

Building Production Systems with Realistic Spiking Neurons

Terrence C. Stewart (tcstewar@uwaterloo.ca)

Chris Eliasmith (celiasmith@uwaterloo.ca)

Centre for Theoretical Neuroscience, University of Waterloo
Waterloo, ON, Canada, N2L 3G1

Abstract

We present a novel cognitive architecture built from realistic spiking neurons that exhibits basic production system capabilities. It uses Holographic Reduced Representations to encode structured information and the Neural Engineering Framework to create detailed neuro-biologically plausible networks of spiking neurons for storing and manipulating these representations. We demonstrate this system's abilities to encode IF-THEN rules and manipulate its own representations in response to the current state. This leads to predictions about the sorts of rules that would be difficult for neurons to encode, and the maximum complexity of such rules. Our system bridges the gap between high-level cognitive architectures (including ACT-R) and modern neuroscience research, allowing details such as the numbers, types, and connections of neurons to be related to cognitive behaviour.

Keywords: neural production systems; neural engineering framework; holographic reduced representations; ACT-R; cognitive architectures; compositionality

Introduction

Human cognition requires representation and combination of concepts, and the manipulation of these concepts. These representations must also allow for internal structuring, so that “dog chases cat” is different from “cat chases dog”. Furthermore, cognition entails modifying, recognizing, and acting on the basis of these structures. Exactly how such compositionality can occur is a fundamental question for cognitive science. Aspects of this process have been identified as challenges that must be met by any theory of human cognition (e.g. Jackendoff, 2002).

Cognitive theories approach this problem in different ways. Some, such as ACT-R (Anderson & Lebiere, 1998), simply posit that such symbolic manipulation and storage is possible, and build up the architecture from there. However, models are starting to be developed which describe how symbolic representations can be built using neurons. Both LISA (Hummel & Holyoak, 2003) and Neural Blackboard Architectures (van der Velde & de Kamps, 2006) can form such structures, although neither is used to manipulate such structures to perform cognition.

Neural Modelling Rationale

Before describing our model, it is important to justify the creation of such a model. Many cognitive scientists have noted that neurons are at the wrong level of description for cognitive behaviour (e.g. Pylyshyn, 1984). Even the models that are “neurally inspired” generally use highly idealized neurons, avoiding the issues of how a series of voltage spikes can represent a given value, how to deal with random

firing noise, the effects of different neurotransmitters, and the hundreds of different types of neurons in the human brain. Furthermore, if a neural level of description is desired, then it is unclear as to whether we should require a molecular, atomic, or quantum level of description. Below a certain level, there may be no advantage to going deeper, as the lower level implementation does not significantly affect the upper level behaviour.

We believe that the neural level should be included in cognitive science theories. Thanks to modern neuroscience advances, the neural level is one that we know a great deal about. The major processes involved in neuron behaviour and their responses to a wide range of stimuli are significantly more understood than any other level of brain phenomena. Furthermore, there is a large body of evidence as to the effects of various influences (neurotransmitters, alcohol, nicotine, etc.) both on neurons and on high-level overt behaviour. Theories of cognition that bridge this gap can be tested on the basis of this sort of evidence.

We also believe that neural-level effects do trickle up to the higher levels in useful and interesting ways. The effects of approximations and variability due to neural implementation may be empirically detectable in overt behaviour. A neural cognitive theory will perform differently from a purely symbolic model, and these differences can inform our understanding of cognition.

Our Modelling Approach

The goal of our research is a realistic neural model of high-level cognitive behaviour. This is the sort of behaviour that is traditionally modelled using symbolic approaches such as ACT-R. We have developed a model that uses spiking neurons, connected in a plausible manner, that is capable of performing basic cognitive tasks. Furthermore, while its overt behaviour is similar to that of a purely symbolic model, it differs in important ways. For example, performance degrades as structures become more complex.

This neural cognitive architecture makes use of two separate areas of research. First, we use Holographic Reduced Representations (HRRs; Plate, 2003) for storing and manipulating structured representations. Importantly, this allows us to represent atomic concepts, structures of concepts, and manipulations of those structures using the same representational form: a simple vector of numbers.

The second area used is the Neural Engineering Framework (NEF; Eliasmith & Anderson, 2003). This describes how groups of neurons can represent and transform numerical values in a robust manner that is plausible given current neuroscientific evidence. It includes

details handling different types of neurons, the various neurotransmitters, connectivity issues, and firing patterns.

From these, we have developed a basic neural production system. Our system can follow a set of rules (productions), where each rule contains a condition in which it should be applied and an action to perform. For example, a rule might state “if dog chase cat, then scold dog” or “if asked three plus four, then say seven”. These sorts of rules are the basis of symbolic cognitive architectures, and have been widely successful in modelling aspects of human cognition. They are able to detect complex situations, manipulate symbols to determine a response, and generate that response.

In this paper, we describe our system and present some preliminary results. The similarities and differences between our system and ACT-R are provided. We further highlight the advantages that a neural cognitive model provides, including aspects of the neural level that affect understanding of high-level behaviour (and vice-versa).

Holographic Reduced Representations

A Vector Symbolic Architecture (VSA) is a method for using a distributed representation to encode structure (see Gayler, 2003 for an overview). In this work, we use a VSA known as Holographic Reduced Representations (HRRs), developed by Plate (1995). Information is represented as a vector in a high-dimensional space (usually more than 100 dimensions). For example, the symbol **cat** might be represented by the vector [-0.3,0,0.4,-0.6,0.6]. For our work, the representations for all such atomic symbols are chosen randomly from a high-dimensional unit sphere.

The important feature of HRRs is that representations can be combined to form a new value that is also a vector in the *same number of dimensions*. That is, if a neural structure is capable of storing a simple, atomic HRR, it can also represent combinations of HRRs. Importantly, these combined representations can later be decomposed into their original constituents. For example, an HRR representing “dogs chase cats” might be calculated as follows:

$$\text{dogs} \otimes \text{subject} + \text{chase} \otimes \text{verb} + \text{cats} \otimes \text{object} \quad (1)$$

Each of the elements in this equation (**dogs**, **subject**, etc.) are atomic HRRs (defined as per the previous paragraph as random unit vectors). The + sign represents standard numerical addition. The \otimes sign indicates a more complex process, *cyclic convolution*. For $C=A \otimes B$, this is defined as:

$$c_j = \sum_{k=0}^{N-1} a_k b_{j-k \bmod N} \quad (2)$$

The key feature that makes HRRs (and VSAs in general) useful for representing structure is that the cyclic convolution can be *inverted*. That is, given a structured HRR such as that in (1), we can extract out the original atomic symbols that formed it. For example, if we want to know what the subject is, we perform a cyclic convolution between (1) and the *inverse* of the HRR for **subject**. The inverse is formed by rearranging the original values:

$$(a_0, a_{N-1}, a_{N-2}, \dots, a_1) \quad (3)$$

Writing the inverse of A as A^T , in general $B \otimes A \otimes A^T \approx B$. After cyclic convolution, the resulting value will be an approximation of the original value that was combined with **subject**; in this case, **dogs**. This makes HRRs a lossy form of representation, so we cannot pack an infinite amount of information into a single HRR. Plate (1995) showed that the storage capacity increases exponentially with more dimensions.

For HRRs, similarity is defined as the cosine of the angle between the two values (equivalently, the normalized dot product). This means that $A+B$ is similar to both A and B, while $A \otimes B$ is not similar to either A or B. This will be important for using HRRs to build production systems.

Production systems

Since HRRs (and other VSAs) exhibit compositionality, they could be used to implement systems similar to standard symbolic production systems. This requires a representation of the *current state* and a set of IF-THEN rules which identify what to do in different states. The current state is normally represented as a set of slot/value pairs. For example, if the model is in the process of counting and is currently at the value three, the current state might have the value **counting** in the slot **mode** and the value **three** in the slot **value**. This would be represented in an HRR as:

$$\text{mode} \otimes \text{counting} + \text{value} \otimes \text{three} \quad (4)$$

In some production systems (such as ACT-R), the state is divided into multiple buffers. In an HRR, these can be represented by nesting the representations. For example, if, while counting, the model is also looking at a picture of a cat, the visual buffer would hold this information while the goal buffer holds the information about counting.

$$\text{goal} \otimes (\text{mode} \otimes \text{counting} + \text{value} \otimes \text{three}) + \text{visual} \otimes (\text{isa} \otimes \text{picture} + \text{value} \otimes \text{cat}) \quad (5)$$

Given these state representations, an IF-THEN rule could then indicate that if the **mode** is **counting** and the **value** is **three**, then we should change the **value** to **four**. The IF portion of this production rule would be defined by

$$\text{goal} \otimes (\text{mode} \otimes \text{counting} + \text{value} \otimes \text{three}) \quad (6)$$

We can determine whether the rule applies to the current state by measuring the similarity between this value and the current state. Since the state (5) is the same as the rule's IF pattern (6) plus the extraneous extra values from the visual buffer, these will be similar (i.e. have a dot product significantly above zero), indicating the rule can be applied.

To apply a rule, we need a representation of the THEN portion. Since this is meant to be an action to be performed, this can be represented by an HRR value that is to be *combined* via cyclic convolution with the current state. For example, to output **four** given that the current counting value is **three**, the THEN portion of the above rule could be:

$$\text{four} \otimes \text{three}^T \quad (7)$$

When combined with the current state given in (5), we get a value that is approximately the following:

$$\text{goal} \otimes \text{value} \otimes \text{four} \quad (8)$$

More complex production rules (both in terms of the IF and THEN portions) are discussed later. The accuracy of such a system, in terms of the amount of extraneous information that can be successfully ignored and the amount of nesting that can occur, can be increased by using more dimensions. The system is also similarly robust to random noise being added onto the represented value. This is important when using realistic neurons to implement an HRR-based production system.

Neural Engineering Framework

The Neural Engineering Framework (NEF; Eliasmith & Anderson, 2003) provides a methodology for understanding how physical neurons represent and manipulate information. This is based on the idea that information is represented by *neural groups* and the connection weights between neural groups can be seen as *transformations* of these representations. It has been used to model a variety of neural systems, including the owl audition (Fischer et al., 2007) and rodent navigation (Conklin & Eliasmith, 2005).

A neural group is a set of neurons with a realistically heterogeneous range of neural properties (i.e. maximum firing rates, refractory periods, neurotransmitters, etc.). The pattern of firing across these neurons can be seen as a representation of a particular value. For example, one particular firing pattern might represent the vector $[-0.3, 0.4, -0.6, 0.6]$ (used in the previous section to represent the symbol **cat**). Importantly, the number of dimensions in the vector is *not* the same as the number of neurons in the neural group. This is in contrast to standard neuron representation schemes where there is a direct mapping between particular neuron firing rates and values in the vector being represented.

More precisely, we can define a mapping from a particular value we want to represent and the firing pattern for each neuron in the neural group. This is done by assigning *encoding vectors* to each neuron. This encoding vector is the vector for which the neuron will fire the strongest. The precise details of this mapping will vary depending on the type of neuron (and the degree of accuracy to which the neuron is being simulated). In general, the activation a of a particular neuron i to represent a value \mathbf{x} is:

$$a_i = G_i(\alpha_i \tilde{\phi}_i \cdot \mathbf{x} + J_i^{bias}) \quad (9)$$

Here, α is the neuron gain or sensitivity, $\tilde{\phi}$ is the encoding or preferred direction vector, and J^{bias} is a fixed input current to model background neural activity. G is the *response function*, which is determined by what sort of neuron is being modelled, including its particular resistances, capacitances, maximum firing rate, and so on. In our work, we use the response function for the leaky integrate-and-fire (LIF) model, which is widely used for its reasonable trade-off between realism and computational requirements. NEF can easily make use of more detailed models simply by changing this response function.

Using this approach, we can directly translate from a particular value that we want to represent (\mathbf{x}) to the spike

trains over time of each neuron in the group (a_i). The value being represented is distributed across all of the neurons.

Decoding Vectors

If we have the firing pattern for a neural group, we can also determine what value is currently being represented. This is the reverse of the encoding process. This is more complex than encoding, and in general it is impossible to perfectly recover the original value from the firing pattern. However, we can determine an *optimal linear decoder*. This is a set of vectors ϕ_i (one for each neuron) which are multiplied by the activation level of each neuron to get a sum that is the best possible linear approximation of the original value (Eliasmith & Anderson, 2003).

$$\phi = \Gamma^{-1}Y; \Gamma_{ij} = \int a_i a_j dx; Y_j = \int a_j x dx \quad (10)$$

Importantly, this formula gives an approximation that is highly robust to random variations in the firing rates of neurons (and to neuron death). The representations can thus be made as accurate as desired by increasing the number of neurons used, as shown in Figure 1.

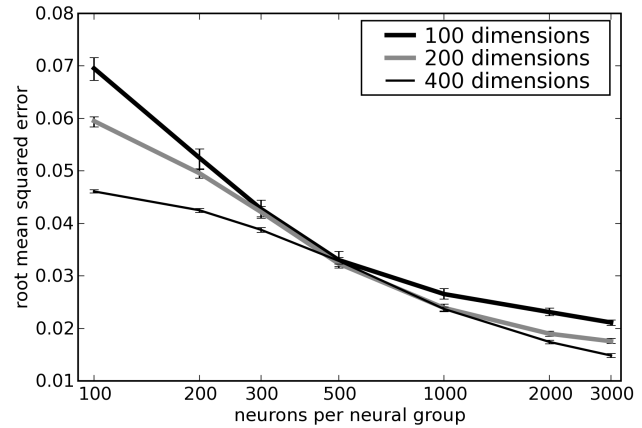


Figure 1: Representational accuracy for varying numbers of neurons per neural group. HRRs with 100, 200, and 400 dimensions are shown, using default neural parameters.

Connecting Neural Groups

To make use of neural groups, we need to connect them so as to perform the desired manipulations of the values they are representing. As with most neural models, interactions between neurons are via the synaptic connection weights.

The most common way of connecting neural groups in NEF is to directly calculate what the connection weights should be. This bypasses the question of how these weights would have developed over time.

The simplest case is a transformation that does nothing. That is, if we have two neural groups (A and B), and we set the value of group A to be \mathbf{x} , we want group B to also represent \mathbf{x} . This can be seen as the direct transmission of information from one location in the brain to another. For this situation, the optimal connection weights between each neuron i in group A and each neuron j in group B are:

$$\omega_{ji} = \alpha_j \tilde{\phi}_j \cdot \phi_i \quad (11)$$

If this formula is used to determine the strength of the synaptic connection between the neural groups, then group B will be driven to fire such that it represents the same value as group A. As noted in the previous section, the accuracy of this representation will be dependent on the number of neurons in the groups. Importantly, this system works even though none of the neurons in the two neural groups will have exactly the same encoding vector (and thus firing pattern). That is, there will generally not be a one-to-one correspondence between any neurons in the groups.

We can also connect neural groups in such a way as to *transform* the value from A to B. That is, we can set the synaptic weights so that B represents the result of any desired function $f(x)$. This is done using the same formula as (11), but the direct decoding vector ϕ_i is replaced by a specialized one determined using the following modification of (10):

$$Y_j = \int a_j f(x) dx \quad (12)$$

As we have previously shown (Eliasmith, 2005), this is sufficient for implementing accurate circular convolution, requiring only a single intermediate neural layer.

Learning Transformations

Instead of specifying the transformation function between two neural groups, and thus manually specifying the synaptic connection weights, we can also have these weights be learned. Given a set of desired input activations (a_i) and the corresponding output activations (b_i), the following learning rule can be used.

$$\Delta \omega_{ij} = -\kappa \left(\sum_i \omega_{ij} a_i - \sum_j \omega_{ij} b_j \right) \quad (13)$$

This is a local, Hebbian learning rule, where updating each neuron's synaptic weights only requires information that is available to that neuron, making it biologically plausible. However, a precise biochemical mechanism for this sort of learning has not yet been identified.

Neurobiological Realism

An important feature of the Neural Engineering Framework is that the general approach continues to be applicable no matter how detailed the neural model is. It can be applied to rate neurons, leaky integrate-and-fire neurons, adaptive LIF neurons, and even the highly complex compartmental models that require supercomputers to simulate the firing pattern of a single neuron. This means that as we determine more accurate details of the particular neurons involved in a cognitive behaviour, we can add those constraints into the cognitive model without disrupting the overall system. Furthermore, simulations can first be done using a simplistic neural model requiring less computing power, and then once a suitable model is created a more detailed neural model can be used to generate precise predictions about firing patterns, representational accuracy, etc.

One example of this involves the synaptic connection weights. In general, the approach described above results in both positively and negatively weighted connections. This

is not consistent with what is found in real brains, where positive (excitatory) and negative (inhibitory) weights use different neurotransmitters and different types of neurons. However, Eliasmith and Anderson (2003, section 6.4) show an alternate approach to determining weights that separates excitatory and inhibitory connections.

A Neural Production System

While the NEF has been used successfully to create models of complex perception activity, our current work aims to apply it to a cognitive domain. This is an extension of previous work (Eliasmith, 2005) where state-based rule following behaviour was shown using NEF and HRRs. However, the state which identified the HRR transformation rule to apply was merely a single numerical value (1 or 0). In full production systems, the current state which determines the rule to follow is much more complex.

Figure 2 depicts the core organization of our neural production system. One neural group stores the current state. This is used by the associative memory system to determine which production rule is to be used next. To do this, we can train a network using the learning rule given in (13), which would form a simple associative memory between particular states and the rule to be applied.

Once this rule is selected, it can be applied by performing a cyclic convolution with the current state. This allows rules to generalize over multiple situations, as will be demonstrated below. The resulting value can be interpreted as the chosen actions to perform (the THEN part of the production rule). This output can also feed back to the current state, allowing complex sequenced action.

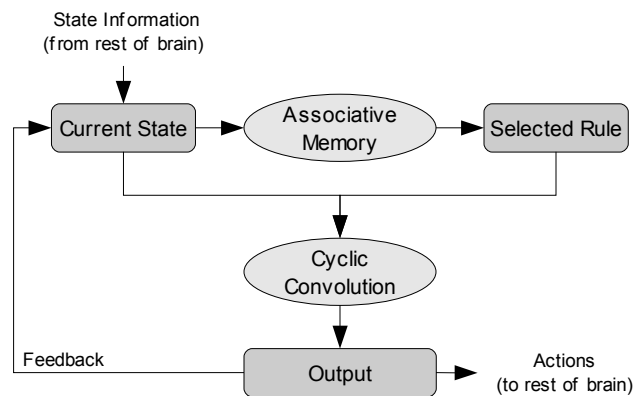


Figure 2: The core of a neural production system. Boxes represent neural groups and ovals represent calculations performed by synaptic connections between neural groups.

Simple Rule Following

To demonstrate the basic capabilities of this system, we can implement a model of simple addition. This is based on the model in the ACT-R unit 1 tutorial, where it is presumed that experts at addition can add numbers directly without recourse to declarative memory to explicitly recall addition facts. While implementing a declarative memory system can be done with HRRs and NEF, this is not covered here.

The productions in this system are shown in Table 1. They can be implemented by training the associative memory using each pair. This will cause the correct rule to be represented in the Selected Rule neural group, which will be convolved with the current state to produce an output. As many productions as desired can be added in this way.

Table 1: Production rules for expert addition.

IF	THEN
state@adding+add1@one+add2@one	two@state ^T @adding ^T
state@adding+add1@one+add2@two	three@state ^T @adding ^T
state@adding+add1@one+add2@three	four@state ^T @adding ^T
state@adding+add1@two+add2@two	four@state ^T @adding ^T
state@adding+add1@two+add2@three	five@state ^T @adding ^T

With this associative memory defined, if the state is similar to state@adding+add1@one+add2@three, then the selected action will be four@state^T@adding^T. This rule is then applied by convolving it with the current state.

$$(state@adding+add1@one+add2@three)@(four@state^T@adding^T)$$

Due to the nature of the \otimes and $+$ rules for VSAs, this is equivalent to the following value

$$four+(add1@one+add2@three)@four@state^T@adding^T$$

As a result, the output will be similar to the value for **four**. The system is thus capable of simple rule following. The accuracy of this rule following will be dependent on the complexity of the rule, and the number of neurons in the representation, as shown in Figure 3. This is a lower bound on accuracy; performance improvements are underway.

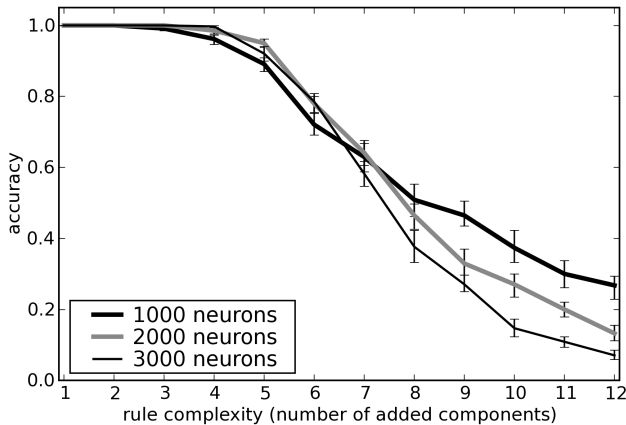


Figure 3: Accuracy of rule selection as the number of components in the rule is changed.

Production Rules With Variable Binding

Production systems do not generally have specific rules for every single possible state. We often want to be able to have values in the output be based on current state values. This is referred to as *variable binding* in ACT-R. For example, consider a production system for scolding any animal that chases any other animal. If this is represented with states such as subject@dog+verb@chases+object@cat, and we want an output such as scold@dog, we would not want to have to add productions for each possible type of animal. Instead, we can just add the rule shown in Table 2.

Table 2: Production rule for scolding animals

IF	THEN
verb@chases	scold@subject ^T

This will match any situation where an animal is chasing anything. Since the THEN portion is convolved with the current state, this results in the following:

$$(subject@dog+verb@chases+object@cat)@(scold@subject^T) = scold@dog+(verb@chases+object@cat)@(scold@subject^T) \approx scold@dog$$

Iteration

Since the output value can be fed back to adjust the current state, the production system can be used to perform a sequence of actions. Given the following simple production rules and an initial state of **one**, the system will count from one to five.

Table 3: Production rules for counting.

IF	THEN
number@one	two@one ^T
number@two	three@two ^T
number@three	four@three ^T
number@four	five@four ^T

Given to the temporal aspect of this example, it is worth observing that the actual behaviour is affected by the biochemical characteristics of the neurons. If the neurons use a neurotransmitter that leads to a quickly decaying post-synaptic current, then the representations will change quickly, while longer lasting neurotransmitters will require significantly more time. As our model is further developed, we believe these details will provide an explanation for the amount of time needed for a production to fire (generally assumed to be ~50msec regardless of rule complexity).

Slot Detection

The production rule examples thus far are based on a simple similarity measure between the state and some particular value specified by the rule. While this is straightforward to implement using the Hebbian synaptic connection weights given in (13), there are types of rules which cannot work in this way. For example, in ACT-R, it is possible for a rule to indicate that it should only fire if a particular slot *exists* in the current state, regardless of its value.

As there is no value that all matching states will be similar to, this situation cannot be identified using the associative memory system described above. However, similar functionality can be achieved by adding a *slot detection module* to the system. This is a separate set of neural groups that can be controlled by the output in Figure 2 and whose output affects the current state of the system. Using a variation of a butterfly cleanup memory (Singh, 2005), this system can be instructed to inspect a particular slot and add detected@yes to the state if any valid value is in that slot (and detected@no otherwise). Production rules can thus match on this state, rather than directly specifying the slots they require to be valid.

This approach differs from the standard production rules in two ways. First, an extra step is required to tell the slot detection module what slot to detect, thus requiring more time. Second, only one slot can be detected at a time, so it is impossible to have two rules which need to detect different slots at the same time. While this could be improved by adding a few more independent slot detection modules, another option is to re-examine existing cognitive models to see whether we do in fact need to detect separate slots at the same time.

The slot detection module can also be used to detect whether two (or more) different slots have the same value. This is done by asking the detection system to look for the sum of the slots, since $A \otimes X + B \otimes X = (A+B) \otimes X$. This is also a common feature of ACT-R models.

Negative Matching

We can also enhance the model to account for rules that should only be applied if a particular pattern does not exist. For example, a rule might match on $A \otimes B$, but should not be applied if $A \otimes B + C \otimes D$. To achieve this, we add a second associative memory system in parallel to the first one, but whose output value is subtracted from the Selected Rule neural group in Figure 2. The rule $A \otimes B$ (and its associated THEN value) is put in the first associative memory, while the rule $A \otimes B + C \otimes D$ is put in the second one. This allows the rule to be selected if $A \otimes B$, but not when $C \otimes D$.

This capability is necessary for many ACT-R models. However, it is still unclear how to form a rule that uses negative matching on variables, which is also present but less common in ACT-R.

Conflict Resolution

The previous examples have all assumed that only one rule can apply at a given time. In general, multiple rules could match at once, but only one should be used at a time. In ACT-R, each rule has a strength value, used to select a rule probabilistically when multiple rules could apply.

The associative memory system described here does support rules having a certain strength by scaling the stored vector. However, it will always choose the rule with the highest strength. A more flexible approach would be to use a variation of a cleanup memory system (Singh, 2005) with a winner-take-all mutual inhibition system.

Results and Current Research

Our system demonstrates the basic functionality needed for a neural production system. We can model rule-following behaviour using arbitrarily realistic spiking neurons, and determine the behavioural properties of such a system. However, further work must be done to explore the connection between the neural and behavioural levels.

First, we have shown that the number of atomic concepts available in the system relates to the number of neurons in the neural groups (Figure 1). Given this, we estimate that 1,000,000 neurons is sufficient to store 10,000 atomic

concepts (plus structures built up from them). For comparison, other methods which do not make use of VSAs (van der Velde & de Kamps, 2006; Hummel & Holyoak, 2003) require many billions of neurons, while there are only 100 billion neurons in the brain. This result uses the characteristics of typical neurons, and will be further tuned as parts of our model are matched to particular brain regions and the characteristics of neurons found within.

Second, the implementational details of our model affect the high-level behaviour. The time taken to select a rule and apply it is based on the properties of the neurons themselves, rather than being imposed on the system via a "clock" signal or other pattern generator. This allows for investigation into situations that can affect this timing. We also found (Figure 3) that accuracy is decreased as the number of components in a rule increase. This forms a natural limitation on the complexity of cognitive rules, and relates to the ACT-R suggestion to only use 7 ± 2 slots. We have also indicated that particular sorts of rules that are standard in production systems are problematic to implement in a neurally plausible manner. Adding these limitations to ACT-R provides new constraints that may lead to more accurate cognitive models.

References

- Anderson, J. R. & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Erlbaum.
- Conklin, J., & Eliasmith, C. (2005). An attractor network model of path integration in the rat. *Journal of Computational Neuroscience*, 18, 183-203.
- Eliasmith, 2005. *Cognition with neurons: A large-scale, biologically realistic model of the Wason task*. Cognitive Science Society Conference. Stresa, Italy: July 2005.
- Eliasmith, C., & Anderson, C. H. (2003). *Neural engineering: Computation, representation and dynamics in neurobiological systems*. Cambridge, MA: MIT Press.
- Fischer, B., Pena, J.L., & Konishi, M. (2007). Emergence of multiplicative auditory responses in the midbrain of the barn owl. *Journal of Neurophysiology*, 98, 1181-1193.
- Gayler, R. (2003). *Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience*. ICCS/ASCS International Conference on Cognitive Science, Sydney: University of New South Wales.
- Hummel, J. E., & Holyoak, K. J. (2003). A symbolic-connectionist theory of relational inference and generalization. *Psychological Review*, 110, 220-264.
- Jackendoff, R. (2002). *Foundations of language: Brain, meaning, grammar, evolution*. Oxford University Press.
- Plate, T. (1995). Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3), 623-641.
- Pylyshyn, Z. (1984) *Computation and cognition*. Cambridge: MIT Press.
- Singh, R. (2005). *Cleanup memory in biologically plausible neural networks*. (MASC, University of Waterloo).
- van der Velde, F., & de Kamps, M. (2006). Neural blackboard architectures of combinatorial structures in cognition. *Behavioral and Brain Sciences*, 29, 37-70.