Chapter 6

Biological cognition – memory and learning

6.1 Extending cognition through time

A unlike in a traditional computer architecture, memory and learning are tightly coupled to computation in biological brains. For many, this marks one of the most fundamental differences between neural computation and standard digital computer computation. Consequently, any neural architecture would be incomplete without a consideration of memory and learning.

I have grouped memory and learning together because both allow biological systems to extend their cognition through time. That is, memory of recent or long past events can be very important for determining an appropriate behaviour in the present. Similarly, learning can often be thought of as the tuning of behaviour based on past experience, again resulting in current behaviour being sensitive to events that occurred in the past.

Conceptually speaking, memory and learning are very difficult to distinguish. Traditionally, in the neural network literature, when researchers speak of "learning", they are referring to changing the connection weights between neurons in the network. The ubiquitous "learning rules" proposed by these researchers are a means of specifying how to change connection weights given the activity of neurons in the network. However, notice that the example of learning that we have seen earlier in the book (section ref?) looks like a standard case of learning, but depends only on a memory system. That is, no connection weights are changed in the network even though the system learns a general rule that allows past experi-

ence to change what behaviors are chosen in the present.

Similarly, the traditional contrast between short-term (or 'working') memory[1] (which is often taken to be activity in a recurrent network; refs?), and long-term memory (which is taken to be connection weight changes; refs?), is difficult to sustain in the face of neural evidence. For example, there is good evidence that in the hippocampus there are very rapid weight changes (refs?). Consequently, some 'short-term' memories might be stored in the system in virtue of connection weight changes. In sum, the brain adapts to different timescales in many different ways, some are more closely related to neural activity, others are more closely related to connection weight changes.

Undoubtedly, the neural systems important for long-term memory, short-term memory, working memory, etc. are tightly integrated in the brain. And, furthermore, they exploit activity-based, and connection weight-based mechanisms in sophisticated and poorly understood ways. All of this is to say that I am not going to provide a general description of memory and learning, but rather considerations of how activity based and connection weight-based mechanisms for adaptation can be integrated into the semantic pointer architecture.

Specifically, this chapter is going to address two challenges, one related to activity-based explanations of working memory, and the other related to learning connection weight changes. There are many neurally plausible models of working memory that rely on recurrently connected networks (ref?). These models are able to explain the sustained firing rate found in many parts of cortex during the delay period of memory experiments. Some are even able to explain more subtle dynamics, such as ramping up and down of single cell activity seen during these periods (ref?). However, none of these models address complex working memory tasks that are essential for cognition, such as serial working memory (i.e. remembering an ordered list of items). This is largely because methods for controlling the loading of memories, clearing the network of past memories, and constructing sophisticated representations, have not been devised. In the next two sections, I describe a model that addresses many of these limitations. As well, the tutorial at the end of this chapter demonstrates how to build a simple working memory.

The second challenge, of introducing a biologically plausible mechanism for

[1] Sometimes these terms are distinguished and sometimes they are not (refs?). These additional distinctions do not challenge the point being made here, which is a consequence of the fact that kinds of memory are typically behaviourally defined, and the timescales of different neural mechanisms (some activity-based, some connection-based) overlap. Consequently, there are medium-time-scale behaviors that are activity-based, and others that are connection-based (and no doubt others that are a combination of the two).

connection weight adaptation, is dealt with in the last two sections. There, I describe a biologically realistic learning rule that is able to learn linear and nonlinear transformations of the representations used in the SPA. I demonstrate the rule by showing that it can be used to perform action selection in the model of the basal ganglia (section 5.2), that it can be used to learn reasoning strategies in different contexts to solve language processing tasks, and also that it can be used to learn the binding operation employed in the SPA (i.e., circular convolution). The tutorial at the end of the next chapter demonstrates how to use this rule in Nengo.

6.2 Working memory

Working memory is typically characterized as an actively engaged system used to store information that is relevant to the current behavioral situation. Typical tests of working memory in monkeys consist of having a monkey fixate at a central location, and presenting a target stimulus somewhere in the periphery of the monkeys visual field. The stimulus is removed, a delay of about 3 to 6 seconds long is then initiated, and finally the fixation target changes to indicate to the monkey that it should respond by saccading or pointing to the remembered location of the stimulus. Many experiments like these have been used to characterize the activity of single cells during working memory tasks (refs?). And, many models have been built that explain a wide variety of the properties of single cell tuning curves observed during such tasks (refs?).

However, a very common feature of working memory tasks encountered by humans, and also probably many other animals, is that they require not only the memory of a single object, but of several objects at the same time. In addition, often the order in which the information is presented is relevant to using it appropriately (e.g. a telephone number). The ability to store and recall items in *order* is called serial working memory.

Serial working memory has been studied since the 1960s, and two fundamental regularities in this behavior have been observed: primacy, and recency. Both primacy and recency can be understood by looking at an example of a serial recall task, like the one shown in figure 6.1. Primacy is identified by the observation that items appearing earlier in the list have a greater chance of being recalled accurately, regardless of the length of the list. Recency is identified by the observation that the items most recently presented to subjects have an increased chance of being recalled as well. Together, primacy and recency account for the typical U-shaped response probability curve seen in serial working memory tasks.

Figure 6.1: primacy and recency human data, xuan

Figure 6.2: updated version of xuan's 4.9 figure

Interestingly, this same U-shape is seen in free recall tasks (where order information is irrelevant) as well (see figure 6.6). So it seems likely that the same mechanisms are involved in both free recall and serial recall. In fact, as I discuss in more detail in the next section, it seems likely that all working memory behavior can be accounted for by a system that has serial working memory. In contrast, simple models of working memory that can account for the behavior observed on the monkey tasks previously described, cannot account for serial working memory behavior.

Consequently, serial working memory is of more fundamental importance when considering human cognition. The first model of serial working memory implemented in a spiking neural network that I am aware of was constructed by Xuan (pronounced 'Shawn') Choo in my lab. Based on a consideration of several models from mathematical psychology (refs?), and cognitive psychology (refs?), Xuan has proposed the ordinal serial encoding (OSE) model of working memory (ref?). In the next section, I present several examples of how this model can account for human psychological data on serial and free recall tasks. First however, let me describe the component parts, and basic function of the model.

The encoding and decoding of items to be remembered is described by figure 6.2. As can be seen here, the OSE model consists of two main components, an input buffer and an episodic buffer. These are taken to map onto cortical and hippocampal memory systems respectively. Both buffers are currently modeled as neural integrators (section ref?), although a more sophisticated hippocampal model (e.g. ref? becker, etc.) would improve the plausibility of the system, while generating a similar behavior. Modeling these buffers as neural integrators amounts to including past simple models of working memory as a component of this model.

The input to the model consists of a semantic pointer representing the item, and another semantic pointer representing its position in a list. These two representations are bound, using a convolution network (section ref?), and fed into the two buffers. Simultaneously, the item vector alone is fed into the two buffers to help enforce the item's semantics. Each buffer adds the new representation to the list that is currently in memory, and the overall representation of the sequence is

the sum of the output of the two buffers. Generation of this result completes the process of encoding new information into a memory trace.

Decoding of such a memory trace consists of subtracting already recalled items from the memory trace, convolving the memory trace with the inverse of the position vector, and passing the result through a cleanup memory (section ref?). In short, the decoding consists of unbinding the desired position from the total memory trace, and cleaning up the results. Concurrently, items which have already been generated are subtracted so as to not be generated again. In the case of free recall (where position information is not considered relevant), the unbinding step is simply skipped. The equations describing both the encoding and decoding processes can be found in appendix ref?.

As can be seen from figure 6.2, this model has two free parameters, α and γ. The first of these captures the effect of rehearsing items early in the list during the encoding process. The value of this parameter was set by fitting an experiment conducted by Rundus (1971) ref? in which the average number of rehearsals was found as a function of the item number in a serial list tasks. The second parameter, γ, captures the natural fading of the representation from cortical working memory over time. The value of this parameter was set by reproducing the simple working memory experiment described in Reitman (1974) ref? and choosing the value for which the accuracy of the model matches that of humans. Reitman's experiment showed that human subjects could recall on average 65% of the items that they recalled immediately after list presentation after a 15 second delay (when rehearsal was not permitted).

It is important to note that the two free parameters of this model were set by considering experiments which are not being used to claim that the model is a good model of serial recall. That is, they were set independently of the test experiments considered in the next section. Consequently, the good performance of the model on these other tasks in *no way* depends on tuning free parameters to those tasks.

Also note that the preceding description of this model is compact. This is because we have seen the implementation of many of the underlying components, the necessary representations, and the basic transformations before. So, in many ways this working memory system is not a new addition to the semantic pointer architecture, but rather a recombination of functions already identified as central to the architecture (i.e., integration, binding, and cleanup). This is encouraging because it means our basic architecture does not need to get more complicated in order to explain additional, even reasonably sophisticated, functions.

Figure 6.3: Model recency & primacy: note that the model always has 95% confidence intervals, human data is only averages, because that is all that is available

Figure 6.4: Transposition gradients figure

6.3 Example: Serial list memory

Figure 6.3 shows the OSE model capturing the basic recency and primacy effects observed in the human data, shown earlier in figure 6.1. This demonstrates not only that the model captures basic features of serial memory, but that setting the parameters of the model using different experiments did not limit the system's ability to explain this behavior.

As is clear from the fact that the model, and the subjects, do not have perfect recall accuracy, mistakes are made in recalling serial lists. These mistakes have been analyzed in several experiments (refs?). Figure 6.4 shows a comparison of the transposition gradients found in the model with those measured from human subjects. The transposition gradient measures the probability of recalling an item outside of its correct position in a list. Both the model and the human data show that errors are most likely to occur in positions near the original item position, as might be expected.

As you might also expect, the similarity between items in a list can affect how well items can be recalled. Henson et al. (1996) refs? designed an experiment in which they presented subjects with lists containing confusable and non-confusable letters. Because the stimuli were heard, confusable letters were those which rhymed (e.g. "B", "D", and "G"), while non-confusable letters did not (e.g., "H", "K", and "M"). The experimenters presented four different kinds of lists to the subjects: lists containing all confusable items, those containing confusable items at odd positions, those containing confusable items at even positions, and those containing no confusable items. The probability of successful recall and the transposition gradients for these lists for both the human subjects and the model are shown in figure 6.5.

Again, this example shows that the model does a good job of capturing behavioral data, both the likelihood of successful recall and the pattern of errors that are observed in humans. The same model has also been tested on several other serial

Figure 6.5: Confusable lists data, 3 figures combined

Figure 6.6: Free recall figure, also demonstrating primacy and recency with the U-shape

Figure 6.7: a) no normalization and b) Ideal normalization

list tasks, including delayed recall tasks, backwards recall tasks, and combinations of these (ref?).

More important for present purposes, however, the same model can also explain the results of free recall experiments. As shown in figure 6.6, the accuracy of recall for the model is very similar to that of humans for a wide variety of list lengths. In these tasks, there is no constraint on the order in which items are recalled from memory. Nevertheless, both the model and human data show the typical U-shaped response probability curves.

Taken together, these examples of good model performance on a variety of tasks (none of which were used to tune the model) can make us reasonably confident that some of the principles behind human working memory are captured by the model. Because it uses the same kinds of representations as the rest of the SPA, we can be confident that it will integrate easily into larger scale models.

Before leaving consideration of this model, I want to highlight what I think is perhaps its most theoretically interesting feature: namely, that this model only works if it is implemented in neurons. As demonstrated by figure 6.7a, if we directly simulate the equations that describe this model (see appendix ref?), it is unable to accurately reproduce the recency and primacy effects observed in the human data.

Initially, it seemed that this failure might have been caused by the semantic pointer vectors in the system becoming arbitrarily long as additional items were added to the memory trace. Consequently, we also implemented the model using standard vector normalization, which guarantees that the vectors always have a constant length. But again, as seen in figure 6.7b, the model is unable to capture the human data.

Consequently, we realized that one of the main reasons that this model is able to capture the human data as it does, is that the individual neurons themselves saturate when participating in the representation of large vectors. This saturation serves as a kind of "soft" normalization, which is neither ideal mathematical normalization, nor a complete lack of normalization. Instead, it is a much more subtle kind of constraint placed on the representation of vectors in virtue of neuron

response properties. And, crucially, this constraint is directly evident in the behavioral data. This is theoretically interesting, because it provides an unambiguous example of the importance that constructing a neural implementation can have on explaining high-level psychological behavior. Without constructing the neural model, we would have considered the mathematical characterization a failure, and moved on to other, likely more complex, models. However, it is now clear that we would have been doing so unnecessarily. And unnecessary complexity, of course, is the bane of intelligible explanations.

6.4 Biological learning

- mostly general considerations: the NEF is learning independent for good reasons, but obviously learning happens too... how to connect (note that in the paper with Dave MacNeil, we show you need to get 'near' the optimal in order to learn well, we also show learning is better than the 'optimal' weights).
- reinforcement learning (the three kinds of learning, and my 'new' one???)

It is difficult to specify what counts as a 'biologically plausible' learning rule, as there are many mechanisms of synaptic modification (i.e., connection weight change) in the brain (Feldman, 2009; Caporale and Dan, 2008). However, the vast majority of characterizations of synpatic plasticity still adhere to Donald Hebb's original suggestion that: "When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased" (Hebb, 1949, p. 62). Or, more pithily: "Neurons that fire together wire together." Most importantly, this means that any modification of synaptic connection weights must be based on information directly available to the cell whose weight is changed (i.e., pre-synaptic and post-synaptic activity).

Recently, there has become increasingly strong evidence that this modification can be highly dependent on the timing of the pre- and post-synaptic spikes (refs?). Such spike driven learning has become known as STDP (spike-timing dependent plasticity). STDP refers to the observation that, under certain circumstances, if a presynaptic spike occurs before a postsynaptic spike, there is an increase in the likelihood of the next presynaptic spike causing a postsynaptic spike. However, if the postsynaptic spike precedes the presynaptic spike, then there is a decrease in

the likelihood of the next presynaptic spike causing a postsynaptic spike (refs?). Or, more succinctly, a presynaptic spike followed closely in time by a postsynaptic spike will potentiate a synapse, and the reverse timing depresses it: fire together, wire together.

- ???STDP figures??? explaining the mechanism

The learning rules proposed to explain STDP computationally, focus, as the original experiments did, on comparing two spikes at a time. However, in order to explain more recent plasticity experiments, it has become necessary to consider triplets of spikes (Pfister and Gerstner, ref?). Specifically, Pfister and Gerstner show that by adding an additional depression with a pre-post-pre triplet, and additional potentiation with post-pre-post triplets, a better fit to the experimental data is achieved.

Even more recently, Trevor Bekolay, a member of my lab, has devised a method for using triplet-based STDP rules to learn high dimensional vector transformations. The mathematical details of this rule can be found in appendix ref?. This rule simultaneously addresses the limitations of both standard learning rules, and STDP. In particular, unlike most standard rules this rule is able to take into account precise spike times, and unlike most STDP rules, it is able to relate synaptic plasticity directly to values represented by an ensemble of neurons. Thus, the rule can be usefully employed in a network that can be characterized as representing a value in spiking neurons – precisely the kinds of networks we find in the SPA (a tutorial demonstration of this method is in section 7.4).

Better yet, this rule can incorporate error information coming from other parts of the system, to affect the interaction between spikes in a synapse. There is direct evidence for this kind of interaction in parts of the brain stem (ref? mcneil), and it also captures a broadly accepted view of a role for the neurotransmitter dopamine (refs). There is strong evidence that dopamine neurons increase their firing rate both when rewards that are not predicted occur, and when predicted rewards do not occur (ref?). Consequently, it has been suggested that dopamine can act as a modulatory input to cortex and parts of the basal ganglia (i.e., striatum), helping to determine when connections should be changed in order to account for unexpected information in the environment.

Figure 6.8 shows a simple example of applying this rule to a scalar representation, with feedback error information. The left-hand side of the figure shows the structure of a simple circuit that generates an error signal, and the right hand side shows an example run during which the improvement in the receiving population's ability to represent the input signal is evident.

Figure 6.8: stdp network structure and sample run

Figure 6.9: figure and caption from trevor's tech report

Figure 6.9 shows the statistics of the effectiveness of the rule over many example runs. This figure also demonstrates that the spike-based rule is as effective as a less neurally plausible, rate-based rule. These simple examples of learning show the basic functionality of the learning rule. That is, it demonstrates that a biologically realistic STDP rule can be used to learn connection weights that compute the identity function between a scalar represented in one population and a scalar represented in another population. However, given the general characterization of computation and representation provided by the NEF, we can generalize this rule to learn (nearly) arbitrary transformations over high-dimensional vector representations.

A particularly important transformation over high-dimensional vectors for the SPA is circular convolution: recall that it is used for binding, unbinding, and content sensitive control. Given the central role of convolution in the architecture it is natural to be concerned that it might be a highly specialized, hand-picked, and possibly unlearnable transformation. If this was the case, that would make the entire architecture seem much less plausible. After all, if binding cannot be learned, we would have to tell a difficult-to-verify evolutionary story about its origins. Fortunately, binding can be learned, and it can be learned with this same learning rule.

Figure 6.10 shows the results of this STDP rule being applied to a network which is given examples of binding of six-dimensional vectors. These simulations demonstrate that the binding operation we have chosen is, at least in principle, learnable using a biologically realistic learning rule. However, it would be premature to claim that we therefore know how cognitive systems learn binding. This is because there are questions to answer regarding how the target representations would be generated, how the appropriate error signals can be targeted to the appropriate systems that perform binding, and so on. In fact, it should not be surprising that telling a story about how binding is learned might be difficult: many of the highly cognitive behaviors that require binding are not evident in people until about age 3 (e.g., analogy, A not-B processing, two-place relations in language, etc. (refs?)). Nevertheless, this demonstration should allay concerns that there is something especially implausible about choosing convolution as a binding

Figure 6.10: stdp learning binding figure, showing nonlinear high-d function learning, and making SPA more palatable

operation.

6.5 Example: Learning new long-term strategies

- including control strategies? actions? rules? two examples, one 'low level' habitual action learning (reinforcement), the other 'high-level' syntax/context manipulation (wason)
- raven's example is consistent with the idea that there is some way to start 'fixing' the strategies learned during one session (though people are notoriously unstable in their performance on different sessions... talk to dan about exact learning effects?).

6.5.1 Learning new control strategies

- intro on central role of BG in reinforcement learning...
- a seciton on reinforcement learning to add new elements to the M and M_t matrices in the bg model!
- There is strong evidence for dopamine being used to mediate the learning between striatal neurons and cortical neurons Calabresi et al. (2007).

6.5.2 Learning new syntactic manipulations

- and cortical example of structural long-term learning (wason task)

While Neumann considered learning syntactic transformations, here I consider the slightly more complicated task of learning a transformation **T** that is context-dependent (i.e., one that might change in different contexts). Essentially this demands an associative memory that associates contexts to transformations, and that also learns what an appropriate transformation is in a context. As shown in appendix B.2, we can use the NEF to determine the following rule that implements this associative transformation learning:

- $$putInWhat'sDerived$$

  where k is the learning rate, cl are neurons representing the context signal, and tj are neurons representing the transformation vector. The connection weights wil are learned by this rule and the wij are learned by a similar rule that performs simple representation matching (see appendix C).

This rule is 'Hebbian' because only information locally available to a particular neuron is needed to update the relevant connection weight, and, as the coincidental activity of the pre- and post-synpatic neurons increases, so does their connection weight.

The application of this learning rule in a neurobiological simulation is illustrated in figure 6.11. This associative memory makes it possible to learn how to represent an unknown transformation space, and then associate transformations in that space to different contexts. This means that we can learn, instead of handcrafting, syntactic transformations. This simulation does not explicitly demonstrate that the learned vectors are syntactic transformations. That demonstration is left for section 6.5.2, in which this learning is incorporated into a more complete SPA model.

4. Modeling the Wason selection task In sections 3.1 and 3.2, we demonstrated structured symbolic representation, manipulation, and learning via neurally plausible mechanisms for low-dimensional vectors. This work provides theoretical foundations for our proposed biologically plausible cognitive architecture, but it must be scaled up and functionally organized to perform a concrete task. As an example, we have chosen to generate a neural cognitive model of the Wason selection task {{2683 Wason,P.C. 1966; }}. This task is challenging because it requires symbolic reasoning and strategy changes across contexts. In the Wason task, participants are given a conditional rule of the form "if P, then Q". They are then presented with four cards, each of which has either P or not-P on one side, and either Q or not-Q on the other. The visible sides of each card show P, not-P, Q, and not-Q. The task is for the participant to indicate all of the cards that would have to be flipped over to determine whether the conditional rule is true (i.e., is being obeyed). The logically correct answer is to select the cards showing P and not-Q. Figure 7: The Wason selection task. Four cards are presented to the participant, each with a piece of information on both sides. A rule is stated, and the participant must select the cards that must be flipped over to confirm whether the rule is being followed. The most commonly selected cards are indicated with a circle.

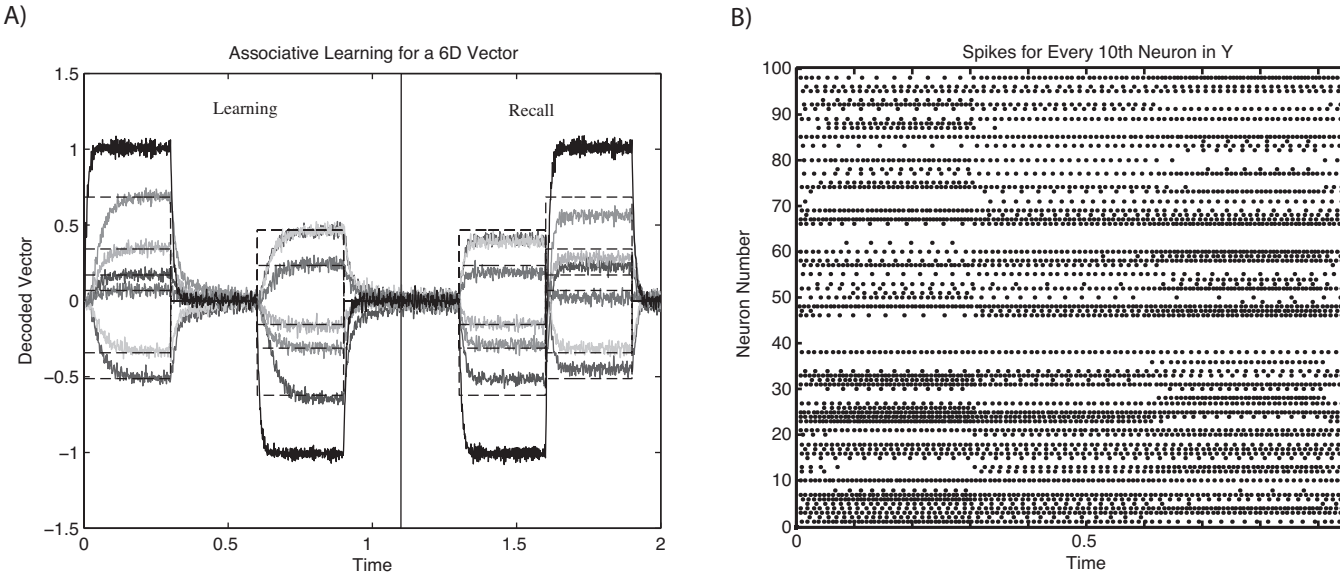

Figure 6.11: Associative learning using the derived rule in a spiking network for two 6D vectors. Part A shows vector estimates decoded from the neural spikes in part B. A) The thick black line indicates the context signal. There are three contexts, +1, -1, and 0. The thin lines each indicate the represented value of one element of the 6D vector. The dotted lines indicate the ideal answers. During the first half of the simulation, two random vectors are being learned under two different contexts. During the second half of the simulation, only the context signal is applied to the population, which then responds with the vector encoded in that context. The network successfully associates the context signal with a close approximation to the vector. B) The spike trains (action potential times) of the neurons in the associative memory population (for the first half of the run, and every tenth neuron, for legibility). There are 1000 neurons in this population. The synaptic weights are initialized to random values.

If the task is given in an abstract form, such as "if a card has a vowel on one side, then it has an even number on the other", the majority of participants choose P (the vowel) and Q (the even number) {{1858 Oaksford,M. 1994;396 Cheng,P.W. 1985; }}. However, if the task is presented using familiar content, such as "if a person is drinking beer, then they must be over 21 years old", the majority choose P (drinking beer) and not-Q (under 21) {{2823 Cox,J.R. 1982; }}. In other words, structurally identical tasks with different contents lead to differing performance. Explaining this "content effect" in a neural model requires changing symbolic manipulation strategies based on the content of the task, a challenging control problem that ultimately demonstrates that the proposed architecture can perform sophisticated transformations as well as represent complex structure – a feature of past architectures that we criticized as largely absent (section 2). 4.1 Theoretical characterizations of the Wason task Initial explanations of Wason task performance were based on the familiarity of the contexts. However, later evidence focused on the distinction between deontic and non-deontic domains. In deontic situations the rule expresses a social or contractual obligation, and here human performance matches logical explanations, regardless of familiarity or abstractness {{2398 Sperber,D. 1995/pfor references see /f, p. 34;}}. To explain this, Cheng and Holyoak {{396 Cheng,P.W. 1985/a;}} proposed that people reason using neither context-independent rules of inference nor memory of specific experiences, but rather abstract context-sensitive knowledge structures induced from ordinary life experiences, such as permissions, obligations, and causations. These structures, or pragmatic reasoning schemas (PRS) are general strategies that activate based on the current situation. In particular, a "permission" schema might contain the simple rule that "if the precondition (P) is not satisfied then the action must not be taken (not-Q)". When this schema is triggered, this rule is directly available, allowing the not-Q card to be selected without recourse to indirect logical derivation via reductio ad absurdum. Importantly, PRS differs from propositional logic in that it is context sensitive and does not require long chains of reasoning. Alternatively, Cosmides {{519 Cosmides,L. 1989/a;}} developed social contract theory (SCT), an explanation based on evolutionary psychology. SCT proposes that natural selection has produced special-purpose, domain specific mental algorithms for solving important recurring adaptive problems. When triggered by the specific content of a problem, these algorithms call forth procedural knowledge which leads to appropriate inferences and responses in the given domain. The domain of "social exchange" (adaptive cooperation between two individuals for mutual benefit) is theorized to activate a domain specific "cheater detection" algorithm, which operates in terms of benefits and costs. This mechanism draws

attention to any individual who has either "taken the benefit" (P) or "not paid the cost" (not-Q) in the social exchange domain in order to detect potential cheaters. This leads to a logically correct response in selection tasks that trigger the cheater detection mechanism. PRS and SCT both agree that people lack an inherent "mental logic," in that people do not use a domain general propositional calculus. They also agree that people reason using different rules of inference for different content domains. Both theories use this context sensitivity to explain why different formulations of the selection task elicit different responses. The theories differ in the source of these rules: PRS schemas are induced through experience while SCT algorithms are genetically endowed. Indeed, Cosmides {{519 Cosmides,L. 1989/a;}} has challenged any theory based on induction to lay out the mechanistically defined domain-general procedures that can take modern human experience as statistically encountered input, and produce observed domain specific performance in the selection task as output. Responses to this challenge have been made {{2398 Sperber,D. 1995; 1858 Oaksford,M. 1994;}}, but none present a neural mechanism. To develop our model, we suppose that there is a content-sensitive domain general inference mechanism that learns to map linguistic structures to appropriate responses in a context. This hypothesis makes no assumptions about how contexts are partitioned (i.e. as deontic or not) and is consistent with PRS, although PRS is silent on possible implementations, specific mechanisms, or neural plausibility in general. Our goal is not to unequivocally establish a new theoretical characterization of the task, but to show how a high-level characterization such as the one we have proposed can be tested for plausibility by implementing it in a biologically constrained simulation. Doing so, we suggest, meets the challenge that Cosmides (1989) has offered to domain general hypotheses. 4.2 Anatomical and physiological constraints The PET and fMRI studies on symbolic reasoning do not unambiguously determine a single brain area dedicated to such tasks {{2828 Goel, V. 2005; }}. However, there is some indication that right brain areas are preferentially activated during the processing of deductive reasoning tasks, including the middle temporal lobe and inferior frontal cortex {{1894 Parsons,L. 2001; }}. Since the middle temporal lobe has been indicated to be more closely related to language processing than actual deduction {{2828 Goel, V. 2005; }}, our model centers on the right inferior frontal cortex. Three other brain areas provide inputs to this system in accordance with their functions (described below; see figure 8): the left language areas, ventromedial prefrontal cortex (VMPFC), and anterior cingulate cortex (ACC). The Wason task rule is presumed to be encoded by the left language areas {{2884 Heimer, L. 1994/pi.e., the Perisylvian language zone, ;}}. This is consistent with Parsons' {{1895 Parsons,L. 1999/a;}} suggestion that

rules are provided to the deductive mechanism by left language areas and the general characterization of language processing involving left frontal cortices {{205 Binder,J.R. 1997; }}. VMPFC provides context information, used in the model to select the appropriate reasoning strategy through an associative memory. Context is determined based on the rule stored in the left language areas and transmitted via hippocampal and limbic projections to VMPFC {{2885 Kalisch,R. 2006; }} (modeled here as a direct projection for simplicity). There are direct anatomical connections between VMPFC and right inferior frontal cortex, and VMPFC has been implicated in carrying context information {{26 Adolphs,R. 1995; }}. Finally, a feedback mechanism is needed to induce the correct responses in different contexts based on experience. Holroyd and Coles {{1182 Holroyd,C. 2002/a;}} propose that a high-level error-processing system in the anterior cingulate cortex (ACC) sends an error signal to frontal cortex via the mesencephalic dopamine system, facilitating the development of adaptive behaviors. Our model thus receives an error signal from ACC, which we take to be a representation of the correct responses to give in a particular context. This is used by our model to tune the context-sensitive mapping from rule to response. It is difficult to determine appropriate neurophysiological constraints on these networks because there is little data on the physiological properties of neurons in the implicated areas of cortex in humans. As a result, we have based our general modeling assumptions on data from monkey frontal areas {{2156 Romo,R. 1999; }}, with a slight increase in maximum firing rates to reduce computational overhead. 4.3 A biological model of the selection task The basic functioning of the model is that VMPFC takes the task rule (R), and calculates a context (C). This context is used by the associative memory to determine a transformation (T) that will extract the appropriate responses in this context from the rule. In some contexts (such as the drinking/age version of the Wason task), this would be the transformation from "if P then Q" to "P and not-Q", while in other contexts (such as the abstract vowel/even version) the appropriate transformation is from "if P then Q" to "P and Q". This transformation is then applied to the particular rule (R) to produce a response (A). This would be "alcohol and under 21" and "vowel and even", respectively. To support learning through experience, the input from ACC is used to provide a "correct" response (A'). This can be combined with R to determine T', the correct transformation to apply. T' is then used to train the associative memory. In more computational terms, the model loads an appropriate transformation (or reasoning method) based on a context signal and then uses that transformation to determine a response in this context. When the provided response is incorrect, the context transformation association is updated to make it more likely that subsequent responses will be

correct.

Figure 8: High-level description of the Wason model.

Given this high-level characterization, we need to describe precisely how each of these mechanisms operates neurally. To do this, we combine the elements of the cognitive architecture described in section 3, resulting in the mechanism shown in figure 9. The two transformations (combining R and T to produce A, and combining R and A' to produce T') are implemented as convolutions (A=R⊗T and T'=R-1⊗A'), using the techniques described in section 3.1.2. The associative memory uses the learning mechanism described in section 3.2.

Figure 9: Neural diagram for the Wason model. Circles indicate neural populations (N is the number of neurons, D is the dimensionality of the value represented by the neurons). Arrows indicate neural connections.

To encode sufficiently complex rules for this task, we use 100-dimensional vectors, rather than the 6-dimensional ones used in section 3. Rules are structured as in section 2.3, so "if a card has a vowel on one side, then it has an even number on the other" would be implies(vowel,even), encoded as relation⊗implies+antecedent⊗vowel+consequent
Similarly "if a person is drinking beer, then they must be over 21 years old" would be implies(alcohol, over21), encoded as relation⊗implies+antecedent⊗alcohol+consequent⊗over21
The term over21 is built from over and 21 (over21=over⊗21), demonstrating the use of embedded structure. The expected responses for these rules are even+vowel and alcohol+not⊗over21, respectively. Each atomic symbol is a random 100-dimensional vector with each element chosen from independent Gaussian distributions with zero mean and variance 1/100. All populations are composed of spiking leaky-integrate-and-fire (LIF) neurons, with absolute refractory periods of 1 ms, membrane time constants of 10 ms, and synaptic time constants of 5 ms, corresponding to typical AMPA-type receptors. The biophysical properties of each neuron are uniformly distributed to produce a range of peak and background firing rates that is similar to that found in frontal cortices of apes. Postsynaptic currents are characterized using a standard decaying exponential model, $h(t)=e^{-t/\tau}$, where τ is the synaptic time constant. As per the NEF, each neuron has a preferred direction vector (the direction in concept space for which it is most active). For the majority of neural populations, we choose these to be distributed along the dimensional axes (i.e. standard unit basis vectors) . However, the two populations performing circular convolution (d and h in figure 9) use vectors evenly distributed along the diagonals of the high dimensional space (between the axes of any two elements which are multiplied) to allow the nonlinear circular convolution to be computed. All simulations described below use this exact same model, simulated at a time resolution of 0.5 milliseconds. The

only adjustment across simulations is varying the input, as described below. 4.4. Simulations Two simulations demonstrate the capabilities of this model. First, we show that it can learn the basic task: to perform different symbolic transformations based on the context. Second, we show that these transformations are general, in that they can be applied to new rules for which the system was not initially trained. 4.4.1. Simulating the Wason task across contexts The basic Wason task is modeled by considering two different rules and their appropriate responses: relation⊗implies+antecedent⊗vowel+consequent⊗even vowel+even relation⊗implies+antecedent⊗alcohol+consequent⊗over21 alcohol+not⊗over21 In the first context, the system must extract the consequent and the antecedent, while in the second the antecedent must be further modified by binding it with not. To perform this task, the model must learn the appropriate transformation for the two contexts. We incorporate a training period during which the model is presented with a rule (via LLA) and the appropriate response (via ACC). The context signal is derived from the rule by VMPFC . Afterwards, we test the model by presenting just the rule and recording its output. The process covers six seconds of simulated time, with 1.5 seconds each for training and 1.5 seconds each for testing. The output from the system (i.e. the structure representing the chosen response) is the spiking pattern in Figure 10. These spikes encode the 100-dimensional vector representing the chosen actions. To interpret this, we compare the neural representation (from the Decoding equation of section 3.1.1) of the vector to each of the possible symbols (vowel, relation, over21, not⊗over21, etc.) via a dot product. The results of this simulation are provided in Figure 11. Figure 10: Spikes of every 30th neuron in the e population over 200ms of simulation time. This is the raw spike data that is decoded to estimate the elements of the response vector.

Figure 11: Model results for learning and applying inferences across the two contexts. The time-varying solid lines indicate the similarity of the model's decoded spiking output (population e) to all possible responses. The darker lines trace the expected answers on the two versions of the Wason task. The vector names and numerical similarity values of the top two results (over the last 100ms of that part of the task) are shown above. After learning, the system reliably performs the correct context-dependent transformation, producing the appropriate response.

For the abstract case, the expected response is vowel and even. This 'logically incorrect' answer may be learned as a result of past experiences where bi-conditional interpretation of the implication {{2830 Feeney, A. 2000; }} coupled with a preference for positive information or matching strategies {{2078 Reich,S.S. 1982;2831 Manktelow, K.I. 1979; }} have been used and reinforced. In

the content specific task, the expected response is the 'logically correct' answer alcohol and not⊗over21, which is assumed to be learned through experiences where the content specific cues have been relevant and indicate this specific transformation is most appropriate (i.e., situations where one is reasoning about legal drinking). During these first three seconds, the appropriate transformations to produce these expected results are learned in the weights between the context signal and associative memory as described in section 3.2. We have compressed the learning history by having a high learning rate (k in the learning rule equation). During the last three seconds, feedback of the correct solution from ACC is removed, and the model then performs the correct transformation for each context based only on the context signal. This performance is 95% reliable (N=20) over independent simulations. The variability is due to the fact that the 25 different symbol vectors are randomly chosen at the beginning of each run. As the number of symbols increases, the model reliability is reduced (70% correct (N=20) with 35 symbols). This is expected, as we have implemented a relatively low dimensional VSA. However, as described earlier, capacity (and hence accuracy) increases exponentially with the number of dimensions.

4.4.2. Syntactic generalization within contexts The preceding results are potentially uninteresting insofar as they could be generated by a neural network model that simply memorizes an answer in a given context. For the model to demonstrate truly cognitive capacities, it is essential to show that the transformations being learned generalize to novel rules in the same context. And, furthermore, that this generalization be content insensitive – that is, driven by the syntactic structure of the examples. It is important to note that it is precisely this kind of generalization that has been thought to distinguish cognitive from non-cognitive systems {{799 Fodor,Jerry 1988; 1241 Jackendoff,R. 2002; 2792 Hummel, John E Holyoak,Keith J. 2003; }}. As a result, it is crucial to show this kind of syntactic systemmaticity is achievable by models employing our cognitive architecture. To demonstrate this, we train the model on two rules in the same context, and then test it on a third rule, again in the same context. All other aspects of the simulation are identical to the previous one. relation⊗implies+antecedent⊗vote+consequent⊗over18 vote+not⊗over18 relation⊗implies+antecedent⊗alcohol+consequent⊗over21 alcohol+not⊗over21 relation⊗implies+antecedent⊗drive+consequent⊗over16 drive+not⊗over16 As before, the model learns the correct response in this context for the first two rules. Next, learning is turned off and the model is shown the third rule, which is syntactically similar but not previously seen. Since the rule produces the same context signal (via VMPFC), the same transformation is applied, producing the correct response (drive+not⊗over16).

Figure 12: Simulation results demonstrating syntactic generalization within a context. The format is the same as described in Figure 11. In the first half of the simulation, two different deontic rules are presented, and the relevant transformation is learned. In the third quarter, a novel deontic rule is shown and the model correctly infers the appropriate response. This demonstrates that the model can infer based solely on the syntactic structure of the training examples.

Figure 12 demonstrates that the system is capable of applying its learned syntactic transformations to novel contextually similar situations, which have different content. As a result, we have implemented a specific, domain general cognitive mechanism that is able to reproduce the basic Wason task and the related content effects. More generally, these simulations together indicate that we have successfully met our goal of implementing a truly cognitive architecture in a biologically plausible network. 4.5 Discussion of Wason task simulations In the context of the Wason task, the model meets Cosmides' {{519 Cosmides,L. 1989/a;}} original challenge to provide a plausible domain general mechanism able to explain task performance while being inductive. In particular, Cosmides suggests that the inductive evidence during the normal course of human development cannot serve to explain the superior performance on the content facilitated task because the relative amount of evidence would not favor the correct response. However, the inductive mechanism provided here does not rely on the relative amount of evidence available in different contexts. Instead, it relies on a minimum threshold of available evidence (i.e., enough for the learning rule to induce the relevant transformation) within a given context. It should also be clear that the reasoning mechanism itself does not change as the context does. Instead, an externally generated signal determines how the mechanism is employed (i.e. which transformation is applied). That is, the model suggests that the central difference between the abstract and content facilitated versions of the Wason task is the context signal being provided from the VMPFC. This, we take it, is very different from suggesting that evolutionarily distinct reasoning mechanisms are necessary to account for performance differences on these two versions of the task. Experiments performed after we first presented an early version of this model {{692 Eliasmith,C. 2004; }} have highlighted the VMPFC as the locus of major cognitive processing differences on these two versions of the task {{2832 Canessa,N. 2005; }}, supporting this suggestion. However, our explicit and heavy reliance on learning in this model is unique in the theoretical approaches to the Wason task. Specifically, pragmatic reasoning schemas have been heavily criticized as leaving 'learning' or 'induction' unspecified and overly powerful {{519 Cosmides,L. 1989; }}. In contrast, we have provided a detailed, neurally plau-

sible mechanism in our model. Further, our implementation offers an advantage over PRS, in that the transformations used to solve the problem are not limited to a theoretically pre-determined group of "schemas" which are identical for all subjects (e.g., the 'permission' schema). Instead, each individual will solve the problem using his or her own estimation of the correct transformation in the given context, as determined by his or her personal experience and learning history in that context. The model is thus not restricted to a clear-cut deontic/non-deontic distinction, but rather the similarity of context signals will determine which transformations are applied. As a result, the similarity space provided by context signals will be mapped onto a transformation space. This mapping is discovered by the model based on past examples. This is consistent with findings that successful performance of the task does not align neatly with intuitive theoretical distinctions {{1859 Oaksford,M. 1996; }}{{2827 Almor, A. 1996; }}, and can vary with past experience {{2126 Rinella,K. 2001;}}. We do not suggest that our model is completely satisfactory. A number of improvements to the model will be the focus of future development. A first improvement will be to extend the work of Singh {{2811 Singh,R. 2005/a;}} on biologically plausible cleanup memory. This models how the noisy output from such a system can be used to identify the symbols that are most strongly represented (the top lines in figures 11 and 12). Second, it is crucial to explicitly explore the scalability of the model. In theory, the accuracy and symbol capacity scales exponentially with the number of dimensions, but this remains to be tested. This is especially important given the interaction of this scalabilty with the amount of neural noise in the system. Third, there are a number of learning-related improvements that could be made to the model. For instance, it should be possible to extend the learning rule to permit learning based only on a valence signal (i.e., reinforcement learning) without specific information regarding the correct answer. This likely better reflects learning in everyday circumstances. Relatedly, it would be useful to have the learning consist of a large number of distinct examples of reasoning across similar contexts. This would likely improve the semantic generalization results in figure 12, since the learned transformation would be an average over a large number of syntactically similar and semantically dissimilar examples (effectively removing semantic effects). Finally, it would be useful to investigate the number of distinct contexts that can be effectively stored in the associative memory. It would then be possible to predict and test how specific associative memory limitations would be expected to affect performance.

- other syntactic generalization discussion, not wason specific.

Syntactic generalization is well-demonstrated by considering simple examples of deductive reasoning. In such circumstances, it is the syntactic structure, not the content, that is expected to drive behaviour. While content effects are well-documented in human deductive reasoning (Wason, 1966; Cheng & Holyoak, 1985), syntactic generalization must still be accounted for. The model we present here incorporates both.

Syntactic generalization requires the integration of the transformation and learning functions. Learning is used to determine what the relevant transformation is in a given circumstance, and that transformation is then applied by a transformation subnetwork. Because not all input/output pairs are taken by the system to be examples of the same transformation, we have introduced a context signal into the learning component of the model to allow it to employ different reasoning strategies in difference circumstances. Since we allow the content of a processed sentence to determine the context, we can naturally capture content effects.

PET and fMRI studies on symbolic reasoning do not unambiguously determine a single brain area dedicated to such tasks (Goel, 2005). However, there is some indication that right brain areas are preferentially activated during the processing of deductive reasoning tasks, including the middle temporal lobe and inferior frontal cortex (Parsons & Osherson, 2001). Since the middle temporal lobe has been indicated to be more closely related to language processing than actual deduction (Goel, 2005), our model centers on the right inferior frontal cortex. Deductive rules are presumed to be encoded by the left language areas (i.e., the Perisylvian language zone, Heimer, 1994). This is consistent with Parsons (1999) suggestion that rules are provided to the deductive mechanism by left language areas and consistent with the general characterization of language processing as involving left frontal cortices (Binder et al., 1997).

We assume that context is determined based on the rule contents and transmitted to VMPFC (Kalisch et al., 2006). There are direct anatomical connections between VMPFC and right inferior frontal cortex, and VMPFC has been implicated in carrying context information (Adolphs et al., 1995).

Finally, a feedback mechanism is needed to induce the correct responses in different contexts based on experience. Holroyd and Coles (2002) propose that a high-level error-processing system in the anterior cingulate cortex (ACC) sends an error signal to frontal cortex via the mesencephalic dopamine system, facilitating the development of adaptive behaviors. Our model thus receives an error signal from ACC, which we take to be a representation of the correct responses in a particular context. This is used by our model to learn the context-sensitive mapping from rule to response. Figure 11 summarizes the model and its mapping

to the neuroanatomy.

[Warning: Image not found]

Figure 11: Neural diagram for the syntactic generalization model. Circles indicate neural populations (N is the number of neurons, D is the dimensionality of the value represented by the neurons). Arrows indicate neural connections.

The specific tasks we simulate are based on simple 'if-then' deductive reasoning, and patterned after the Wason task (Wason, 1966): e.g. "If A then B". In this task, a subject is shown four cards with the positive and negative cases of the elements displayed (i.e., A, not-A, B, not-B), and told that on the back of each card is another case. The task is to determine which cards must have their back value checked to determine if they violate the provided rule.

In our simulations all if-then rules are encoded as:

R = relation[F056?]implication + antecedent [F056?]A + consequent [F056?]B

If the rule is taken as a material conditional, the violating cases are **A** and **not-B** (encoded as **not[F056?]B**). In some contexts human subjects correctly make this judgement (Cheng & Holyoak, 1985). However, in others, where subjects may be mistaking the conditional as a bi-conditional (Feeney & Handley, 2000), the most common response is **A** and **B**. This is a straightforward example of a content effect, where the same sentential structure is reasoned about differently based on the content.

Figure 12 demonstrates the performance of the model in these two different contexts. Here we have used two rules, an abstract rule "If vowel, then even number," and a facillitated rule "If drinking alcohol, then over 21." The former is treated as a bi-conditional, the latter as a material conditional. In the first half of the simulation, the model learns different transformations in these two contexts. In the second half, it applies these transformations across those same contexts. It is evident here that the system is able to learn and apply deductive reasoning strategies in a content/context sensitive manner.

[Warning: Image not found]

Figure 12: Model results for learning and applying inferences across the two contexts. The time-varying solid lines indicate the similarity of the model's decoded spiking output (population *e*) to all possible responses. The darker lines trace the expected answers on the two versions of the Wason task. The vector names and numerical similarity values of the top two results (over the last 100ms of that part of the task) are shown above. After learning, the system reliably performs the correct context-dependent transformation, producing the appropriate response.

This performance is 95% reliable (N=20) over independent simulations. The variability is due to the fact that the 25 different symbol vectors are randomly chosen at the beginning of each run. As the number of symbols increases, the model reliability is reduced (70% correct (N=20) with 35 symbols). This is expected, as we have implemented a relatively low dimensional VSA. However, as described earlier, capacity (and hence accuracy) increases exponentially with the number of dimensions.

Because the simulation is run in spiking neurons, we are also able to look at the detailed spiking patterns produced during the task. Figure 13 provides an example of the spike patterns produced during the application of the learned transformation to a rule.

[Warning: Image not found]

Figure 13: Spikes of every 30th neuron in the *e* population over 200ms of simulation time. This is the raw spike data that is decoded to estimate the elements of the response vector.

More importantly, this same model can be shown to learn syntactic generalizations within a given context. Figure 14 shows results from a simulation where the context is held fixed (i.e. all are treated as material conditionals), and different rules are provided. This figure demonstrates that when a novel rule is provided in the second half of the simulation, the *syntactic* reasoning strategy learned in the first half is applied successfully.

[Warning: Image not found]

Figure 14: Simulation results demonstrating syntactic generalization within a context. The format is the same as described in Figure 11. In the first half of the simulation, two different deontic rules are presented, and the relevant transformation is learned. In the third quarter, a novel deontic rule is shown and the model correctly infers the appropriate response. This demonstrates that the model can infer based solely on the syntactic structure of the training examples.

We can employ the model to attempt to gain some insight into how we would expect humans to perform given various examples of novel syntactic transformation. The generally expected improvement is consistent with evidence of practice effects in reasoning tasks (Rinella et al., 2001). However, we can more specifically predict performance as a function of the number of examples. In Table 3 we show that generalization improves rapidly with the first three examples, and then improves less quickly. We predict that this would translate behaviorally as more rapid and more consistent responses after four examples compared to two, for these simple cases of syntactic generalization. However, the difference between 3 and 4 examples, while positive, is much less pronounced.

Table 3: Improvement in generalization with learning history. Accuracy is the difference between the VSA representations of the correct answers and the largest incorrect answer, scaled by the largest incorrect answer. A number below 0 indicates that the wrong answer is produced, and larger numbers indicate the separation between the correct and incorrect answers.

# of examples shown	Generalization accuracy
1	-0.10
2	0.05
3	0.23
4	0.26

In summary, these simulations together demonstrate that the same system is able to switch reasoning strategies (transformations) in different contexts, as well as apply the same syntactically-based reasoning strategy to novel structures within a context. In all cases, these strategies are learned based on (few) examples.

6.6 Nengo: Neural dynamics

- Theoretical point: Bringing representation, dynamics, and transformations together in one example.
- Do Controlled integrator (and/or maybe the subtracting integrator for better loading), in Nengo; appendix with code, or appendix with derivation, slight extension to simplest kind of working memory models (i.e. with a bit of control)... marc H might have evidence that it takes 100ms to load a memory?

6.6.1 ***from comp neuro paper*** Neural integrator

The line attractor, or 'neural integrator', has recently been implicated in decision making **??**, but is most extensively explored in the context of oculomotor control **???**. It is interesting to note that the terms 'line attractor' and 'neural integrator' actually describe different aspects of the network. In particular, the network is called an 'integrator' because the low-dimensional variable (e.g., horizontal eye position) $x(t)$ describing the network's output reflects the integration of the input

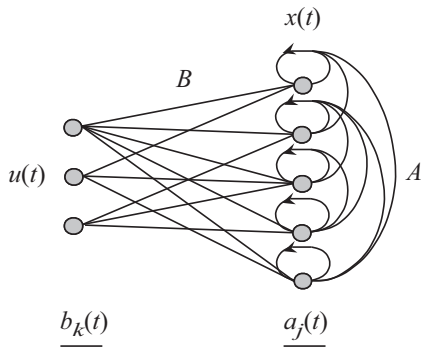

Figure 6.12: Line attractor network architecture. The underline denotes variables that are part of the neuron-level description. The remaining variables are part of the higher-level description. (taken from eliasmith, 2005)

signal (e.g., eye movement velocity) $u(t)$ to the system. In contrast, the network is called a 'line attractor' because in the high-dimensional activity space of the network (where the dimension is equal to the number of neurons in the network), the organization of the system collapses network activity to lie on a one-dimensional subspace (i.e. a line). As a result, only input that moves the network along this line changes the network's output.

In a sense, then, these two terms reflect a difference between what can be called 'higher-level' and 'neuron-level' descriptions of the system (see figure 6.12). As modelers of the system, we need a method that allows us to integrate these two descriptions. Adopting the principles outlined earlier does precisely this. Notably, the resulting derivation is extremely simple, and is similar to that already presented in Eliasmith and Anderson (2003). However, all of the steps needed to generate the far more complex circuits discussed later are described here, so it is a useful introduction (and refered to for some of the subsequent derivations).

We can begin by describing the higher-level behavior as integration, which has the state equation

$$\dot{x} = Ax(t) + Bu(t) \tag{6.1}$$

$$x(s) = \frac{1}{s}\left[Ax(s) + Bu(s)\right], \tag{6.2}$$

where $A = 0$ and $B = 1$. Given principle 3, I can determine A' and B', which are needed to implement this behavior in a system with neural dynamics defined by

$h'(t)$ (see (**??**)). The result is

$$\begin{aligned} B' &= \tau \\ A' &= 1, \end{aligned}$$

where τ is the time constant of the PSC of neurons in the population representing $x(t)$.

To use this description in a neural model, we must define the representation of the state variable of the system, i.e., $x(t)$. Given principle 1, let us define this representation using the following encoding and decoding:

$$a_j(t) = G_j\left[\alpha_j \left\langle x(t)\tilde{\phi}_j \right\rangle + J_j^{bias}\right] \tag{6.3}$$

and

$$\hat{x}(t) = \sum_j a_j(t)\phi_j^x. \tag{6.4}$$

Note that the encoding weight $\tilde{\phi}_j$ plays the same role as the encoding vector in (A.2), but is simply ± 1 (for 'on' and 'off' neurons) in the scalar case. Figure 6.13 shows a population of neurons with this kind of encoding. Let us also assume an analogous representation for $u(t)$.

Working in the time domain, we can take our description of the dynamics,

$$x(t) = h'(t) * \left[A'x(t) + B'u(t)\right]$$

and substitute it into (6.3), to give

$$a_j(t) = G_j\left[\alpha_j \left\langle \tilde{\phi}_j h'(t) * \left[A'x(t) + B'u(t)\right] \right\rangle + J_j^{bias}\right]. \tag{6.5}$$

Substituting our decoding (6.4) into (6.5) for both populations gives

$$\begin{aligned} a_j(t) &= G_j\left[\alpha_j \left\langle h'(t) * \tilde{\phi}_j \left[A' \sum_i a_i(t)\phi_i^x + B' \sum_k b_k(t)\phi_k^u\right] \right\rangle + J_j^{bias}\right] && (6.6) \\ &= G_j\left[h'(t) * \left[\sum_i \omega_{ji} a_i(t) + \sum_k \omega_{jk} b_k(t)\right] + J_j^{bias}\right] && (6.7) \end{aligned}$$

where $\omega_{ji} = \alpha_j A' \phi_i^x \tilde{\phi}_j$ and$\omega_{jk} = \alpha_j B' \phi_k^u \tilde{\phi}_j$ are the recurrent and input connection weights respectively. Note that i is used to index population activity at the previous

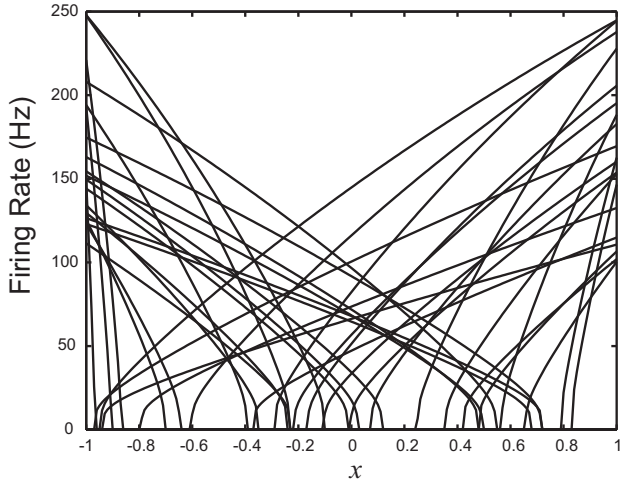

Figure 6.13: Sample tuning curves for a population of neurons used to implement the line attractor. These are the equivalent steady state tuning curves of the spiking neurons used in this example. They are found by solving the differential equations for the LIF neuron assuming a constant input current, and are described by: $a_j(x) = \frac{1}{\tau_j^{ref} - \tau_j^{RC} \ln\left(1 - \frac{J_{threshold}}{\alpha_j x + J^{bias}}\right)}$. (taken from eliasmith 2005)

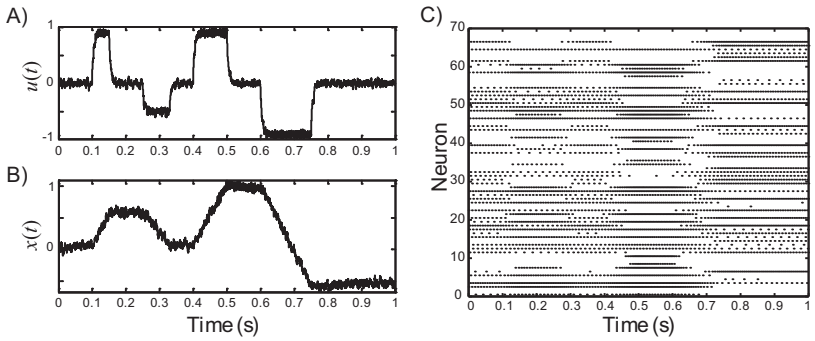

Figure 6.14: A) The decoded input $u(t)$, B) the decoded integration $x(t)$ of a spiking line attractor with 200 neurons under 10% noise, and C) spike rasters of a third of the neurons in the population. (taken from eliasmith 2005)

'time step'[2] and G_i is a spiking nonlinearity. It is important to keep in mind that the temporal filtering is only done once, despite this notation. That is, $h'(t)$ is the same filter as that defining the decoding of both $x(t)$ and $u(t)$. More precisely, this equation should be written as

$$\sum_n \delta_j(t-t_n) = G_j\left[\sum_{i,n} \omega_{ji} h'_i(t) * \delta_i(t-t_n) + \ldots \right. \tag{6.8}$$

$$\left. \sum_{k,n} \omega_{jk} h'_k(t) * \delta_k(t-t_n) + J_j^{bias}\right]. \tag{6.9}$$

The dynamics of this system when $h'_i(t) = h'_k(t)$ are as written in (6.5), which is the case of most interest as it best approximates a true integrator. Neverthless, they do not have to be equal and model a broader class of dynamics when this is included in the higher-level analysis.

[2] In fact, there are no discrete time steps since this is a continuous system. However, the PSC effectively acts as a time step, as it determines the length of time that previous information is available.

For completeness, we can write the sub-threshold dynamical equations for an individual LIF neuron voltage $V_j(t)$ in this population as follows:

$$\frac{dV_j}{dt} = -\frac{1}{\tau_j^{RC}}\left(V_j - R_j\left[\sum_{i,n}\omega_{ji}h_i'(t) * \delta_i(t-t_n) + ... \right.\right. \tag{6.10}$$

$$\left.\left.\sum_{k,n}\omega_{jk}h_k'(t) * \delta_k(t-t_n) + J_j^{bias}\right]\right) \tag{6.11}$$

where $\tau_j^{RC} = R_jC_j$, $R_j$is the membrane resistance and C_j the membrane capacitance. As usual, the spikes are determined by choosing a threshold voltage for the LIF neuron (V_{th}) and placing a spike when $V_j > V_{th}$. In our models, we also include a refractory time constant τ_j^{ref}, which captures the absolute refractory period observed in real neurons. Figure 6.14 shows a brief sample run for this network.

To gain insight into the network's function, both as an attractor and an integrator, it is important to derive measures of the networks behavior. This has already been done to some extent for line attractors, so I will not discuss such measures here **??**. What these analyses make clear, however, is how higher-level properties, such as the effective time constant of the network, are related to neuron-level properties, such as membrane and synaptic time constants. Because the previous derivation is part of a general method for building more complex attractor networks (as I discuss next), it becomes evident how these same analyses can apply in the more complex cases. This is a significant benefit of generating models with a set of unified principles. More importantly from a practical standpoint, constructing this network by employing control theory makes it evident how to control some of the high-level properties, such as the effective network time constant (see section **??**). It is this kind of control that begins to make clear how important such simple networks are for understanding neural signal processing.

****control**** As described in section **??**, a line attractor is implemented in the neural integrator in virtue of the dynamics matrix A' being set to 1. While the particular output value of the attractor depends on the input, the dynamics of the attractor are controlled by A'. Hence, it is natural to inquire as to what happens as A' varies over time. Since A' is unity feedback, it is fairly obvious what the answer to this question is: as A' goes over 1, the resulting positive feedback will cause the circuit to saturate; as A' becomes less than one, the circuit begins to act as a low-pass filter, with the cutoff frequency determined by the precise value of

A'. Thus, I can build a tunable filter by using the same circuit and allowing direct control over A'.

To do so, we can introduce another population of neurons d_l that encode the value of $A'(t)$. Because A' is no longer static, the product $A'x$ must be constantly recomputed. This means that our network must support multiplication at the higher level. The two most obvious architectures for building this computation into the network are shown in figure 6.15. Both architectures are implementations of the same high-level dynamics equation

$$x(t) = h'(t) * \big(A'(t)x(t) + \tau u(t)\big) \tag{6.12}$$

which is no longer LTI, as it is clearly a time-varying system. Notably, while both architectures demand multiplication at the higher level, this does not mean that there needs to be multiplication between activities at the neural level. This is because, as mentioned in section **??** and demonstrated in section **??**, nonlinear functions can be determined using only linear decoding weights.

As described in **?**, the first architecture can be implemented by constructing an intermediate representation of the vector$\mathbf{c} = [A', x]$ from which the product is extracted using linear decoding. The result is then used as the recurrent input to the a_i population representing x. This circuit is successful, but performance is improved by adopting the second architecture.

In the second architecture, the representation in a_i population is taken to be a 2D representation of $\mathbf{x}$ in which the first element is the integrated input and the second element is A'. The product is extracted directly from this representation using linear decoding and then used as feedback. This has the advantage over the first architecture of not introducing extra delays and noise.

Specifically, let $\mathbf{x} = [x_1, x_2]$ (where $x_1 = x$ and $x_2 = A'$ in 6.12). So, a more accurate description of the higher-level dynamics equation for this system is

$$\begin{aligned} \dot{\mathbf{x}} &= h'(t) * \big(\mathbf{A}'\mathbf{x} + \mathbf{B}'\mathbf{u}\big) \\ \begin{bmatrix} \dot{x_1} \\ x_2 \end{bmatrix} &= h'(t) * \left(\begin{bmatrix} x_2 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \tau & 0 \\ 0 & \tau \end{bmatrix} \begin{bmatrix} u \\ A' \end{bmatrix} \right) \end{aligned} \tag{6.13}$$

which makes the nonlinear nature of this implementation explicit. Notably, here the desired A' is provided as input from a preceding population, as is the signal to be integrated, u. To implement this system, we need to compute the transformation

$$\hat{p}(t) = \sum_i a_i(t)\phi_i^p,$$

where $p(t)$ is the product of the elements of $\mathbf{x}$. Substituting this transformation into (6.6) gives

$$\begin{aligned} a_j &= G_j\left[\alpha_j\left\langle h' * \tilde{\phi}_j\left[\sum_i a_i(t)\phi_i^p + B'\sum_k b_k\phi_k^u\right]\right\rangle + J_j^{bias}\right] \\ &= G_j\left[\sum_i \omega_{ij}a_i(t) + \sum_k \omega_{kj}b_k(t) + J_j^{bias}\right] \end{aligned} \tag{6.14}$$

where $\omega_{ij} = \alpha_j\tilde{\phi}_j\phi_i^p$, $\omega_{kj} = \alpha_j\tilde{\phi}_j B'\phi_k^u$ and

$$a_i(t) = h' * G_i\left[\alpha_i\left\langle \mathbf{x}(t)\tilde{\phi}_i\right\rangle + J_i^{bias}\right].$$

The results of simulating this nonlinear control system are shown in figure 6.16. This run demonstrates a number of features of the network. In the first tenth of a second, the control signal $1 - A'$ is non-zero, helping to eliminate any drift in the network for zero input. The control signal then goes to zero, turning the network into a standard integrator over the next two-tenths of a second when a step input is provided to the network. The control signal is then increased to .3, rapidly forcing the integrated signal to zero. The next step input is then filtered by a low pass filter, since the control signal is again non-zero. The third step input is also integrated, as the control signal is zero. Like the first input, this input is forced to zero by increasing the control signal, but this time the decay is much slower because the control signal is lower (.1). These behaviors show how the control signal can be used as a reset signal (by simply making it non-zero), or as a means of determining the properties of a tunable low-pass filter.

So, the introduction of control into the system gave us a means of radically altering the attractive properties of the system. It is only while $A' = 1$ that we have an approximate line attractor. For positive values less than one, the system no longer acts as a line attractor, but rather as a point attractor, whose basin properties (e.g., steepness) vary as the control signal.

As can be seen in (6.14), there is no multiplication of neural activities. There is, of course, significant debate about whether, and to what extent, dendritic nonlinearities might be able to support multiplication of neural activity (see, e.g., **?????**). As a result, it is useful to demonstrate that it is possible to generate circuits without multiplication of neural activities that support network level nonlinearities. If dendritic nonlinearities are discovered in the relevant systems, these

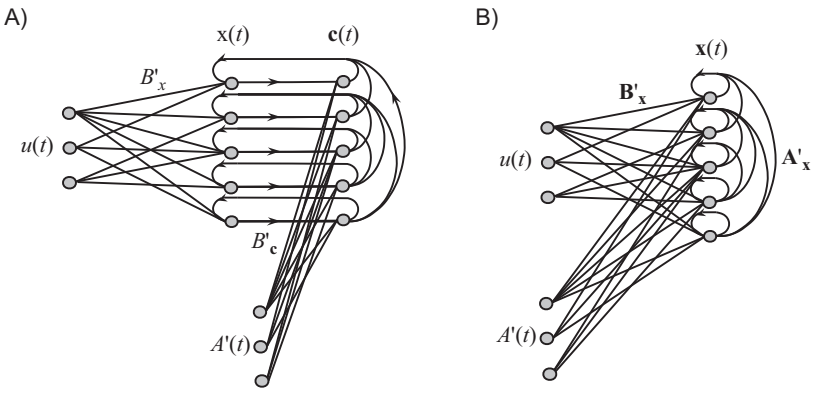

Figure 6.15: Two possible network architectures for implementing the controllable filter. The *B* variables modify the inputs to the populations representing their subscripted variable. The *A* variables modify the relevant recurrent connections. The architecture in A) is considered in **?**, the more efficient architecture in B) is considered here. (taken from eliasmith 2005)

networks would become much simpler (essentially we would not need to construct the intermediate **c** population).

- Relate to working memory model and rat path integration, etc.

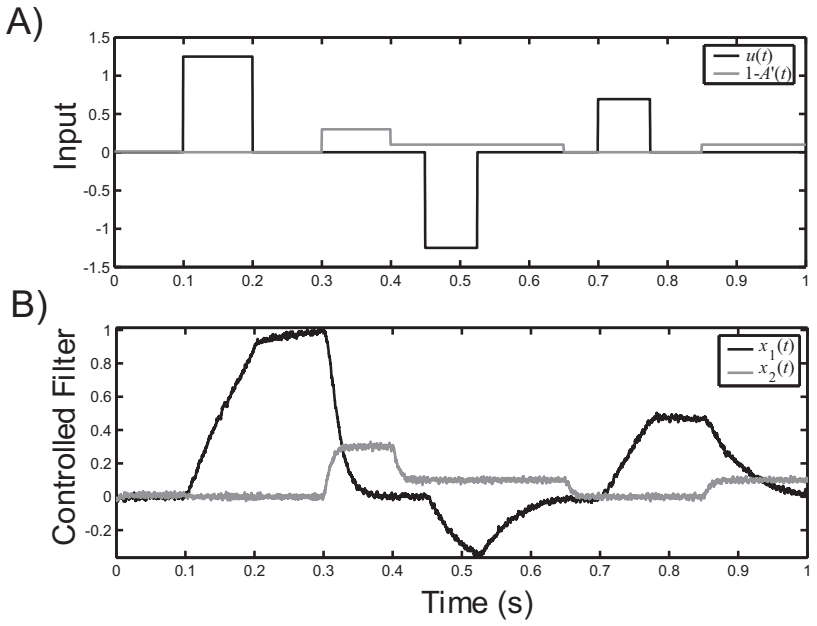

Figure 6.16: The results of simulating the second architecture for a controllable filter in a spiking network of 2000 neurons under 10% noise. A) The input signals to the network. B) The high-level decoded response of the spiking neural network. The network encodes both the integrated result and the control signal directly, to efficiently support the necessary nonlinearity. See text for a description of the behavior. (taken from eliasmith, 2005)