Semantic Holographic Memories

Igor Peric, Stefan Ulbrich, Alexandru Lesi, Rüdiger Dillman, Marius Zöllner
FZI Forschungszentrum Informatik
Karlsruhe, Germany
Email: {peric, ulbrich, dillmann, zoellner}@fzi.de

***Abstract*—Vector Symbolic Architectures (VSA) define a set of operations for storage (compression), manipulation and retrieval of symbolic concepts, represented as fixed-length vectors. A specific instance of VSA, Holographic Reduced Representations (HRR), has proven to exhibit properties similar to human short-term memory and is, as such, interesting for computational modeling.**

In this paper we extend HRR approach by introducing topology-preserving, invertible encoding/decoding procedure for all major data types used in robotics and engineering. These encoding schemes play the role of otherwise used clean-up modules after memory retrieval (e.g. self-organizing maps), thus eliminating necessity for them.

We test our memories in three different robotic tasks: mapping and recall, visual object classification and state machine scripting (holographic controllers).

I. Introduction

Vector Symbolic Architectures (VSA) define a set of operations for storage (compression) and retrieval of symbolic concepts in fixed size vectors. In its basic form 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 compression technique, since binary operations on N-element vectors return N-element vectors as well.

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

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

Superposition is operation of extending a set of already memorized items with another item (or binding of two or more of them). As an example, a memory of "red triangle and blue circle" can be formed in following way:

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

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

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

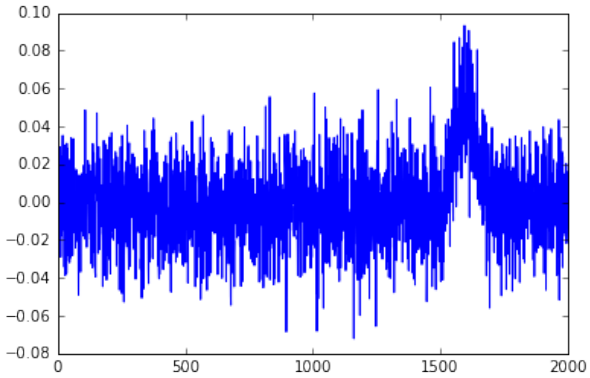


Fig. 1: Result of $(80 \otimes A) \oslash A$ for HRR from

Probing operation assumes existence of inverse element to association, so previous equation can be interpreted and explained in following way:

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

where $triangle'$ represents inverse of $triangle$. Resulting vector will be composition of something that is very similar to red and something that doesn't resemble anything meaningful from symbol table, so it can be treated as neglectable noise.

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

II. Holographic reduced representations

Holographic Reduced Representations are VSAs which implement operation of binding as circular convolution:

$$z_i = \sum_{k=0}^{N} x_i y_{i-k} //TODO \tag{5}$$

where z is the result of convolution, x and y are operands of size N. Superposition is implemented as simple element-wise addition. Probing is implemented using inverse of circular convolution - circular correlation:

$$y_i = \sum_{k=0}^{m} x_i y_{i-k} //TODO \tag{6}$$

- Implementation and building up of HRRs (similarity measures and cosine, lookup table...)

III. Topology-preserving encoding

- necessity (benefit of having it, querying smth that was not there) - implementation

In order to use various types of inputs with HRRs it is necessary to encode them in vectors of fixed size. The basic approach is to generate these vectors randomly for each distinct input and store these pairs in a map. This will consequently take its toll on the space complexity and may become infeasible, depending on the type of input used. The following section presents ways to encode values in a deterministic manner, removing the need to store these, as long as decoding is robust against noise.

A. Scalars

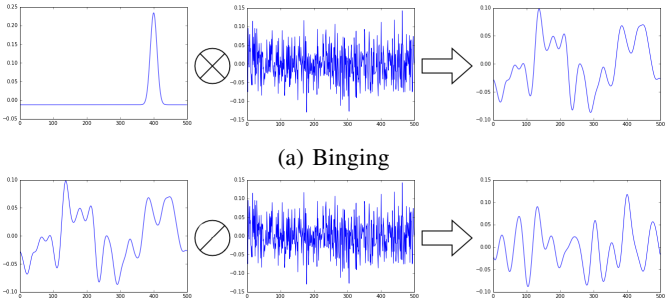
Encoding scalars is a good example for the necessity of preserving topology. Depending on what dataset is in use it may be required to store mappings for any number of values. Furthermore, using random encoding, there would be no similarity in neighbouring values. As such the following probe will yield no result:

$$(10000 \otimes A) \oslash 10001 \tag{7}$$

To facilitate these two qualities scalar encoding can be achieved with the help of Gaussian distributions. The encoded vector will simply represent a Gaussian bell sampled over a fixed input range. 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 lots of information is lost. If this result is probed with the same random vector used in binding something similar to the initial Gaussian bell 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 and the lack of frequencies present in encoded vectors becomes apparent in Fourier space as well. Figure 3 highlights this. All the information of a Gaussian bell is crammed into the lowest and highest values of the Fourier transformation. In these circumstances, when probing for a scalar, the vector received is in turn recreated of too few frequencies and does not come close to resembling the original Gaussian.

This inconvenience is removed with the help of permutations. Depending on the size of the memory vectors used a fixed permutation is generated. Whenever a scalar is encoded the Gaussian bell is first sampled over the range, after which the sampled values swap places accordingly. As such the resulting vector will present lots of jumps in value, which



(a) Binging

(b) Probing

Fig. 2: Plot of vector values. Figure 2a shows the binding of the value 8 on a scale to 10 with the symbolic representation of "ball", which is random. In figure 2b "ball" is being probed, and the expected result is a Gaussian bell for 8, mixed with some noise.

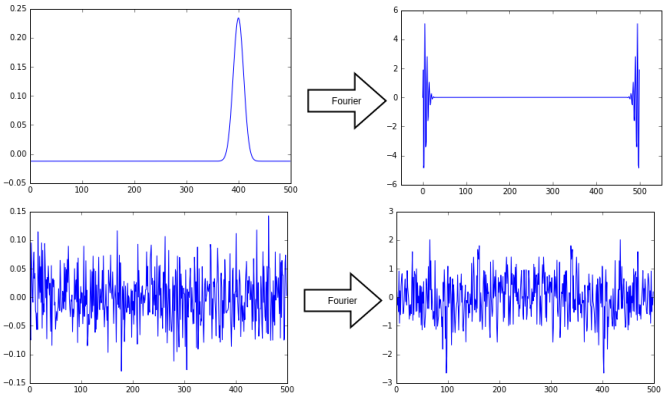


Fig. 3: Fourier space conversions of a Gaussian bell and a random vector.

have a significant impact on the Fourier space, as can be seen in figure 4.

Working with such a vector makes it possible to successfully probe for scalars and receive a Gaussian bell 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 bell. The result in figure 5b clearly shows an according spike. This vector can be smoothed out, after which the peak can easily be identified and the initial scalar can be recovered.

On a technical note it became apparent that simply sampling Gaussian bells in a vector with all the values greater zero brought up a small issue. Adding multiple such bindings on

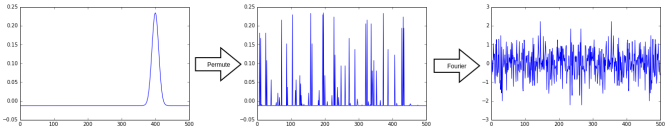
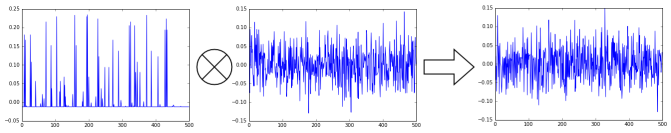
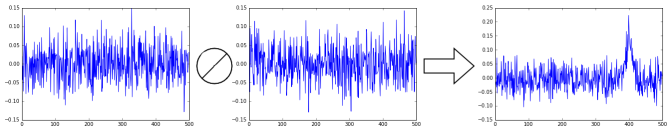


Fig. 4: Plots on the effect of permuting a Gaussian bell, and the impact on Fourier space.



(a) Binging



(b) Probing

Fig. 5: Figure 5a shows the binding of the permuted Gaussian for value 8 on a scale to 10 with the symbolic representation of ”ball”, which is random. In figure 5b ”ball” is being probed, and the expected result is a Gaussian bell for 8, mixed with some noise.

top of each other would constantly increase the values in the resulting vector. Due to this, when probing, we would receive vectors which had the correct distance between their values, i.e. the correct ”shape”, but would be shifted on the Y-axis. This can be solved by editing the permuted Gaussian bell to have $\sum_{i=0}^{n} x_i = 0$, and is achieved with this simple operation:

$$\forall x_i \in x : x_i = x_i - \frac{\sum_{i=0}^{n} ||x_i||}{|x|}$$

B. Coordinate encoder

C. String encoder

IV. Properties of SHM

- you can’t extract the items memorized, number of them, you can just query - they are probabilistic - you can modify them online (no separate learning/probing phases)

A. Capacity

1) SHM size vs. number of items:

2) Model of human forgetting:

3) Model of intentional forgetting:

B. Fan-out effect - multiple probing outputs

C. Limitations

- introducing problem of lookup (symbol grounding) - solution using implicit encoding and decoding

V. Experiments

A. Visual object classification

- SIFT + label

B. Localization and mapping

- SIFT + location

C. Holographic state machines (holographic controllers)

- sensory_data + current_state + next state

D. Reinforcement learning

- sensory_data + action + reward

E. Language understanding

- TensorFlow (role) + word = question answering

F. Implicit inductive reasoning

VI. Conclusions

Acknowledgment

The authors would like to thank... Manchester developers. :slowclap: