Available online at www.sciencedirect.com

ScienceDirect

Cognitive Systems Research 12 (2011) 84–92

Cognitive Systems RESEARCH

www.elsevier.com/locate/cogsys

A biologically realistic cleanup memory: Autoassociation in spiking neurons

Action editor: Richard P. Cooper

Terrence C. Stewart *, Yichuan Tang, Chris Eliasmith

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

Received 15 January 2010; accepted 23 June 2010
Available online 10 August 2010

Abstract

Methods for "cleaning up" (or recognizing) states of a neural network are crucial for the functioning of many neural cognitive models. This process takes a noisy approximation of a state and recovers the original information. As a particular example, we consider the cleanup required for the use of Vector Symbolic Architectures, which provide a method for manipulating symbols using a fixed-length vector representation. To recognize the result of these manipulations, a mechanism for cleaning up the resulting noisy representation is needed, as this noise increases with the number of symbols being combined. While these symbolic manipulations have previously been modelled with biologically plausible neurons, this paper presents the first spiking neuron model of the cleanup process. We demonstrate that it approaches ideal performance and that the neural requirements scale linearly with the number of distinct symbols in the system. While this result is relevant for any biological model requiring cleanup, it is crucial for VSAs, as it completes the set of mechanisms needed to provide a full neural implementation of symbolic reasoning.
© 2010 Elsevier B.V. All rights reserved.

Keywords: Auto-associative memory; Neural engineering framework; Vector Symbolic Architectures; Holographic reduced representation

1. Introduction

A fundamental component of many cognitive architectures is an auto-associative memory. This is a system that can be provided with a partial or noisy version of a previously stored memory and will in turn provide a complete and more accurate version of that memory. This can be seen in ACT-R's declarative memory system (Anderson & Lebiere, 1998), CLARION's non-action-centered subsystem (Sun, 2006), RAAM's compressor and reconstructor (Pollack, 1988), and many other cognitive models. This capability can be implemented using a wide variety of approaches, including multilayer perceptrons, Hopfield networks, and any prototype-based classifier.

The particular use of auto-associative memory of importance to this paper is as a *cleanup memory* for cognitive operations. Recent research has shown that the storage and manipulation of cognitive symbol systems can be implemented as mathematical operations on fixed-length, high-dimensional vectors. These approaches are known as Vector Symbolic Architectures (VSAs; Gayler, 2003) and include Holographic Reduced Representation (HRR; Plate, 2003), MAP Coding (Gayler & Wales, 2000), Binary Splatter Codes (Kanerva, 1997), and others. Each of these provides an alternate method for converting symbols and symbol trees into vectors, combining vectors to perform symbolic manipulations, and extracting out the original components of that symbol tree. However, when symbols are combined and manipulated using these techniques, the results are *approximate*, in that the output is close to, but not exactly the same as, the original symbols.

In previous research we have shown how VSAs can be implemented using biologically realistic spiking neurons

* Corresponding author.
E-mail addresses: tcstewar@uwaterloo.ca (T.C. Stewart), y3tang@uwaterloo.ca (Y. Tang), celiasmith@uwaterloo.ca (C. Eliasmith).

1389-0417/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.cogsys.2010.06.006

(Eliasmith, 2005; Stewart & Eliasmith, 2008). This approach is orders of magnitude more efficient[1] than alternate theories of how symbolic manipulations could be performed by the brain (Stewart & Eliasmith, in press), and is the basis for our model of a neural production system (Stewart, Choo, & Eliasmith, 2010a,b). However, one common criticism is that we have not previously shown how these systems can cleanup their representations. Performing symbol manipulations using VSAs is an inherently noisy process, and these operations must be performed by spiking neurons, adding a significant amount of random variation. When symbols are extracted from a combined representation, the brain needs a reliable method for identifying which symbol it is, allowing it to respond appropriately.

The purpose of this paper is to present an auto-associative memory constructed from spiking neurons which is appropriate for cleaning up the representations resulting from cognitive manipulations using VSAs. We start by describing the characteristics of VSAs that define the statistical properties of the noise that must be removed. Next, a general method is described for encoding (and decoding) high-dimensional vectors across a population of spiking neurons. We then show that standard approaches to deriving connection weights have difficulty when scaled up to the number of symbols required for human vocabularies. Our cleanup memory model is then presented, followed by an analysis of its behaviour.

1.1. Vector Symbolic Architectures

Vector Symbolic Architectures (VSAs) map symbols and symbol structures to vectors. To begin, each symbol is represented by a high-dimensional vector. These vectors can be randomly chosen, or they can be chosen to respect semantic similarity, so that the vectors for two conceptually similar symbols, like **cat** and **dog**, will also be similar. This notion of *similarity* is important for VSAs, and is generally defined based on having a small angle between two vectors, or, equivalently, having a large dot product. For simplicity, in this paper we randomly choose the vectors for each symbol. For high-dimensional vectors, this means that any two symbols will be highly dissimilar (i.e. will have a dot product near zero).

To create symbol structures, we combine the vectors for symbols using mathematical operations. The two core operations are superposition (+) and binding (⊗). Two vectors can be combined by superposition (+) to produce a new vector that is similar to both of the original vectors. Alternatively, two vectors can be combined by binding (⊗) to produce a new vector that is *dissimilar* to both of the original vectors. Importantly, this operation can be approximately reversed by binding with the inverse of a vector (denoted with an underline), such that $A \otimes B \otimes \underline{B} \approx A$. These operations are similar to standard addition and multiplication in terms of being associative, commutative, and distributive. With such a system we can represent a structure such as **chase(dog,cat)** by performing the following calculation:

$$\mathbf{chase} \otimes \mathbf{action} + \mathbf{dog} \otimes \mathbf{actor} + \mathbf{cat} \otimes \mathbf{patient}$$

The result is a single vector of the same length as the vectors for the basic symbols (**chase**, **action**, **dog**, etc.).

This one vector can be interpreted as a representation of the entire structure because it is possible to extract the original components. For example, to determine the patient of the above structure, we take the whole vector and bind it with the inverse of **patient**.

$$\begin{aligned}
&\mathbf{chase} \otimes \mathbf{action} + \mathbf{dog} \otimes \mathbf{actor} + \mathbf{cat} \otimes \mathrm{patient}) \otimes \underline{\mathbf{patient}} \\
&\quad = \mathbf{chase} \otimes \mathbf{action} \otimes \underline{\mathbf{patient}} + \mathbf{dog} \otimes \mathbf{actor} \otimes \underline{\mathbf{patient}} \\
&\qquad + \mathbf{cat} \otimes \mathbf{patient} \otimes \underline{\mathbf{patient}} \\
&\quad \approx \mathbf{cat} + \mathbf{chase} \otimes \mathbf{action} \otimes \underline{\mathbf{patient}} \\
&\qquad + \mathbf{dog} \otimes \mathbf{actor} \otimes \underline{\mathbf{patient}}
\end{aligned}$$

The result is a vector that is similar to cat, but is not exactly the same since it has two additional terms superposed on it. Due to the properties of the binding operation, however, these two terms $\mathbf{chase} \otimes \mathbf{action} \otimes \underline{\mathbf{patient}}$ and $\mathbf{dog} \otimes \mathbf{actor} \otimes \underline{\mathbf{patient}}$ will be vectors that are *dissimilar* to all of the original symbols, and can thus be treated as randomly distributed noise. It is this noise that must be removed by the cleanup memory system.

While the above discussion applies to all VSAs, if we choose one particular type of VSA we identify the properties of the symbol and noise vectors. For this, we use Holographic Reduced Representations (HRRs; Plate, 2003). Here, each basic symbol vector is a point on the high-dimensional unit sphere (i.e. a random vector normalized to a length of one). Superposition is performed by vector addition and the binding operation is circular convolution.

We can now define our requirements for a cleanup memory. This system will take as input a single vector and it will output the vector that it is closest to. In other words, given the vector $\mathbf{cat} + \mathbf{chase} \otimes \mathbf{action} \otimes \underline{\mathbf{patient}} + \mathbf{dog} \otimes \mathbf{actor} \otimes \underline{\mathbf{patient}}$ (the result of trying to determine the patient of **chase(dog,cat)**) it will output the vector **cat**.

To do this, our cleanup memory needs the following properties:

(1) Be able to output one of M unit vectors (one per symbol), distributed uniformly over a high-dimensional unit sphere.
(2) Handle additive noise produced by adding k unit vectors uniformly distributed over the same sphere (in the above case, $k = 2$).

To be useful for cognitive operations, on the order of 100,000 symbols (M) must be able to be identified. The complexity of the structures that can be encoded is

[1] For realistic vocabulary sizes, our approach uses three orders of magnitude fewer neurons than the Neural Blackboard Architecture (van der Velde & de Kamps, 2006) and seven orders of magnitude fewer than LISA (Hummel & Holyoak, 2003).

determined by k, indicating the number of terms that can be superposed and still lead to accurate recognition. This should be at least 7 ± 2 to conform to the standard chunk sizes used in cognitive modelling.

Since the cleanup is to be performed by spiking neurons, the result will not be a perfect representation of the desired value; there will always be some inaccuracy. To measure this error, we take the dot product of the correct vector and the output of the memory; the larger the value the greater the chance that the symbol is successfully recognized. Plate (2003, p. 100) provides a method for determining how large this accuracy value should be in special cases where k is fixed and known, and generally values above 0.7 are sufficient for recognition. Without a cleanup memory, the dot product will be quite small: for $k = 6$ the average value will be around 0.38 (or $1/\sqrt{6}$).

The final factor to consider when using Vector Symbolic Architectures is the number of dimensions. In an ideal case (where vectors are represented exactly, rather then via noisy spiking neurons), Plate (2003) derived the following formula for determining the minimum number of dimensions D required to represent combinations of k vectors out of M symbols and have a probability of error q:

$$D = 4.5(k + 0.7)\ln(M/30q^4) \quad (1)$$

From this, we note that 700 dimensions would be sufficient to represent chunks of up to seven symbols out of a vocabulary of 100,000 with an accuracy of 95%. However, this formula assumes an ideal cleanup memory, rather than one constructed using spiking neurons. This ideal performance will be used as a benchmark for comparison.

1.2. Distributed representation

There are a variety of methods whereby a numerical vector can be represented by a population of spiking neurons. The most simplistic approach is to have one neuron per dimension, and the firing rate of that neuron indicates the value in that dimension. However, this approach is highly fragile to neuron death and does not correspond to known methods of spatial representation by neurons. It is well established (e.g., Georgopoulos, Schwartz, & Kettner, 1986) that two-dimensional movement directions are encoded in a highly distributed manner by having a large population of neurons, each of which is most sensitive to a different direction. That is, each neuron has a preferred direction: a particular vector for which it will fire most strongly. If a movement is in a slightly different direction, then the firing rate for that neuron will be decreased. These preferred directions are generally observed to be uniformly distributed around the unit circle.

We generalize this approach to encode high-dimensional vectors. If each neuron has a preferred direction vector $\boldsymbol{e}$ then we can capture the observed behaviour by setting the ionic current entering the neuron based on the dot product between $\boldsymbol{e}$ and the vector $\boldsymbol{x}$ being represented. If α is the neuron gain and $\boldsymbol{J}_{bias}$ is a fixed background current, then the total current $\boldsymbol{J}$ flowing into neuron i is:

$$\boldsymbol{J} = \alpha\boldsymbol{e}\cdot\boldsymbol{x} + \boldsymbol{J}_{bias} \quad (2)$$

This current can be used as the input for any spiking neural model, such as the standard leaky integrate-and-fire (LIF) model, which is what we use here. This allows us to convert any vector into a pattern of spikes that represents that vector. Since this spiking pattern is meant to represent the original vector $\boldsymbol{x}$, it should be possible to determine an estimate of $\boldsymbol{x}$ given only this spiking pattern. We compute this by determining a decoding vector $\boldsymbol{d}$ for each neuron using Eq. (3), where $a_i(\boldsymbol{x})$ is the average firing rate for neuron i when representing the value $\boldsymbol{x}$, and integration is over the range of possible values of $\boldsymbol{x}$. The resulting $\boldsymbol{d}_i$ values are the least-squares linearly optimal decoding vectors for recovering the original value $\boldsymbol{x}$ given only the outputs from each of these neurons. That is, we can take the outputs from each neuron, multiply them by $\boldsymbol{d}_i$, and sum the results to produce an estimate of $\boldsymbol{x}$. This method has been shown to uniquely combine accuracy and neurobiological plausibility (e.g., Salinas & Abbott, 1994).

$$\boldsymbol{d} = \Gamma^{-1}\gamma \quad \Gamma_{ij} = \int a_i(\boldsymbol{x})a_j(\boldsymbol{x})dx \quad \gamma_i = \int a_j(\boldsymbol{x})\boldsymbol{x}\,dx \quad (3)$$

The representational error introduced by this method is dependent on the particular neural parameters and encoding vectors, but in general is inversely proportional to the number of neurons in the group (Eliasmith & Anderson, 2003). That is, by doubling the number of neurons involved, the root mean squared error of the representation will be halved.

While the decoding vectors $\boldsymbol{d}$ are useful for determining what value a spike pattern represents, a more important feature is that they can also be used to derive optimal synaptic connection weights between neural groups. That is, if one group of neurons represents $\boldsymbol{x}$ and we have another group of neurons that we want to represent $\boldsymbol{Wx}$ (i.e. any linear transformation of $\boldsymbol{x}$), then this can be achieved by setting their synaptic connection weights ω_{ij} using the following equation

$$\omega_{ij} = \alpha_j\boldsymbol{e}_j\boldsymbol{W}\boldsymbol{d}_i \quad (4)$$

While Eq. (4) allows us to compute any linear function of $\boldsymbol{x}$, we can also compute non-linear functions by adjusting Eq. (3). By changing $a_j(\boldsymbol{x})\boldsymbol{x}$ to $a_j(\boldsymbol{x})f(\boldsymbol{x})$, we find decoding vectors that will approximate the function $f(\boldsymbol{x})$. The accuracy of this approximation is dependent on the characteristics of the function, and in general the more non-linear interactions there are between the dimensions of $\boldsymbol{x}$, the lower the accuracy (see Eliasmith & Anderson, 2003 for further details). This will be used in Section 2.2.

These results provide a generic framework for representing vectors of any dimension using spiking neurons. These neurons can be made as realistic as possible (given computational processing constraints), including effects of adaptation, neurotransmitter re-uptake rates, refractory periods, and so on. The only limitation is that there be some steady-state average firing rate output from the neuron model given a constant input determined using Eq. (2). Furthermore, we can use Eq. (4) to derive the optimal synaptic connection weights that will cause the neurons to approximately calculate any linear or non-linear function of the represented values. These techniques form the basis of the Neural Engineering Framework (Eliasmith & Anderson, 2003), and are suitable for modelling a wide range of sensory and cognitive systems.

2. Standard approaches

Given the above representation system, we have two groups of neurons: one representing the input (noisy) vector, and one that should represent the output (cleaned) vector. The goal is to determine how to connect these neurons so as to achieve the best cleanup.

For this work, we are only considering feed-forward networks. That is, we do not consider models where activity flows backwards from the output to the input, or where the output is the same group of neurons as the input, but at a later time. These models, such as the Hopfield network, must wait for their output to "settle", requiring significantly more time than purely feed-forward models. If cleanup systems are to be used by cognitive agents directly interacting with the world, state recognition needs to proceed as quickly as possible.

2.1. Linear autoassociation

The simplest autoassociation memory merely performs a linear transformation on the input to produce the output (Hinton & Andersen, 1989). If the matrix $\boldsymbol{X}$ consists of a set of noisy vectors and the matrix $\boldsymbol{Y}$ holds the corresponding cleaned vectors, then we want to find $\boldsymbol{W}$ such that $\boldsymbol{WX} \approx \boldsymbol{Y}$. Given the subsequent noisy vector $\boldsymbol{x}$, it can then be multiplied by $\boldsymbol{W}$ to produce the estimated cleaned up item $\boldsymbol{y} = \boldsymbol{Wx}$. Once $\boldsymbol{W}$ is found, we derive the synaptic connection weights for this linear transformation using Eq. (4).

A variety of methods exist to find the $\boldsymbol{W}$ that minimizes the error between $\boldsymbol{WX}$ and $\boldsymbol{Y}$. Fig. 1 shows the result of using the Penrose–Moore pseudoinverse, which was chosen since $\boldsymbol{X}$ is generally not full rank.

These results show that using a linear autoassociation model does not approach ideal performance. The ideal system is able to deal with large vocabulary sizes (M) when the number of dimensions (D) is increased sufficiently, giving accuracy values above 0.9. However, the linear autoassociator is unable to do this, and for $M > 10$ shows little improvement (if any) over not having any cleanup memory at all.

2.2. Direct function approximation

A second possibility is to directly determine the optimal connection weights for computing the cleanup function. This can be thought of in two mathematically equivalent ways. First, we can follow the same method as in the previous section, but instead of $\boldsymbol{X}$ and $\boldsymbol{Y}$ being the vectors being represented, we use the average firing rate of the individual neurons when representing those vectors. Alternatively, we replace $\boldsymbol{x}$ with $f(\boldsymbol{x})$ when calculating the decoders in Eq. (3), where $f(\boldsymbol{x})$ is the cleanup function (i.e. a function which returns the closest vector in the vocabulary for a given $\boldsymbol{x}$). This approach is used extensively in the Neural Engineering Framework (Eliasmith & Ander-

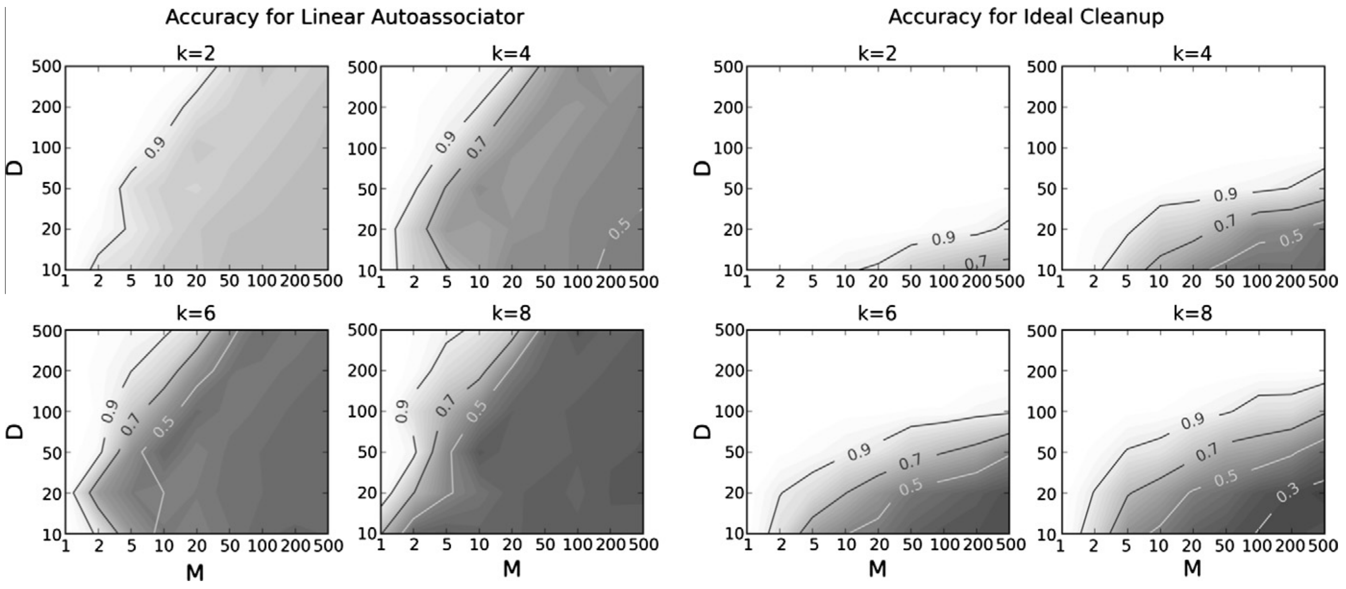

Fig. 1. Accuracy of the linear autoassociation network for varying D (dimensions in the vector), M (vocabulary size), and k (number of terms) compared to an ideal (non-neural) cleanup. The larger the value, the closer the output comes to the correct (clean) result. Without cleanup, the values would be 0.58, 0.45, 0.38, and 0.33 for $k = 2$, 4, 6, and 8, respectively.

son, 2003) to derive synaptic connection weights that can perform non-linear operations.

The results in Fig. 2 show that, while this approach is a significant improvement over Fig. 1 in terms of handling for complex structures (larger k) at a smaller number of dimensions (D), it is still unable to accurate deal with large vocabularies (M).

2.3. Multilayer perceptron

One potential reason for the failure of the direct function approximation approach discussed in Section 2.2 is that the function being computed is highly non-linear. To address this, we can make use of a multilayer perceptron, capable of computing much more complex functions. This involves introducing a new hidden layer of neurons between the input and output.

The multilayer perceptron (MLP) is the most famous and widely used artificial neural network (Rumelhart, Hinton, & Williams, 1986). Using a two layer MLP, a mapping is learned to convert noisy input vectors into their cleaned (or prototype) vectors.

Instead of directly calculating the weights for these networks, a learning rule (such as the classic backpropagation of error rule) must be used. This allows the system to find a suitable intermediate representation in the hidden layer which makes the cleanup operation most accurate. For this task we trained the MLP using gradient descent on the sum of the squared error.

In theory, given enough time, hidden nodes, and a sufficiently powerful optimization algorithm, this approach should be able to find the optimal synaptic connection weights to perform this task. However, as the results in Fig. 3 show, due to limited computational resources we were unable to successfully train this network for large M. This is in part due to the fact that the MLP requires many more hidden nodes than the vector dimension in order to generalize across the entire input domain.

More importantly, the standard strengths of a backpropagation network are not applicable to the cleanup task. Crucially, there is no inner structure in the data being modelled; each symbol is a randomly chosen unit vector. This means that the network cannot use its hidden layer to form an internal representation that simplifies the task.

Overall, it is likely possible to improve on this approach to training a network to perform cleanup. However, such a method may require significantly larger amounts of computing resources as M increases.

3. A cleanup memory model

From the MLP model (Section 2.3), it is clear that while transforming the initial representation through a middle layer of neurons can provide a significant improvement, it is impractical to learn the required synaptic connection weights for large M. Instead, for our cleanup memory model we choose to directly derive connection weights suitable for this task. To do this, we first identify how we want the middle layer of neurons to respond. This involves defining their preferred direction vectors $\boldsymbol{e}$, gain α, and $\boldsymbol{J}_{bias}$ as per Eq. (2). Given these, we can use Eq. (4) to derive the neural connection weights that will result in this behaviour. Since no transformation of the represented vector itself is to be performed by the weights, $\boldsymbol{W}$ in Eq. (4) is set to be the identity matrix.

For the preferred direction vectors $\boldsymbol{e}$, we choose exactly those vectors that must be cleaned up. For redundancy, we have ~10 neurons for each of the M vectors, meaning that

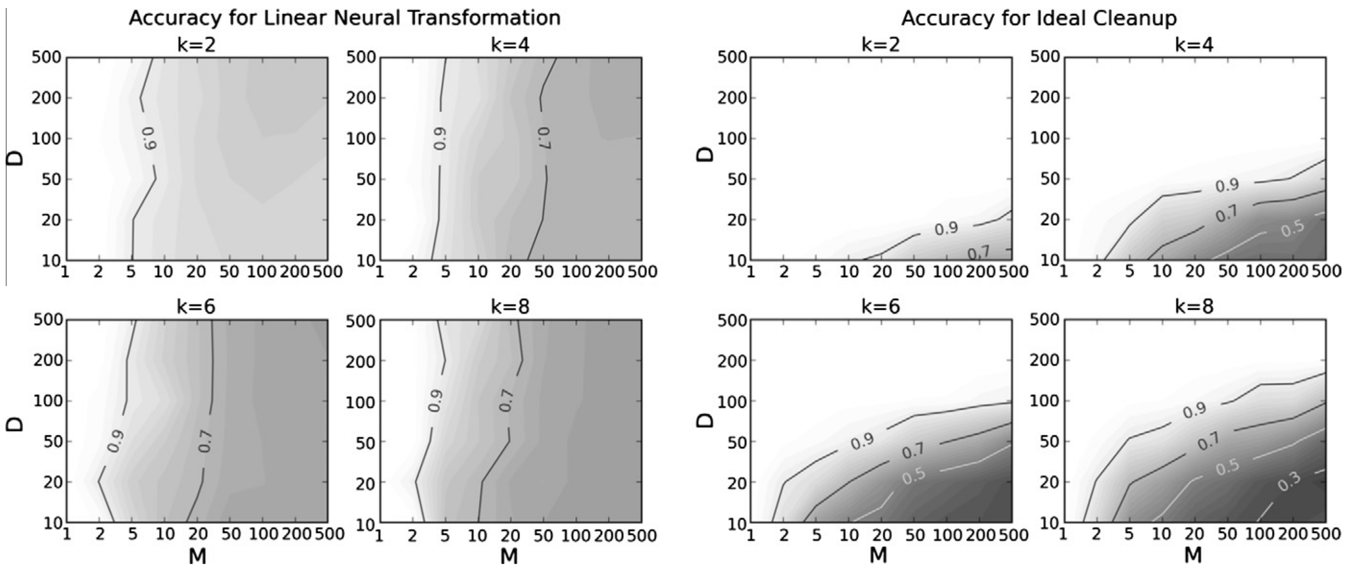

Fig. 2. Accuracy of the direct function approximation approach for varying D (dimensions in the vector), M (vocabulary size), and k (number of terms) compared to an ideal (non-neural) cleanup. The larger the value, the closer the output comes to the correct (clean) result. Without cleanup, the values would be 0.58, 0.45, 0.38, and 0.33 for $k = 2$, 4, 6, and 8, respectively.

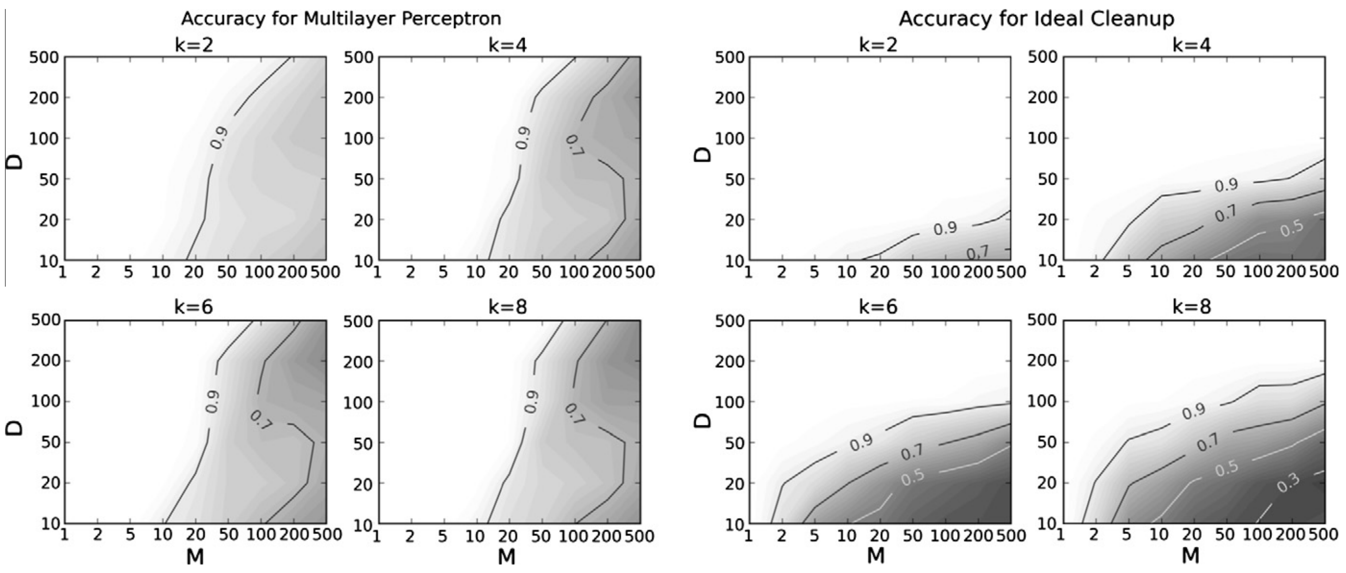

Fig. 3. Accuracy of the multilayer perceptron for varying D (dimensions in the vector), M (vocabulary size), and k (number of terms) compared to an ideal (non-neural) cleanup. The larger the value, the closer the output comes to the correct (clean) result. Without cleanup, the values would be 0.58, 0.45, 0.38, and 0.33 for $k = 2$, 4, 6, and 8, respectively.

there are particular neurons that fire maximally for each symbol. Furthermore, we set $\boldsymbol{J}_{bias}$ to be slightly negative for each neuron, but allow α and the maximum firing rates to vary as is typical in neural populations. The resulting connection weights ω_{ij} cause the middle layer neurons to only fire if the dot product of the input vector with the corresponding clean vector is greater than some small threshold.

In effect, the inherent non-linearity of the neurons (the fact that they do not fire if their input current is too low) is being used to perform cleanup. This middle layer is good at representing the cleaned vectors, but is poor at representing small vectors in those directions. Since the noise that we are trying to cleanup consists of randomly chosen vectors, these will generally have small dot products with each of the preferred direction vectors, and so will not cause sufficient activation for the neuron to fire. The presence of a slight background inhibition (the negative $\boldsymbol{J}_{bias}$) allows the neurons to be insensitive to the noise.

The firing rates of ten sample middle layer neurons are shown in Fig. 4. Their activity varies as the dot product of the input and the neurons' preferred direction vector changes. The closer the represented value is to this particular preferred vector, the higher the firing rate for each neuron, much like is commonly found in sensory and motor neurons.

Given this middle layer representation we can then calculate the optimal connection weights with the output neural group. This output group can have any arbitrarily chosen preferred direction vectors and other neural properties, much like the original input population. Eq. (4) is used to calculate these weights, again setting $\boldsymbol{W}$ to be the identity matrix.

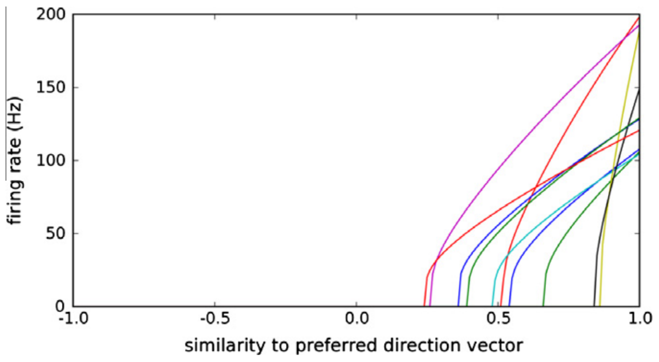

Fig. 4. Middle layer neuron tuning curves. Average firing rates for 10 neurons are shown as the input to the cleanup memory changes. Similarity is the dot product of the input vector with the preferred direction vector.

3.1. Performance

We evaluated this implementation of cleanup memory with an optimized middle layer of neurons in the same manner as the previous models and the results are shown in Fig. 5.

The important result here is that this implementation of a cleanup memory, unlike the previous ones, does begin to scale up for large vocabulary sizes (M). Indeed, it is the only one of the models that was found to work with $M = 500$, and its performance accuracy is only slightly worse than the theoretical ideal.

To further explore this model's capabilities, we examine significantly larger vocabulary sizes in Fig. 6. Here we see that the model continues to scale up well. Importantly, our neural cleanup memory system was able to successfully cleanup combinations of eight symbols out of a vocabulary of 10,000 using 500-dimensional vectors. Furthermore, its

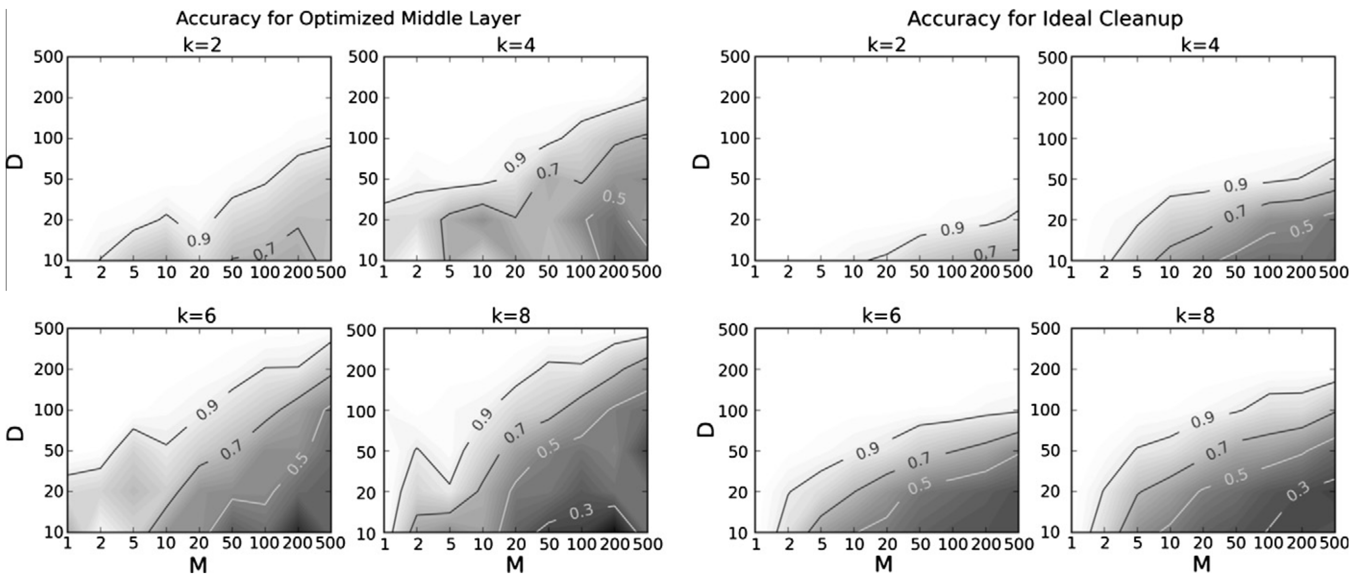

Fig. 5. Accuracy of our neural cleanup memory for varying D (dimensions in the vector), M (vocabulary size), and k (number of terms) compared to an ideal (non-neural) cleanup. The larger the value, the closer the output comes to the correct (clean) result. Without cleanup, the values would be 0.58, 0.45, 0.38, and 0.33 for $k = 2$, 4, 6, and 8, respectively.

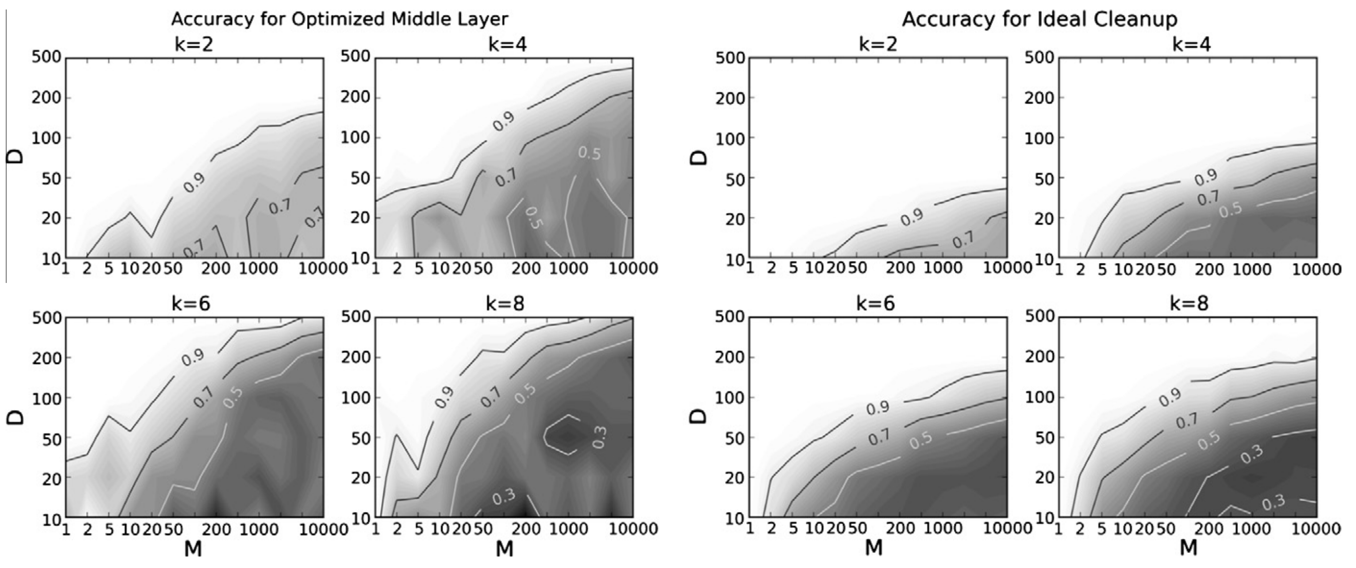

Fig. 6. Accuracy of our neural cleanup memory for varying D (dimensions in the vector), M (vocabulary size), and k (number of terms) compared to an ideal (non-neural) cleanup for larger values of M. The larger the value, the closer the output comes to the correct (clean) result. Without cleanup, the values would be 0.58, 0.45, 0.38, and 0.33 for $k = 2$, 4, 6, and 8, respectively.

capabilities increase rapidly with the number of dimensions. We have evaluated this model up to $M = 100{,}000$ and $D = 1000$, producing consistently high quality cleanup results.

We have thus demonstrated an effective implementation of a neural autoassociator as a cleanup memory for Vector Symbolic Architectures. The number of neurons required for cleanup scales linearly with M, while the number of neurons required for storing the resulting cleaned vector is linear in D. For a realistic vocabulary of 100,000 terms, we only require 20,000 neurons to represent a 1000 dimensional vector. If this vector is any combination of up to 8 slot/value pairs (as per the VSA representation discussed in Section 1.1), then we can perform operations on this vector to extract the relevant information and cleanup the result using 1,000,000 neurons. This allows complex symbol manipulation to be implemented in realistic spiking neurons using relatively few neurons (0.005% of the cortex).

3.2. Dynamics and timing

Since a cleanup memory is meant to be a component to support symbolic manipulations by spiking neurons, it must not only be efficient in terms of numbers of neurons, but also in terms of the amount of time required to perform cleanup. This is why we did not consider models that require feedback loops and a long settling time (such as a Hopfield network).

To evaluate how our model performs over time, we can decode its output using the $\boldsymbol{d}$ values from Eq. (3). That is, we take the output from each neuron i, multiply by $\boldsymbol{d}_i$, and sum the result. Since this result is a high-dimensional vector (the result of the cleanup), we can evaluate its accuracy by comparing it to the correct ideal vector by taking the dot product. Since the output from each neuron varies over time, this result will also vary over time.

The precise timing characteristics of the neural model will vary based on the neural parameters. We used typical values for cortical neurons: a refractory period of 2 ms, a membrane time constant of 20 ms, and a maximum firing rate of 200 Hz. We applied random noise in the input current to each cell of $\sigma = 10\%$. We also assumed NMDA neurotransmitter receptors, giving a time constant of $\tau = 5$ ms for the post-synaptic current. That is, instead of the output from each neuron being an instantaneous spike, we model it as having a gradual exponential decay of the form $e^{-t/\tau}$, which is a common first-order approximation of the effects of a neural spike (e.g., Jack & Redman, 1971).

To observe the resulting dynamics, we ran a cleanup memory using 500-dimensional vectors ($D = 500$) with a vocabulary size (M) of 10,000. For the input, we used five different noisy vectors comprised of eight terms each ($k = 8$), presented over the course of 250 ms of simulated time. The output from the system was measured at each time step. Fig. 7 shows the result of comparing the output of the model (the cleaned up vector) with the corresponding five original vectors. As in the rest of this paper, comparison was done by the dot product of the output vector and the desired clean vector.

These results indicate that the network reliably cleans the input vector and does so within 5–10 ms. This makes our cleanup memory suitable for fast recognition, which is needed for symbolic manipulations at a cognitive time scale.

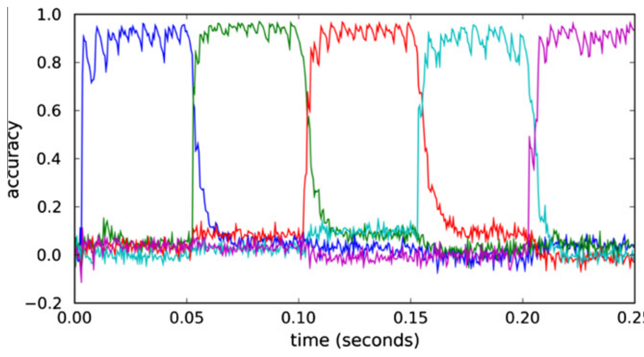

Fig. 7. Temporal accuracy of the cleanup memory. Five noisy vectors are presented for 50 ms each. Graphed lines show the dot product of the output from the network and the five original clean vectors.

4. Discussion

The model we have presented differs from standard auto-associative memories in a variety of respects. First, it is in the family of purely feed-forward autoassociation models. On the particular task of interest here (cleaning up a high-dimensional unit vector that has had k randomly distributed unit vectors added to it), it is significantly more effective than the traditional linear autoassociation models. This provides an accurate and highly powerful memory using realistic noisy neurons, but without resorting to recurrent feedback connections which require a longer time to settle before producing a clean output.

Of particular interest for us is that this memory can be used to cleanup the results of Vector Symbolic Architecture operations. We have argued elsewhere (Stewart & Eliasmith 2008, in press; Stewart et al., 2010b) that VSAs are highly suitable for implementing high-level symbolic operations using realistic spiking neurons. The required superposition (+) and binding ($\otimes$) operations (vector addition and circular convolution, respectively) are well approximated using synaptic connections calculated using Eq. (4), and this results in a flexible method for creating neural cognitive models that manipulate symbolic structures. However, as demonstrated in Section 1.1, each VSA operation introduces noise into the representation. Given a representation of **chase(dog,cat)**, extracting one of the components does not produce exactly **cat**, but rather **cat** plus some random noise. In the non-neural VSA models of Plate (2003), Gayler (2003), and Kanerva (1997), it is simply assumed that there is some mechanism to find the closest known vector given a noisy representation. We have shown that this cleanup memory can be implemented in spiking neurons, and that its accuracy approaches that of their idealized version.

The fact that the number of neurons required for the cleanup memory scales linearly in the size of the vocabulary (M) also has interesting consequences. We note that the vocabulary size is generally much larger than the number of dimensions (D), as we previously saw via Eq. (1) that 700 dimensions is adequate for a vocabulary of 100,000 items. However, given the distributed representational scheme discussed in Section 1.2, the number of neurons required to represent a vector is linear in the number of dimensions of that vector. This means that, if we use VSAs to represent symbol structures, we can use a small number of neurons (on the order of tens of thousands) to represent, store, and manipulate a complex symbol structure. However, if the brain needs to extract a particular symbol from that symbol structure, many more neurons are needed (on the order of millions).

We believe this means that cleanup should be a relatively rare process in a neural cognitive architecture based on VSAs. For example, this could mean that within a cognitive module, complex representations can be manipulated and stored, and perhaps cleanup only occurs between cognitive modules, when particular discrete symbols need to be sent (for motor behaviour or some other low-level task). These sorts of considerations are vital for the development of a neural cognitive architecture based on VSAs, and we believe VSAs hold the most promise for a neural account of high-level symbolic reasoning (Stewart & Eliasmith, in press).

5. Conclusions

First, we have demonstrated a novel method for implementing an auto-associative memory using spiking neurons. This method is fast, in that it does not require a settling time or make use of feedback loops, and well-suited for situations where the stored values are a set of high-dimensional vectors and the input consists of one of these stored values plus a set of randomly distributed noise vectors. The number of neurons required for performing this autoassociation increases linearly in the number of items, while the number of neurons required for representing the input and output are linear in the number of dimensions in the vector.

Second, we have demonstrated that this auto-associative network is suitable as a cleanup memory when implementing a Vector Symbolic Architecture using spiking neurons. The model presented here is the first such cleanup memory that can be efficiently implemented by realistic spiking neurons for large vocabulary sizes. The number of neurons required to build this memory increases linearly in the number of distinct symbols that can be recognized. The accuracy approaches that of an ideal mathematical cleanup, and can perform cleanup in 5–10 ms using realistic noisy spiking neurons.

References

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Erlbaum.

Eliasmith, C. (2005). Cognition with neurons: A large-scale, biologically realistic model of the Wason task. In *Proceedings of the 27th annual meeting of the cognitive science society*.

Eliasmith, C., & Anderson, C. (2003). *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. Cambridge: MIT Press.

Gayler, R. (2003). Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. In *ICCS/ASCS international conference on cognitive science*.

Gayler, R. W., & Wales, R. (2000). Multiplicative binding, representation operators and analogical inference. In *5th Australasian cognitive science conference*.

Georgopoulos, A. P., Schwartz, A. B., & Kettner, R. E. (1986). Neuronal population coding of movement direction. *Science, 233*(4771), 1416–1419.

Hinton, G., & Andersen, J. (1989). *Parallel models of associative memory*. Lawrence Erlbaum Associates, Inc..

Hummel, J. E., & Holyoak, K. J. (2003). A symbolic-connectionist theory of relational inference and generalization. *Psychological Review, 110*(2), 220–264.

Jack, J. J. B., & Redman, S. J. (1971). The propagation of transient potentials in some linear cable structures. *Journal of Physiology, 215*, 283–320.

Kanerva, P. (1997). Fully distributed representation. In *Proceedings of 1997 real world computing symposium*.

Plate, T. (2003). *Holographic reduced representations*. Stanford, CA: CSLI Publication.

Pollack, J. B. (1988). Recursive auto-associative memory: devising compositional distributed representations. In *Proceedings of the 10th annual conference of the cognitive science society*.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533–536.

Salinas, E., & Abbott, L. F. (1994). Vector reconstruction from firing rates. *Journal of Computational Neuroscience, 1*, 89–107.

Stewart, T. C., & Eliasmith, C. (2008) Building production systems with realistic spiking neurons. In *Proceedings of the 30th annual meeting of the cognitive science society.*

Stewart, T. C., & Eliasmith, C. (in press). Compositionality and biologically plausible models. In W. Hinzen, E. Machery, & M. Werning (Eds.), *Oxford handbook of Compositionality*. Oxford University Press.

Stewart, T. C., Choo, X., Eliasmith, C. (2010a). Symbolic reasoning in spiking neurons: A model of the cortex/basal ganglia/thalamus loop. In *Proceedings of the 32nd annual meeting of the cognitive science society*.

Stewart, T. C., Choo, X., Eliasmith, C. (2010b). Dynamic behaviour of a spiking model of action selection in the basal ganglia. In *Proceedings of the 10th international conference on cognitive modelling*.

Sun, R. (2006). The CLARION cognitive architecture: Extending cognitive modeling to social simulation. In Ron Sun (Ed.), *Cognition and multi-agent interaction*. New York: Cambridge University Press.

van der Velde, F., & de Kamps, M. (2006). Neural blackboard architectures of combinatorial structures in cognition. *Behavioral and Brain Sciences, 29*, 37–70.