

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 are, as such, interesting for computational modeling. In this paper we extend the HRR approach by introducing implicit, topology-preserving encoding and decoding procedures. We propose moving away from traditional exact symbolic representation to probabilistic one, replacing unique symbol identifiers with probability density function. We show this requires randomly chosen permutations for encoding and their corresponding inverse for decoding. These are necessary for the uniform distribution of signals across Fourier space where embedding takes place. These novel encoding schemes, together with the extended operation pipeline, are eliminating the need for so-called *clean-up modules* after memory retrieval (e.g., self-organizing maps). Effectively, encoding implicitly represents the scalar symbol, so no further lookup is needed. We show that our encoding scheme has a positive impact on memory capacity in comparison to the original capacity benchmark for HRRs (?). We also evaluate our memories in two different robotics tasks: visual scene memory and state machine scripting (holographic controllers).

WS

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 (?), every VSA has to implement three basic operations:

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

In addition, every VSA can be considered 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

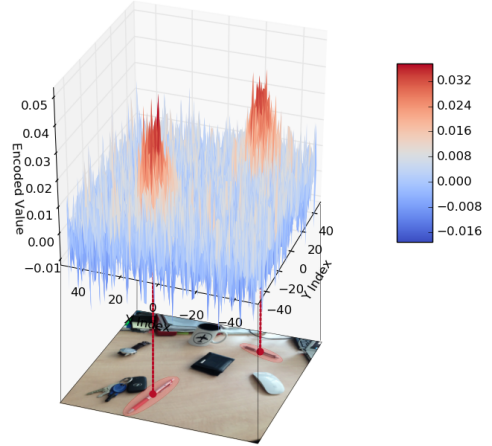


Figure 1: In-memory representation of visual scene short-term memory. The result shows probability distribution for a symbol "pen" being at certain position in scene.

(?), Circular HRRs (?), Binary Spatter Codes (?), Holographic Graph Neurons (?), Geometric Algebras

(?) and Random Permutations (?). The most widely used ones are HRRs and they are briefly explained in section 2.1. Authors in (?) have used them to solve word embedding problem for extremely large dataset by replacing computationally expensive big matrix factorization with an iterative estimation algorithm based on HRRs. Some of the other work is centered around pattern recognition (?) and functional large-scale cognitive architecture models (?). Preliminary theoretical assumptions have been made about the ability to use them as agent controllers (?), where a symbolic program is represented with a set of combined symbolic actions associated with actuator symbols and symbolic sensing.

Authors in (?) 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 concept of weight sharing are still missing, it is important to reuse proven concepts, such as convolutional filtering, in as general way 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:

$$memory_1 = red \otimes triangle \quad (1)$$

Superposition is the operation of extending a set of items, regardless of what is already contained in them. As an example, a memory of "red triangle and blue circle" can be formed in the following way:

$$memory_2 = memory_1 + blue \otimes circle \quad (2)$$

Probing is the operation of extracting a noisy version of an associated item from a joint set memory. I.e. in this scenario the question "What color is the triangle?" could be asked like this:

$$red \approx memory_2 \oslash triangle \quad (3)$$

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

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

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. A usual way of computing similarity is the cosine distance, which for two vectors \mathbf{x} and \mathbf{y} is calculated as follows:

$$cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|} \quad (5)$$

Obtained values of similarity 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, in short FFTs. Taking $f(x)$ to describe an FFT, with $f^{-1}(x)$ as its inverse, circular convolution can be calculated as follows:

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

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

$$\mathbf{m} = \mathbf{a} + \mathbf{b} \quad (7)$$

The resulting vector of the superposition operation \mathbf{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" of a vector in Fourier space:

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

Following the previous formulas as a series of operations the vector \mathbf{y}^* will be similar to vector \mathbf{y} , but not entirely the same due to the noisy influence of vector \mathbf{b} . The inverse in Fourier space referred to here is gained by calculating the complex conjugate of the vector's entries. Complex conjugates are simply the same values with negated 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:

$$x_i' = x_i - \frac{\sum_j x_j}{|\mathbf{x}|}, \quad (9)$$

and

$$\mathbf{x}'' = \frac{\mathbf{x}'}{\|\mathbf{x}'\|}, \quad (10)$$

which normalize \mathbf{x} to a mean value of 0 and a length of 1.

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:

$$(10000 \otimes a) \odot 10001 \quad (11)$$

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 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 same random vector used in binding something similar to the initial Gaussian distribution 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.

This effect is due to the fact that circular convolution is based on Fourier transformations. A lack of

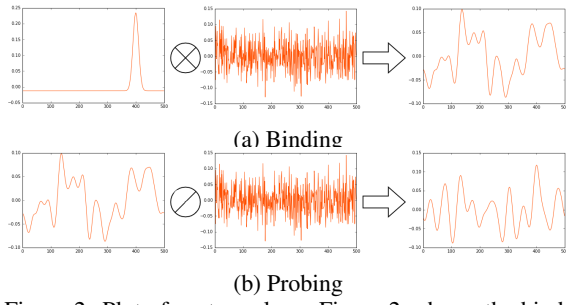


Figure 2: Plot of vector values. Figure 2a shows the binding of the value 8 from representational range [0,10] with the symbolic representation of "ball" (which is randomly chosen), giving a representation of association. In figure 2b this resulting representation is probed with "ball" and the expected result is a Gaussian distribution for 8, mixed with some noise. As it can be seen, result is far from expected and numeric value cannot be inferred.

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.

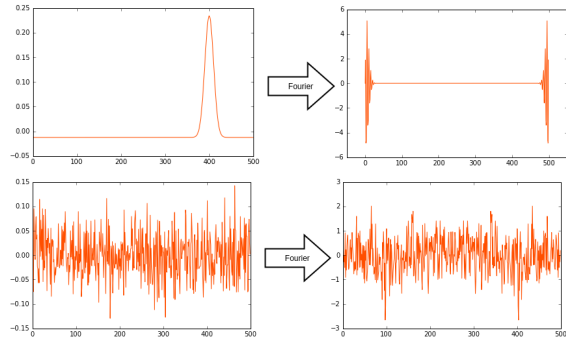


Figure 3: Fourier analysis of 1D symbols: a Gaussian distribution (top row) and a randomly sampled vector of the same size (bottom row). It is obvious that the latter's information in Fourier space is uniformly spread across the whole spectrum, while the former's is concentrated around 2 boundary points.

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.

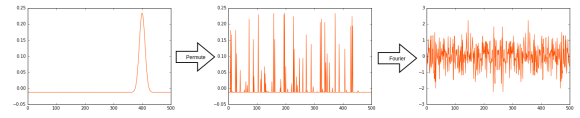


Figure 4: Applying random permutation to a sampled Gaussian distribution results in more discrete changes in signal, which correspond to high-frequency components in Fourier space. This makes frequency spectrum more uniformly spread, which is a desirable property for HRRs. Although this scrambling of initial signal makes it seem random, the fact that the same permutation is used in all encodings keeps two semantically close values having very similar representations, thus preserving their semantic proximity.

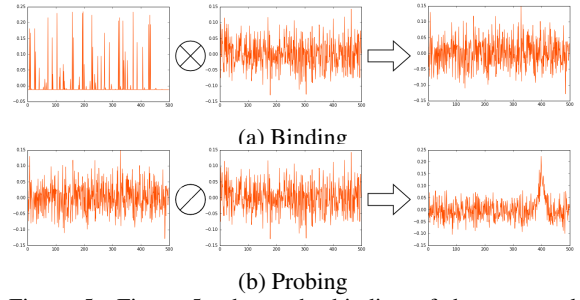


Figure 5: Figure 5a shows the binding of the permuted Gaussian distribution for the value 8 from range [1, 10] with the symbolic representation of "ball", which is random. In figure 5b "ball" is being probed, and the result after inverse permutation is a Gaussian distribution for the value 8, mixed with some noise.

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.

Algorithms 1 and 2 show the operations pipeline before and after the suggested modification are introduced in pseudo code.

```

$ \ForAll{$symbol$} $
  \State $t_{symbol}$ \gets chooseRandomVector() $
\EndFor
\State $m$ \gets initEmptyMemory() $
\ForAll{$(symbol_1, symbol_2)$} $
  \State $enc_1$ \gets FFT($t_{symbol_1}$) $
  \State $enc_2$ \gets FFT($t_{symbol_2}$) $
  \State $m$ \gets $m$ + IFFT($enc_1$ * $enc_2$) $
\EndFor
\Function{ProbeMemory}{memory, op}
  \State $result$ \gets IFFT(FFT(memory) * FFT(op)^{-1}) $
\EndFunction
\ForAll{$symbol$} $
  \State $s$ \gets cosineSimilarity($t_{symbol}$, $result$) $

```

```

\If {$s \geq cleanResultSimilarity$}
\State $cleanResultSimilarity \gets s$
\State $cleanResult \gets t_{\{symbol\}}$
\EndIf
\EndFor \
\Return $cleanResult$
\EndFunction

```

Lines 1-3 assign a random vector representation to each symbol known to the system prior to any other operations. This lookup table is stored in an array t . Lines 4-9 initialize the memory m with an arbitrary amount of pairs of associated scalars. Function *ProbeMemory*(*memory*, *op*) returns a symbol associated with provided *op*. Line 11 performs circular correlation and stores the noisy result of unbinding in s . Lines 12-18 loop through the whole lookup table and find the maximum similarity vector to s .

Algorithm 1 Modified pipeline

```

1:  $P \leftarrow \text{chooseRandomPermutation}(n)$ 
2:  $m \leftarrow \text{initEmptyMemory}()$ 
3: for all ( $symbol_1, symbol_2$ ) do
4:    $enc_1 \leftarrow \text{FFT}(\text{permute}(enc(symbol_1, n), P))$ 
5:    $enc_2 \leftarrow \text{FFT}(\text{permute}(enc(symbol_2, n), P))$ 
6:    $m \leftarrow m + \text{IFFT}(enc_1 * enc_2)$ 
7: end for
8: function PROBE MEMORY(memory, op)
9:    $r \leftarrow \text{IFFT}(\text{FFT}(\text{memory}) * \text{FFT}(op)^{-1})$ 
10:   $r \leftarrow \text{permute}(r, P^{-1})$ 
11:   $cleanResult \leftarrow \text{argmax}(r)$ 
    return  $cleanResult$ 
12: end function

```

In contrast, Algorithm 2 shows a pipeline with our proposed changes. The first line chooses a fixed random permutation of n elements used throughout the whole process. There is no initial loop to assign symbolic representations to all symbols. They are, instead, constructed according to need using function *enc*(*value*, n). This function returns a discrete sampled Gaussian distribution vector of n elements, centred around a chosen *value*. The vector is then permuted using a permutation P in lines 4 and 5, so its frequency representation contains uniform spread over the whole spectrum, as explained earlier in this chapter. The probing operation consists of applying an inverse permutation P^{-1} after IFFT (line 10). Since this returns a noisy version of the initial Gaussian distribution (shown in figure 5b), the only thing that needs to be done is to identify the peak of the distribution. The complexity of this operation is linear w.r.t. length of the vector n , while in the case of Algorithm 1 it is linear w.r.t. to number of symbols stored in the lookup table.

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.

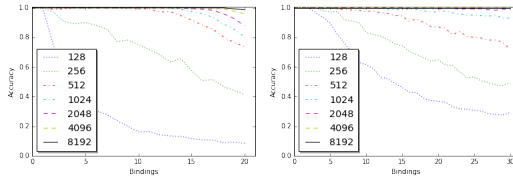
An attempt has been made to overcome the curse of dimensionality by splitting the vector up in the number of dimensions and simply encoding the value for each dimension in the resulting sub-vectors. Unfortunately it then becomes impossible to differentiate among multiple coordinates encoded in the same 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

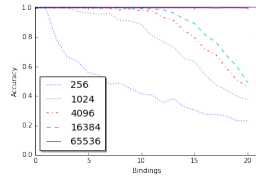
Vector Symbolic Architectures in general have the following properties:

- 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) Binding pairs of random vectors. Accuracy scalar and a random symbol decreases with more bindings.



(b) Binding pairs of one 2D coordinate and a random symbol, then probing for the scalar.

Figure 6: Memory capacity analysis for storing a variable number of pair associations in a single memory vector. Different lines show various vector lengths, the horizontal axis shows the number of pairs and vertical axis accuracy of successful recall. Both false positives and false negatives are regarded.

4.1 Capacity

Just as Plate has done in the original HRR paper (?) 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 binding a random symbol to the encodings of both scalars and to 2D coordinates, to compare performance. 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.

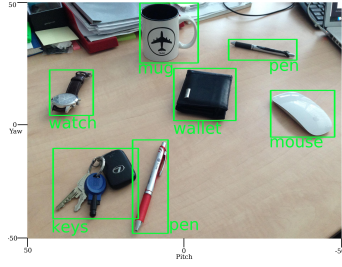
5 EXPERIMENTS

5.1 Visual Short Term Memory and Top-Down Attention Model

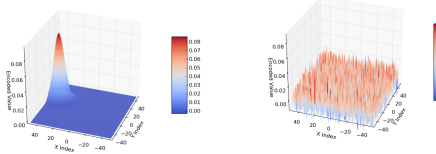
In this experiment a system is given a monocular stream as an input. An arbitrary algorithm is used to perform object detection and recognition, resulting in a number of 2D coordinates associated with respective object names (figure 7a). Object names are strings and are encoded as random vectors of length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

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.

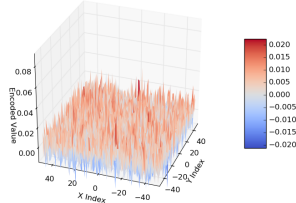
As we can see all symbols that are nowhere near the position present similarities well below 0.05 (values may also be negative), while both close matches scored over 0.2. Spot on probes may even attain values of over 0.5. The example in chart 8b is for a probe



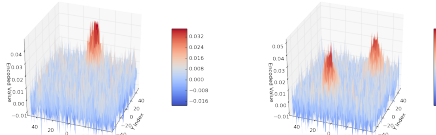
(a) Original scene and a set of detected objects



(b) Encoding of location (x,y) for "watch" (c) Encoding of the "watch" object ID as a randomly sampled vector



(d) Representation of the complete visual scene, containing information about all 7 objects and their locations scrambled together



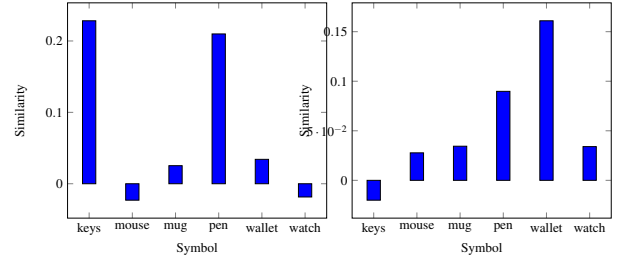
(e) Probing for "mug" returns location probability turn mixture of Gaussian distributions (f) Probing for "pen" returns location probability turn mixture of Gaussian distributions

Figure 7: Representation of visual scene in short term symbolic memory.

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)

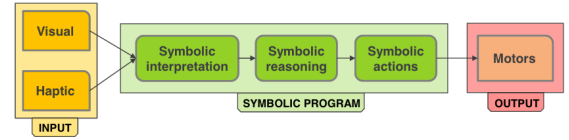
We have used our system to create a mechanism for encoding neural programs and processes through a se-



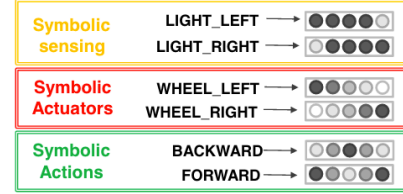
(a) Probe at $(19, -26.75)$. (b) Probe at $(-13, 21.5)$.

Figure 8: Similarity test for each symbol from the lookup table after probing the entire scene memory for a specific location.

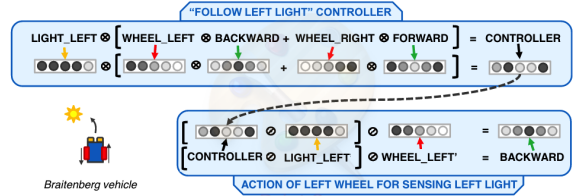
ries of associated action, perception and actuator symbols. We call this mechanism for defining controllers using holographic memories "symbolic scripting".



(a) Symbolic scripting pipeline



(b) Symbol table



(c) Holographic program (script)

Figure 9: Symbolic scripting illustration for object following behavior of a two-wheeled robot

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.

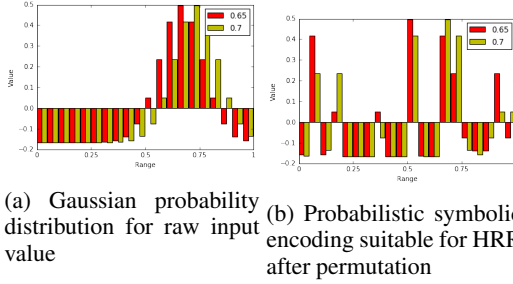


Figure 10: Sensory input encoding of a target objects position in the visual field at two successive points in time. Left graph shows original encodings for two adjacent values and right graph shows permuted encodings. It can be seen that semantically close input values result in similar HRR encodings.

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 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). 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 ~ 0.5 , the actions for both wheels switch to *forwards* (figure 11i) and the agent approaches the target, constantly decreasing its distance (figure 11g).

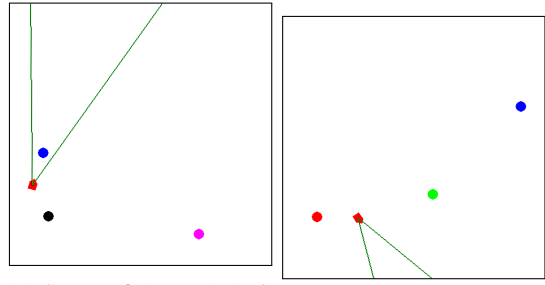
In the right column of figure 11, the target is not yet in the visual field of the agent. This is the result of the agent previously reaching an old target (at approximately tick -13) which lead to the removal of all current objects and random placement of new ones, including a new target. As the new 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

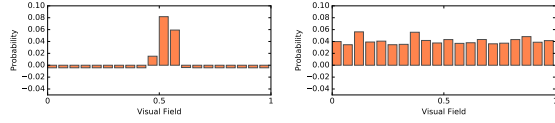
We have introduced a mechanism for intrinsic symbol encoding/decoding for Holographic Reduced Representations. As a consequence, our approach eliminates the need for clean-up memories as they are described in literature. Instead, results of probing operations directly give interpretable symbols, which 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.



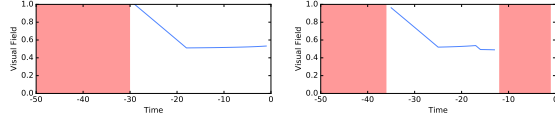
(a) Scene of Agent 1 (red) with three randomly placed items, the blue one being the target object. The green fan represents the visual field.

(b) Scene of Agent 2 with three randomly placed items. The agent does not have the target object in his field of view yet.



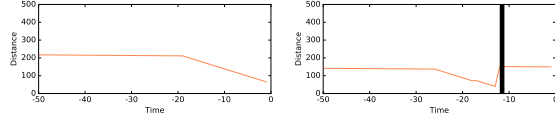
(c) Symbolic sensing with the target in the visual field, which results in a scalar field showing the represent value representing the target direction (~ 0.55 , almost straight forward).

(d) Symbolic sensing without the target in the visual field, out of the target in the visual field, results in a scalar field showing the represent value representing the target direction (~ 0.55 , almost straight forward).



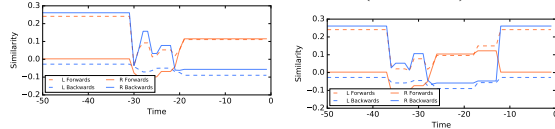
(e) The targets position over time in the visual field of the target object (red background). At tick -30 , the target enters the far right edge of the field of view and then prompts a new target creation, which is not in the field of view.

(f) Agent 2 lost track of the target object (red background). This is due to it having been reached, which prompts a new target creation, which is not in the field of view.



(g) Distance between Agent 1 and its target over time. Here, the blue object in sub-figure a is being approached.

(h) Distance between Agent 2 and its target over time. At tick -12 a new target was created (black bar).



(i) Similarities of *left_wheel* and *right_wheel* to the *forwards* and *backwards* actions respectively. Here, Agent 1 approaches its target because *forwards* has the highest similarity for both wheels.

(j) Similarities as described in sub-figure i for Agent 2. As no target is in the visual field, the default action for the symbol *no_object* is performed (rotate right).

Figure 11: Excerpts from our symbolic scripting experiment, controlling a Braitenberg vehicle with one linear sensor. The left column (sub-figures a, c, e, g and i) illustrates an agent approaching a target in its visual field. The right column (b, d, f, h, j) shows an agent with currently no target in its visual field, rotating clockwise (j) to find it.