Results of Simulations of a System with the Recommendation Architecture

Tamás D. Gedeon [1], L. Andrew Coward [2]
and Bailing Zhang [3]
[1] School of Information Technology
Murdoch University, Australia
[2] Nortel Networks, Canada
[3] School of Electrical and Information Engineering
University of Sydney, Australia
tom@it.murdoch.edu.au

Abstract

Functionally complex electronic systems are organized into functional components exchanging unambiguous information. The requirement to exchange unambiguous information results in difficulties in implementing parallel processing and extreme difficulty in implementing any capability to heuristically change functionality based on experience.

The recommendation architecture allows the exchange of ambiguous information between functional components ant therefore offers a way to reduce these difficulties. A system with the recommendation architecture uses a device imprinting mechanism to heuristically organize its inputs into a portfolio of ambiguous information repetition conditions on a range of levels of detail. The presence or absence of these conditions contains enough information to be used by a separate subsystem to determine appropriate behavior.

Simulations of a simple system with the recommendation architecture demonstrate that sequences of inputs of wide range of different types can be heuristically organized into a functionally usable set of repetition conditions. Organization is successful even though there are no exact repetitions of input conditions. Learning effectiveness measures which make no use of information on the consequences of system actions can be used to adjust architectural parameters to organize even wider ranges of input types. These results demonstrate the feasibility of developing functionally complex systems with the recommendation architecture.

Introduction

Current systems design technology is based on the partitioning of functionality between functional components which can be designed relatively independently.

There are several limitations to this approach to functional partitioning which derive from the sharing of unambiguous information by different components. Any active component may need to use any element of information, so all information must be stored in one accessible central point known as memory. Because an active component may change the information it is using, the version stored in memory is not usable by another component until it has been updated.

Components therefore tend to execute sequentially. Achievement of parallel processing requires partitioning of information into orthogonal segments which can be operated on independently by different components. In a functionally complex system this partitioning must be dynamic and is very difficult to achieve. Another limitation is that if a component were to change its functionality in response to experience (i.e. self modifying code) it would be extremely difficult to maintain the unambiguous context for information generated by that component. Heuristic modification of functionality has therefore not been achieved in a functionally complex system.

An approach which allowed the exchange of ambiguous information between functional components would avoid these limitations. The recommendation architecture (Coward 1999) makes it possible to maintain an ambiguous but meaningful context for information exchanged between functional components and therefore potentially could make possible the design of functionally complex, parallel processing systems which could heuristically modify their own functionality. This paper reports the results of simulations of a relatively simple system with the recommendation architecture with a range of different types of inputs and discusses the implications of the results for the design of functionally complex systems.

Architecture of the Simulated System:

In the recommendation architecture, functional components detect repetitions of input information combinations on different levels of complexity. These combinations are heuristically defined based on a device level mechanism in which components will be imprinted with its current set of active inputs. This imprinting means that the device produces an immediate output, and produces the same output in the future if the active input combination repeats, even in the absence of the higher level conditions. Higher level functional components are sets of devices which produce a representative output if a large set of their constituent devices are active. Yet higher level components are made up of sets of these and so on.

The functional hierarchy is thus a hierarchy of repetition similarity with the detection of similarity at any level being indicated by an output from the corresponding component. The competition for control of action must take place in a subsystem completely separate from the clustering in the sense that the consequences of action affect the competitive subsystem but cannot affect the similarity hierarchy. Another way of looking at this problem is that the use of consequence information introduces the use of information with an unambiguous context. It is then a major problem to ensure that all heuristically defined components share that context.

In a functional architecture, the information exchange between functional components must be minimized. In an instruction architecture this minimization is an important part of the design process. If component functionality is defined heuristically there must also be a process to heuristically minimize information exchange. Coward (1990) has argued that information exchange minimization is the function of sleep with dreaming in biological systems. The differences between the familiar instruction architecture and the recommendation architecture have a number of implications for practical application.

The nature of the imprinting process produces a separation between recognized familiar conditions and new conditions requiring more processing in a manner analogous to human behavior on habitual activities and new tasks. As the system outputs are initially arbitrary recommendations, the mapping between the recommendations and actions will be suggested by the expert. This task is significantly simpler and more abstract than the classification of individual network states into instruction architecture commands or strict categorical groups. Note that this task is simpler partly because of the grouping of input conditions which give rise to particular recommendations, and partly because each recommendation can even be wrong, as it is the totality of recommendations which give rise to actions.

The combination of recommendations into an instruction is analogous to the process of defuzzification, as a final step when interfacing with the crisp Boolean logic, the fuzzy sets / fuzzy numbers are transformed using mathematical formulae to crisp sets and single numerical values. The benefits of fuzzy logic come from the propagation of the notion of partial membership of sets throughout a computation and only truncating to the less rich traditional sets at the end of the computation. In this fashion the elimination of the requirement for unambiguous context information during the computation provides a benefit (discussed below), and the outputs can be 'sharpened' into instructions.

Architectural Structure and Parameters

To explore constructing a system for recommendation architecture, simple functional modules or clusters were defined. These clusters identify repetition similarity conditions, and interact to organize a sequence of system inputs (or experiences) into a limited set of such conditions. Each cluster has three layers, α, β and γ. The α layer has two functions, it receives system inputs and also inhibits the creation of new clusters. The β layer detects the presence of an input similar to the heuristically defined similarity condition for the cluster and triggers modification of that condition to include the new input. The γ layer also has two functions. One is to provide a cluster output that indicates the presence of the similarity condition within the system input and also discriminate between different inputs meeting the cluster similarity condition, the other is to prevent any further modification to the cluster similarity condition once an output has been generated. In a functionally extremely complex system the layers with multiple functions would be split into one layer per function to allow independent functional optimization. Each layer in a cluster is made up of devices which can be imprinted with a currently present information combination to detect any future repetition of that combination. This imprinting occurs under the control of the layer functions described above.

The devices are initially provided with a randomly selected set of inputs from the preceding layer, or from the possible binary input vector elements in the case of the α layer. The random selection is biased in favor of certain inputs which play a significant functional role in other devices in the same layer as described below. This bias is an important element in the minimization of information exchange between functional components mentioned earlier. Devices when initially configured are called virgin devices, and will not produce an output even if all their randomly assigned inputs from the preceding layer are active. However, if in response to a system input a cluster has significant device firing in its β layer but no firing in its γ layer, the number of inputs required to fire virgin devices throughout the cluster is gradually lowered, resulting in virgin devices being imprinted until either the additional firing results in an output from the cluster or

a minimum virgin device threshold is reached. For the first imprinting of a cluster the presence of enough significant α inputs as described below is sufficient to trigger device imprinting. Once a deice is imprinted it becomes a regular device which will produce an output if a large proportion of the inputs with which it is imprinted are present, independent of the degree of firing in different cluster layers.

The number of γ outputs generated in response to a system input is an indication of the degree to which the cluster similarity condition is present in that system input. A system constructed with these functional components heuristically defines a set of repetition similarity conditions from a sequence of inputs such that eventually every input includes a repetition of one or more of the defined conditions. The presence of a repetition condition of a particular type is indicated by a gamma output from the corresponding cluster, while the identity of the actual γs producing the output provides some discrimination between different conditions of the same type.

Initial configuration of a module	**Similarity criteria**	**Learning conditions**
Number of initial inputs to α, β, and γ devices	Minimum number of γ devices firing to indicate recognition	Limits on numbers of α, β, and γ devices imprinted at initial learning
Statistical bias on selection of sensory inputs to α devices	Minimum number of β devices firing to indicate similarity	Limits on numbers of α, β, and γ devices imprinted at regular learning
Numbers of α, β, and γ devices	Criterion for selection of module for learning	Minimum number of inputs to a virgin device which must be firing to trigger imprinting under learning conditions
Allowed level of input duplication	Minimum number of devices to indicate cluster familiarity	Regular device threshold after imprinting
Resource addition to existing modules	Minimum number of statistically favoured inputs present in an object to initiate new cluster	**Other**
Number of virgin devices added to α, β, and γ levels during sleep	Correlation criterion to indicate cluster duplication	Adjustment to regular device thresholds with changes in number of inputs
Number of inputs to virgin devices added to α, β, and γ levels during sleep		Limits on frequency of creation of new modules
Statistical bias on selection of inputs to α, β, and γ virgin devices added during sleep		Elimination of devices which rarely or never fire after initial imprinting
Number of inputs added to regular devices during sleep		Modification of device thresholds in clusters which rarely respond to any object
Statistical bias on selection of inputs to α, β, and γ regular devices added during sleep		
Allowed level of input duplication		

Table 1. Parameters which can be adjusted to modulate learning effectiveness

The architectural requirements are that it be possible to sort an actual sequence of input conditions into a limited set of repetition clusters, and that the outputs are limited but adequate to manage the system functionality. There are a range of parameters which can be set in advance or modified heuristically to meet these requirements. However, the output of a cluster is used by a range of later competitive functions to heuristically determine system actions. Cluster outputs therefore cannot be changed using information on the correctness of the output for one function because such a change would change the context of the cluster output information for other functions. Parameter values can be set in advance based on knowledge of the type of problem which the system will be required to solve, but during learning can be modified only using indicators of general effectiveness of learning, not indicators of learning effectiveness for specific types of function. A number of parameters which can be varied to influence learning effectiveness are shown in fig. 1.

Such parameters can be set in advance for a particular problem, but this will limit the generality of the architecture. Alternatively they can be adjusted in response to indications of general learning effectiveness. In practice some combination will be appropriate for a real system. For example, it could be decided in advance that somewhere between 5 and 10 clusters would be appropriate to organize a particular type of experience, but parameters like the input counts and output targets of virgin devices could be adjusted if clusters were not forming or tending to form in too large numbers. The purpose of the experiments with artificial data described in the next section is to begin to understand the effects of changing different parameters and the learning effectiveness indicators which could be used to adjust these parameters.

Description of the simulation process

The system being simulated is exposed to a sequence of input combinations, followed by a sleep period. Input and sleep period alternate for a total of thirty of each. In each input period 25 input patterns from each of five classes were alternated. All 3750 input patterns were different, so no input condition was repeated.

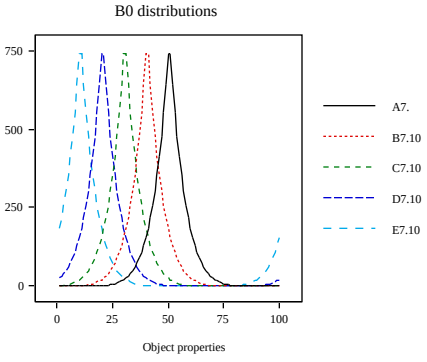
In the first input period no cluster existed and there was no device imprinting, but records were kept indicating the frequency of occurrence of input vector elements. In the first sleep period one cluster was configured with a random selection of inputs to α, β, and γ neurons but with a statistical bias in the inputs to α neurons in favor of input vector elements which most frequently occurred in the same input combination.
In the second input period, the first input pattern which contained a high proportion of these frequently coincident inputs generated device imprinting until an output resulted from the new cluster. From that point on, any input pattern that resulted in significant β firing in the new cluster generated device imprinting until either a β output resulted or device resources were exhausted.

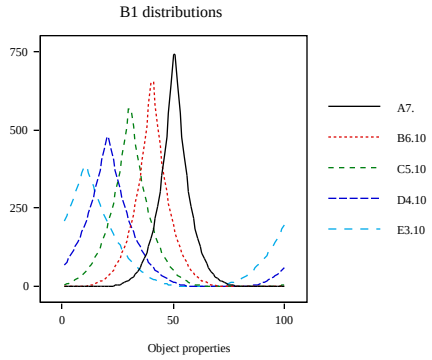
In the second sleep period, new virgin neurons were configured within the imprinted cluster. Inputs were randomly selected but with a statistical bias in favor of inputs that had frequently contributed to the firing of regular neurons in the same layer as the virgin neuron being configured.

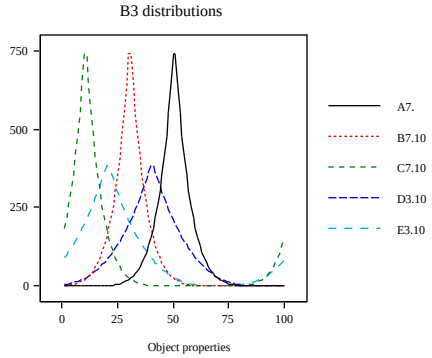
In subsequent sleep periods new clusters could be configured provided that two conditions were met. One condition was that no new cluster had been configured in the previous sleep period. The other was that a significant number of input patterns produced no activity in any existing cluster. The inputs to the a layer of the new cluster were than biased in favor of input vector elements which frequently occurred together in these input patterns. These conditions were designed to limit proliferation of clusters.

Results with artificial distribution data:

Four initial distributions (B0 to B3) were constructed as follows:







Each input pattern consists of a 100 element binary vector. Each input pattern belongs to one of the five classes A to E. An input pattern is generate by random assignment of ones and zeros based on the distributions shown above. For example, for a pattern of class A from distribution B3, each element of the vector is assigned a value of zero or one based on the rightmost curve in the rightmost diagram. In this case, only in the range 25 to 75 will there be any ones, and the probability of ones is highest around element (or property) 50.

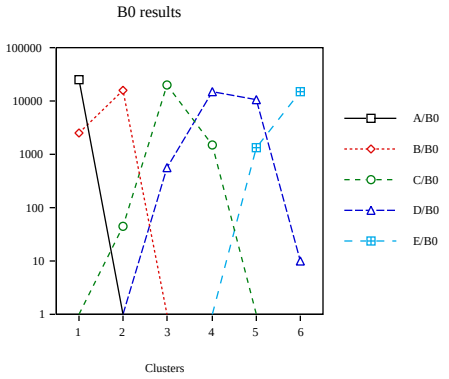
Thus individual input patterns may not appear characteristic of their particular class, rather as a group their statistical properties will be characteristic of their class. This is appropriate to the objective, which is to determine whether the architecture can heuristically identify repetition conditions in its inputs which can usefully be associated with system actions under conditions in which no input condition ever repeats exactly. A single event will not change functionality unless it is repeated sufficiently often.

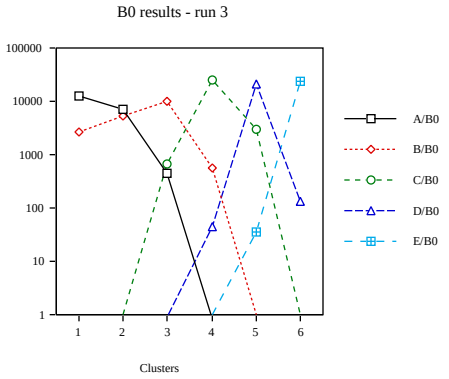
The distributions above were chosen to represent a range of likely distributions of properties of real data. The B0 distributions have high peaks, with some overlap in the distributions of the classes. In the B1 distributions the peaks are successively reduced and the distribution widened, increasing the overlaps in properties. We would expect that this would be harder

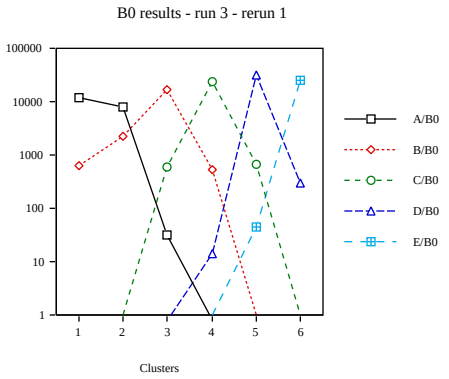
to learn. In B2 the distributions are the same as B1 except that they are twice as closely packed, leading to significantly greater overlapping of properties. In B3 three of the high peak distributions are mixed with two of the lowest peaked distributions.

The results of simulation runs on each of these distributions are summarized below. The two objectives were to discover the degree of similarity of results of different runs on each of the distributions, and to discover how sensitive the architecture was to the actual input patterns used. That is, firstly we generate a population of input patterns and then train the architecture to determine the degree of similarity between different runs. Then, to detect the sensitivity one of the sets of input patterns generated for the first part of the experiment is used. Below we call this re-running.

Results B0:





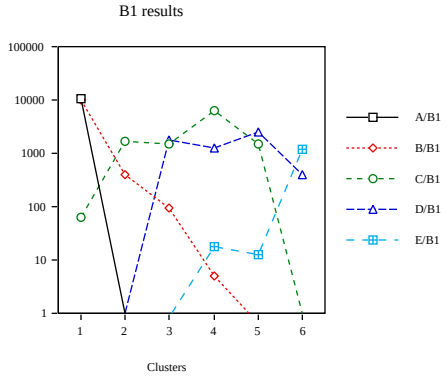


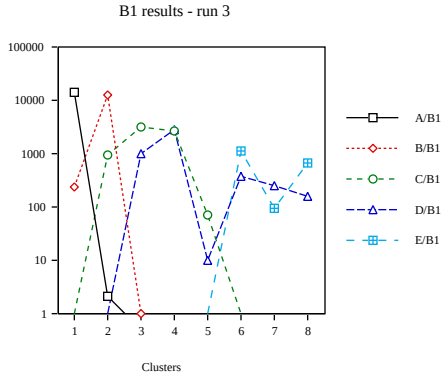
The above diagrams show the number of γ outputs from each cluster for each kind of object. The results can be summarized as follows: each of A to E are the major source of γ outputs from a cluster, and there is another lesser output from each cluster (less by at least one order of magnitude) by another adjacent kind of object. For example, cluster 1 produces most γs for A, and about 10% as many γs for B. Similarly cluster 2 largely represents B, and secondarily C. (Here and subsequently, we have sorted the clusters in order of decreasing γs on A, then B and so on. This aids the comparison of different simulations, and is acceptable as there is no particular meaning to the ordering of clusters originally.) As six to seven clusters are formed, this is not completely consistent, for example in run 1 with D producing the majority of γs in both of the clusters 4 and 5, with secondary γs for E and C

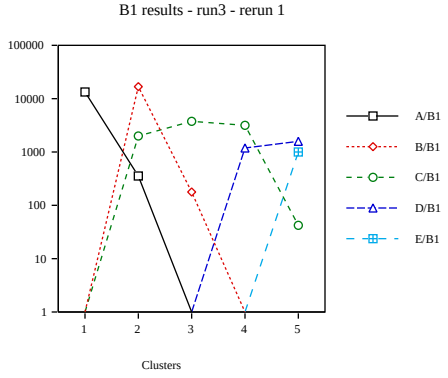
respectively. The fourth diagram (run 3 rerun 1) appears most similar to run 3, though with some variation.

From these results, it appears that the architecture will provide reasonably consistent results on different populations of inputs drawn from the same distributions, and that the variation in results when re-run using the same population is less.

Results B1:





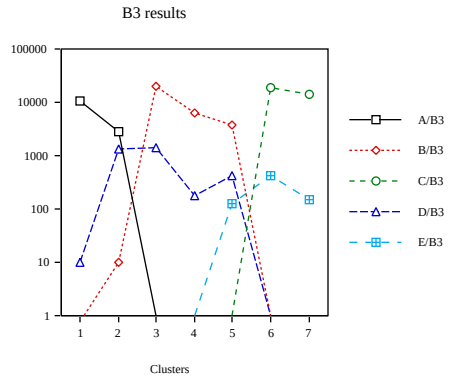


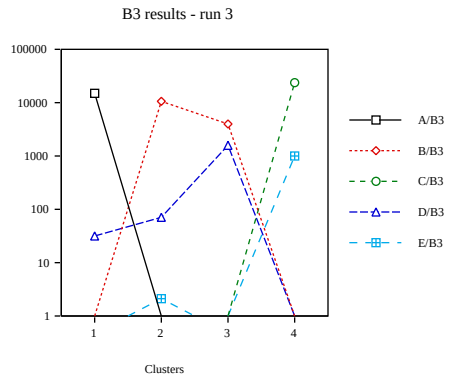
Note that we would expect C to lead to the first cluster being imprinted, as all of the other kinds of objects share some common object properties. After the first 125 object presentations, the system accumulates the most common object property and imprints using this and the most commonly co-occurring other properties. Thus, the center or core of C is most likely to be most common, and the commonly co-occurring properties are most likely to be other object properties of C. This is not happening, we will investigate this later.

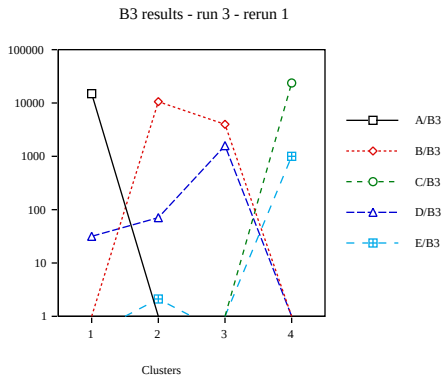
The high peak for A has again occurred, but is shared almost exactly by B. That is, there is no longer the factor of 10 decrease in γs for the second kind of object sharing that cluster. In further analysis we will look at the actual identity of the γs , rather than just the number of γs produced. We also observe that in run 1, for example, 12 clusters were created, but 6 of these had no γs , producing again 6 functional clusters. We comment on this further below.

From these results, our conclusion from before remains, except that results when re-run using the same population appears no different than using new populations. This is useful, establishing that the architecture is sensitive to the statistical distribution of properties but not overly sensitive to specific distributions.

Results B3:



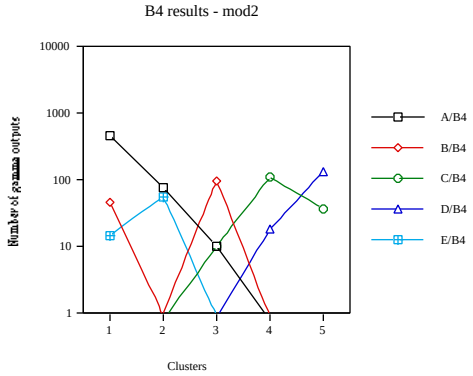




Objects D and E are again harder for the architecture to learn. Categories A, B and C can easily be discovered as producing very high γ outputs. Category D can be discovered as producing much lower γ outputs from clusters 1 and 3 in all diagrams above. Similarly category E can be discovered as the patterns belonging to this category produce much lower number of γ outputs on the last cluster in each of the above diagrams.

The results for a repetition of a B3 run were identical. The reason for this is that the random element in the selection of inputs to virgin neurons is provided in the simulations by a random number generator. The seed for this random number generator is a hash created from the state of the computer at the start of the simulation. If that state happens to be identical, runs in which the same objects are presented in the same order will be identical. The randomization represents the ambiguity of information exchange between functional components.

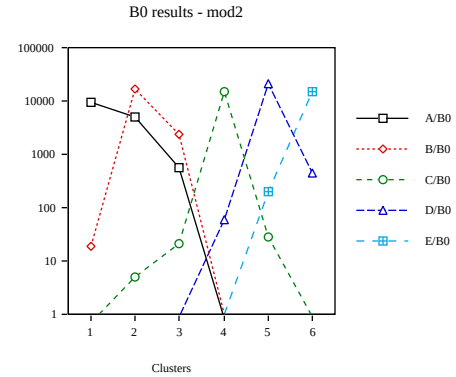
Overall, the results from B0 to B3 indicate that we can statistically differentiate between the categories on number of γ outputs . These results lead us to define a further set of distributions B4, with completely evenly distributed probability distributions of object properties, fully covering the space of properties. The results are shown below.

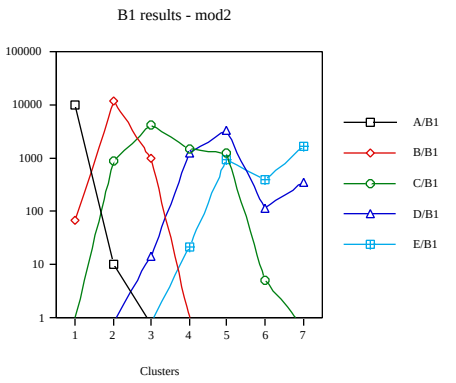


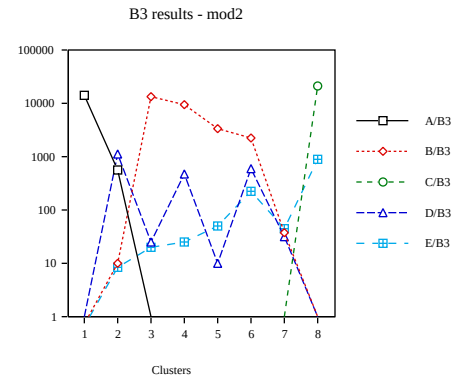
Note that all properties are equiprobable. This is a hard case, as overall occurrence of properties carries no meaningful information. In simulations it was found that once a cluster was imprinted it responded to very few other objects. A mechanism to heuristically tune learning parameters under these conditions was investigated. The mechanism was that firstly, before a new cluster is configured, a test is made to ensure that the most recently previously imprinted cluster produced a response to at least 5 objects in previous 125 input patterns. Secondly, during sleep each cluster is checked if it has responded to at least 1 input pattern in 125 with a minimum review period of 250 input patterns. If not, the thresholds of all existing regular α and β neurons is reduced by 1 to a minimum of 4.

The results are illustrated, below. The number of γ outputs is still much reduced compared to runs on B0 to B3, however the categories could now be separated based on these results. These results also demonstrate that research work remains to be done.

Below we show the results with this modification on B0 to B3 remain unchanged. The purpose of the modification was to adjust the learning parameters heuristically so that the system could effectively sort a wider range of experience types into a useful set of repetition clusters. Hence it is an appropriate change because it extends the range to include B4 without affecting the ability to handle the other types.







Discussion and Conclusions

Previous simulations (Coward 1999) were limited to object classes with distribution type B0. Following experimentation the parameters in table 1 were set to values which gave a separation into clusters which correlated strongly with object classes of this distribution type. The simulations demonstrate that the same parameters can generate useful clusters for a wide range of other types of input patterns, and that for one particularly different type it is possible for the system to heuristically modify some parameters to achieve a useful cluster set using only its learning experience and without prior knowledge of the input type.

The conclusion is that the architectural approach appears able to heuristically organize a wide range of possible input types into repetition similarity clusters which generate outputs that contain enough information to control functionality.

The next stage of work will require a thorough investigation of the effects on learning effectiveness of a number of variations in the table 1 parameters. Variations in these parameters must be studied on the basis of their statistical effect on the results produced. That is, we must be able to find a statistical correlation between the parameter variation and the effect on outputs. This will lead to mechanisms by which a system can heuristically adjust itself to different input types based on measurements of its outputs.

References

Coward, L. A. (1990). *Pattern Thinking*, New York: Praeger.

Coward, L. A. (1999). A functional architecture approach to neural systems. To be published in *The International Journal of Systems Science and Information Technology*.