An Auto-Associative Unit-Merge Network

Kieran Greer *

Posted Date: 1 January 2026

doi: 10.20944/preprints202412.1209.v3

Keywords: auto-associative; unit; cohesion; compound key; noise

 doi:10.20944/preprints202412.1209.v3

Article

An Auto-Associative Unit-Merge Network

Kieran Greer

Distributed Computing Systems, Belfast, UK; kgreer@distributedcomputingsystems.co.uk

Abstract

This paper describes a new auto-associative network called a Unit-Merge Network. It is so-called because novel compound keys are used to link 2 nodes in 1 layer, with 1 node in the next layer. Unit nodes at the base store integer values that can represent binary words. The word size is critical and specific to the dataset and it also provides a first level of consistency over the input patterns. A second cohesion network then links the unit nodes list, through novel compound keys that create layers of decreasing dimension, until the top layer contains only 1 node for any pattern. Thus, a pattern can be found using a search and compare technique through the memory network. The Unit-Merge network is compared to a Hopfield network and a Sparse Distributed Memory (SDM). It is shown that the memory requirements are not unreasonable and that it has a much larger capacity than a discrete Hopfield network, for example. It can store sparse data, deal with noisy input and a complexity of O(log n) compares favourably with these networks. This is demonstrated with test results for 4 benchmark datasets. Apart from the unit size, the rest of the configuration is automatic, and its simplistic design could make it an attractive option for some applications.

Keywords: auto-associative; unit; cohesion; compound key; noise

1. Introduction

This paper describes a new design that can be used as an auto-associative memory. In this respect it is compared with a version of the Hopfield Network [6], which was the original inspiration, and also Sparse Distributed Memory (SDM) [10]. Content-addressable memories can retrieve a full memory from any subpart of sufficient size. They are also able to retrieve the original memory when some of it has been corrupted by noise. As Hopfield also explains [6]: 'Computational properties of use to biological organisms or to the construction of computers can emerge as collective properties of systems-having a large number of simple equivalent components (or neurons).' This notes computational properties of neurons, which suggests some type of functional behaviour. Recent biological research suggests that it could be possible to have memory structures that are less functional as well, such as with glial cells. The main problem with discrete Hopfield networks is their memory capacity, which is only about 0.15 times the number of nodes. For example, a network with 100 nodes would only be able to store about 15 different patterns before they started to corrupt each other. This has now largely been solved with the new Modern Hopfield Network [11] that also does not need to be discrete. The memory structure tends to be dense however and so it may not deal very well with sparse data. One model designed to cope with sparse data is presented in [8], but it also notes that apart from some very nice advantages, the modern Hopfield models have been shown to be computationally heavy and vulnerable against noisy queries. In particular, the dense output alignments of the retrieval dynamics can be computationally inefficient, making models less interpretable and noise-sensitive. Sparse Distributed Memory is another content addressable model for human memory that can handle large amounts of data. It is designed to be sparse and has similarities and differences with the Unit-Merge network model. These are described in section 2.

This paper proposes a new model called a Unit-Merge Network. Unit nodes at the base store integer values that can represent binary words, for images, for example. Thus, a unit size is the first design feature and it is critical and specific to the dataset. It also provides a first level of consistency

 doi:10.20944/preprints202412.1209.v3

over the input patterns, because specific binary sets must be present for the related integer values to be flagged. A second cohesion network then links the unit nodes list, through novel compound keys that create layers of decreasing dimension, until the top layer contains only 1 node for any pattern. The cohesion network therefore joins the units together, thus allowing whole patterns to be retrieved again. The structure looks to be different, but one might think about a standard auto-encoder, with similar input and output layers (the unit nodes) and then a hidden layer (the cohesion network) that contains the common features. In this respect, that type of functionality is present in the Unit-Merge network as well.

The rest of the paper is organised as follows: Section 2 gives some related work. Section 3 describes the new Unit-Merge network, including algorithms and section 4 gives some theory about the design. Section 5 gives some test results, while section 6 gives some conclusions on the work.

2. Related Work

The Modern Hopfield Network [11] has resulted in a plethora of new designs, where the architecture has been further evolved, or combined with transformers or deep learning structures, for example [7][8][9][14]. One variation described in [4], includes setwise links, creating a simplical complex. This is interesting because the simplical sets group base nodes, but into higher levels and degrees. Basically, the corollary 2.2 on page 5 states that the network capacity is proportional to the number of connections. To store more patterns, you need to add more connections. It cites earlier papers that also consider setwise links, but for a binary pairwise Hopfield network. The idea is to link non-linearly, in concentrations, where the nodes are associated. The linking process is entirely different to the cohesion network, however and continually changes the network topology. The appendices of that paper have biological reference, including neurons and synapses. Another new design, described in [1], might in fact be closer. Their system still reduces an energy function, but it constructs a graph representation (discrete-graphical model) as the internal model, which is why it may be more similar. They argue for local learning rules, both in the human brain and their system. Because Deep Learning is often implemented with backpropagation, it is non-local and generally considered biologically implausible. They use a single forward-pass and backward-pass for retrieval, which is again like the Unit-Merge network and also, the one-shot integer value retrieval.

The original work on SDM can be found in [10]. It used a Hamming distance with the precept of: 'The pursuit of a simple idea led to the discovery of the model, namely, that the distances between concepts in our minds correspond to the distances between points of a high-dimensional space. Strictly speaking, a mathematical space need not be a high-dimensional vector space to have the desired properties; it needs to be a huge space, with an appropriate similarity measure for pairs of points, but the measure need not define a metric on the space.' Long (high-dimensional) binary vectors, or words, are used as the memory model, but in fact, it is necessary to reduce the dimensionality so that these large words can be managed. Larger words are thus broken down into smaller bitstrings that point to the locations of whole memories, typically through a threshold 'radius' given by a Hamming distance. Thus, there are whole binary memories and an indexing system to find them, where the number of hard locations or addresses actually used, is much less than the possible number from the memory size. There would therefore be a redundancy close to the Hopfield network capacity, but this also makes the problem tractable. The original SDM could handle 20% noise, but examples of noisy patterns were included in the memory.

In [16], the SDM binary numbers were converted into decimal, which is also done with the Unit-Merge network. Instead of cohesion, a Euclidean range around the selected memory was used to retrieve matching possibilities, which were then summed and a majority vote produced the final result. Different methods of encoding the data were tested in [13], including: Natural Binary Code (NBC), NBC with a different sorting of the numbers, integer values and a sum-code. The paper states that 'Another big weakness of the original SDM model is that of using bit counters. This results in a low storage rate, which is about 0.1 bits per bit of traditional computer memory, huge consumption of processing power and a big complexity of implementation.' Problems with using Hamming

 doi:10.20944/preprints202412.1209.v3

distances for converting binary to decimal are noted in [16], as follows: 'The integer representation has several advantages, such as a simpler encoding, it diminishes the effect of normalization when several vectors are combined and it avoids other undesirable effects. But decimal and Hamming differences are not the same. For example, 0111 and 1000 would give decimal values of 7 and 8, with a hamming distance of 4, but a decimal difference of 1.' The method of this paper still prefers the hamming distance. In their case, the decimal number is required to locate the position for a pattern in a vector list, whereas in this paper, patterns want to be clustered based on word similarity. It is described in [13] however, that a conversion to binary from some other form that also includes noise, also has problems. For example, if there are greyscale images and 8-bit patterns are being stored, then a grey-level value of 127, or 01111111, could be converted with very little noise into 128, or 10000000, but the hamming distance between these two values is very large. The best method is therefore application-dependent, because either option is possible.

3. The Unit-Merge Network

3.1. Network Construction

The model proposed in this paper is not a recursive Hopfield design, but is more like a Sparse Distributed Memory (SDM). The focus is on binary inputs only, with a value of 0 or 1. As the name suggests, the input pattern can be split up into units that each contain a certain number of the binary inputs. It could be 8 bits or 16 bits, for example and in fact the selected size is quite critical. When stored in the network, these binary numbers are converted to decimal first. Each network node therefore references words of this size and not individual binary values. This not only helps to reduce the size of the network but it adds a certain amount of coherence, where these smaller 'patterns' need to occur for the unit value to be flagged. Then a second indexing network makes the retrieval and comparisons more efficient. It also adds cohesion across the units, by clustering the ones with shared values together. This cohesion network links all the base unit nodes through a number of layers that converges to a single final layer node. The dimension is reduced at each level by combining 2 values from the previous level into a compound one, and also leaving out 2 values. Figure 1 is a schematic of a Unit-Merge network.

While the unit nodes may be generated from binary value lists, it would be possible to reduce that list size to 1 and allow them to store a decimal value directly. The node pairing is spatial, or between adjacent units, as determined by the input pattern. This would mean that the base units can retrieve the exact same pattern again. The nodes at any level may therefore contain links for more than 1 base pattern and so there is a certain amount of search and comparison required, but the network structure is able to keep that to a minimum. While it may appear that this network simply stores every memory and then retrieves what is closest to the input, it does this using a lot of overlap in values that it stores and thereby reduces the memory requirements by a lot. In fact, it is probably more economic than a discrete Hopfield network for the same number of patterns. It is not completely deterministic however and can produce slightly different results on different test runs, but the only critical criterion is really the size of unit to use. The rest is mostly automatic. One unanswered question is how to cluster sibling nodes in a layer using the hamming distance, or at least the most economic way to do it. This would allow small traversals to neighbouring nodes, for example, but the current algorithm does not require that.

 doi:10.20944/preprints202412.1209.v3

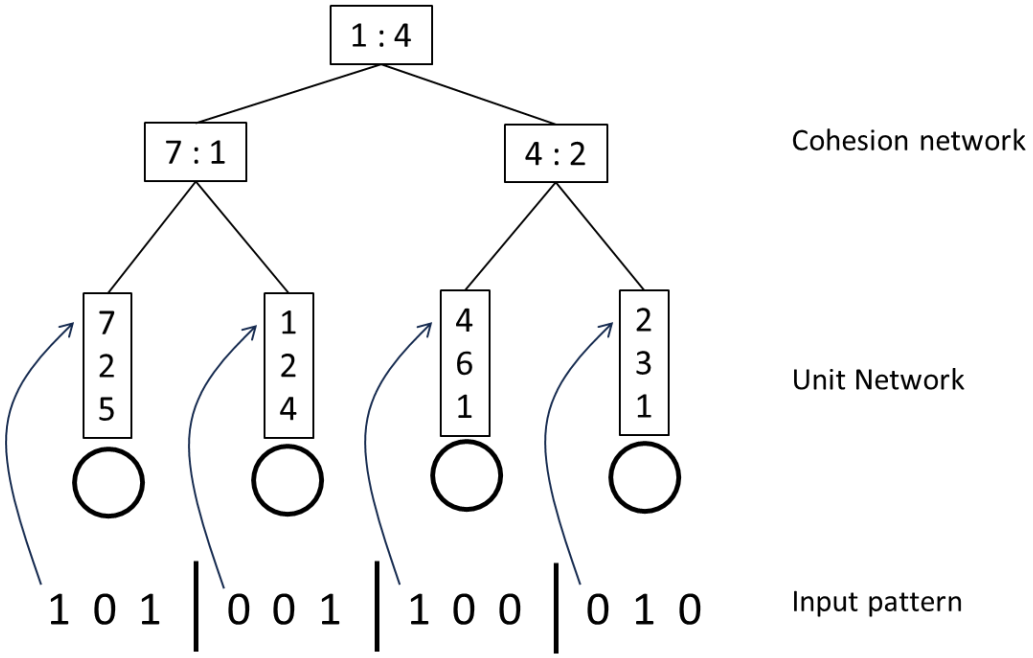

Figure 1. Unit and Cohesion Networks. The current binary input is shown at the bottom. It has been split into unit sizes of 3 and converted into the integer numbers 7, 1, 4, 2, which are added to the related unit list. The cohesion network then combines adjacent numbers to produce compound keys. 7 and 1 produce the first key, while 4 and 2 produce the second. These are then merged into 1 and 4 for the top node key.

3.2. Network Use

Using the network also consists of two stages. To train the network, the first stage creates the base unit nodes and assigns each node a list of integer values that represent any of the binary array words assigned to the unit. The second stage converts each pattern into a hierarchy of compound integer values and stores these in a cohesion network. This network can then search through the hierarchy to find the base unit patterns again. The cohesion network is also used to match new input with the base units, but with some additional rules, as follows: A new network is created for the new input pattern only, which results in a cohesion network for the input pattern only that is also a binary tree. The unit node values for the new input may be missing in the trained network and so if an input node does not match exactly with a corresponding trained node, then the unit value is set to 'null.' The trained network is then traced through to the base nodes using the input pattern's binary tree and if a null value is encountered, then any key value is allowed at that branch. This results in a small set of base unit values that are converted back to their binary equivalents, to produce a small set of whole train patterns. Each pattern is compared with the new input pattern and the one with the least hamming differences is selected as the correct pattern.

3.3. Train Algorithm

The structure consists of a unit list and a cohesion network and so both need to be built. Figure 1 illustrates what the data looks like after it has been added to the two structures.

3.3.1. Unit List

- Create a unit list, where each node stores integer values relating to an indicated number x of base binary values.
- For each input pattern, split it up into words of the indicated unit size x. Note that the unit size is important and will separate out the values (and features) differently.
    - Convert a list of base words, representing a unit, from binary to decimal.

 doi:10.20944/preprints202412.1209.v3

- Store the integer number only once in a value list, for the corresponding unit.

3.3.2. Cohesion Network

- Link and add all decimal numbers for the current pattern to the cohesion network.
  - Create a layer of base compound keys by combining the newly added unit node values, using 2 adjacent nodes each time.
  - Then to reduce the dimensions, take the second compound value of the first node and the first compound value of the second node and create a new compound value and node from it, in a new layer.
  - Repeat until there is only 1 node in the top layer.
- This creates a path which if followed, will lead to the input pattern at the end.
  - The paths are not unique however and because some information has been lost, there can be several instances at each node. But the numbers will be much smaller than for a flat list.
  - Thus, when there are several choices, include all possibilities and leave the final decision to a comparison metric at the end.

3.4. Test Algorithm

The retrieval process therefore also requires traversing both the unit list and the cohesion network.

3.4.1. Unit List

- Split the test input pattern into unit sizes and create the corresponding unit values.
- Compare each number directly with the corresponding unit node value list.
  - If the value list contains the number, then keep the value for that position.
  - If the value list does not contain the number, then add a 'null' value for the position.
- This produces a list of integer or null values that represent the test pattern in the trained network.

3.4.2. Cohesion Network

- Create a 1-instance cohesion network for the test pattern only, using the test unit values.
- Traverse the trained cohesion network using these values.
  - If there is a null value at a node, then any key that includes the other value can be selected.
  - While there will be multiple options, the final set of patterns will be much smaller than the train dataset.
- Retrieve all the selected base patterns and convert them back into binary patterns.
- Use a Hamming distance or some other similarity count, to select the pattern that is closest to the test input pattern.
  - If an appropriate match is not available, the current result pattern can be convoluted once and the test run again on the convoluted pattern.
- The finally selected pattern can thus have different values to the test pattern and therefore might remove noise from it.

Note that if there is no exact unit match, then it is not good to include a closest match, because this leads to a combinatorial explosion that cannot be managed. It was much more economic to add a null value and leave the decision to the final set of patterns.

4. Theory and Complexity

The purpose of the unit nodes is to reduce the dimensionality of the input pattern, without losing information. In the binary case it adds information, because individual bits are converted into more

 doi:10.20944/preprints202412.1209.v3

meaningful words. The purpose of the cohesion network is to maintain links over the units while providing a more compact structure that can be searched over, instead of an exhaustive search. In fact, storing values as they are, instead of using a weight transposition, should make the information more reliable and so a different method to reduce the memory size is required. The compound key therefore removes information at each level, but its oblique construction process means that value sets will still obtain unique paths if 2 or more levels are traversed. This therefore fulfils the requirements of large number of patterns to be stored in a more compact structure and for it to be economically searched over. If knowledge is shared between patterns, then the network can also generalise. This is achieved in a lightweight way when nodes are shared between the patterns.

At first sight, storing patterns as they are does not seem to be a very economic solution. However, a direct comparison with the memory requirements of a discrete Hopfield network, for example, show that it is reasonably economic. Consider the following example: an auto-associative network with 100 nodes is to be used. The Hopfield network stores a weight from each node to every other node, which is 100 x 100 = 10000 weights. Even if this is symmetric, so that ab == ba, then that still requires 5000 weights. If a Unit-Merge network was to use units of size 4, then there would be 25 unit-network nodes. The cohesion network reduces these in binary style, from 13 to 7 to 4 to 2 to 1. Summing this gives 25 + 13 + 7 + 4 + 2 + 1 = 52 nodes. A Hopfield network can reliably store ≈ 0.15N nodes, or 15 patterns in this case. An upper limit would therefore be if every pattern value is different, leading to 52 x 15 = 780, which is much less than for the Hopfield network. This is only a crude estimate, but it shows that the memory requirements should be OK. For example, the Semeion dataset (see section 5) requires possibly 54579 values to be stored, whereas a fully-connected Hopfield network would require (256 x 256) = 65536 weight values, or if symmetry is considered then 32768 values. But that would be for 39 patterns, not the 1597 patterns that the Unit-Merge network stores.

Binary reduction is the chosen method for the network construction, but it could be an n-ary reduction, for example. When values are added from different patterns, any node can be used more than once and so the trained cohesion network will typically have multiple branches at any node. However, when searching this network, it is done using a binary tree from the single test pattern, with the caveat of selecting what the next values at a node might be, from a small list. The train phase is quicker than the test phase, because the data simply needs to be converted into units and then compound keys created for the cohesion network. Most of the time complexity therefore relates to the test stage, when search and other algorithms are required. The relative times varied quite widely, where the smallest was a test time of about 3-4 times the train time and the largest was over 200 times the train time. For these small test sets, both were in milliseconds. If measuring the amount of time taken per unit, then the relative amount decreased as the number of units increased. This suggests a logarithm time complexity, or O(log n).

5. Testing

A computer program was written in the Java programming language and runs on a standard laptop. The source code for this can be found on GitHub [19]. The network was tested with 4 different sets of images. The Chars74k dataset [17] is a set of hand-written numbers, where only the numbers 1 to 9 were used. This produced approximately 55 examples for each number, or 360 images in total. The image was converted into a 32x32 black and white ascii image, which would also be a binary image with the values 1 or 0. The second dataset was another set of handwritten numbers called the Semeion dataset [2][3]. These were converted into 16x16 binary images, with a total of 1597 images. A third benchmark dataset, obtained from the UCI Machine Learning Repository [18], was a small subset of the Caltech 101 Silhouettes dataset [12]. These were converted into 64x64 binary images and placed into 8 different categories, with up to 10 images in a category, with 68 images in total. This was not pre-determined, but depended largely on the number of available images. The fourth set of images was the set of ‘apple’ images from the MPEG-7 core experiment CE-Shape-1 [15]. This was only 20 images, all from the same category, but with a pixel size of 256x256. This was used to show that the images can be larger in size.

 doi:10.20944/preprints202412.1209.v3

5.1. Test Strategy

The only parameter that needs to be set is the unit size. Thus, a number of test runs could determine which unit size would give the best result for each dataset and this could be done automatically as part of a train loop. The results were not the same every time, but were close and so the results presented in Table 2 have been averaged over 50 test runs each. A standard deviation value for the test results was only about 0.5 – 1 and a single testing phase would take seconds or less to run. After determining what the best unit size was, the trained network was presented with the dataset again, but with a certain amount of noise added to each row. This included switching both the 1 and the 0 values, where a 10% noise factor would randomly switch 10% of the binary values, for example. With no noise, the network would be expected to recognise each train image again, but with noise, it would have to match partially to the image and then retrieve the train image that was closest. Because the images are placed in categories, the test also measured what category the finally selected image was from.

This resulted in two different strategies, as follows:

- If testing the row accuracy, then the test result was compared with the rows in the same category only. An exact match with any row was then required to indicate a positive score.
    - If there was not an exact match, the current result pattern would be convoluted only once, and the test run again using the convoluted pattern.
- If testing the category accuracy, then the test result was compared with every row in each category. A percentage match was calculated, with the best score from any row determining the selected category.
- The metric was a basic count of how many (binary) pattern elements were the same.

Thus, two percentage scores were produced for each test run. The first was if the input row returned an exact match with a row from the same category and the second was if the input row category was selected from all categories.

5.2. Row and Category Test Results

The preferred unit size for each dataset is shown in Table 1, while Table 2 gives the accuracy for both row and category matching. Because the apples dataset has only 1 category, it is only evaluated for the row matching.

Table 1. Preferred unit Sizes.

Dataset	Dimensions	Unit Size
Chars74	32 x 32	12
Semeion	16 x 16	16
Silhouettes	64 x 64	4
Apples	256x256	24

Table 2. Test results show the accuracy for exact row recall or correct category recall, with varying amounts of noise in the test pattern.

Dataset	Row Accuracy for Noise %					Category Accuracy for Noise %				
	0%	10%	20%	30%	40%	0%	10%	20%	30%	40%
Chars74	100	98.7	91	75.9	60	100	98.9	91.5	76.1	59.3
Semeion	100	96.2	90.2	85.1	83.4	99.7	95.8	90	84.9	83.5
Silhouettes	100	98.4	95.3	87.4	77.2	100	98.5	94.9	87.2	78.9
Apples	100	100	100	100	100	n/a	n/a	n/a	n/a	n/a

 doi:10.20944/preprints202412.1209.v3

The category matching could theoretically perform worse than exact row matching. It did involve a best match over all rows and that might be different to a single exact match. The apples images are larger but they kept a good accuracy, even with larger amounts of noise. The shapes are relatively uncomplicated however and the best unit sizes were relatively smaller – 24 or lower. It should be noted that binary numbers can become quite large after that and many different ones would then be stored for a single unit. It should also be noted that a test set with different images was not recognised very well, but that would not be the objective of an auto-associative network. It should also be noted that two fairly-basic discrete Hopfield networks failed this test completely. The digit images are quite sparse and so a Hopfield network may tend to move each node's score to 0. This is what happened, with the returned patterns being all 0's, because that was also the input to most nodes. Modern Hopfield networks [11] are addressing this problem by placing the minima more accurately, but they also need to be specialised to cope with sparse data [8].

5.3. Occlusion Test Results

To show that occluded data can also be managed, the apples dataset was used again, where the images were altered to have the top 70 from 256 lines removed. This resulted in a 27% occlusion, or removal of the leaf feature in the image. The retrieval of the correct original image was 100% successful, where the result from 3 images is shown in Table 3.

Table 3. Occluded data test result for the 20 Apples images. All images had the top 70 lines removed (27%) and were successfully recognised again. Three are shown in the table.

Image	Original	Occluded	Test Result
1			
2			
3			

The unit size was still critical. A 16 bits unit size could return the wrong image, for example, whereas a 24 bits unit size returned the correct one each time. So even if the row accuracy result was the same for these two unit sizes, it was different for the occlusion test.

5.4. Comparison with Other Classifier Types

The datasets tested here are quite small and could be converted to an ascii format of 1's and 0's, which allowed some preliminary results to be quickly obtained. The MNIST dataset [5] has been used by other researchers, which contains similar handwritten digits, but with many more examples. The accuracy is expected to degrade when noise is added and the value is comparable with what other systems have produced. For example, the paper [4] quotes an 87% accuracy for its particular test, while [16] quotes 65% accuracy for 30% noise. The paper [7] states that their design can perfectly store 60,000 patterns, but for classification only. There is no mention of noise. Another new model [9]

 doi:10.20944/preprints202412.1209.v3

trained 10000 images from MNIST for a classification problem and quotes a 90% accuracy, which is similar to backpropagation networks. They note that the accuracy drops to about 53% for noisy data, but only state that it relates to a standard deviation of 4. The tests in [14] were carried out on modern versions of the Hopfield network and also SDM. They introduce the idea of a Universal Hopfield Network and the test results showed that 50% noise could be managed, including some images from the MNIST dataset, but not to 100% accuracy.

5.5. Limitations of the Unit-Merge Network

This paper presents only preliminary results, which are very encouraging. The patterns have been converted into binary black and white images, while a complete model would need to handle coloured images as well. How this is achieved will be a critical next step. But the units can be made to accept input from single sources that contain integer not binary numbers instead and so, converting to a more detailed input pattern should be possible.

6. Conclusions

A Unit-Merge Network has been shown to be very efficient, both in terms of processing time and memory requirements. It can handle dense or sparse data and does not require large basins of attraction, thereby allowing it to store a larger number of patterns safely. This relates to the energy landscape, like a minimum and is the area of influence for each pattern that is stored, which is global for the original Hopfield network. The Unit-Merge network is possibly a leaner design, where any overlap in the values can be easily separated out again. Noisy input is the primary concern in this paper and it is shown that for even 40% noise, the retrieval accuracy is as good as for other methods. The results of this paper therefore appear to be competitive with those of other modern architectures. The model has an input/output layer (unit network) and a hidden layer (cohesion network) analogy and could be compared with any associative network. A sparse quantized variant of the Hopfield Network also shows that a graphical representation of the hidden layer is interesting. If the Unit-Merge network was to have an energy function, then it might be a shortest-path version, which could be a local rule. The best unit size would need to be decided first, but then online learning would be possible, which is another reason cited in [1] for their design. While the Unit-Merge network has a much simpler mathematical framework, this should not detract from the obvious importance of the Hopfield models. In fact, determining exactly what functionality they represent could make them all-the-more interesting. For example, they are thought to be a model for human neural memory, as is SDM and a Unit-Merge network may also be a memory model.

References

1. Alonso, N. and Krichmar, J.L. (2024). A sparse quantized hopfield network for online-continual memory, Nature Communications, 15:3722.
2. Bouaguel, W. and Ben NCir, C.E., 2022. Distributed Evolutionary Feature Selection for Big Data Processing. Vietnam Journal of Computer Science, pp.1-20.
3. Buscema, M. (1998). MetaNet: The Theory of Independent Judges, in Substance Use & Misuse, Vol. 33, No. 2, pp. 439 - 461.
4. Burns, T.E. and Fukao, T. (2023). Simplical Hopfield Networks. https://arxiv.org/abs/2305.05179.
5. Deng, L. (2012). The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web], in IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141-142, doi: 10.1109/MSP.2012.2211477.
6. Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. Proc. Nat. Acad. Sci., 79(8):2554–2558.
7. Hu, J.Y.C., Wu, D. and Liu, H., (2024). Provably optimal memory capacity for modern hopfield models: Transformer-compatible dense associative memories as spherical codes. Advances in Neural Information Processing Systems, 37, pp.70693-70729.

 doi:10.20944/preprints202412.1209.v3

8. Hu, J.Y-C., Yang, D., Wu, D., Xu, C., Chen, B-Y. and Liu, H. (2023). On Sparse Modern Hopfield Model, 37th Conference on Neural Information Processing Systems (NeurIPS 2023).
9. Joshi, S.A., Prashanth, G. and Bazhenov, M. (2023). Modern Hopfield Network with Local Learning Rules for Class Generalization. In Associative Memory {\&} Hopfield Networks in 2023.
10. Kanerva, P. (1992). Sparse Distributed Memory and Related Models, NASA Technological Report, Or, In M.H. Hassoun, ed., Associative Neural Memories: Theory and Implementation, pp. 50 - 76. New York: Oxford University Press, 1993.
11. Krotov, D. and Hopfield, J. (2018). Dense Associative Memory Is Robust to Adversarial Inputs. Neural Computation, 30(12):3151–3167.
12. Marlin, B., Swersky, K., Chen, B. and Freitas, N., (2010). Inductive principles for restricted Boltzmann machine learning. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, pp. 509 - 516.
13. Mendes, M., Coimbra, A.P. and Crisostomo, M. (2009). Assessing a Sparse Distributed Memory Using Different Encoding Methods, Proceedings of the World Congress on Engineering 2009 Vol I, WCE 2009, July 1 - 3, 2009, London, U.K.
14. Millidge, B., Salvatori, T., Song, Y., Lukasiewicz, T. and Bogacz, R., 2022, June. Universal hopfield networks: A general framework for single-shot associative memory models. In International Conference on Machine Learning (pp. 15561-15583). PMLR.
15. Shape data for the MPEG-7 core experiment CE-Shape-1. http://www.cis.temple.edu/~latecki/TestData/mpeg7shapeB.tar.gz. (last accessed 30/8/25).
16. Snaider, J. and Franklin, S. (2012). Integer sparse distributed memory, In Twenty-fifth international flairs conference.
17. The Chars74K dataset, http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/. (last accessed 30/8/25).
18. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/. (last accessed 30/8/25).
19. Unit-Merge Source Code. (2025). https://github.com/discompsys/Unit-Merge-Network.