A chunking mechanism in a neural system for the parallel processing of propositional production rules *

E. Burattini, A. Pasconcino, and G. Tamburrini
Istituto di Cibernetica C.N.R.
I-80072 Arco Felice, Italy
e-mail: ernb@sole.cib.na.cnr.it

Abstract

The problem of extracting more compact rules from a rule-based knowledge base is approached by means of a chunking mechanism implemented via a neural system. Taking advantage of the parallel processing potentialities of neural systems, the computational problem normally arising when introducing chunking processes is overcome. Also the memory saturation effect is coped with using some sort of “forgetting” mechanism which allows the system to eliminate previously stored, but less often used chunks.

Even though some connection weights are changed in the process of storing or discarding chunks, we emphasize that this neural system cannot be regarded as a “connectionist” system, since a localist semantic interpretation is adopted and no classical learning algorithm is employed.

*This work was partially supported by “Progetto Finalizzato Trasporti 2” of C.N.R. and by the jointly supported French-Italian project “Galileo”.

85
1 Introduction

The power law of practice is an experimental law of psychology asserting that reaction times in the execution of certain motor-perceptual and higher cognitive tasks decrease exponentially with the number of trials—until a lower bound, depending on the task, is reached. A tentative theoretical account of this regularity detected in human cognitive behaviour is the chunking theory of learning of Newell and Rosenbloom (1981), which hinges upon the chunking hypothesis formulated in Miller (1956):

A human acquires and organizes knowledge of the environment by forming and storing expressions, called chunks, which are structured collections of the chunks existing at the time of learning.

Miller does not specify the nature of the primitive chunks on the basis of which humans form and store new chunks. In the SOAR system, which embodies the fundamental assumptions of Newell and Rosenbloom’s chunking theory of learning, these “pre-existing” chunks are identified with production rules. And, again under the form of production rules, the chunking mechanism of SOAR enables the system to store in memory new chunks isolated on the basis of previous problem solving activity. (See Laird et al. (1986), Laird et al. (1987), Newell (1990), p. 185f.)

Independently of their cognitive plausibility, chunking mechanisms on production rules may play a significant role in AI applications, both for the automatic acquisition of knowledge bases and for the design of more efficient problem solving strategies. In this paper, we are concerned with the use of chunking mechanisms for addressing the latter problem. In particular, we describe a chunking mechanism generating production rules which codify associations between initial data and final outcome of an inferential path, and thus enabling a rule-based system endowed with it to bypass that inferential path upon successive presentations of the same initial data.

Chunking mechanisms generally give rise to what, following Tambe et al. (1990), may be called cognitive and computational effects. The cognitive effect is the reduction of the number of (inferential) steps
needed to carry out a given task. The computational effect is the increase of the time needed to carry out each individual step. Thus, what is gained in efficiency by reducing the number of steps is often lost by an increase of execution time for each step. Clearly, when chunks take the form of production rules, the time required for executing the matching process between data in working memory and the antecedents of production rules may increase. In fact, more rules have to be scanned, and the newly introduced rules may contain more complicated antecedents than those present in the original system of rules. Another related phenomenon may be called the memory saturation effect: given preassigned finite memory capacities, a system endowed with a chunking mechanism, but incapable of “forgetting” some of the previously stored chunks, will be eventually unable to make room for newly acquired and possibly more useful chunks.

In view of the computational and memory saturation effects, an efficient use of chunking mechanisms requires a computational agent capable of

(a) leaving unaltered the access time to knowledge when new chunks are added, and

(b) attenuating the incidence of the memory saturation phenomenon.

The chunking mechanism described below—extracting associations between initial data and final outcome of a forward chaining inferential path—does satisfy condition (a). Condition (b) is satisfied as well, if the relative frequency of use of chunks is regarded as a satisfactory criterion for deciding which chunks have to be “unlearned” by the system.

This chunking mechanism is embedded into a system for parallel forward chaining on propositional production rules. This rule-based system can also perform goal-directed queries, in order to gather information useful to establish its current goals. And its justification module enables the user to get information about inferential paths that have been followed in establishing a given goal. The system is entirely formed by non-linear threshold neural elements. It is worth noticing that also the control of the inferential, query, chunking, and justification processes is entirely carried out by a neural network, too.
One may therefore legitimately qualify this system as a purely neural system. However, it cannot be regarded as a “connectionist” system, insofar as a localist semantic interpretation (as opposed to a distributed one) is adopted, and the classical learning algorithms used in connectionist systems are not employed here.

In view of these qualifications, it should be evident that the choice of a neural architecture for this system is chiefly motivated by the idea of exploiting at best the parallel processing potentialities inherent in propositional production rule sets. AI applications of this system in diagnostic problem solving domains have been developed (see for more extensive discussion Burattini and Tamburrini (1992), and Aiello et al. (1995a, b)).

Sections 2 to 6 provide an overall description of the system modular organization, neural architecture, and the implementation of forward chaining, query, and justification processes. Section 7 is exclusively devoted to the chunking mechanism. The appendix reports a trace of a forward chaining session, the recording of the appropriate chunk, and a new run of the augmented system of rules on the same initial data.

2 System overview

In this section we describe the overall functional organization of the system. Figure 1 shows the main modules of the system. The labels “Maker”, “Process”, and “Recorder” associated to arcs in that figure refer to the algorithms that are respectively used for constructing the neural network, carrying out its inferential activity, and recording the new chunks isolated on the basis of previous runs of the system.

Given a set of propositional production rules presented in a certain canonical form, the algorithm Maker yields a neural network for executing forward chaining, goal-directed query, and chunking on this system of rules, in addition to building an appropriate neural justification module for answering user’s queries.

When this neural system has been constructed, and a list of \( n + m \) propositional literals representing \( n \) initial data and \( m \) goals is given in input, the Process algorithm simulates the activity of the network, by a synchronous updating of the state of neural units. Time is treated as
A chunking mechanism in a neural system...

Figure 1: General Scheme

a discrete variable, and the time interval needed by any neural element for processing the incoming information is assumed to be independent of the number of input channels and to be equal to 1.

The $n$ initial data are transferred to the Chunk Module ($CM$) in order to recall the $r$ (with $r \geq 0$) propositional literals that the system inferred if and when presented in previous runs with (a subset of) the same initial data. More specifically, $CM$ checks whether the initial data match the antecedent parts of stored chunks (these chunks can be viewed as particular rules as well). Then the $r$ distinct literals appearing in the consequents of those chunks are retrieved by $CM$ and immediately made available to the forward chaining layer of the Rule Module ($RM$).

The $RM$ module performs forward chaining on the system of production rules and goal-directed query. $RM$ is started on the set of $n$ initial data independently of what is the outcome of the chunk retrieval process, since (i) it may still infer other literals in addition to the $r$ literals retrieved by $CM$ or (ii) it may have to provide, upon request, an appropriate justification for (the trace of an inferential path to) the literals obtained by the chunk retrieval process. Case (i) may apply if the antecedents of the chunks retrieved in $CM$ contain only a proper subset of the set of $n$ initial data, or if responses to queries make available additional premises.

The activity of $RM$ terminates when no other rules can be acti-
vated by forward chaining, and no more information can be accrued
by goal-directed query. At this stage, the system shows which, if any,
of the \( m \) goals have been established, and displays any other proposi-
tional literal which is (i) included in the initial data, (ii) declared by
the external source to hold in response to a goal-directed query, (iii)
obtained via the chunking mechanism or (iv) inferred by forward chain-
ing starting from data obtained by (i) or (ii). The user can activate
the Justification Module (JM) on any of the displayed propositional
literals. JM indicates whether this literal satisfies (i) or (ii); if it does
not, JM provides a trace of the shortest inferential paths to it. \(^1\)

At the end of each run of the system, the algorithm Recorder
isoi-
lates a chunk \(< I, C >\), where, for \( k, m \geq 1 \), \( I = \{ p_1, \ldots, p_k \} \) is
the set of initial data provided to the system in that run and \( C = \{ q_1, \ldots, q_m \} \) is the set of literals derived by forward chaining in RM
starting from \( I \). These chunks should be interpreted as conditional
statements \( p_1 \land \ldots \land p_k \rightarrow q_1 \land \ldots \land q_m \), where the literals in the an-
tecedent are the elements of \( I \), and the literals in the consequent are
the elements of \( C \). If the chunk \(< I, C >\) was already stored in CM as
a result of previous runs of the system, this fact is signalled by CM,
and the algorithm Recorder merely modifies the weights of the neural
units that are devoted to storing data about the relative frequency of
use of chunk \(< I, C >\). Otherwise, the chunk \(< I, C >\) is stored in
CM. In the latter case, the new chunk \(< I, C >\) may replace one of
the previously stored chunks that have been less frequently used during
the system operation.

3 Processing units and their semantic inter-
pretation

The network is formed by threshold neural units, whose activation at
time \( t \) is determined by the equation

\[
u_h(t) = 1 \left( \sum_{j=1}^{n} u_j(t-1)w_{jh} - \theta_h + E X T_h(t) \right)
\]

\(^1\)Notice that JM provides such a trace also for literals obtained by chunking.
A chunking mechanism in a neural system...

where

- $u_h(t)$ is the state of unit $h$ at time $t$;
- $u_j(t-1)$ is the state at time $(t-1)$ of unit $j$;
- $\theta_h$ is the threshold value of unit $h$;
- $w_{jh}$ is the weight of the direct connection, if any, from unit $j$ to unit $h$, which can be modified only during the intervals between different runs of the system;
- $\text{EXT}_h(t)$ is an input signal to unit $h$ incoming at time $t$ from a source external to the system.

- $1(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$ is the step function determining the state of the neural units;

In view of the fact that the state function can assume only values 1 and 0, these threshold elements can provide, under a localist semantic representation, boolean-valued information about the literals they are associated to. For example, the system includes a layer of neurons called GOAL (see figure 6 in Appendix), which is formed by as many units as the distinguished consequents in the system of production rules; the unit in this layer associated to, say, literal $p$ is in state 1 if and only if $p$ is a current goal of the inferential process.

The various kinds of boolean information provided by the neural units belonging to the main submodular components of RM, JM and CM are briefly described hereinafter. (For an example of how neural activity propagates through the network, and a figure sketching out the flow of information between the submodular components of the system, the interested reader is referred to the example reported in the Appendix). Let $x$ be a variable ranging over literals. Then,

- if the neural representative of $x$ in the block ANTECEDENT (and DATAIN) is active (its state is 1), then $x$ belongs to the set $I$ of literals constituting the initial data provided by the user;
- if the neural representative of $x$ in FORWARD (and FORWARDBF) is active, then $x$ follows by application of the forward chaining process from the set $I$;

- if the neural representative of $x$ in INFERREDBF is active, then $x$ follows by application of the forward chaining process from the set $I$; with respect to the representative of $x$ in FORWARD, however, the neural elements of INFERREDBF play the role of “temporal filters” as described in sect. 6 below;

- if the neural representative of $x$ in GOALACTIVE is active, then $x$ belongs to the set of goals of the system;

- if the neural representative of $x$ in SUBGOAL is active, then $x$ is a temporary subgoal, generated by the system while attempting to establish its current goal;

- if the neural representative of $x$ in QUERY is active, then the external source is to be queried in order to decide whether $x$ holds;

- if the neural representative of $x$ in QUERYTRUE is active, then the external source has asserted that $x$ holds;

- if the neural representative of $x$ in QUERYFALSE is active, then the external source has asserted that $x$ does not hold (or, equivalently, that the negation of $x$ holds);

- if the neural representative of $x$ in QUERYNOTKNOW is active, then no information on $x$ has been obtained from querying the external source;

- if the neural representative of $x$ in CONTRADICTION is active, then the set of literals comprising initial data, data asserted by the external source, and results of forward chaining includes both $x$ and $\neg x$;

- if the neural representative of $x$ in ACTIVEJ is active, then the user has asked information about the inferential and query paths that led the system to assert $x$;
- if the neural representative of $x$ in $\text{PREMISEJ}$ is active, then the system signals to the user that $x$ was asserted because an element of the set $I$ of initial data;

- if the neural representative of $x$ in $\text{INFERREDJ}$ is active, then the system signals to the user that $x$ was inferred by forward chaining;

- if the neural representative of $x$ in $\text{RESPONSEJ}$ is active, then the system signals to the user that $x$ was asserted to hold as a response to query;

- if the neural representative of $x$ in $\text{CONTRADICTIONJ}$ is active, then the system signals to the user that a contradiction occurred, since both $x$ and $\neg x$ are asserted to hold;

- if the neural representatives of $x$ in $\text{PATTERNON}$ and in $\text{PATTERNOFF}$ are active, then $x$ is an element of the set $I$ of initial data provided to the system;

- if the neural representative of $x$ in $\text{PATTERNOUT}$ is active, then $x$ was inferred by forward chaining from the set $I$.

Furthermore, let $j$ be a variable ranging over indexes which can be associated by the system to any chunk, as described in sect.7 below. Then

- if the neural representative of $j$ in $\text{INDEXBF}$ is active, then $j$ is already associated to a chunk; thus the system is not going to select $j$ as index for a new chunk;

- if the neural representative of $j$ in $\text{INDEX}$ is active, then the set $I$ matches the first part of the chunk associated to the index $j$;

- finally, if the neural representative of $j$ in $\text{SUBINDEX}$ is active, then the set $I$ contains a subset of literals which matches the first part of the chunk associated to the index $j$. 
4 Rule model and forward chaining

The algorithm Maker constructs the network formed by the modules CM, RM, and JM for any finite set of production rules of the following form

\[ p_1 \land \ldots \land p_k \rightarrow q \tag{2} \]

with \( p_1, \ldots, p_k \) and \( q \) propositional literals\(^2\). Since the inferential process carried out by RM is forward chaining, and no inference rule for negation is introduced, the inferential power of the system is not extended by permitting literals, rather than just propositional letters, to appear in rules of form (2). However, the introduction of literals (i) furnishes the external source (typically, the expert user of the system) with greater expressive power during the query process (it may directly communicate to the system that the negation of a certain proposition holds), and (ii) has enabled us to introduce a control mechanism for explicit contradictions, which detects whether contradictory pairs of literals occur in the set of literals containing initial data, literals asserted by the external source, and literals inferred by forward chaining.

Let us now consider the neural representation of such production rules, and the neural implementation of forward chaining. The subnet FORWARD of RM (see figure 6 in Appendix) is the neural implementation of forward chaining. Within this subnet, each rule of form (2) is represented as a net having \( k \) neurons \( p_1, \ldots, p_k \) connected to a neuron \( q \) (see fig.2) with the following settings:

\[
w_{jq} = 1 \quad (1 \leq j \leq k) \\
\theta_k = k - \epsilon \quad (0 < \epsilon < 1) \quad (3)
\]

By (1) and the settings in (3), one has that:

\[
u_q(1) = 1 \left[ \sum_{j=1}^{k} u_{p_j}(0) - (k - \epsilon) \right].
\]

\(^2\)One has to notice that chunks can be interpreted, as stated at the end of section 2, as conditional statements of the more general form \( p_1 \land \ldots \land p_k \rightarrow q_1 \land \ldots \land q_m \). But clearly, each statement of this form is equivalent to a conjunction of \( m \) production rules of the standard form.
Thus, $u_q(1) = 1$ if and only if $\forall j \ u_{p_j}(0) = 1$ that is, neuron $q$ is active (its state is 1) at time $t = 1$ if and only if all of the $p_1, \ldots, p_k$ are simultaneously active at time $t = 0$. More in general,

$$u_q(t + 1) = 1 \iff \forall j \ u_{p_j}(t) = 1.$$

Thus, the behaviour of the net formed by the neurons $p_1, \ldots, p_k$ and $q$ reflects faithfully the behaviour of a rule interpreter applied to a rule of the form $p_1 \land \ldots \land p_k \rightarrow q$.

The system of rules given as input to Maker is allowed to contain cycles (unlike, e.g., the KBANN neural production systems of Shavlik and Towell (1994)): a literal appearing as consequent in one rule can also appear in the antecedent of another rule. Moreover, several rules may share the same consequent. This latter possibility requires the introduction of a slight complication, with respect to the scheme in fig. 2, in the neural implementation of the forward chaining process: if a literal $p$ occurs as the consequent of $m$ production rules, then $m$ distinct neurons—each one representing an occurrence of $p$ in the conclusion part of those rules—are to be introduced. This additional condition is needed to avoid an incorrect activation of a neuron representing $p$, which may derive from a combination of premises belonging to different rules having $p$ as conclusion. A specific example of such situation is presented in figure 3, which illustrates the network for forward chaining relative to the following rules:

$$(R1) \quad a \land b \rightarrow c$$

$$(R2) \quad d \land h \land g \rightarrow c$$
One has to notice that in the forward layer of this network, the two occurrences of the propositional letter $c$ in (R1) and (R2) are represented by distinct neurons (which are labeled $c$ and $c_0$, respectively). Notice that if the neural representatives of the antecedents in (R2) are active, then $c_0$ becomes active at the next instant of time, and sends out an excitatory signal which is sufficient to activate $c$ as well.

5 Goal-directed query

The propagation of neural activity from the SUBGOAL to the QUERY block (see figure 6 in Appendix) is to be interpreted as the search for information by an external source on data useful for establishing a goal, say literal $q$, of the system. This propagation triggers the goal-directed query process. This is done by isolating, in the various rules of form $p_1, \ldots, p_k \rightarrow q$, the antecedents $p_i$ about which the system has not obtained any information (by inference or user declaration). The weights of the connections between the SUBGOAL and QUERY blocks ensure that if an element representing $q$ in SUBGOAL is active at time $t$, then the neural representatives in QUERY of these antecedents $p_i$ become active at time $t + 1$.

The answers a user can provide as a response to a system query on, say, literal $p_i$ are TRUE, FALSE, DONOTKNOW. These answers are recorded by the self-sustaining neural representatives of $p_i$ in QUERY-TRUE, QUERYFALSE, QUERYNOTKNOW, respectively. When a neural representative of $p_i$ in one of these blocks becomes active, it
sends permanent inhibitory signals to the representative of $p_i$ in the QUERY block, so that no additional consultation of the user on $p_i$ can be undertaken. More specifically, (a) if the answer on $p_i$ is TRUE (resp., FALSE), then the neural representative of $p_i$ in QUERYTRUE (resp., QUERYFALSE) is activated; (b) if the answer is DONOT-KNOW, the system obtains no information on $p_i$ and proceeds to investigate the antecedents of rules which have $p_i$ as their consequent: the representative of $p_i$ in QUERY activates at time $t + 1$ the representative of the same literal in SUBGOAL. In turn, this element activates, at time $t + 2$, the representatives in QUERY of those literals which appear as antecedents in rules having $p_i$ as their consequent, and about which the system has no information available. The user is asked to provide new information about these literals. This process is iterated until no more literals in QUERY can be activated.

6 Justification

When the forward chaining and goal-directed query processes triggered by a set $I$ of initial data and a set of goals have come to an end, the following neural units remain active (see the figure 6 in Appendix for the submodular organization of the JM module):

- the units in PATTERNON representing the elements of $I$;
- the units in INFERREDDBF representing the literals inferred from $I$ by forward chaining;
- the units in QUERYTRUE storing the asserted literals provided by the external source in response to queries;
- the units in CONTRADICTION representing contradictory pairs of literals, if explicit contradictions have been arrived at.

This information provides the means for discriminating between literals asserted by external sources (i.e. initial data and literals asserted in response to queries) and literals inferred by forward chaining. In the latter case, the system can also display inferential paths.
The user, by activating the neural unit in ACTIVEJ representing literal $x$, can start, when the forward chaining and goal-directed query processes have come to an end, the justification process concerning literal $x$ asserted by the system. At the next step, the impulse outputted by this unit in ACTIVEJ is transmitted to the neural representatives of $x$ in PREMISEJ, INFERREDJ, and RESPONSEJ. But only one of these neural representatives of $x$ will become active in its turn, and namely,

(i) the unit representing $x$ in PREMISEJ becomes active if the representative of $x$ in PATTERNON is active, that is, if $x$ is an element of the set $I$ of initial data;

(ii) the unit representing $x$ in RESPONSEJ becomes active if the representative of $x$ in QUERYTRUE is active, that is, if $x$ was asserted by the external source in response to a system’s query.

(iii) the unit representing $x$ in CONTRADICTIONJ becomes active if the representative of $x$ in CONTRADICTION is active, that is, if both $x$ and $\neg x$ are included in the set of literals formed by the elements of $I$, the literals asserted as response to queries, and those derived by forward chaining.

(iv) the unit representing $x$ in INFERREDJ becomes active if the representative of $x$ in INFERREDBF is active, that is, if $x$ was obtained by applying forward chaining to rules in RM. In this case, the justification process is iterated on sets of premises from which $x$ was inferred. In this connection, one has to notice that, given a set $I$ of initial data, $x$ may have been derived traversing several inferential paths. A simple example of this situation is when $x$ appears as consequent in more than one production rule, and every literal in the antecedent parts of these rules can be derived by forward chaining from $I$. Even though the subnet for forward chaining traverses all inferential paths leading from the initial data to $x$, it seems redundant to exhibit in the justification mode the trace of every such path. Thus, we have designed

\[3\text{As stated in section 4, forward chaining on input } I \text{ terminates only when no more rules can fire.}\]
a justification module exhibiting only the trace of the “minimal” inferential paths to \( x \), and namely, those paths which require the least number of parallel processing steps from the activation of the input layer of neurons to the activation of the neural representative of \( x \) in \textsc{forwardbf}.

Let us consider, for example, the following system of rules:

\[
\begin{align*}
(R1) & \quad a \land b \rightarrow e \\
(R2) & \quad b \land c \rightarrow e \\
(R3) & \quad d \rightarrow c
\end{align*}
\]

If the set of initial data is \( I = \{a, b, d\} \), then the system will reach \( e \) in one step by applying (R1), and in two steps by applying (R3) and (R2). Since the first inferential path requires less processing steps than the second one, the justification module exhibits only the trace of the first inferential path to \( e \). Figure 4 shows the neural net providing a competitive filter for this system of rules, which enables the justification module to select the “minimal” inferential paths to \( e \). Crucial to this filtering function are the inhibitory connections (represented by dashed arrows) between the neural representatives in \textsc{inferredbf} of the occurrences of \( e \) in (R1) and (R2). Indeed, if \( e \) becomes active at time \( t \), then \( e_0 = e \) cannot become active at any later time, and conversely. Thus, when the user asks the justification for the assertion \( e \) by activating the neural representative of \( e \) in \textsc{activej}, at the next instant of time only the neural representative of the occurrence of \( e \) in (R1) will become active in \textsc{inferredj}. The justification procedure is then applied to the literals constituting the antecedent of (R1), whose neural representatives in \textsc{premisej} become active, thus signaling that \( a \) and \( b \) are element of \( I \).

### 7 Chunking mechanism

In the previous sections we have described the modules for parallel forward chaining, query, and justification. Let us now turn to consider the module \textsc{cm} for the chunking process. The chunks stored in \textsc{cm} take the form \(<I, C>\), where \( I = \{p_1, \ldots, p_k\} \) is a set of literals
provided as initial data to the system and $C = \{q_1, \ldots, q_m\}$ is the set of literals derived by forward chaining in $\textbf{RM}$ starting from $I$. These associations enable the system to reduce processing time: whenever a set of input data coincides with or strictly contains the literals in the first element of a stored chunk, at the next step the system outputs the literals in its second element.

Constructing a chunking mechanism of this sort, efficiently coping with the computational and memory saturation problems (see sect.1), involves solving the following problems:

(i) recognizing an input pattern previously presented to the system in order to recall the chunks with antecedents matching the input pattern (or else storing an input pattern presented for the first time to the system);

(ii) keeping track of how often the stored chunks are used during the system operation, in order to discard less often used chunks when new chunks have to be acquired;

(iii) executing operations (i)-(ii) in a preassigned time, independently of the size of input patterns and the number of stored chunks.

It is unlikely that problems (i)-(iii) can be efficiently solved using the typical learning methods used in connectionist systems. In fact,
(a) multilayer connectionist networks require, for effectively storing any newly presented input pattern, repeating the training session on the whole set of input patterns. This indicates the difficulty of using such networks for solving problem (i), in view of the fact that training by backpropagation may require exponential time in the number of input patterns (see Judd (1990)).

(b) Hopfield’s nets (see Hopfield (1982), (1984)), when used as autoassociative memories, enable one to store patterns by a procedure (one-shot learning) determining the appropriate weights in linear time in the number of patterns. There are, however, interference effects between non-orthogonal patterns, and considerable processing time may be needed to converging into a stable state.

(c) While Grossberg’s ART networks (see Carpenter and Grossberg (1987)) may provide a sensible solution for the plasticity-stability problem, they seem unsuitable for addressing problem (iii). Indeed, the performance of ART networks depends on

(c1) the time $t$ needed to compare a previously unclassified input vector $X$ with the stored prototypes having non-empty intersection with $X$, before assigning to $X$ a new class; $t$ is a linear function of the number of such prototypes and therefore, if the number of prototypes is greater than the number of steps $RM$ requires to complete the forward chaining process, then the introduction of an ART-like CM module would not improve the performance of the system;

(c2) the time $t$ needed by the two competitive layers to converge to a stable state; the value of $t$ cannot be fixed a priori, and a bound on it can be estimated only experimentally, because reaching a stable state depends essentially on the number of components in each competitive layer and on the relative intensity of the inputs to the competitive layers.

The module CM affords a sensible solution to problems (i)-(iii). The algorithm Recorder stores each chunk $<I,C>$ by constructing two different links, respectively connecting (*) every literal in $I$ with
a given index $j$, and (**)) index $j$ with all the literals in $C$. This mechanism enables one to eliminate interferences between non-orthogonal patterns affecting traditional neural associative memories, such as the multilayer perceptron (see Rumelhart and McClelland (1986)) and the linear associators (see Kohonen (1977), (1982)): links (*)-(**) establish a one-one correspondence between first and second element of each chunk.

Let us now turn to describe the operation of CM. Upon presentation to the system of an input set of literals $I$, three different cases may occur.

**Case 1:** For every index $j$ the first element of the chunk $<I_j, C_j>$ is different from $I$ and case 3 below does not hold. In this case, CM does not recall any stored chunk. However, when forward chaining on $I$ is completed and outputs a non-empty set of literals $C$, the algorithm Recorder selects a new index $j'$ and stores a new chunk $<I_{j'}, C_{j'}>$, with $I = I_{j'}$ and $C = C_{j'}$.

**Case 2:** There is an index $j$ such that $I = I_j$ for some chunk $<I_j, C_j>$. CM recognizes this situation, and provides the elements of $C_j$ as outputs of the forward chaining process.

**Case 3:** For every index $j$ the first element of the chunk $<I_j, C_j>$ is different from $I$, but there are indexes $j_1, \ldots, j_k$ such that $I_{j_1}, \ldots, I_{j_k}$ are strictly included in $I$. Then CM provides the elements of $S = \cup\{C_{j_1}, \ldots, C_{j_k}\}$ as outputs of the forward chaining process. When forward chaining on $I$ is completed and outputs a non-empty set of literals $C$, the algorithm Recorder selects a new index $j'$ and stores a new chunk $<I_{j'}, C_{j'}>$, with $I = I_{j'}$ and $C = C_{j'}$.

The overall functional organization of the CM module is shown in figure 5. Let us now describe in more detail how the network behaves in cases 1-3.

**CASE 1:** When $I$ is given as input to CM at time $t = 0$ (and the neural units representing its elements in PATTERNON and PATTERNOFF are active).

---

4This configuration is not optimal with respect to the use of resources. In fact, a single layer could be designed which is functionally equivalent to the two layers PATTERNON and PATTERNOFF. The use of two such distinct layers was motivated by considerations of a semantic character. For more details see Pasconcino (1994).
NOFF become active at $t = 1$), the relation $\forall j \ I \neq I_j$ is recognized to hold at time $t = 2$ at the next instant of time since, for every $j$, the neuron representing index $j$ in the layer INDEX is inactive at that time. At time $t = 3$, the impulse from control neuron CLOCK2 activates the other control neuron CTRIND, whose activity is necessary and, together with impulses from PATTERNON (which are absent in this case), sufficient to activating subpatterns in SUBINDEX. Thus, at time $t = 4$ (resp. at time $t = 5$) all neurons of layers SUBINDEX (resp. of PATTERNOUT) are inactive.

As a consequence, the neural activity triggered by the input set $I$ in the CM module does not recall the second element of any chunk represented in the layer PATTERNOUT. So, at time $t = 6$ only the input set $I$ is transferred to the RM module. Finally, when forward chaining on $I$ is completed and outputs a non-empty set of literals $C$, the algorithm Recorder selects\(^5\) a new index $j'$ and stores a new chunk $<I_{j'}, C_{j'>}$, with $I = I_{j'}$ and $C = C_{j'}$.

The recording of a new chunk consists of the change of the weights of the following connections: (1) the connections between the neural representatives of the elements of $I$ in PATTERNON and PATTERNOFF on the one hand, and the neural representatives of the selected

---

\(^5\)See below the paragraphs under the heading “SELECTING INDEXES” for a description of the selection procedure and the role of the INDEXBF layer.
index \( j' \) in INDEX and SUBINDEX on the other hand; (2) the connections between neuron \( j' \) of SUBINDEX and the neural representatives of the elements of \( C \) in PATTERNOUT.

The weights of the connections between PATTERNON and INDEX are initialized with the following values

\[
w_{\text{Pon}_i \text{Index}_j} = 0 \quad j = 1, 2, \ldots, M; \quad i = 1, 2, \ldots, N;
\]

where \( M \) is the maximum number of patterns which can be stored by the system (that is, the maximum number of index neurons available in INDEX), \( N \) is the dimension or number of components of input patterns (that is, the number of neural units in each layer PATTERNON and PATTERNOFF), \( \text{Index}_j \) and \( \text{Pon}_i \) are neurons of the INDEX and PATTERNON layers, respectively.

The weights of the connections between PATTERNON and \( \text{Index}_{j'} \) are updated according to the rule

\[
w'_{\text{Pon}_i \text{Index}_{j'}} = \begin{cases} 
\frac{1}{|I|} & \text{if } \text{Pon}_i \text{ active;} \\
w_{\text{Pon}_i \text{Index}_{j'}} & \text{otherwise;}
\end{cases} \quad (6)
\]

where

\[
|I| = \sum_{j=1}^{n} u_{\text{Pon}_j}
\]

The weights of the connections between PATTERNOFF and INDEX are initialized with the following values

\[
w_{\text{Poff}_i \text{Index}_j} = -1; \quad j = 1, 2, \ldots, M; \quad i = 1, 2, \ldots, N;
\]

The weights of the connections between PATTERNOFF and \( \text{Index}_{j'} \) are updated according to the rule

\[
w'_{\text{Poff}_i \text{Index}_{j'}} = \begin{cases} 
0 & \text{if } \text{Poff}_i \text{ active;} \\
w_{\text{Poff}_i \text{Index}_{j'}} & \text{otherwise;}
\end{cases} \quad (7)
\]

Rules (6) and (7) ensure that exactly one neuron in INDEX becomes active when case 2 occurs. (See Pasconcino (1994) for a detailed justification of this claim.)
The algorithm *Recorder* initializes the weights of the connections from units of SUBINDEX to the units of PATTERNOUT in the following way

\[ w_{\text{Subindex}, \text{Pout}_i} = 0 \quad j = 1, \ldots, M, \quad i = 1, \ldots, N_c; \]

where \( N_c \) is the number of “consequents” in the system of rules, i.e., the number of the production rules. The weights of the connections from unit \( j \) active in SUBINDEX to the neural representatives of those “consequents” that are active\(^6\) in PATTERNOUT are updated by the following rule

\[ w'_{\text{Subindex}, \text{Pout}_i} = \begin{cases} 1 & \text{if } \text{Pout}_i \text{ active;} \\ 0 & \text{otherwise;} \end{cases} \quad (8) \]

Rules (6), (7), and (8) ensure that when the input set \( I_j \) is presented to the network the neurons representing the elements of \( C_j \) in PATTERNOUT are activated, in the way described in CASE 2 below.

**CASE 2:** \( \exists j : I = I_j \). When \( I \) is given as input to \( \text{CM} \) at time \( t = 0 \), the neural units representing its elements in PATTERNON and PATTERNOFF become active at \( t = 1 \). At time \( t = 2 \), the neuron representing index \( j \) in the layer INDEX becomes active, as determined by rules (6) and (7). The system has identified \( I \) with \( I_j \).

When this identification is successfully completed, \( \text{CM} \) has to retrieve from index \( j \) the second element of the chunk \( < I_j, C_j > \). Since, at time \( t = 2 \), the index neuron \( j \) in INDEX is active, then at time \( t = 3 \) the neural unit in SUBINDEX representing the same index \( j \) will become active as well. And, at time \( t = 4 \), the neurons in PATTERNOUT representing the elements of \( C_j \), as associated to the input pattern \( I = I_j \) by rule (8), will become active. Finally, at time \( t = 5 \), the neural representatives of elements of \( C_j \) in the forward chaining layer of \( \text{RM} \) are activated by direct connections from the PATTERNOUT layer of the \( \text{CM} \) module to the FORWARD layer of the \( \text{RM} \) module.

**CASE 3:** \( \forall j \ I \neq I_j \), but \( \exists j_1 \ldots j_k : I_{j_1} \subset I \land \ldots \land I_{j_k} \subset I \). \( I \) contains subpatterns \( I_{j_1}, \ldots, I_{j_k} \) which consist of the literals in the first

\(^6\)These neurons are directly activated by the neurons representing the same literals in the layer INFERRERDBF of the \( \text{RM} \) module.
element of already stored chunks. Then, the input set $I$ activates in PATTERNOUT layer of $\text{CM}$ the neural representatives of the elements of $S = \cup \{C_{j_1}, \ldots, C_{j_k}\}$.

Let us describe in more detail how this result is achieved. As in case 1, all neurons of INDEX are inactive at time $t = 2$ when $I$ is presented to $\text{CM}$ and, at time $t = 3$, the impulse of control neuron CLOCK2 (see figure 5) activates the control neurons CTRINDOFF and CTRIND. At time $t = 4$, the impulse from CTRIND, combined with the impulses from the neural representatives of the elements of the sets $I_{j_1}, \ldots, I_{j_k}$ in PATTERNON, activates the neurons $j_n$ in SUBINDEX with $n = 1, \ldots, k$. Thus, at time $t = 5$, all neural representatives of the elements of $S$ in PATTERNOUT are activated and, finally, at time $t = 6$, the neural representatives of the elements of $S$ in the FORWARD layer of $\text{RM}$ module fire.

The activation of the right neurons $j_n$ upon presentation of input pattern $I$ is determined by the weight values of connections from the PATTERNON layer to the SUBINDEX layer. These weights are initialized with the following values

$$w_{Pon_i,SubIndexj} = 0 \quad j = 1, \ldots, M, \quad i = 1, 2, \ldots, N.$$ 

The updating rule for these weights is analogous to (6):

$$w'_{Pon_i,SubIndexj} = \begin{cases} 
1/[I] & \text{if } Pon_i \text{ active;} \\
 w_{Pon_i,SubIndexj} & \text{otherwise}.
\end{cases} \quad (9)$$

One can easily show that rule (9) guarantees that, if $I_j$ is a subpattern of the new input set $I$, then at time $t = 4$ neuron $j$ in SUBINDEX is active.

SELECTING INDEXES: Let us now turn to describe the role of the layer INDEXBF and the selection criterion of indexes which is used by $\text{Recorder}$ to store a new chunk.

The algorithm $\text{Recorder}$ modifies the weights of connections from neurons CTRIND and CTRINDOFF to the neurons of INDEXBF, in order to codify, for every chunk $<I_j,C_j>$, its frequency of recall, relative to the total number of network runs. In particular, the weight of the connection from the CTRIND neuron to neuron $j$ in INDEXBF
is increased by one if and only if the presentation of set $I$ activates the neuron $j$ in INDEXBF, that is to say, when the network recognizes the situation $I = I_j$, such as described in CASE 2. At the same time, the weight of the connection from the CTRINDOFF neuron to each neuron of INDEXBF is decreased with a real value (let us call it $\alpha$). The parameter $\alpha$ can be interpreted as a frequency threshold suitably chosen by the user. Thus, if the difference between the weighted impulses from CTRIND and CTRINDOFF to the neuron $j$ is positive, then neuron $j$ of INDEXBF is activated. This signals that the frequency of recall of chunk $<I_j, C_j>$ is greater than the $\alpha$ threshold (see Pasconcino (1994) for more details).

Thus, when the CTRIND and CTRINDOFF neurons are activated (at time $t = 3$), at the next instant of time only those index neurons which have frequency of recall greater than $\alpha$ are active in INDEXBF. Every other neuron in INDEXBF is inactive either because no chunk is associated to it or because the associated chunk has recall frequency below the $\alpha$ threshold. In order to acquire a new chunk in CM Recorder randomly selects an index $j'$ among the inactive neurons of INDEXBF. It may be the case that such “drawing” procedure selects an index $j'$ to which a chunk $<A_{j'}, B_{j'}>$, with $A_{j'} \neq I$, is associated, and therefore the new chunk $<I_{j'}, C_{j'}>$ replaces the old one with a low frequency of use (below $\alpha$).

This simple criterion based on recall frequency provides a mechanism for managing the prefixed limited resources of the CM module. In other words, this drawing procedure provides a simple version of a garbage collector which, as is well known, does not eliminate the phenomenon of memory saturation, but simply reduces its incidence.

In concluding this paper, we wish to emphasize that the algorithm Recorder acquires a new chunk when presented only once with input set $I$ and output set $C$. In fact, rules (6) and (7), which enable the system to acquire new chunks, eliminate the interference between the first elements of recorded chunks, as they change exclusively the weights of the input connections to the selected index neurons in the INDEX and SUBINDEX layers. Furthermore, let us point out that one can easily

---

7Except for the first updating of this weight which increases the initial value by one plus the total number of system runs plus the total number of indexes.
show that the computational complexity of the procedure Recorder is linear with respect to the size of the sets $I$ and $C$.

A system endowed with this chunking mechanism can improve its performances on the basis of previous activity. Reduction of processing time is due both to the parallelism inherent in the general neural architecture of the system and to the specific architecture of the CM module; the latter guarantees that the computational cost of access to stored chunks is independent of the number of chunks that are in memory. It goes without saying that the actual exploitation of a reduction in processing times is contingent on the availability of a computational agent capable of modifying neural weights and executing the parallel computations allowed by this neural model.

**Appendix**

Let the rule knowledge base of the system be formed by the following set of rules

\begin{align*}
(R1) & \quad a \land b \implies c \\
(R2) & \quad c \land d \implies f \\
(R3) & \quad f \implies h
\end{align*}

and let the system be started on the following set of initial data $I = \{a, b, d\}$ and goal set $G = \{h\}$. What follows is the list of the neural units that are active at each synchronous computational step of the network, whose submodular blocks are represented in figure 6 below. From this list and the connections between neural blocks sketched out in fig. 6, the interested reader may get a more precise idea of how neural activity flows within the network.

The suffixes of the neuron labels listed below indicate which submodular block the active neuron belongs to (For example, the suffix in “d, datain” indicates that neuron $d$ belongs to the block DATAIN.) We recall that section 3 describes what is the intended semantic interpretation of the neural elements in the main blocks. The first list, clock 0 to 13, is relative to the case in which the system is started, for the first time ever, on the given sets $I$ and $G$. 
A chunking mechanism in a neural system...

Figure 6: A simplified block representation of CM, RM, and JM. Dotted lines represent inhibitory connections.
ACTIVE NEURONS( CLOCK : 0 )
Neuron Label( clock0 ); /* control neuron */
Neuron Label( d_patternon );
Neuron Label( b_patternon );
Neuron Label( a_patternon );
Neuron Label( h_patternon );
Neuron Label( h_goalin );

ACTIVE NEURONS( CLOCK : 1 )
Neuron Label( d_patternon );
Neuron Label( a_patternon );
Neuron Label( b_patternon );
Neuron Label( d_patternoff );
Neuron Label( a_patternoff );
Neuron Label( b_patternoff );
Neuron Label( clock1 ); /* control neuron */
Neuron Label( h_goal );
Neuron Label( h_goalin );

ACTIVE NEURONS( CLOCK : 2 )
Neuron Label( h_goal );
Neuron Label( h_goal );
Neuron Label( ctr_goal ); /* control neuron */
Neuron Label( clock2 ); /* control neuron */
Neuron Label( h_patternoff );
Neuron Label( a_patternoff );
Neuron Label( b_patternoff );
Neuron Label( a_patternoff );
Neuron Label( d_patternoff );

ACTIVE NEURONS( CLOCK : 3 )
Neuron Label( d_patternon );
Neuron Label( a_patternon );
Neuron Label( b_patternon );
Neuron Label( d_patternoff );
Neuron Label( a_patternoff );
Neuron Label( b_patternoff );
Neuron Label( clock3 ); /* control neuron */
Neuron Label( ctrInd ); /* control neuron */
Neuron Label( ctr_patternoff ); /* control neuron */
Neuron Label( ctr_goal ); /* control neuron */
Neuron Label( h_goal );
Neuron Label( h_goal );

ACTIVE NEURONS( CLOCK : 4 )
Neuron Label( h_goal );
Neuron Label( h_goal );
Neuron Label( ctr_goal ); /* control neuron */
Neuron Label( clock4 ); /* control neuron */
Neuron Label( ctr_patternoff ); /* control neuron */
Neuron Label( a_patternoff );
Neuron Label( b_patternoff );
Neuron Label( b_patternon );
Neuron Label( a_patternoff );
Neuron Label( d_patternon );

ACTIVE NEURONS( CLOCK : 5 )
Neuron Label( d_patternon );
Neuron Label( a_patternon );
Neuron Label( b_patternon );
Neuron Label( d_patternoff );
Neuron Label( a_patternoff );
Neuron Label( b_patternoff );
Neuron Label( clock5 ); /* control neuron */
Neuron Label( ctr_patternoff ); /* control neuron */
Neuron Label( h_goal );
Neuron Label( h_patternoff );
Neuron Label( ctr_patternactive );
Neuron Label( h_patterngoal );
Neuron Label( a_patternactive );

ACTIVE NEURONS( CLOCK : 6 )
Neuron Label( h_forward );
Neuron Label( a_forward );
Neuron Label( d_forward );
Neuron Label( h_backward );
Neuron Label( query );
Neuron Label( h_subgoal );
Neuron Label( h_patterngoal );
Neuron Label( h_goal );
Neuron Label( h_patternoff );
Neuron Label( a_patternoff );
Neuron Label( d_patternoff );

ACTIVE NEURONS( CLOCK : 7 )
Neuron Label( d_patternon );
Neuron Label( a_patternon );
Neuron Label( b_patternon );
Neuron Label( d_patternoff );
Neuron Label( a_patternoff );
Neuron Label( b_patternoff );
Neuron Label( ctr_patternactive );
Neuron Label( h_patterngoal );
Neuron Label( h_forward );
Neuron Label( query );
A chunking mechanism in a neural system...

Neuron Label( fforward );
Neuron Label( cforward );
Neuron Label( dforward );
Neuron Label( aforward );
Neuron Label( hforward );

ACTIVE NEURONS( CLOCK : 8 )
Neuron Label( bforwardbf );
Neuron Label( aforwardbf );
Neuron Label( dforwardbf );
Neuron Label( aforwardbf );
Neuron Label( hforwardbf );
Neuron Label( hgoal );
Neuron Label( inferredbf );
Neuron Label( inferredbf );
Neuron Label( inferredbf );
Neuron Label( inferredbf );
Neuron Label( inferredbf );
Neuron Label( inferredbf );
Neuron Label( inferredbf );
Neuron Label( inferredbf );
Neuron Label( inferredbf );
Neuron Label( inferredbf );
Neuron Label( inferredbf );
Neuron Label( inferredbf );
Neuron Label( inferredbf );

ACTIVE NEURONS( CLOCK : 9 )
Neuron Label( bpatternout );
Neuron Label( apatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternoff );
Neuron Label( dpatternoff );
Neuron Label( dpatternoff );
Neuron Label( dpatternoff );
Neuron Label( dpatternoff );
Neuron Label( dpatterngoal );
Neuron Label( dpatterngoal );
Neuron Label( saturat );

ACTIVE NEURONS( CLOCK : 10 )
Neuron Label( bpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );

ACTIVE NEURONS( CLOCK : 11 )
Neuron Label( bpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );
Neuron Label( dpatternout );

When processing of the input terminates, as signalled by the active state of the neuron end, the algorithm Recorder stores in CM the chunk $<I_6 = \{a, b, d\}, C_6 = \{c, f, h\}>$ which is associated to index 6, represented by the neuron $\text{index6}_6$ in the block INDEX and $\text{index6}_6$ in SUBINDEX.

The second list below concerns the case in which the same set of inputs is presented to the system: $I = \{a, b, d\}, G = \{h\}$. There is a reduction in processing time from the first to the second session: the steps needed by the system to infer the same set of literals are now 11 (instead of 13). Naturally, more marked reductions in processing time are possible for systems of rules involving longer inferential chains. The simulation of activity of the whole neural system is carried out by means of a PROLOG program running on LPA MacPROLOG Version 3.5 environment.
A chunking mechanism in a neural system...

ACTIVE NEURONS (CLOCK : 0)
Neuron Label( d_patternon ); /* control neuron */
Neuron Label( d_datain );
Neuron Label( h_datain );
Neuron Label( h_patternoff );
Neuron Label( a_patternoff );
Neuron Label( h_goalin );

ACTIVE NEURONS (CLOCK : 1)
Neuron Label( d_patternon );
Neuron Label( a_patternon );
Neuron Label( h_patternon );
Neuron Label( d_patternoff );
Neuron Label( a_patternoff );
Neuron Label( h_patternoff );
Neuron Label( clock1 );
Neuron Label( h_goal );
Neuron Label( h_goal );
Neuron Label( index );
Neuron Label( indexout ); /* recalled literal */
Neuron Label( index ); /* identified chunk */

ACTIVE NEURONS (CLOCK : 2)
Neuron Label( d_patternon );
Neuron Label( a_patternon );
Neuron Label( h_goal );
Neuron Label( ctr_goal );
Neuron Label( clock2 );
Neuron Label( h_patternoff );
Neuron Label( a_patternoff );
Neuron Label( h_patternoff );
Neuron Label( a_patternon );
Neuron Label( h_patternon );
Neuron Label( a_patternoff );
Neuron Label( index6_index ); /* identified chunk */

ACTIVE NEURONS (CLOCK : 3)
Neuron Label( index6_indexbf );
Neuron Label( index6_subindex );
Neuron Label( index6_index );
Neuron Label( d_patternon );
Neuron Label( a_patternon );
Neuron Label( h_patternon );
Neuron Label( d_patternoff );
Neuron Label( a_patternoff );
Neuron Label( h_patternoff );
Neuron Label( index6_index );

ACTIVE NEURONS (CLOCK : 4)
Neuron Label( d_datain );
Neuron Label( d_goal );
Neuron Label( clock4 );
Neuron Label( h_patternoff );
Neuron Label( a_patternoff );
Neuron Label( h_patternoff );
Neuron Label( h_patternon );
Neuron Label( a_patternnon );
Neuron Label( h_patternnon );
Neuron Label( d_patternon );
A chunking mechanism in a neural system...

References


Judd J.S. (1990), Neural Network Design and the Complexity of Learning, The MIT Press.
Miller G.A. (1956), The magic number seven plus or minus two: Some limits on our capacity for processing information, Psychological Review, 63, 81-97.