A Study on Merging Mechanisms of Simple Hopfield Network Models for Building Associative Memory

Peter Kimani Mungai
Graduate School of Computer and Information Sciences
Hosei University
Tokyo, Japan
mungai.peter.69@stu.hosei.ac.jp

Runhe Huang
Faculty of Computer and Information Sciences
Hosei University
Tokyo, Japan
rhuang@hosei.ac.jp

***Abstract*— We are in the era of artificial intelligence which is fulfilling the vision of ubiquitous intelligent machines and systems. Just like in us humans, the most critical elements for machine intelligence are those related to associative memory and recall. It is urgent to have a structure or model that can emulate human-like associative memory and recall functionality. This research is aimed at building a model based on a hybrid of neural networks like Hopfield network, recursive neural network (Recursive NN), and recurrent neural network (Recurrent NN). In our study, a Hopfield network is used as a core element for learning associative relations among concepts or objects and the Hopfield matrix as a basic unit memory for association knowledge. Hopfield networks in a sequence are recursively merged by applying a Recursive NN and associative relations between concepts/objects nodes in two Hopfield networks are learned and their sequences of merging are kept in a Recurrent NN. When there is a stimulus, the model can retrieve its associative concepts or objects as its recall. This paper shows the feasibility of the proposed model with some case study and a proof of concept prototype.**

Keywords*— *Artificial intelligence, Deep learning, Hopfield network, Associative memory, and recall, Cognitive model

I. INTRODUCTION

The attainment of general artificial intelligence is still a pipe dream to AI researchers. In the early days of AI, many scientists believed that computers would quickly gain human-level intelligence. This belief gave rise to over expectations by the early adopters of AI which were not well managed. Due to the inability of AI to meet those ambitious expectations, there was a widespread skepticism which slowed down the research and development of the field. However, deep learning which has attracted many researchers in the recent years has breathed new life to the initial goal of AI.

In the contemporary world, the use of computer applications that perform sophisticated tasks perceived to require intelligence to solve have become ubiquitous and most of them rely on deep learning techniques. Some of these applications include image processors, voice recognition and machine translations systems. Many researchers in the field are of the view that, deep learning has been the missing link in the chain leading to the achievement of artificial general intelligence [1].

Deep learning allows computational models composed of multiple processing layers to learn a representation of data with multiple levels of abstraction [2]. Deep learning techniques came as a relief to the conventional machine learning techniques which do not learn representation from raw data. Representation learning has poised deep learning as the best bet towards the attainment of general AI since these models can take natural raw data as input and discover necessary representation for classification to take place. Initially, representation of raw data had to be crafted by hand which was not only time consuming but also an error-prone exercise [3]. Automation of this process through the use of deep learning techniques has accelerated the research in the field.

A major hurdle to be overcome before general AI can be achieved however is to have intelligent agents that are able to learn and recall different tasks [4]. This can be only achieved if the agents have access to an efficient memory and recall mechanism. In the case where the intelligent agent is a neural network, the network needs to learn and remember a wide array of tasks. In most neural network models, there is a limitation to the number of tasks that can be learned and recalled mainly because learning of new tasks normally calls for altering the weights associated with other tasks. This alteration of weights associated with previous tasks when learning new ones in neural networks eventually leads to the problem of catastrophic forgetting. Reducing the interference of weights associated with already learned tasks is highly in tune with the goals of this study.

Our aim in this study is to create an associative memory and recall model that establishes associations between different Hopfield networks. Our choice of Hopfield networks in this study was informed by their simplicity and does not mean therefore that the model cannot work with other learning models. The proposed model will in addition to establishing associations among different learning results also merge the weight matrices of associated Hopfield networks into one weight matrix. Doing this will ensure that we improve the association ability of Hopfield models while at the same time improve the efficiency of the recall process since related weights will have been merged.

Most real world tasks are naturally dependent on continuous events [5]. This reality has made recursive and recurrent learning models an obvious choice for many deep learning researchers. These learning models have been widely applied to the processing of continuous data. In this study, we borrow the concept of Recurrent NN to create and maintain a sequence of Hopfield matrices and the concept of Recursive NN to recursively create and update an association graph in which the merging of associated Hopfield matrices takes place. More details are discussed in later sections of this study.

Proc. 2017 IEEE 16th Int'l Conf. on Cognitive Informatics & Cognitive Computing (ICCI*CC'17)
N. Howard, Y. Wang, A. Hussain, F. Hamdy , B. Widrow & L.A. Zadeh (Eds.)
978-1-5386-0771-8/17/$31.00 ©2017 IEEE

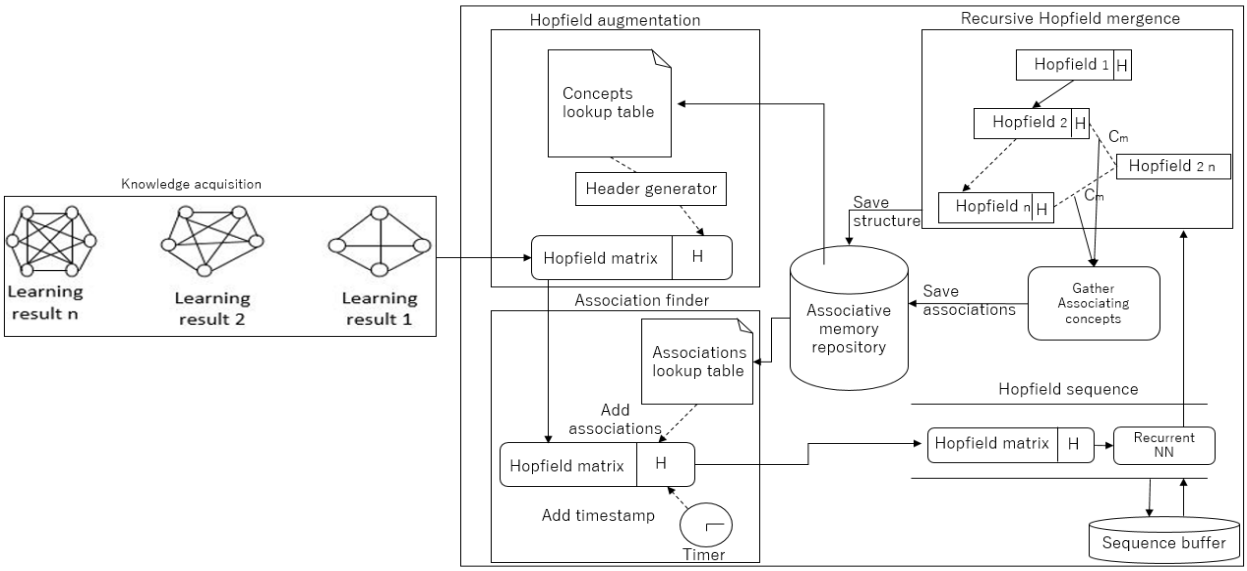

Fig. 1. Associative memory and recall (AMR) model

This paper is structured as follows. In Section II we will introduce our proposed model. In section III we will describe the merging mechanism used in the proposed system. In Section IV we will demonstrate the functionalities of the proposed model using hypothetical use case scenarios. In Section V we will present our proof of concept prototype. In Section VI we will review some of the related works, and finally, the conclusion and future work will follow in Section VII.

II. Associative Memory and Recall (AMR) Model

Fig. 1 shows the conceptual view of our proposed model. The model is composed of five modules i.e. knowledge acquisition, Hopfield augmentation, association finder, Hopfield sequence and recursive Hopfield mergence. Learned association knowledge which is the result of a Hopfield network is fed to the Hopfield augmentation where the weight matrix is augmented by annexing a header vector to it. The augmented matrix is then sent to the association finder where the associations between the matrix and previous matrices are established. The timestamp is also noted and included in the header vector by the association finder. The matrix is then sent to the Hopfield sequence where a recurrent NN is used to learn and maintain a sequence of each received matrix. The sequence is then sent to the recursive Hopfield merging where associated matrices are merged.

A. Knowledge Acquisition

This is a key module block of the proposed system which is aimed at using Hopfield network to learn the association between concepts and among objects in a task or some tasks. Like us humans, we learn via interaction with the real world, machines learn via interaction with its external environment. The module block acts as a conduit between the proposed model and learning models such as Hopfield neural networks.

Hopfield learning models were popularized in 1982 by a researcher known as John Hopfield [6]. These models are inspired by the concept of magnetism where it has been proven that particles within a magnetic field will vibrate continuously (attracting or repelling each other) until they reach the most energy efficient state among themselves. This borrowed concept is implemented in Hopfield neural networks whose fully interconnected neurons influence each other's output until the conditions of a pre-defined function are met. The result of these models is usually in form of a matrix whose diagonal elements are all zeros since loopbacks on a single neuron are disallowed. The diagonal of these matrices also act as mirrors since elements in the upper part of the diagonal have the same values as those in the lower part of the diagonal. This property makes it easy to implement Hopfield network models in a computer program since only one-half of the matrix on either side of the diagonal is calculated and results copied to the other part.

Hopfield models are used for image processing, pattern matching and data deconvolution [7]. However, these models use simple association architecture which we plan to leverage on when merging various networks to make more complex associative memory architectures.

The proposed model will act on the learning results of Hopfield learning models. In the model, learning results from a learning model are acquired and used sequentially over time as shown in Fig. 1.

B. Hopfield Augmentation

The goal of this module is to assign a concept to the received Hopfield matrix. Concepts data is stored in a look-up table and it contains real world facts and their corresponding representations by the Hopfield network. The concept look-up functionality imitates the functionality of a DNS in a network system. It is assumed that the concept look-up table is pre-learned and established in the system.

This module augments the Hopfield matrix representation by associating it to concepts. The mapping between a Hopfield matrix and a concept can happen in many ways. For simplicity purposes, we have decided to map the order (size) of each incoming matrix to a given concept in the look-up table.

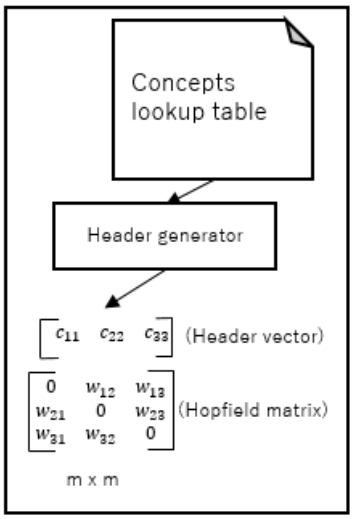

Fig. 2. Hopfield Augmentation process

The header generator is the bridge between the received weight matrix and the concept look-up table. The generator annexes a header vector to the received weight matrix indicative of what concept the weight matrix belongs to. As shown in Fig. 2, the header is made of elements (for example, c_{11}, c_{12}, c_{13}) in the header vector c_1 which are mapped to the number of concepts in the look-up table. The sequence of elements in the header vector is same as the sequence of the concepts in the look-up table.

Based on the order of the received matrix, the header generator establishes a related concept from the look-up table. Consequently, the header vector element corresponding to that concept is changed to a true value while those corresponding to the other concepts are changed to false value.

The Hopfield matrix together with its header vector is then passed to the association finder module block. The header vector is composed of only those concepts that describe the current weight matrix. In a future study, we will enhance the vector generation function not only to rely on the size of the matrix.

C. Association Finder

This module is responsible for establishing associations between different concepts. The module updates the header vector annexed to the Hopfield weight matrix by the previous module. On receiving the augmented weight matrix, the association finder reads its header vector to determine the concept value of the received matrix. Using this value, the association finder consults the association look-up table to find how this concept relates to other concepts. Similarly, it is assumed that the association look-up table is also pre-learned and established in the system.

The association look-up contains an association description of all the concepts in the system. The association finder reads the header vector of the received Hopfield matrix to determine which concept it relates to. The association finder then checks the association look-up to identify all the other concepts related to the identified concept.

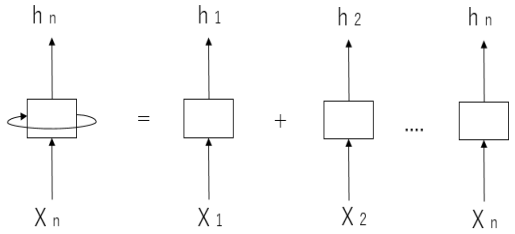

Fig. 3. An example of a recurrent neural network model

After the establishment of associations, the timer adds a new element to the header vector to hold the timestamp. The timer continuously keeps track of all the received Hopfield matrices and increments a counter each time a new one is received. The extra element added to the header vector is updated with the current value of the timer. This ensures that the original sequence of the Hopfield matrices is known.

D. Hopfield networks sequence

This function module is responsible for maintaining the sequence of all the received Hopfield matrices and saving it in the Hopfield sequence buffer. To accomplish this, the principle of recurrent neural networks is used. The module adds a frequency element to the header vectors which represents the number of times a Hopfield matrix has been simulated. The module is then trained using all the frequencies already in the system thereby making it possible for the module to predict the sequencing order of a new input. Matrices with a higher frequency are given priority in a queue.

Recurrent neural networks (RNNs) are special types of neural networks designed to process continuous data. As shown in Fig. 3, a recurrent neural network allows for loopbacks within a single neuron. This means that the current output of the network is dependent on its previous outputs. The ability to define current output in terms of previous output has given recurrent neural networks an edge when it comes to processing natural continuous data e.g. speech recognition and machine translation where there is a higher degree of dependency among input values.

Recurrent NNs, however, suffer from the problem of exploding and vanishing gradient as the sequence gets long [8]. The Long term short term memory (LSTM) which is a variant of Recurrent NN is used to address but can add significant overhead to the model. We deal with this challenge by introducing a sequence buffer connected to the Hopfield sequence model making it possible to store sizeable chunks of the sequences and retrieved only when required.

E. Recursive Hopfield mergence

This is the final module in the proposed model. The aim of the module is to combine associated weight matrices into one matrix. Based on the sequence generated by the Hopfield sequence block, this module creates a graph of Hopfield matrices using the principle of recursive NN.

Under the recursive NN framework, the perceived structure of a problem is captured and expressed using graphical models

[10]. These networks are similar to recurrent NN only that they have multiple branching.

In generating the graph, the header vector in each of the received matrix is used to determine whether to combine it with any other previous matrix. If an association with a previous matrix exists, the combining function is invoked otherwise the input is added to the graph and the next input sort until all the elements in the sequence have been included in the graph.

The merging of two Hopfield matrices works by adding all the elements of one matrix into another. For the newly formed matrix to be a valid Hopfield matrix, however, all neurons must be fully connected. As such, the elements that belonged to different weight matrices will have zero-valued synaptic weights between them.

The newly formed matrix represents an association that the normal simple Hopfield network could not achieve. This is like creating new knowledge from the already existing knowledge in form of disjoint Hopfield matrices.

III. MERGING MECHANISM OF HOPFIELD MODELS

Merging of different Hopfield models into one is the backbone of this study. This process broadens the recall abilities of the Hopfield models since they can recall patterns that they were not exclusively trained on.

In its basic structure, a Hopfield network consists of n interconnected neurons. From a set of x input vectors (learning data), the weight of the network is computed using the formula 1.

$$\boldsymbol{w = \sum_{n=1}^{N}(x_n)(x_n^T),\ \ w_{ii} = 0} \tag{1}$$

Where w is the weight matrix, N the total number of input vectors, x the actual input vector and T the transposition. The resulting matrix a zero diagonal since the weights connecting a neuron to itself is zero.

In Hopfield neural networks, the Hebbian learning rule is used to train the network [9]. This rule was proposed by Donald Hebb who held that learning takes place by reinforcing connections among the learned states. The maximum recall capacity of the network is 0.14N where N is the number of neurons [11]. Using the rule, the weight matrix is updated as shown in formula 2 where w is the synaptic weight, σ is the learning rate and p the pattern in its matrix form being learned.

$$W^{old} = W^{old} + \ \sigma\ (\mathrm{P}.P^T) \tag{2}$$

To recall a pattern P from a weight matrix W, the recall function shown in formula (3) is used. X represents the state of a neuron σ represents the learning rate and θ represents a threshold value.

$$X_i^{new} = \sigma\ (\sum_{i \neq j} W_{ij} X_j^{old} - \theta_j) \tag{3}$$

In the proposed model, the merged matrix will follow the conventional learning and recall rules. Suppose P and Q are two Hopfield matrices we want to merge as shown in formula (4):

$$p = \begin{pmatrix} 0 & a & b \\ a & 0 & b \\ b & b & 0 \end{pmatrix} \quad \text{and} \quad Q = \begin{pmatrix} 0 & m \\ m & 0 \end{pmatrix} \tag{4}$$

$$\begin{pmatrix} 0 & a & b & 0 & 0 \\ a & 0 & b & 0 & 0 \\ b & b & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & m \\ 0 & 0 & 0 & m & 0 \end{pmatrix} = \begin{pmatrix} 0 & a & b & 0 & 0 \\ a & 0 & b & 0 & 0 \\ b & b & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & m \\ 0 & 0 & 0 & m & 0 \end{pmatrix}$$

$M \qquad P \qquad Q$

Fig. 4. matrix merging

The following algorithm is used to merge the matrices in the model:

- Add P and Q
- Get the size of P and Q i.e. 3 x 3 and 2 x 2
- Add n and m zero elements to P and $Q \in$ new size P= $(3+n)\ X\ (3+n)$ and $Q = (2+m)\ X\ (2+m)$ AND $(3+n)\ X\ (3+n) = (2+m)\ X\ (2+m)$
- Sum P and Q

Following the above algorithm, the merged matrix M of P and Q will be as shown in Fig. 4. M can be used to recall patterns stored in both P and Q as well as a new pattern that is a hybrid of P and Q.

IV. HYPOTHETICAL USE CASE

To demonstrate the feasibility of our proposed associative memory and recall model, in this study, we use a case scenario which comprises of 3 initial hypothetical weight matrices. As shown in Fig. 4, Hopfield networks are trained to remember different aspects of an object (i.e. red apple, green apple and spheroid tennis ball).

A. Knowledge Acquisition and Hopfield Augmentation

Each resulting Hopfield matrix from the Hopfield networks shown Fig. 5 is initially received by the Hopfield augmentation module of the proposed associative memory model. This module annexes a header vector to the Hopfield matrix. The header vector indicates the concept in which the matrix belongs to according to the entries in the concepts look-up table. As shown in Fig. 6, the case scenario has three concepts on its look-up table namely fruit, shape and color. The header generator annexes a vector on each of the case scenarios Hopfield matrixes as shown in Fig. 6.

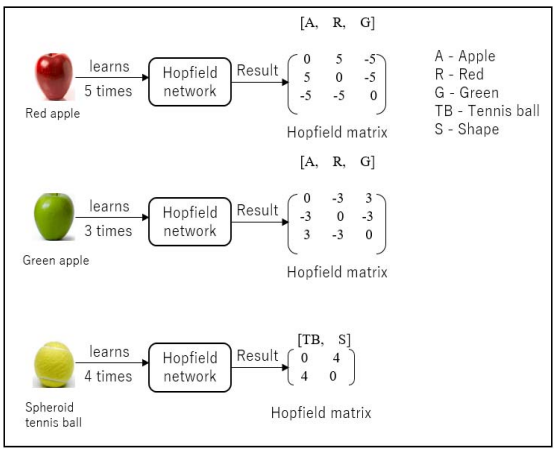

Fig. 5. Hypothetical case scenario

Since there are only three concepts in the concept look-up table, the header vector has three elements each corresponding to one concept. According to the look-up table, the first and second Hopfield matrices belong to the fruit and color concepts while the third matrix belongs to the shapes concept.

B. Association Finder

In the association finder, the header vector of each Hopfield matrix is updated according to the relationships entries in the association look-up table. The vector is analyzed to determine to which concept the Hopfield matrix belongs. The read value is used in the look-up process of the association look-up table.

According to Fig. 7, there is an association between the concept fruits and colors as well as fruits and eatable. The shape concept is however not associated with any other concept. Lack of association between one matrix to another is captured by editing the header vector elements corresponding to other associations with a value -1 for each. The existence of an association is captured by not altering the header vector elements corresponding to the related concepts.

In addition to capturing associations in the header vector, the timestamp of each Hopfield matrix is also added to the header vector. The timestamp is added as the last element of the header vector. As shown in Fig. 7, the timestamp element has been implemented as a counter value which increments by 1 after every new input.

The association finder module then sends the Hopfield matrix together with its updated header vector to the Hopfield sequence module.

C. Hopfield networks sequence

This module uses a recurrent NN to create a sequence of the incoming Hopfield matrices together with their header vectors.

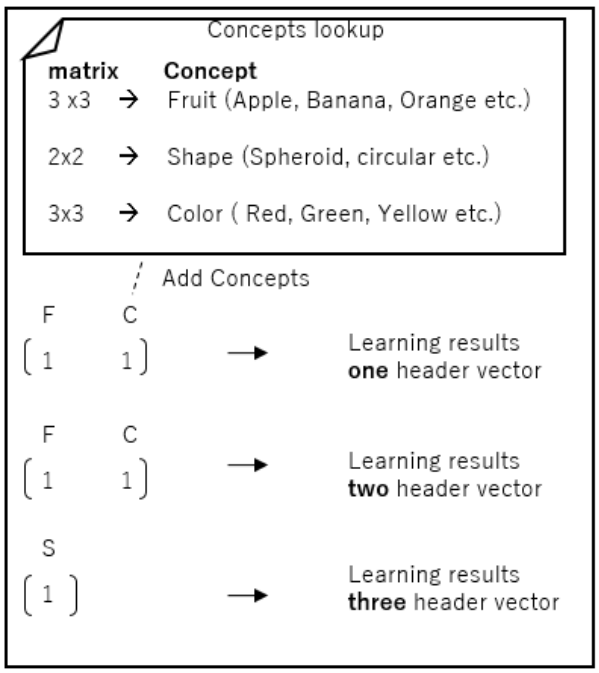

Fig. 6. Generated header vectors

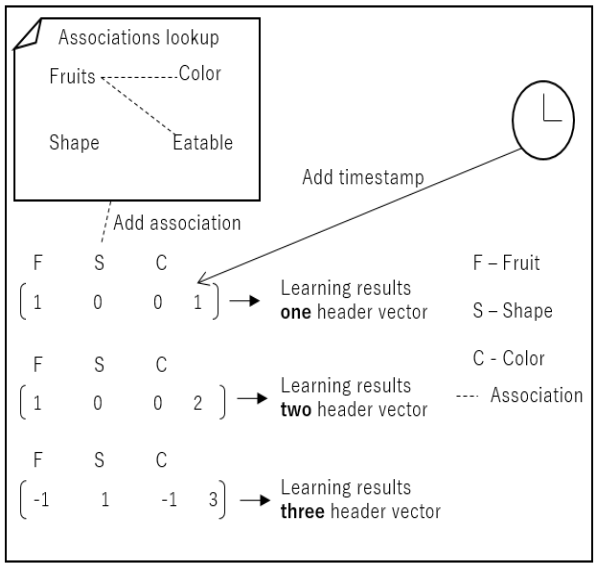

Fig. 7. Associations establishment and timestamp addition

The module is trained using the frequencies of simulation i.e. 5, 3 and 5 respectively as shown in Fig. 5. The state of the recurrent NN is stored in the sequence buffer to increase the memory available to the network. As such, the sequence buffer does not store the actual hypothetical matrices but the weight matrices of the recurrent NN trained to output the sequence of the Hopfield matrices. The sequence of the Hopfield matrices together with their header vectors are then sent to the Hopfield mergence module.

D. Recursive Hopfield mergence

The recursive Hopfield mergence uses the received sequence of the Hopfield matrices to create a graph. The Hopfield matrices are linked to each other and merged to form a bigger matrix if they are associated. With each new addition to the graph, the header vector is checked for the existence of an association with any of the previous learning results. Fig. 8 shows how the merging of associated Hopfield matrices occurs.

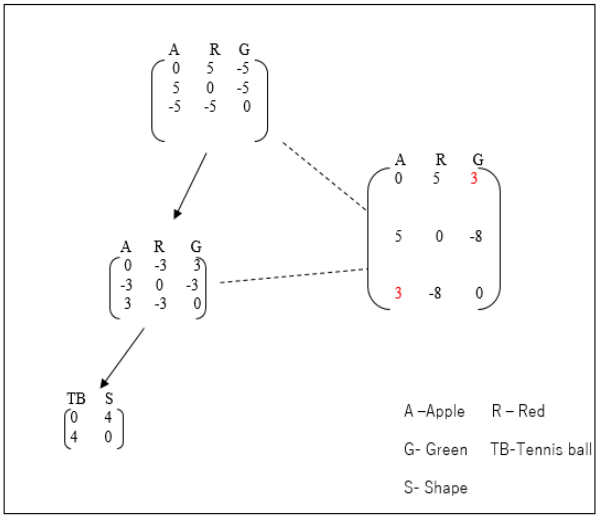

Fig. 8. Recursive Hopfield mergence

The matrix with no association to the other matrices is appended to the graph and the next input sort from the Hopfield sequence. The resulting weight matrix represents an association between different concepts. In this hypothesis, the matrices representing color and fruits are merged to form a fruit-color matrix.

These merged weight matrices represent a Hopfield neural network capable of detecting more knowledge since it now represents more concepts.

The graph is the passed to the mergence structure unit. The structure unit saves the graph in the associative memory repository.

V. Proof of Concept

We implemented a prototype of the proposed model using the Python programming language. Each module of the proposed system was implemented as a python method. The first method was an implementation of the Hopfield neural networks and uses the algorithm shown below:

- Create zero Hopfield matrix *M*
- *Input*: pattern vectors *P*:
- **for each** pattern $p \in P$ ***do***
  - Multiply transpose pattern p^T with *p i.e.* $p^T x\ p$ to get M'
  - $M =+ M'$
  - Append a vector containing the concept value [c]
- **end for**
- Return *M*

Given pattern inputs in form of n x 1 vectors, the class returns the weight matrices of size n x n (n being the number of elements in the pattern vector). To perform the dot product of the patterns, Numpy library is utilized.

The next method is designed to augment the weight matrices by annexing header vectors to them. The method accepts an array of Hopfield matrices as its only parameter and uses the following algorithm:

- Add header vector to Hopfield matrices *M.*
- **for each** $m \in M$ **do**
  - matrix order ← *m* size
  - Check the concepts look-up table for a concept corresponding to matrix order
  - *m* append ← concept vector [c]
- **end for**
- Return *M*

The size of each matrix is then determined and used to select a concept in the look-up table implemented as a Python dictionary. This method creates an array in which every even index contains the header vector of the immediate next odd index i.e. all header vectors are in the even indices of the array while all Hopfield matrices are in the odd indices of the array. This mechanism ensures the preservation of the original weight matrix. The header vector is initially a single element list containing the concept value of the matrix.

The association finder function takes in an array of augmented matrices with their corresponding header vectors as its only augment. The header vector is read to determine the concept to which a Hopfield matrix. The read value is used to retrieve associated concept. Each header vector in the array is updated with a new element indicating associations. In addition to adding associations, this function as adds a timestamp value to the header vector. The time is a counter value incremented at each iteration of the implemented loop. The function follows the algorithm shown below:

- Establish associations among Hopfield matrices
- *Input:* Augmented Hopfield matrices
- **for each** matrix ∈ input:
  - correlate header concept with concepts in the associations look-up
  - **if** associated concept found **then** write it in the header vector
  - Append current timestamp to the header vector
- **end for**
- Return updated Augmented Hopfield matrices

The next method is an implementation of the Hopfield sequence module. It is an implementation of a 2-layer neural network. The method first updates each header vector with a frequency element indicating the number of times a Hopfield matrix has been stimulated. This function updates a file each time a matrix is simulated. The neural network is then trained using the frequencies of all the matrices. The output of the network is *m* vectors of m x 1 size where m is the number of Hopfield matrices currently in the system. Each vector is zero value except one true value indicative of the sequence order of a given matrix.

The last method in the prototype uses the output of the neural network method to generate a recursive graph. It is implemented in form of a loop through an array reading the header vector in each iteration to check for associations. In instances where associations exist, a mergence function is invoked to merge the associated matrices. The function is implemented using Numpy array's concatenate function to merge the matrices. It follows the following algorithm:

```
mergence(matrix_associated)

[[ 0. -2.  2. -2.  2. -2.  0.  0.  0.  0.  0.]
 [-2.  0. -2.  2. -2.  2.  0.  0.  0.  0.  0.]
 [ 2. -2.  0. -2.  2. -2.  0.  0.  0.  0.  0.]
 [-2.  2. -2.  0. -2.  2.  0.  0.  0.  0.  0.]
 [ 2. -2.  2. -2.  0. -2.  0.  0.  0.  0.  0.]
 [-2.  2. -2.  2. -2.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  1. -1. -1.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.  1. -1. -1.]
 [ 0.  0.  0.  0.  0.  0.  1.  1.  0. -1. -1.]
 [ 0.  0.  0.  0.  0.  0. -1. -1. -1.  0.  1.]
 [ 0.  0.  0.  0.  0.  0. -1. -1. -1.  1.  0.]]
```

Fig. 9. Hopfield mergence function

- Create merged matrices
- *Input:* Augmented Hopfield matrices
- **for each** matrix with an association **do**
  - Create merged matrix *m x m* such that this matrix equals the sum of the individual sizes of the matrices.
  - Add individual associated matrices to the merged matrix.
  - Concatenate the header vectors of the associated matrices to form a header vector for the newly merged matrix.
- **end for**
- Return merged matrix

For the mergence to be valid, however, the associated matrices are first made of the same row order (i.e. ***m*** *x n and* ***m*** *x p*) by adding new zero valued elements until they have exactly the same row order.

When the prototype is run using two patterns such that vector [1, -1, 1, -1, 1, -1] represents an apple, vector [-1, 1, -1, 1, -1, 1] represents a banana and vector [1, 1, 1, -1, -1] represents red color, two Hopfield matrices are formed. The first matrix is a combination of apple and banana and the second matrix contains the color red pattern. Both matrices are assigned concept 0 and 1 representing fruit and color concepts respectively.

Both inputs are associated with the association method which updates the header vector of each with the concept value of the other. The Hopfield mergence function combines both matrices giving rise to one bigger matrix as shown in Fig. 9. Since this bigger matrix is a combination of two concepts i.e. fruit and color it can be called fruit color. The fruit color matrix can be queried with the new pattern red apple represented by the vector [1, -1, 1, -1, 1, -1, 1, 1, 1, -1, -1] and retrieve it.

VI. Related Work

There are some research works related to building human-like associative memory models. In particular, recently with the increasing popularity of neural networks and deep learning, there have been some approaches as described below aimed at eliminating catastrophic forgetfulness in learning models and most of them are a recurrent neural network (RNN) or revised RNN based mechanisms. None of the approaches analyzed below employed simple associative memory models like Hopfield networks. We, however, believe that the use of such simple models could prove powerful through reducing model complexities that hinder their optimal performance.

A. Elastic weight consolidation (EWC)

Elastic weight consolidation (EWC) is an algorithm proposed by K. James et al [12]. This algorithm borrows heavily from the inner workings of a mammalian brain which was discovered to be retaining new skills through the strengthening of the synapses associated with that acquired skill. To incorporate this idea into their model, these researchers designed the model such that weights were preserved during learning.

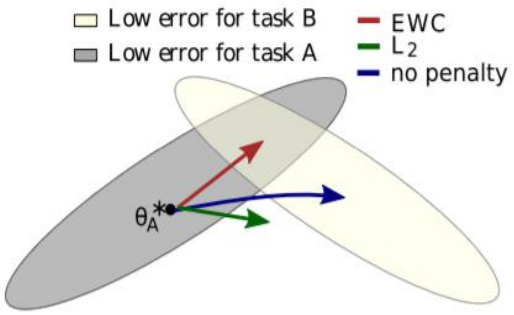

Fig. 12. EWC model shows two planes representing task A and B [12].

To perform a given task A for examples, weights were preserved as much as possible during the learning of another task B. In order to prove this approach was viable, these researchers chose to view the training of a neural network in probabilistic perspective.

Particularly, the conditional probability is used to justify which weights are important to a given task. Fig. 10 illustrates how the weights can be adjusted along different directions when learning a new task. The EWC model works by adjusting its weights when learning a new task along a direction where the previous task plane overlaps with the current task plane.

We borrow the EWC concept of weight preservation in our associative memory and recall (AMR) model. However, unlike the EWC model, in our model, the weights associated with a given task are not altered at all. We achieve this in our model by creating a recursive graph using the idea of a recursive neural network that grows with each new addition to the tree.

B. PathNet

PathNet is a learning algorithm proposed by C. Fernando et al [13] of Google DeepMind. These researchers were of the view that it would be more efficient to train one giant neural network that allows for parameter re-use while avoiding catastrophic forgetting. PathNet has been designed to use agents embedded in a neural network whose main functions is to determine which parts of the networks to re-use when learning new tasks.

It suffices to say therefore that PathNet is a deep neural network consisting of several modules each working autonomously. This structure enables the modeling of different scenarios making the larger network capable of more sophisticated tasks. Our proposed model is similar to this framework since it combines the weight matrices of associated Hopfield networks thereby creating a giant neural network. The difference however is, PathNet uses a static giant network, and our model is dynamic because changes in size with every new learning result inputted into the model.

C. Defferentiable neural computer (DNC)

Differentiable neural computer (DNC) is a machine learning model proposed by A. Gravels et al [14]. This model tries to mimic conventional computing where memory is separated from the processing unit. The model consists of a neural network with an external memory thereby allowing the neural network to learn and remember as many tasks as the external memory can hold.

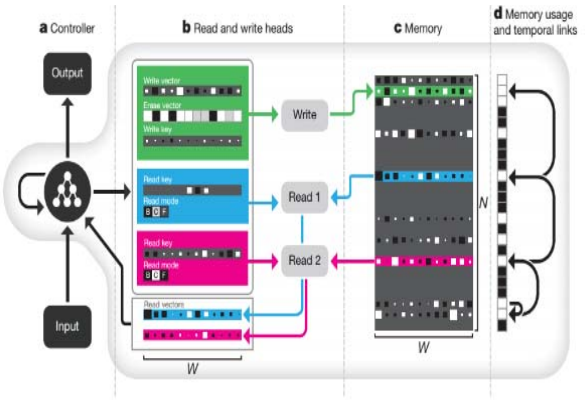

Fig. 13. Differentiable neural computer (DNC) [13].

An RNN is adopted for building the controller to give either a read or write request to the read/ write heads. To ensure that the memory remains differentiable after operation of the control, the read head reads from all the memory locations only to different extents.

Similarly, the write head writes to all memory locations just to different extents. Writing to or reading from every memory location can, however, add significant overhead to the overall performance of the system. We borrow the concept of external memory from this model but our memory does not require reading or writing everywhere in memory for each time step. Fig. 11 shows a schematic view of a DNC.

VII. Conclusion and Future Work

It has been generally affirmed to improve abilities of neural networks, an efficient memory and recall manager is inevitable. In this study we have shown how simple Hopfield neural networks can be leveraged on to create more sophisticated associative memory. Doing this not only creates high-level associations but also catalysis the derivation of new knowledge. It is our believe that this model goes a long way in ensuring learning model are enhanced by creating high-level associations among them that would otherwise not be possible on a single model. As part of our future work, we intend to extend the functionality of the merging mechanism of the proposed model. Such an expansion would allow the merging and recall processed to rely on recentness and relevance rather than frequency of simulation only.

Acknowledgment

This work is partially supported by the Japan Society for the Promotion of Science Grants-in-Aid for Scientific Research (No. 25330270 and No. 26330350). We thank Japan International Cooperation Agency (JICA) for supporting this study through the ABE initiative program.

References

[1] D. Hof. (2017, Jan). Deep Learning With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart [Online]. Available: https://www.technologyreview.com/s/513696/deep-learning/

[2] Y. Lecun, Y. Bengio and G. Hinton, Deep Learning, 3rd ed., vol. 521. Internatoonal Weekly Journal of Science, May 27, 2015.

[3] S.Raschka, Python Machine Learning, 1st ed., Livery Place 35 Livery Street Birmingham B3, 2PB, UK, 2015, pp. 1-276.

[4] S. Legg and M. Hutter, A definition of machine intelligence, 3rd ed., vol. 521. Universal intelligence, 2007, pp. 391-444.

[5] I. Goodfellow,Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016

[6] J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, vol. 79. California Institute of Technology, April, 2 1982, pp.2554-2558.

[7] E. Thierry. (2010). Hopfield Network [Online]. Available: http://perso.ens-lyon.fr/eric.thierry/Graphes2010/alice-julien-laferriere.pdf.

[8] M. Nielsen. (2017, Jan). Deep learning [Online]. Available: http://neuralnetworksanddeeplearning.com/chap6.html

[9] A. Chinea, Understanding the Principles of Recursive Neural Networks: A Generative Approach to Tackle Model Complexity, Departamento de Física Fundamental, Facultad de Ciencias UNED, 2007, pp.1-11.

[10] C. Chuang, "Investigation of Synaptic Plasticity as Memory Formation Mechanism and Pathological Amyloid Fibrillation Caused by β-amyloids Aggregation," Ph.D. dissertation, Dept. Chemical. Eng., MIT., Cambridge, MA,2008.

[11] M.P.Singh, S.S.Pandey, V.Pandey. Performance Evaluation of Hopfield Assocative Memory for Compressed Images, Elk Asia Pacific Journal of Computer Science and Information Systems, vol 1, 2015, pp 13-31.

[12] K. James et al, Overcoming Catastrophic Forgetting in Neural Networks, 3rd ed., vol. 2. National Academy of Sciences of the United States of America, Feb 13, 2017, pp.1-6.

[13] C. Fernando et al, Evolution Channels Gradient Descent in Super Neural Networks, arXiv:1701.08734 [cs.NE], Jan 30 2017, pp. 1-16

[14] A. Graves et al. Hybrid Computing using a Neural Network with Dynamic External Memory, International Weekly Journal of Science, Oct 27 2016, pp 1-21.