Probabilistic Symbol Encoding for Convolutional Associative Memories

Igor Peric, Alexandru Lesi, Daniel Spies, Stefan Ulbrich,
Arne Rönnau, Marius Zöllner and Rüdiger Dillman
FZI Forschungszentrum Informatik, Karlsruhe, Germany
{*peric, lesi, spies, ulbrich, roennau, zoellner, dillmann*}*@fzi.de*

Keywords: Vector Symbolic Architectures, Associative Memories, Symbol Encoding, Symbolic Scripting

Abstract: Vector Symbolic Architectures (VSAs) define a set of operations for association, storage, manipulation and retrieval of symbolic concepts, represented as fixed-length vectors in $\mathbb{R}^n$. A specific instance of VSAs, Holographic Reduced Representations (HRRs), have proven to exhibit properties similar to human short-term memory and as such are interesting for computational modelling. In this paper we extend the HRR approach by introducing implicit, topology-preserving encoding and decoding procedures. We propose to replace unique symbolic representations with symbols based on probability density functions. These symbols must be randomly permuted to ensure the uniform distribution of signals across Fourier space where embedding takes place. These novel encoding schemes eliminate the need for so-called *clean-up modules* after memory retrieval (e.g., self-organizing maps). Effectively each encoding implicitly represents its scalar symbol, so no further lookup is needed. We further show that our encoding scheme has a positive impact on memory capacity in comparison to the original capacity benchmark for HRRs (Plate, 1995). We also evaluate our memories in two different robotics tasks: visual scene memory and state machine scripting (holographic controllers).

1 INTRODUCTION

Vector Symbolic Architectures (VSA) define a set of operations for storage (compression) and retrieval of symbolic concepts in fixed size vectors. As described in (Levy and Gayler, 2008), every VSA has to implement three basic operations:

1. association (binding) $\mapsto \otimes$
2. superposition (appending) $\mapsto +$
3. probing (unbinding) $\mapsto \oslash$

VSAs can be viewed as a lossy compression technique, since binary operations on N-element vectors return N-element vectors as well. These operations will be described in further detail in the following section.

Depending on the way these three operations are implemented we can distinguish several popular VSAs, such as Holographic Reduced Representations (Plate, 1995), Circular HRRs (De Vine and Bruza, 2010), Binary Spatter Codes (Kanerva, 1994), Holographic Graph Neurons (Kleyko et al., 2016), Geometric Algebras (Patyk-Lónska et al., 2011) and Random Permutations (Recchia et al., 2015). The most

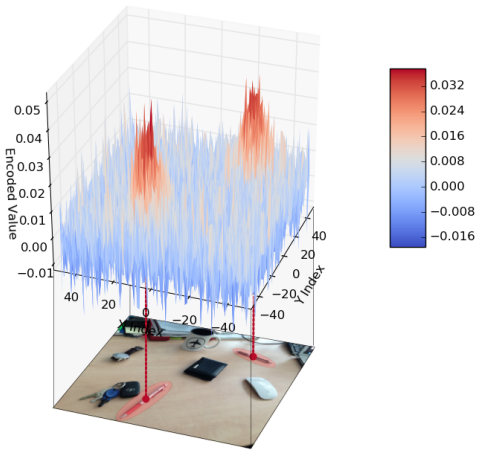


Figure 1: In-memory probability for a symbol ”pen”

widely used ones are HRRs and they are briefly explained in section 2.1. Authors in (Golosio et al., 2015) have used them to solve word embedding problems for extremely large datasets by replacing computationally expensive big matrix factorizations with an iterative estimation algorithm. Some of the other

work is centered around pattern recognition (Kleyko, 2016) and functional large-scale cognitive architecture models (Eliasmith et al., 2012). Preliminary theoretical assumptions have been made about the ability to use them as agent controllers (Levy et al., 2013), where a symbolic program is represented with a set of combined symbolic actions associated with actuator symbols and symbolic sensing.

Authors in (Eliasmith et al., 2012) have demonstrated that the functional cognitive model Spaun is capable of performing 7 different cognitive tasks when implemented as HRRs using biologically plausible models of neuron dynamics. They further show that in serial working memory tasks the model of forgetting exhibits similar behaviour as human working memory due to limited capacity. Lastly they report similarity between the human working memory capacity and the average number of items (symbols) that their model is capable of holding in memory.

This successful model of forgetting is one of the reasons our work is centred around HRRs. Another reason is the fact that HRRs are the only VSAs up to date operating on convolution operations in their core. This is appealing if we take the numerous recent advancement in deep learning and convolutional neural networks into consideration. These networks succeed due to training on big data and using convolution as the main architectural constraint. Although the description of the biophysical mechanism underlying convolution and the concept of weight sharing are still missing, it is important to reuse proven concepts, such as convolutional filtering, in a way as general as possible. In other words, addressing multiple fundamentally different problems with similar computational principles can be considered a step towards a unified brain processing theory.

2 VECTOR SYMBOLIC ARCHITECTURES

In order to use VSA operations all types of required inputs need to first be represented as symbols, which are vectors of fixed size. The basic approach is to pick these vectors randomly once for each input used in any operation and maintain these pairs in a separate data structure.

Association (also called ”binding”) is the operation of linking two operands together, resulting in a vector of the same size which is dissimilar (or orthogonal) to the original vectors, but also contains scrambled information from both of them. For instance, linking the property ”red” to object ”triangle” can be done using the binding operation as follows:

$$memory_1 = red \otimes triangle \tag{1}$$

Superposition is the operation of extending a symbol, regardless of what it already contains. The memory ”red triangle and blue circle” is formed as:

$$memory_2 = memory_1 + blue \otimes circle \tag{2}$$

Probing is the operation of extracting a noisy version of an associated item from a set memory. Asking ”What color is the triangle?” works as follows:

$$red \approx memory_2 \oslash triangle \tag{3}$$

The probing operation assumes the existence of inverse elements, so the previous equation can be interpreted and explained as such:

$$\begin{aligned} &memory_2 \otimes triangle' = \\ &(red \otimes triangle + blue \otimes circle) \otimes triangle' = \\ &red \otimes triangle \otimes triangle' + blue \otimes circle \otimes triangle' = \\ &red + blue \otimes circle \otimes triangle' = red + noise \approx red \end{aligned} \tag{4}$$

Here $triangle'$ represents the inverse of $triangle$. The resulting vector will be a composition of something that is very similar to *red* and something that doesn't resemble anything meaningful from the symbol table, so it can be treated as negligible noise. One similarity measure is the cosine distance, which for two vectors $\mathbf{x}$ and $\mathbf{y}$ is calculated as follows:

$$cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{y}||} \tag{5}$$

Obtained similarity values clearly distinguish the matching vectors. Concrete values that highlight this will be later presented in section 5.

The process of converting the noisy result of probing into the closest known symbol is called the **clean-up procedure**. A straightforward way of implementing clean-up memory is using a lookup table, while other researchers show that any type of attractor network can be used (e.g. self-organizing maps).

In the case of lookup tables, in order to decode a specific symbolic vector, the elements from a symbol table are iterated over, the similarity to each of the entries is calculated and the symbol which presented maximum similarity is returned. With the help of a threshold for the similarity value false-positives can be excluded. A lookup requires $O(n)$, where n is the number of symbols known to the system. This approach works quite well for object identifiers (names), but has some non-desirable properties when used on scalar encoding. The space complexity will increase

and may become infeasible, depending on the number of inputs needed. One way of handling symbolic scalar encoding is to discretize the whole input range into bins and assigning a random vector to each discrete value. This, however may often prove to not satisfy the task needs any more. In this paper, we present an intrinsic, probabilistic encoding and decoding method, which removes the necessity of using a symbol table.

2.1 Holographic Reduced Representations

Holographic Reduced Representations are VSAs which implement the operation of binding as circular convolution. One option for calculating it is with the help of discrete Fast Fourier Transformations (FFTs). Taking $f(x)$ to describe an FFT, with its inverse $f^{-1}(x)$, circular convolution is calculated as:

$$\boldsymbol{a} = f^{-1}(f(\boldsymbol{x}) * f(\boldsymbol{y})) \tag{6}$$

Here $\boldsymbol{a}$ is the result of convolution, and $\boldsymbol{x}$ and $\boldsymbol{y}$ are symbolic vectors of size n. Superposition is implemented as a simple element-wise vector addition. Taking the previous result and superposing it with some vector $\boldsymbol{b}$ is done as follows:

$$\boldsymbol{m} = \boldsymbol{a} + \boldsymbol{b} \tag{7}$$

The resulting vector of the superposition operation $\boldsymbol{m}$ can then be probed by using the inverse of circular convolution, which is circular correlation. Probing can be viewed as binding with the "inverse":

$$\boldsymbol{y}^* = f^{-1}(f(\boldsymbol{m}) * f(\boldsymbol{x})^{-1}) \tag{8}$$

Following this series of operations the vector $\boldsymbol{y}^*$ will be similar to vector $\boldsymbol{y}$, but not entirely the same due to the noisy influence of $\boldsymbol{b}$. The inverse in Fourier space referred to here is gained by calculating the complex conjugate of the vector's entries, which is done by negating the imaginary parts.

Performing FFTs has no invariant regarding the mean of the resulting vector entries and can produce all sorts of results. Due to this, when probing, we would receive vectors which had the correct relative distance between their values, i.e. the correct "shape", but would be shifted, stretched or squeezed along the Y-axis. This lead to vectors falsely being regarded as dissimilar to each other, and can be solved by performing two normalization steps before comparing:

$$x_i' = x_i - \frac{\sum_j x_j}{|\boldsymbol{x}|} \qquad \boldsymbol{x}'' = \frac{\boldsymbol{x}'}{\|\boldsymbol{x}'\|} \tag{9}$$

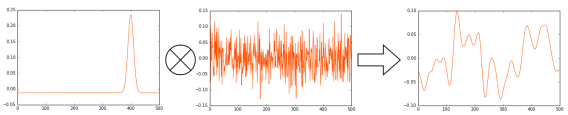
3 TOPOLOGY-PRESERVING ENCODING

3.1 Encoding Scalar Symbols

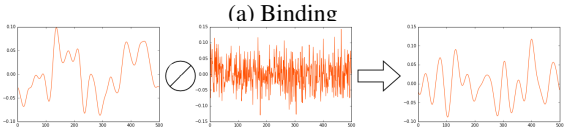
Encoding scalars is a good example for the necessity of preserving topology. Depending on what dataset is in use it may be required to store mappings for any number of values, which can drastically increase space requirements. Furthermore, using a separate random encoding for each scalar, regardless of whether they are very close to each other or not, there would be no similarity in neighbouring values. For example, the following probe will yield no result:

$$(10000 \otimes a) \oslash 10001 \tag{10}$$

To facilitate these two qualities scalar encoding can be achieved using probabilistic instead of exact symbolic representation. The encoded vector can simply be a Gaussian shaped probability distribution sampled over a fixed input range, with a mean equal to the encoded value and arbitrarily chosen standard deviation. Figure 2a shows how such an encoded vector looks like, and what the result of binding it to a generic random vector is. With this result it becomes obvious that unfortunately lots of information is lost. If this result is probed with the symbol for "ball" something similar to the initial Gaussian distribution for the value 8 on a scale to 10 is expected. And yet figure 2b presents no such outcome. While there is a peak at the initial value lots of other peaks also arise, making it impossible to detect which one is correct.



(a) Binding



(b) Probing

Figure 2: Binding 8 with "ball", then probing for "ball"

This effect is due to the fact that circular convolution is based on Fourier transformations. A lack of frequencies in the Fourier space of the encoded scalar becomes apparent, as highlighted in figure 3. All the information of a Gaussian distribution is crammed into the lowest and highest values of the Fourier transformation, resulting in a high loss of information. In these circumstances, when probing for a scalar, the

vector received is in turn recreated of too few frequencies and will not come close to resembling the original Gaussian distribution.

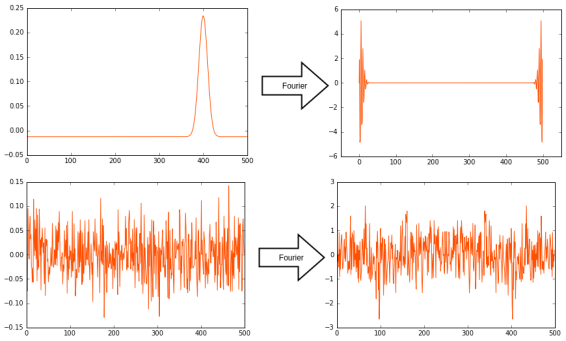


Figure 3: Fourier analysis of scalars and random symbols

This inconvenience is removed with the help of permutations. Depending on the size of the memory vectors used a fixed permutation is generated once. Whenever a scalar is encoded the Gaussian distribution is first sampled over the given range, after which the sampled values swap places accordingly. As such the resulting vector will present lots of jumps in value, which have a significant impact on the Fourier space, as can be seen in figure 4. Decoding scalar values will require a reverse permutations of the result of probing.

Working with such a vector makes it possible to successfully probe for scalars and receive a Gaussian distribution overlapped with some noise. All of this can be seen in figure 5, which depicts the exact same operations as figure 2, but using a permuted version of a Gaussian distribution. The result in figure 5b clearly shows a spike at the expected spot in the sample range. This vector can now be smoothed out, after which the peak can easily be identified and the initial scalar can be recovered.

As such using the scalar encoder simplifies the operations pipeline of working with symbols. Encoding the same scalar over and over will always yield the same result, removing the need to store scalar symbols in a lookup table. Also, when probing for a scalar, the symbol doesn't need to be matched against all other known symbols to determine the highest similarity, which can be computationally expensive. Instead the result of probing is smoothed out and verified for the presence of a Gaussian distribution, which then determines the scalar it represents.

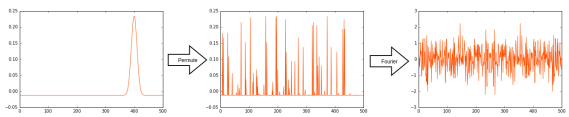
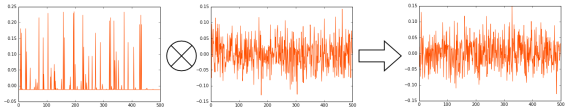
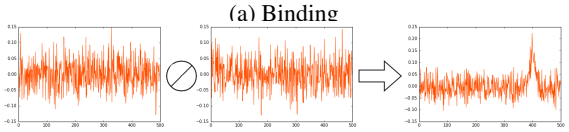


Figure 4: Fourier space of a permuted scalar encoding



(a) Binding



(b) Probing

Figure 5: Operations of figure 2 with permutation

3.2 Encoding Coordinate Symbols

Encoding coordinate symbols works in a manner very similar to encoding scalar symbols. Depending on the number of coordinates the symbolic vectors shape is reinterpreted. 2D coordinates, for instance, require a matrix. This matrix represents the entire coordinate space and on it 2D Gaussian distributions are sampled. It can be viewed as a heat map, as displayed in figure 7b. The matrix will still be treated as a vector when it comes to running operations on it and must also be permuted. Since the value space now has more dimensions the size of the vector has to be raised accordingly to achieve performance similar to that of scalar symbol encoding.

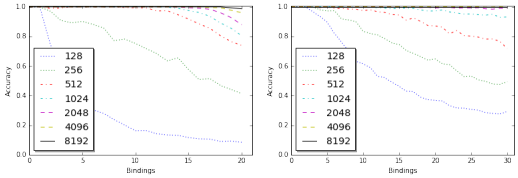
An attempt has been made to overcome the curse of dimensionality by splitting the vector up in the number of dimensions and simply encoding the value for each dimension in the resulting sub-vectors. Unfortunately it then becomes impossible to differentiate among multiple coordinates encoded in the same vector. Specifically, when decoding, there is no way to know which of the values in separate sub-vectors belong to each other and which don't. On the other hand if only a single coordinate needs to be bound to another symbol this approach is still feasible and will permit the use of smaller vectors for high accuracy. Concretely, for d dimensions, splitting up the vector will require the size to be raised by a factor of d to maintain the same accuracy. Reinterpreting the shape will require the size to increase exponentially by the power of d to keep the accuracy.

4 PROPERTIES OF HOLOGRAPHIC MEMORIES

Vector Symbolic Architectures in general have the following properties:

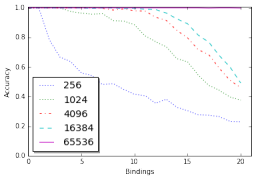
- The number of items stored in a single memory vector is not known based solely on it's content.

- Existence of an item in memory cannot be checked explicitly. The closest equivalent is to probe the memory for the item of interest and to compare the similarity of the decoded result to anything meaningful. Even then it could be that the result of probing for anything might still require to be probed by a further symbol to retrieve the sought out item.
- Chances of successful recall depend on the actual stored data.
- Memories can be changed in an online fashion. There is no need to recreate memories from scratch - symbols can be appended to or removed from already existing memory vectors if present.



(a) Symbols accuracy

(b) Scalars accuracy



(c) Coordinates accuracy

Figure 6: Single vector memory capacity analysis

4.1 Capacity

Just as Plate has done in the original HRR paper (Plate, 1995) we have tested the memory capacity in terms of accuracy of successful recall for variable numbers of bound pairs. Results are shown in figure 6a.

As we can see, a 512 element vector starts decreasing accuracy of recall after 10 pairs stored in a single memory block. A 4096 and 8192 element vectors can almost perfectly recall 20 associated pairs, which matches the results of Plate using a lookup table and a clean-up process.

The experiment was repeated for superposing bounded pairs of a random symbol and a scalar, as well as pairs of a random symbol and a 2D coordinate. Figure 6b shows the result for scalars. In this instance the probe was always done for the scalar value. The total accuracy of the result depends on new factors, the main of which will be the quality of the smoothing function used to clean up the noise from the resulting signal. This will determine how much the peaks position may shift in some direction. In our experiments a Hanning window smoothing technique was used and a 2% deviance from the exact initial scalar was allowed. With these settings probing performed significantly better than solely among random symbols. Vectors of size 8192 even managed to maintain 100% accuracy when storing 30 bindings. Intuitively this can be explained as follows: Gaussian distributions are far more noise tolerant than random vectors. Vectors need to maintain a high similarity to be declared identical, whereas Gaussian distributions only need to present peaks after smoothing.

Finally figure 6c shows the accuracy of stacking bindings of 2D coordinates with random symbols. The same smoothing and deviance settings were used as for the previous experiment. As coordinates are encoded in a vector that is virtually cast to 2D its length would have to be squared to achieve similar performance to scalars. Due to this higher steps in between vector lengths have been used, and most lengths still perform rather poorly. However, increasing the length considerably, like in the case with 65536 elements, near-perfect accuracy can still be upheld up to 20 bindings. In all 3 experiments both fakse positives and false negatives are regarded.

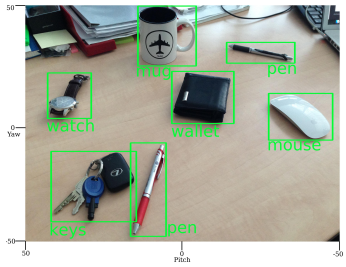
5 EXPERIMENTS

5.1 Visual Short Term Memory and Top-Down Attention Model

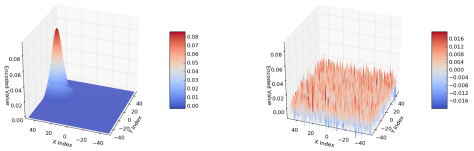
In this experiment a system is given a monocular stream as an input. An arbitrary algorithm is used to perform object detection and recognition, resulting in a number of 2D coordinates associated with respective object names (figure 7a). Object names are strings and are encoded as random vectors of length n (figure 7c). The locations of objects in the image plane are 2D coordinates (x, y) and they are perceptually grounded symbols (figure 7b, non-permuted).

Due to the noisy nature of probing given by the slight loss of information in the binding process each symbol may have some amount of similarity to whatever other symbol it is compared to. However, even though this is the case, the similarity measure of bound symbols is consistently higher and can be clearly distinguished from others. Chart 8a represents the result of probing at $(19, -26.75)$, which is somewhere in between the keys and the red pen, as seen in figure 7a.

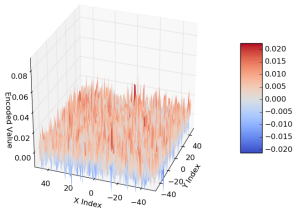
As we can see all symbols that are nowhere near the position present similarities well below 0.05 (val-



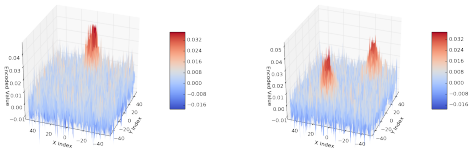
(a) Original scene and a set of detected objects



(b) Location of "watch" (c) Symbol of "watch"



(d) Entire visual scene memory (7 pairs)



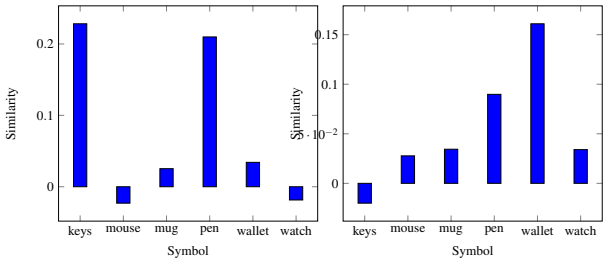
(e) Probing for "mug" (f) Probing for "pen"

Figure 7: Representation of visual scene in memory.

ues may also be negative), while both close matches scored over 0.2. Spot on probes may even attain values of over 0.5. The example in chart 8b is for a probe at $(-13, 21.5)$, which is further away from the center of an object region. But even though the dark pen and the wallet are further apart the result of probing still yields distinctly higher similarities for them as opposed to objects further away.

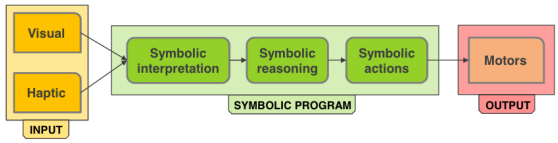
5.2 Symbolic Scripting (Holographic Controllers)

We have used our system to create a mechanism for encoding neural programs and processes through a series of associated action, perception and actuator symbols. We call this mechanism for defining controllers using holographic memories "symbolic scripting".

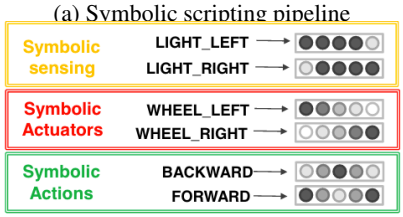


(a) Probe at (19, -26.75). (b) Probe at (-13, 21.5).

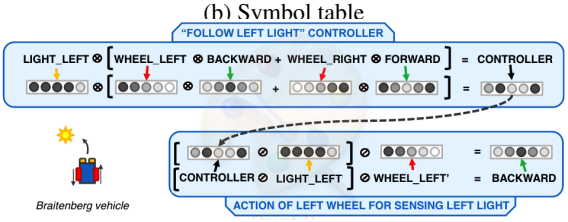
Figure 8: Similarity test after probing for specific locations



(a) Symbolic scripting pipeline



(b) Symbol table



(c) Holographic program (script)

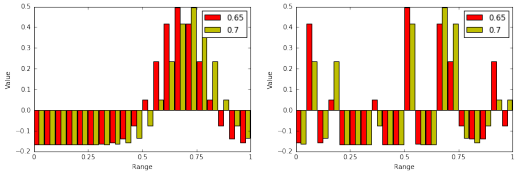
Figure 9: Symbolic scripting illustration: object following

The basic idea is centred around symbolic representations of all three parts involved:

1. **actions** are symbolic constants representing groups of control signals, such as "forward", "backward", "stop", "flex", etc.
2. **actuators** are named controllable parts of the system on which actions can be performed on, such as "right_wheel", "left_wheel", etc.
3. **senses** are preprocessed (classified) sensory stream labels, such as "bright", "dark", "touching", "falling_down", "rolling", etc.

An example of this kind of symbolic script is given in figure 9. Here the memory encodes an object following behaviour for its left wheel.

In our experiment we have extended the detection of light to a position along the visual field, reduced to the X-axis, rather than just binary statements like "light is visible to the left". To visualize the difference chart 10a shows the position of light for two



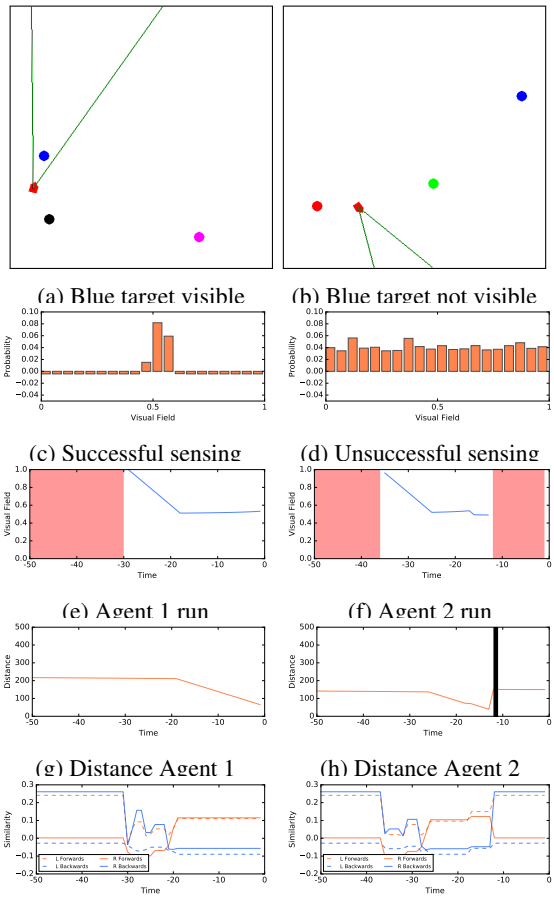
(a) Raw input distribution (b) Permuted distribution
Figure 10: Sensory input at two successive points in time

consecutive time steps. The value 0 represents the far left corner of the sensor, and 1 the far right one. The chart can be viewed as a (normalized) probability of light being at a certain position, and both cases show it being somewhere right of the center. Chart 10b shows what the Gaussian distributions look like when they are permuted. This shape of theirs is then used as a symbol. Concretely, instead of encoding a random symbol for "light_left" we encode a scalar representing the position of light along the visual field and bind it with the rest of the controller components. Accordingly, when running the experiment, once the light reaches a position encoded in the controller it will issue new commands to the vehicle.

The symbolic scripting experiment is set up using a single-sensor Braitenberg vehicle (see figures 11a and 11b). The sensor returns a list of objects in its field of view with their corresponding view space positions in $[0, 1]$ (far left to far right). All returned objects are then compared to the agent's target object. If the target is found, the holographic controller is probed with its view space position which, due to the similarity of scalars, yields results even for values that lie between the originally stored values and can consequently be probed with the symbols $left\_wheel$ and $right\_wheel$ to extract the symbolic actions $forwards$ or $backwards$. If the target is not found or no action could be determined, the agent defaults to probing the controller with the symbolic constant no_object which results in a clockwise rotation when probing for wheel actions (see figure 11j).

In the left column of figure 11, the agent approaches the blue target (figure 11a). Following figure fig:experiment:pos1 in the beginning (tick -50), the agent rotates right (default action) because the target is not visible. At tick -30, the object enters the agent's visual field and continuously nears the fields center, which is reached at tick -19. Then, with the sensory value of $\sim$0.5, the actions for both wheels switch to $forwards$ (figure 11i) and the agent approaches the target, constantly decreasing its distance (figure 11g).

In the right column of figure 11, the target is not yet in the visual field of the agent. This is the result of the agent previously reaching an old target



(i) Symbol similarity Agent 1 (j) Symbol similarity Agent 2
Figure 11: Controlling a vehicle with one linear sensor.

(at approximately tick -13) which lead to the removal of all current objects and random placement of new ones, including a new target. As seen in figure fig:experiment:pos2 since the new target is placed outside of the agent's field of view, the default action of probing no_object is performed. The clockwise rotation can also be seen in figure 11j). Regardless of how many times the agent reaches its target it will continue to find and pursue new ones solely with the help of its symbolic controller.

This experiment shows the effectiveness of our approach in a practical application by encoding symbolic sensing, actuators and actions in a controller.

6 CONCLUSIONS

We have introduced a mechanism for intrinsic symbol encoding/decoding for Holographic Reduced Representations. As a consequence, our approach eliminates the need for clean-up memories as they

are described in literature. Instead, results of probing operations directly give interpretable symbols, which can be decoded in constant computational time. In other words, time needed for encoding/decoding is completely independent of the total number of symbols memorized.

A concrete biophysical evidence and theoretical model of implementation of convolution in the brain is, unfortunately, still missing. Nevertheless, our work assumes convolution as the basic computational substrate of higher level cognition and identifies random permutation as computational requirement without which association of probabilistically encoded symbols is not possible.

REFERENCES

De Vine, L. and Bruza, P. (2010). Semantic Oscillations: Encoding Context and Structure in Complex Valued Holographic Vectors. In *AAAI Fall Symp. Ser.*, number 48-55, pages 48–55, Arlington, Virginia. AAAI Press.

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., and Rasmussen, D. (2012). A Large-Scale Model of the Functioning Brain. *Science (80-.).*, 338(6111):1202–1205.

Golosio, B., Cangelosi, A., Gamotina, O., and Masala, G. L. (2015). A cognitive neural architecture able to learn and communicate through natural language. *PLoS One*, 10(11).

Kanerva, P. (1994). *The Spatter Code for Encoding Concepts at Many Levels*, pages 226–229. Springer London, London.

Kleyko, D. (2016). *Pattern Recognition with Vector Symbolic Architectures*.

Kleyko, D., Osipov, E., Senior, A., Khan, A. I., and Sekercioglu, Y. A. (2016). Holographic Graph Neuron: A Bioinspired Architecture for Pattern Processing. *IEEE Trans. Neural Networks Learn. Syst.*, pages 1–9.

Levy, S. D., Bajracharya, S., and Gayler, R. W. (2013). Learning behavior hierarchies via high-dimensional sensor projection. *AAAI Work. - Tech. Rep.*, WS-13-12:25–27.

Levy, S. D. and Gayler, R. W. (2008). Vector Symbolic Architectures: A New Building Material for Artificial General Intelligence. *Proc. First Conf. Artif. Gen. Intell.*, pages 414–418.

Patyk-Lónska, A., Czachor, M., and Aerts, D. (2011). Distributed representations based on geometric Algebra: The continuous model. *Inform.*, 35(4):407–417.

Plate, T. A. (1995). Holographic Reduced Representations. *IEEE Trans. Neural Networks*, 6(3):623–641.

Recchia, G., Sahlgren, M., Kanerva, P., and Jones, M. N. (2015). Encoding Sequential Information in Semantic Space Models: Comparing Holographic Reduced Representation and Random Permutation. *Comput. Intell. Neurosci.*, 2015:1–18.