the decomposition problem, and to choose the appropriate specialist to provide an answer—the recombination problem. The decomposition can be carried out in various ways, and will be described in the following sections. Recombination

Figure 3: The MINOS reagent. The Worker neural network learns the function $f(x)$ for those (x, y) tuples it has been allocated. The Monitor neural network notes those x learned by the Worker, dynamically building up a confidence function.

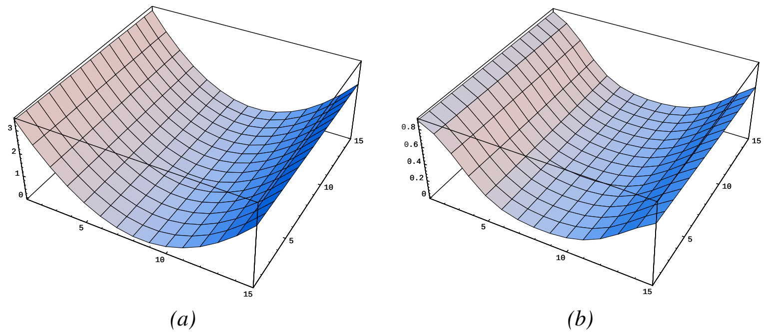

Figure 4: A continuous function (a) as learned by a neural network (b).

is always carried out by choosing the MINOS with the highest confidence for the input pattern in question.

3 Functions with discontinuities

For smooth continuous functions the standard neural network or a simple linear interpolation procedure will generally produce good results. An example for a neural network is shown in Figure 4.

For functions that are mostly continuous but have a number of discontinuities a single neural network does not produce good results (see Figures 5b and 6b). The Pandemonium system can help with such cases. For continuous functions the Monitor is a neural network with one output unit. It simply learns to map the x allocated to be learned by its corresponding Worker to 1, and all other x handled by the Pandemonium to 0. That Worker with the lowest current error for the current examplar is allocated it to learn. The error is calculated using the current state of the Worker. For the parity case we did not use the current state, but a history state that was periodically updated (see section 4.2). This was to prevent the development of a grandmother cell effect: one MINOS learns to map to 1 and another to 0.

It is possible with back-propagation to update the networks either in "batch mode" or in "online mode". Batch mode stores a number of weight-changes (normally those for the entire learning set), and updates the networks with the accumulated changes. Online mode updates after each input pattern is seen.

We tried out the Pandemonium system on two functions that were mostly continuous but had a few discontinuities. Both functions involved learning the mapping $\Re^2 \rightarrow \Re^1$. Function 1 is shown in Figure 5a. This function is characterized by being mostly smooth and non-linear with a piece cut out. The neural network solution (100 hidden units) is shown in Figure 5b. It typically attempts to attain a best continuous approximation over the entire input space. The Pandemonium solution (3 MINOS components with 10 hidden units in each neural network), in Figure 5c and 5d, recognizes the discontinuities, and does not let them disturb the otherwise good neural network approximation to the continuous regions, through automatically sharing the space out among the MINOS modules. Figure 5c is the "batch" solution—where all the 400 patterns are presented before the weights are updated with the accumulated back-propagation changes, and Figure 5d is the "incremental" solution, using online back-propagation learning. The incremental solution is more desirable for dynamic open problems [27], since the agent adapts after each pattern is seen. A characteristic difference in solution (the generalization) between the two methods is the sharper edges in the incremental method, that separate the MINOS modules' contributions. The division of the input space for the solution of Figure 5d is shown in Figure 5f. The actual 400 learning set points for this problem are displayed in Figure 5e. The specialization discovered involved one MINOS (black region) being mostly responsible for the large non-linear part, one (gray region) responsible for the flat base, and the other MINOS (white region) for the remaining non-linear part. The decomposition was achieved completely automatically.

Function 2 is shown in Figure 6a. This function is characterized by being mostly flat and linear with a smaller non-linear piece cut out. The neural network solution (100 hidden units) is shown in Figure 6b. Once again the result is to obtain a best continuous approximation over the entire input space. The Pandemonium solution (3 MINOS components with 10 hidden units in each neural network), in Figure 6c and 6d, again recognizes the discontinuities, producing in both cases (batch is Figure 6c, incremental Figure 6d) significantly better generalizations. The difference in solution (the generalization) between the two methods is again the sharper edges in the incremental method, that separate the MINOS modules' contributions. The division of the input space for the solution of 6d is shown in Figure 5f. The actual 400 learning set points for this problem are displayed in Figure 6e. The specialization discovered involved one MINOS (black region) being responsible for the horizontal hill, one (white region) responsible for the vertical hill and some of the horizontal hill, and the remaining one (gray region) the level floor. The Monitors learnt the almost discontinuous sections that separated the features. This decomposition was likewise achieved

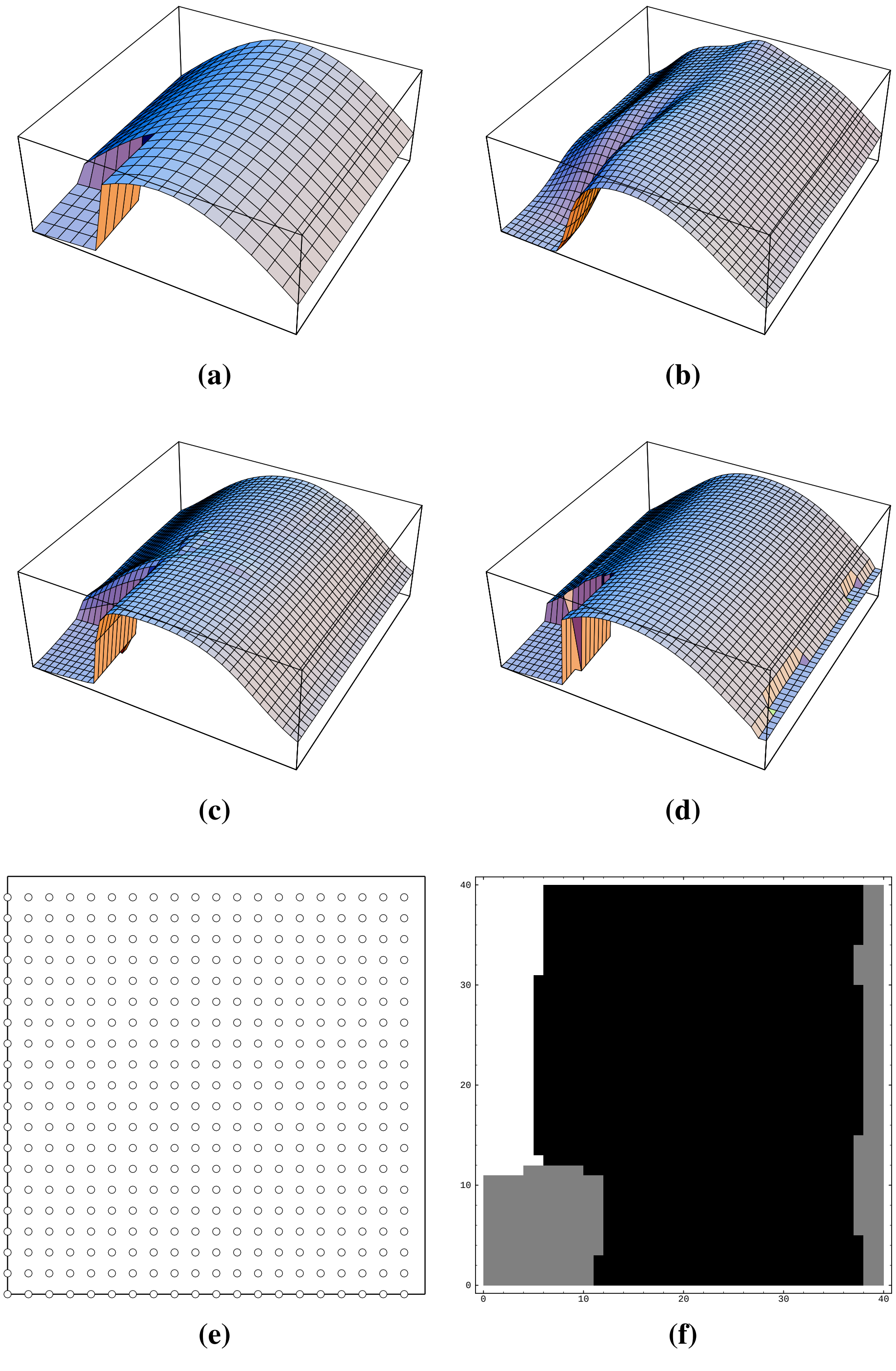

Figure 5: Function with discontinuities 1. (a): original function; (b): neural network solution; (c) and (d): Pandemonium solutions; (e) learning set; (f) Pandemonium decomposition for (d).

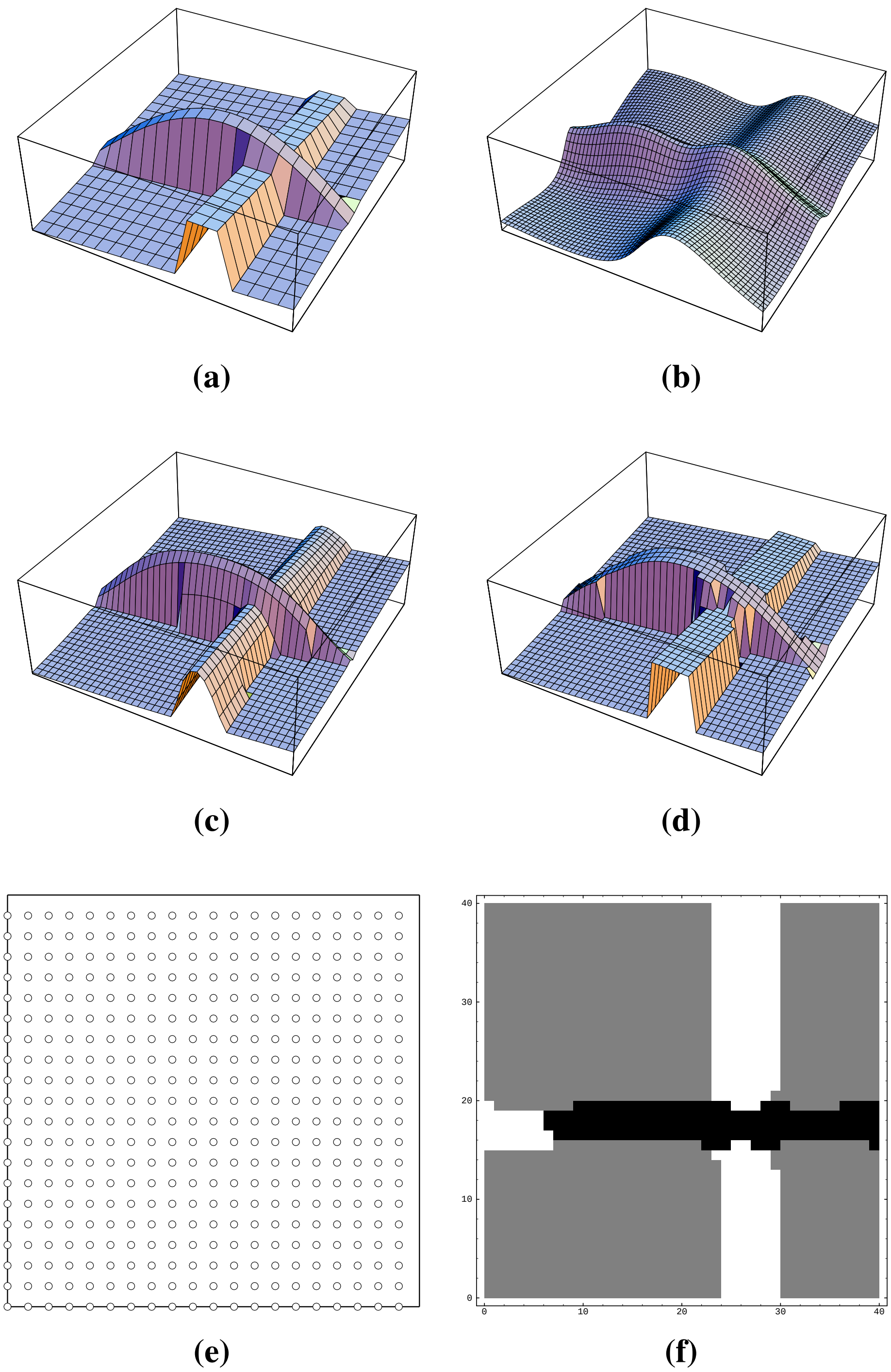

Figure 6: Function with discontinuities 2. (a): original function; (b): neural network solution; (c) and (d): Pandemonium solutions; (e) learning set; (f) Pandemonium decomposition for (d).

completely automatically.

In both case the learning set was randomly permuted each time the entire set was shown, so that the order in which the patterns were seen did not remain static. The 1600 test points chosen were disjunct from the 400 learning points and had a similar grid form. The results are after 4000 learning set iterations. For the incremental cases, the Pandemonium system required nearly 4 times less CPU time in order to complete the 4000 iterations than did the single network (Pandemonium $\approx$ 1.5s per presentation, single net $\approx$ 5.5s per presentation).

The standard back-propagation network smooths the function, rounding off all its discontinuities. This result does not improve with addition of hidden units. On the other hand, the Pandemonium system produces an excellent reproduction of both functions with 3 component MINOS modules, each with a tenfold reduction in number of hidden units.

4 Boolean functions

We describe two methods for decomposition in a discrete problem space. They are both error-based, dynamic, and add MINOS modules constructively to build the Pandemonium system.

The first method splits the (2-D) input space explicitly with metrics, and the second splits a general multidimensional input space in a similar way to that in the last section, but the decomposition occurs based on old Worker weight information. This represents a compromise between incremental learning and batch decomposition.

4.1 The two-spirals problem

The two-spirals problem is defined in [28] by the 192 points making up the learning set. This is shown graphically in Figure 7a. The black points are to be mapped to 1 and the white points to 0. For testing 4900 points are chosen regularly from the space and the output of the system rounded to 0 if the output lies in $[0, 0.4]$, to 1 if the output lies in $[0.6, 1.0]$, and to 0.5 if it lies in $(0.4, 0.6)$. The single net using back-propagation learning was allowed to iterate for 4000 presentations of the learning set to produce the result shown in Figure 7b. It is likely that, given more training time and more hidden units and more refinements to the back-propagation algorithm, the solution could be reached. More useful, however, is a robust method that can quickly reach a solution. The obvious way of simplifying the spiral problem so that it can be quickly solved is to decompose the input space so that individual agents do not need to see the interlocking nature of the classes. This is how the Pandemonium system solves the problem. Figure 7d shows the solution found by a Pandemonium system of 6 MINOS modules, each with 10 hidden units in the Worker.

For this problem the MINOS modules have the following form. Monitors are not neural networks, but consist of a single stored pattern, the reference vector r. Relative confidence between MINOS modules is then defined by

$$c(x) = \frac{1}{\|r - x\|}. \tag{2}$$

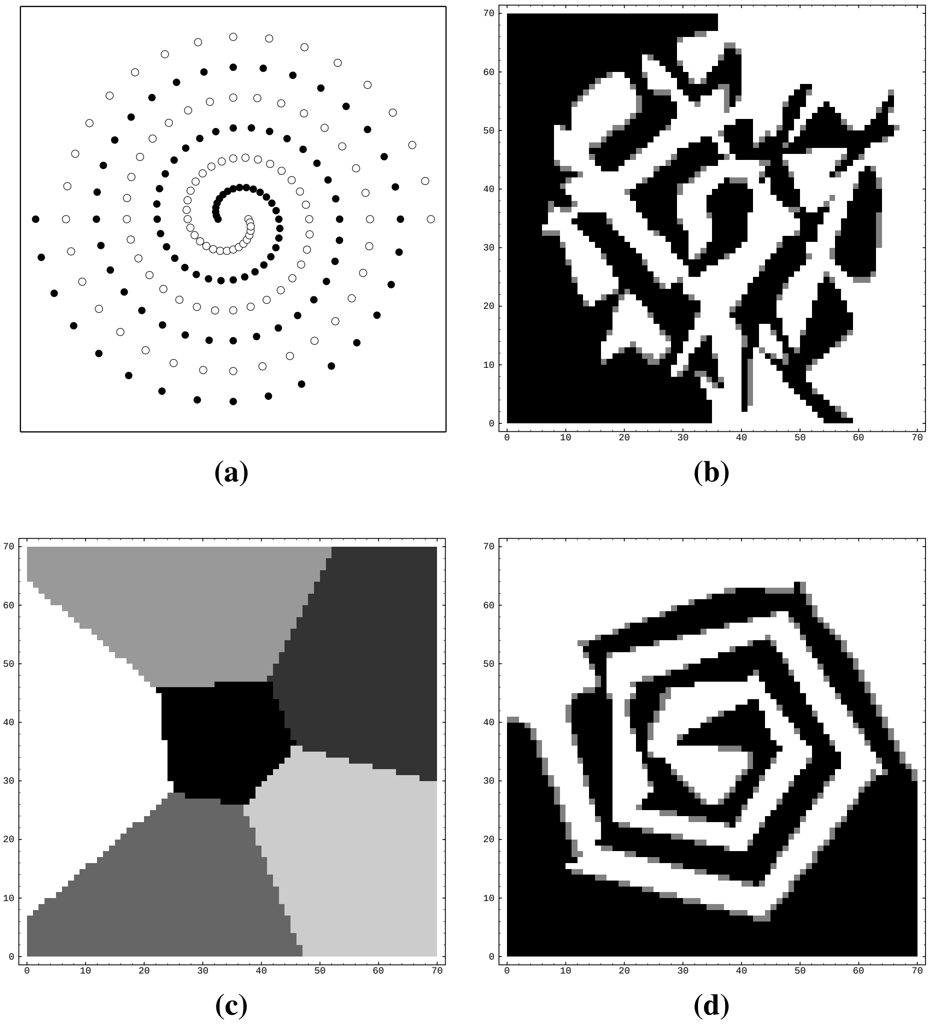

Figure 7: The two spirals problem. (a) shows the learning set points (192 points). One spiral (the black points) are to be mapped to 1 and the other (white points) to 0. In (b) the single neural network (100 hidden units) is shown, and in (d) the dynamic Pandemonium solution (see text). The gray regions represent "undecided" areas, defined to be when the output is in $[0.4, 0.6]$. The sharing of the input space between the 6 MINOS modules (10 hidden units each) is shown in (c).

In other words, that MINOS for which the pattern in question is nearest is the one whose answer is taken. Similarly, in allocating a pattern to be learned, that MINOS is chosen for which the confidence is highest. Thus the modularization consists of convex regions.

The Pandemonium is constructed dynamically. A number of MINOS modules are made available for the problem and the process begins with a single MINOS. The reference vector for each MINOS is adapted each time it receives a new pattern to be learned. It is carried out in the following way, as a function of the number of patterns t the MINOS has learned:

$$r(t) = r(t-1) - \frac{1}{t} \cdot (r(t-1) - \xi(t)). \tag{3}$$

For the first MINOS, the iteration begins with:

$$r(0) = \xi(0), \tag{4}$$

where $\xi(n)$ is the n^{th} pattern to be learned.

Each MINOS has a further reflective component that monitors the initial error of each pattern learned by the Worker. When this ceases on average to decrease, the MINOS signals that the time is ripe for a further addition to the Pandemonium. When the learning is observed to proceed too slowly, or to converge too early, another MINOS is added. Its initial reference vector is set to that of the MINOS that suggested the addition to the Pandemonium. In this way both the new MINOS and the one experiencing difficulties with learning will be responsible for a similar volume of input space, at the start of the new MINOS module's learning. After it has seen a few patterns, largely taken over from the MINOS that was having difficulties, its reference vector converges to a stable point in the input space.

Thus the dynamic modularization is directed to relieve difficult regions of the problem space. The reflection informs as to the problem areas, and guides the addition of new MINOS modules. Figure 7c shows the way in which the input space for the two spirals solution of Figure 7d was allocated among the dynamic Pandemonium consisting of 6 MINOS modules. As expected, the space is distributed radially, with an additional central region. The consequent sub-problems required to be solved by the MINOS modules consist of separating two or three curved "stripes" from one another. The complexity of such a problem is markedly lower than the original one, and the result is a clear improvement on the solution offered by the single large neural network after 4000 presentations. The Pandemonium system was also twice as fast as the single network (Pandemonium $\approx$ 0.30s per learning set presentation, single net $\approx$ 0.61s per presentation). If the single net were also given 10 hidden units it would be faster, but the result would be even worse than that shown in Figure 7b.

4.2 The parity problem

It is possible to use the above method of Pandemonium decomposition for the parity problem as well. However the nature of the problem is such that its complexity would not be significantly reduced by such modularization. The reference vector method is good for problems of low dimensionality. The higher the dimensionality of the space, the less well can a number of hyperspheres cover the whole space without overlapping. This means that the reference vector method would lead to agents receiving extremely distorted regions of space to learn, that are not necessarily of a simplified nature.

Thus it is preferable to allow the Workers, as in section 3, to choose their own regions of specialization. The difficulty is that the Workers would automatically organize themselves so that one MINOS deals with the positive parity patterns, and the other with the negative parity patterns. In such a case Worker 1 only ever says "`1`", and Worker 2 only ever says "`0`". Then the whole complexity of the parity problem has been transferred to the Monitors. This behavior is typical when the output can take on only a few possible values. It was not important in the case of continuous functions, or for OCR for which there are many more classes than MINOS modules used.

In order to avoid the GC scenario, the information as to the appropriateness of a particular Worker is based on a set of history weights. The actual weights are updated after every pattern is learned, and copied to the history weights after every presentation of the complete learning set. In this way, the agents are still adapted incrementally, but the allocation of patterns is based on slower changing information. This allows patterns of both polarities to be learned by a MINOS module, thus stopping the GC convergence.

Parity of dimension 10 was chosen to test the performance of the Pandemonium system. It was compared with a single neural network working with many hidden units and in online update mode. 10 is generally the limiting dimension for neural networks learning parity. At this point the learning becomes extremely slow and generally success is based more on chance than on a good search procedure.

The learning set used was identical with the test set: all 1024 possible patterns were used. Results are shown in Table 1. A "presentation" was understood to mean a complete run through the entire learning set. The patterns were permuted after every presentation. In total 20 runs were performed for each system. Convergence was judged to be when the accuracy reached 99%. Reaching 100% was possible with a similar frequency to that of reaching 99%, but took a lot longer. The simulations ran up to 1000 presentations, after which they were discarded as "unconverged". It is a contentious issue whether one should kill runs that have gone on too long, or let them run until convergence, however long it may take. We feel the convergence time is as valid a factor as the best accuracy in deciding on whether a run is good. In real-world applications the convergence speed and frequency of convergence are important factors for consideration.

From Table 1 it can be seen that the Pandemonium systems all converge more than 50% of the time, while the single net very rarely does. Single nets with a higher number of hidden units were unexpectedly bad at finding solutions in the time allotted. Furthermore they were extremely slow, CPU time per presentation growing superlinearly with number of hidden units. As far as Pandemonium was concerned, the number of MINOS modules used was relatively unimportant. It can be seen that the CPU time per presentation remains quite stable for systems containing more than 3 MINOS modules, as does the convergence frequency. The average number of cycles until convergence is also better for the larger Pandemonium systems. This is a very agreeable scaling behavior. The number of MINOS modules that are required is not known at the beginning of an experiment, and so it is satisfying to see that the quantity does not result in poorer or slower performance.

MINOS modules	hidden units	CPU time per presentation	% trials converged	average cycles to convergence
1	20	6.4	15	727.7
1	100	41.6	0	–
1	200	125	0	–
2	20	10.3	55	575.5
3	20	11.5	45	507.7
4	20	15.7	60	375.2
5	20	16.1	50	395
6	20	16.2	50	395
7	20	16.0	60	388.5

Table 1: Relative performance of single net agents and the Pandemonium system with varying numbers of hidden units and MINOS modules respectively.

One notices that a trade-off is eventually to be expected between the complexity of the task for an individual agent (MINOS) and the complexity of decomposition among the agents. The complexity for individual agent decreases with number of MINOS modules involved in the Pandemonium, but the decomposition increases with this number. This can be expressed in pattern space as the difficulty of the agent discriminations versus the difficulty of separating agent boundaries. Furthermore, it can be observed as the relation between the number of hidden units required by the Workers to those required by the Monitors, to solve their respective problems. In our experiments the number was always the same for Worker and Monitor.

5 Classification problems

Classification problems pose a further question to modular systems: what role is to be played by modularization in areas where the problem consists of separating overlapping classes? In the previous examples we have either had discrete separations (two spirals and parity), or continuous but unique mappings (the functions with discontinuities). Real-world OCR problems are characterized by classes that overlap to such a degree that they are no longer easily separable. The borders between classes have a form typified in Figure 8. A performance of around 80% is generally possible with linear separation, after which non-linear decision regions have to be constructed. The number of regions required for further improvement increases the better the performance becomes [20]. The "last few" difficult patterns become ever more entwined in other classes, and ever further from their fellow patterns.

For such problems the Pandemonium type of modularization will not help significantly in the solution of the problem. Experiments carried out in [25, 19, 20] indicated that the Pandemonium system was significantly faster than a (large) single net in converging to a good solution [29], but did not produced higher accuracy than other, non-modular, systems. Pandemonium is most likely to help, as for the parity problem, in higher convergence speeds

Figure 8: Typical nature of overlap between classes in OCR problems.

and for incremental learning of limited numbers of examples.

It was shown in [20] how marked improvements may be gained in a team method of solution. The performance of a reflective team improved on the best single agent performance of 88.4% by 4.9% for a difficult hand-written upper-case letters problem. In team methods the input space is not necessarily decomposed among agents, but the agents themselves have different ways of looking at the problem and modeling it. Their range of suggestions for each input is used in a committee decision [17] from members that each see the entire input space. Figure 9 illustrates the idea.

6 Conclusion

The Pandemonium system of reflective reagents has been described and applied to benchmark problem domains involving continuous mappings and boolean problems. It application to classification problems was also discussed. It was shown to be successful in terms of convergence speed for all these domains relative to standard neural network methods involving a single network. In the continuous mapping and two-spirals problem cases it was shown to generate a sensible decomposition of the problem space and as a result produce a solution much quicker and better than a non-modular technique. For the parity problem decomposition was not able to simplify the problem space, but allowed significantly quicker convergence with low numbers of hidden units per MINOS together with good scaling properties for the Pandemonium itself.

Three methods of dynamic decomposition were demonstrated. Each method has its advantages depending on the problem to be solved. Thus one method was used for mappings continuous in input and output spaces, another for mappings continuous in a low-

Figure 9: How a team theoretically improves OCR classification tasks. Shown are the distributions of pattern accuracy as a function of pattern (artificially plotted as a single dimension). The idea is that the agents work together to make the team distribution larger. In reality the overlap between individual agents is very small.

dimensional input space but boolean output space (the two-spirals problem), and another for problems with a high-dimensional binary input and boolean output (parity problem). OCR Classification problems are normally sparce high dimensional binary or continuous input spaces with relatively lower dimensional sparce binary output spaces. For such problems any of the decomposition methods could be used, but due to the overlapping nature of the classes one would expect to find significant improvements through using team methods (as described in [20]) rather than decomposition methods.

References

[1] K. M. Hornik, M. Stinchcombe, and H. White. Multilayer feed-forward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[2] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[3] Věra Kůrková. Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, 5:501–506, 1992.

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

[5] 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.

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

[7] 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.

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

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

[10] 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.

[11] 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.

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

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

[14] 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.

[15] D. H. Wolpert. Stacked generalization. Technical Report LA-UR-90-3460, LANL, Los Alamos, NM, 1990.

[16] J. Franke and E. Mandler. A comparison of two approaches for combining the votes of cooperating classifiers. In *Proceedings of the 11th IAPR*, pages 611–614, Den Haag, Netherlands, 1992.

[17] L. K. Hanson and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[18] 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. Also available as GMD technical report 638.

[19] F. J. Śmieja and H. Mühlenbein. Reflective modular neural network systems. Technical Report 633, GMD, Sankt Augustin, Germany, February 1992.

[20] U. Beyer and F. J. Śmieja. Learning from examples, agent teams and the concept of reflection. Technical Report 766, Gesellschaft für Mathematik und Datenverarbeitung, St Augustin, Germany, July 1993. submitted to "International Journal of Pattern Recognition and Artificial Intelligence".

[21] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[22] 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.

[23] G. E. Hinton. Learning distributed representations of concepts. In *Eighth Annual Conference of the Cognitive Science Society*, 1986.

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

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

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

[27] U. Beyer and F. J. Śmieja. Learning from examples using reflective exploration. Technical report, Gesellschaft für Mathematik und Datenverarbeitung, St Augustin, Germany, October 1993.

[28] K. Lang and M. Witbrock. Learning to tell two spirals apart. In D. Touretzky, editor, *Proceedings of the 1988 Connectionist Summer School*, pages 52–59. Morgan Kaufmann, 1988.

[29] F. J. Śmieja. Optical character recognition using pandemonium and a genetically optimized polynomial classifier. In Jon Geist, editor, *The First Census Optical Character Recognition System conference*, pages 145–168, Gaithersburg MD, August 1992. National Institute of Standards and Technology.