Towards developing permanent memory system for artificial intelligence to acquire new knowledge after deployment

Kazunori D Yamada[1,2,*]

[1] *Unprecedented-scale Data Analytics Center, Tohoku University, Sendai, Japan*
[2] *Graduate School of Information Sciences, Tohoku University, Sendai, Japan*

Artificial intelligence that accumulates knowledge and presents it to users is likely to become a good friend to humanity. It is thought to be useful for such artificial intelligence to have the ability to continuously accumulate knowledge while operating as artificial intelligence, rather than having to relearn when acquiring new knowledge. Artificial intelligence that can share past knowledge with users is thought to be a human-like existence. Some dialogue agents developed so far can partially use the memory of past conversations. Such memory is sometimes referred to as long-term memory. However, the ability of existing dialogue agents to continue accumulating knowledge and share that knowledge with users is likely not sufficient at present. Within the framework of their current implementation, their memory capacity is at least finite. It is thought necessary to develop a method to improve this memory capacity, in order to realize memory to store any information in a medium of finite size. This might be a kind of research on data compression. Also, in the current method of developing dialogue agents by repetitively using human conversation data as training data, there is some doubt as to whether or not artificial intelligence can effectively utilize the memory they have acquired in the past. In this study, we demonstrate the impossibility. In contrast, this research proposes a method to store various information in a single finite vector data and retrieve it, utilizing a neural network with a structure similar to a recurrent neural network and encoder–decoder network, which we named mnemonist. This study only yielded very preliminary results, but it is fundamentally different from the conventional learning method of neural networks, and is a fundamental consideration for building artificial intelligence that can store any information in finite size data and freely utilize them in the future.

KEYWORDS: learning with random values, neural networks, long-term memory, permanent memory

1 Introduction

The grand challenge of artificial intelligence research is to construct the strong artificial intelligence or artificial general intelligence. The journey towards the realization of such artificial intelligence is long, and in the process, we will likely need to solve many problems. The study of endowing artificial intelligence with consciousness, that is, the study of artificial consciousness, is one such problem [1], and the construction of entities that possess physicality in the real world is another such problem. Furthermore, an ability to learn indefinitely, without the need for a defined learning process like that found in current machine learning field, may also be necessary. Such artificial intelligence would be capable of continuing to learn even after being deployed, and constructing this kind of artificial intelligence is one of the grand challenges in the field of dialogue agent research [2].

Artificial intelligence that can continue to learn even after being deployed is thought to need at least the following abilities, that is, the ability to acquire new knowledge and use it according to the situation:

(1) The ability to retain the information that has been supplied so far as some kind of memory data.
(2) The ability to add the newly supplied information to that memory data without compromising the information that has already been memorized.
(3) The ability to freely extract necessary data from the memory data.

Generative artificial intelligence, particularly large language models, sparked a boom in 2022. The origin of this trend, ChatGPT, a dialogue agent developed by OpenAI, is capable of having a conversation based on a certain amount (8192 tokens) of past conversation memory. Since the information input into ChatGPT is used in subsequent conversations, it would not be an exaggeration to say that ChatGPT is an entity that continues to learn while operating as artificial intelligence. Also, BlenderBot 2 developed by Meta, and its successor model BlenderBot 3, are dialogue agents that can extract knowledge from past conversations, hold that knowledge, and have conversations based on it. BlenderBot 2 has a component called “long-term memory” in its structure. This component holds information that summarizes past conversations. In response to user input, it also uses the information stored in this long-term memory to make its response. As with ChatGPT, the information remembered is also finite.

In human conversation, people sometimes may forget the content of the conversation they have had. If we are looking for human-like traits in artificial intelligence, artificial intelligence with finite memory may partially meet that demand.

*Corresponding author. E-mail: yamada@tohoku.ac.jp

However, people do not completely forget the content of a conversation from a certain time ago and remember the content of a conversation after a certain time, so the dialogue agents mentioned earlier may not be human-like in that respect.

Furthermore, we should consider whether or not we should demand human-like or imperfect traits in certain abilities of artificial intelligence, such as memory power. As we mentioned earlier, the grand challenge of artificial intelligence research is to build the strong artificial or artificial general intelligence, but should all the abilities of strong artificial intelligence or artificial general intelligence be human-like? In the paper where the term "strong artificial intelligence" was first used [3], strong artificial intelligence is described as an entity with "spirit." This spirit is human nature itself. If strong AI should be human-like, it should have a function that is the very human-likeness called spirit, and at the same time, the human-likeness of strong artificial intelligence should be provided by a function called spirit. It is not necessary for other abilities different from spirit to be human-like, and in some functional aspects, artificial intelligence may well be a "machine-like" existence. Artificial intelligence that does not lose memory like human and can provide knowledge freely without forgetting any memorized information like machine would be good friend to humanity. The aforementioned BlenderBot 2 has long-term memory, but it is not machine-like so far. Perhaps machine-like artificial intelligence should have a device that can memorize any information. Here, we call such memory that can memorize any information "permanent memory." In this study, we will discuss how to realize such permanent memory.

Additionally, this study will consider the machine learning process for implementing such features. There is a suspicion that methods attempting to extract information using medium like long-term memory may not be suited to the learning methods of neural networks. When trying to construct a dialogue agent, we repeatedly train an untrained model with pairs of question sentences and their corresponding response sentences. The same is true for generative language models like the Generative Pretrained Transformer (GPT). For example, it is thought that if a model is trained repeatedly using conversation pair data to accept a question sentence and long-term memory as input values and return some response sentence, the model will stop paying attention to long-term memory information as learning progresses. This is because we think that a model that has undergone training will be optimized to the training data and thus be able to return an appropriate response to a question sentence without referring to the information in long-term memory. Using information that can only exist in external memory like long-term memory to generate a response from the artificial intelligence might be nothing more than a fantasy within the current framework of machine learning.

To clarify this, we conduct an experiment. In this experiment, we will perform encoding and decoding operations. Therefore, we will use two machine learning models, which we will call the "memorizer" and the "recaller." We will refer to the encoding operation as the "memorization" operation, and the decoding operation as the "recall" operation from here. Initially, in the memorization operation, we generate several vectors combining a key vector consisting of random numbers of a certain length and a value vector consisting of integers from 0 to 9, and repeatedly train the memorizer with this information to generate a vector, which is expected to contain all the input information. We will call this vector created by the memorization operation a "memory vector." In the following recall process, we use one of the key vectors used in the memorization process and the memory vector obtained as a result of the memorization operation as inputs to the recaller, and train it to output a value, which corresponds to a key of key–value pair. During the test phase, we feed multiple key and value vectors different from those used during training to the trained memorizer. Next, we observe whether or not the recaller can retrieve and output a correct value, which corresponds to a key, by using only the key vector and the memory vector as input data.

The vector we call a memory vector in this experiment corresponds to the previously mentioned long-term memory. If the recaller can effectively utilize the memory vector, by using the memory vector generated by inputting a key and value vector not included in the training dataset into the trained memorizer, the recaller should be able to output a value corresponding to the key. However, the actual result is likely not to work out. This is because a recaller that has undergone sufficient training with a finite training dataset is a model that is optimized to and merely memorizes the combination of a key and value composed of random numbers and is simply a artificial intelligence for just outputting a value corresponding to the key without function to utilize the memory vector.

In this study, we will conduct this experiment and demonstrate that this learning method does not work well. We also propose a method that allows for such learning to progress successfully by making the effort not to train the model with the patterns inherent in the input data during the learning process. Ultimately, we propose the construction of a system that can memorize multiple pieces of information independently into a memory vector and can extract information freely from it, using the proposed learning method.

2 Experiments and Discussion

2.1 Network Architecture

What we aim to achieve in this study is to train the memorizer and recaller so that they can store N pairs of key and value vectors into one memory vector using the memorizer, as shown in Figure 1, and then output the corresponding value from that memory vector and one key vector using the recaller. In other words, we perform the task of restoring complete information from the key using the memory vector. We call this combination of memorizer–recaller network a mnemonist.

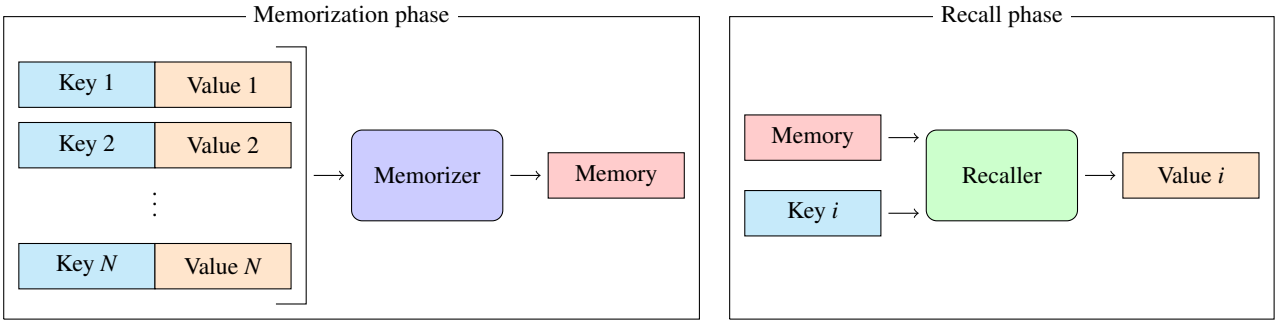

Fig. 1: What we compute in the memorization phase and the recall phase. In the memorization phase shown in the left panel, we use a total of N pairs of key and value combinations in no particular order as inputs to the memorizer, resulting in one output value, the memory vector. In the recall phase shown in the right panel, we use the memory vector generated in the memorization phase and a key, which is part of the input information from the memorization process, as inputs to the recaller to output the value information, which corresponds to the key vector. In the figure, rectangles represent data, and rectangles with rounded corners represent some operation.

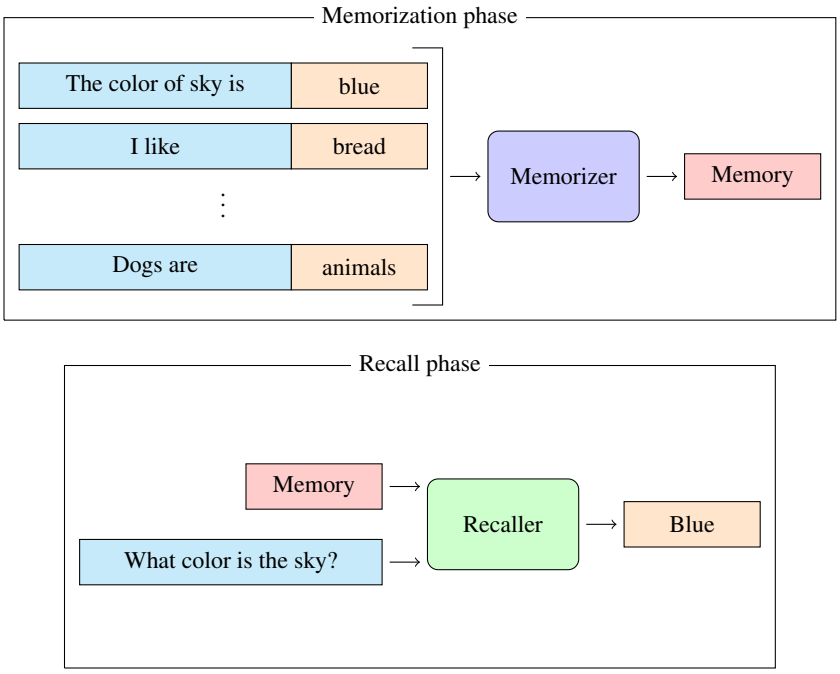

Fig. 2: The example of how memorizer–recaller network could be used to make a dialogue agent accumulate memory. In the diagram, rectangles represent data and rounded with rounded corners represent operations.

This process, using a dialogue agent as an example, can be explained as shown in Figure 2. The combination of key and value corresponds to a statement by an user, and the memory vector represents accumulation of these statements stored in the artificial intelligence. The key represents some piece of information, and the value is an explanation for that information. The role of the recaller is to remember the value corresponding to a key based on its memory. If a artificial intelligence can perform this behavior, it would be able to accumulate knowledge from users or even web searches, and grow without changes that involve updating the parameters of the neural network after learning.

The operations performed by the memorizer–recaller network can be achieved with a key-value store in cases where the key and value are a simple combination, and there is no limit to memory capacity. However, here the size of the memory vector is fixed, and the size of the data to be memorized is also an arbitrary N. Moreover, if we assume natural language as the input value, the key and value are not always in this order, and instead of the tandem structure of key–value, the value may also be located in the middle of the key vector. We cannot handle these kind of data with a simple key-value store.

Furthermore, the operation we want to perform with this memorizer can be realized with a neural network equipped

with an attention mechanism (ANN) [4] or recurrent neural network (RNN). ANNs and RNNs can receive sequential values as input and output values, considering past context information. The operation we want to perform with the memorizer is similar, but one difference of the memorizer from the usual ANNs or RNNs is that there is no teacher data for the memory vector, during the learning of the model.

In this study, we used two neural networks. The first model, the memorizer, is represented by the following formula:

$$u_t \leftarrow f(k_t, v_t), \tag{2.1}$$

$$p \leftarrow \sigma(w_1 u_t + b_1), \tag{2.2}$$

$$q \leftarrow \sigma(w_2 m_{t-1} + b_2), \tag{2.3}$$

$$m_t \leftarrow \sigma(w_3(p+q) + b_3), \tag{2.4}$$

In this paper, the left arrow ($\leftarrow$) is a symbol that assigns the value on its right to the variable on its left. Here, w is the weight parameter of the neural network, b is the bias parameter of the neural network, and σ is the activation function. We used LeakyReLU as the activation function. In addition, f is a function that concatenates two vectors in series, t is an integer from 1 to N, k_t is the t-th key, v_t is the t-th value, and m_t is the t-th memory vector. In the memorizer, the $t-1$-th memory data is used to generate the t-th memory data, and thus the memorizer is a kind of RNN. The variable p in the final formula is a value derived from the input data, while q is a value derived from the $t-1$-th memory vector. Because we need to calculate the element-wise sum of p and q, these vectors need to have the same dimensions. During the memorization phase, we compute this over all N data to finally obtain m_N. The second model, the recaller, is represented by the following formula:

$$r \leftarrow \sigma(w_4 k_i + b_4), \tag{2.5}$$

$$s \leftarrow \sigma(w_5 m_N + b_5), \tag{2.6}$$

$$\hat{v}_i \leftarrow f(r, s), \tag{2.7}$$

$$\hat{v}_i \leftarrow \sigma(w_6 \hat{v}_i + b_6), \tag{2.8}$$

$$\hat{v}_i \leftarrow \phi(w_7 \hat{v}_i + b_7), \tag{2.9}$$

where, ϕ is the activation function, and because this recaller is used to classify and predict values from 0 to 9, we used the softmax function. In addition, k_i is any one of the N keys used as input to the memorizer. In the recall phase, it attempts to output $\hat{v}_i$, which is the value corresponding to k_i.

Figure 3 illustrates this neural network. In the figure, the structure on the left is the memorizer, and the one on the right is the recaller. The memorizer accepts pairs of keys and values as input and outputs a memory vector. By repeating this calculation for all input values, it ultimately generates memory vector that is expected to contain all input information. The recaller accepts a single key and the memory vector generated by the memorizer as input and outputs the value corresponding to the key. What we want to ascertain using the memorizer–recaller network is whether a trained neural network can realize the storage and retrieval of multiple pieces of information, as shown in Figure 1, without requiring relearning. Since we do not pursue enhancing performance of the system, we did not include any luxury mechanisms, which are generally used to improve the performance of neural networks, and designed it with a very simple combination of layers, as shown in the figure.

2.2 Learning Process of the Models

2.2.1 Learning in a Standard Manner

Next, we trained the model. First, we explain the hyperparameters used for training. The size of the intermediate layer was set to 256 for both the memorizer and recaller models. Similarly, the dimension of the memory vector, which is the output of the memorizer, was set to 256. Since the recaller is a predictor that outputs values that are integers from 0 to 9, the dimension of its output vector was set to 10. In addition, the dimension of the key vector was set to 16. Each element of the key vector is a floating point number randomly generated from a continuous uniform distribution following parameters with a minimum value of 0 and a maximum value of 9. The value vector is a one-dimensional vector, and its only value is an integer that follows a discrete uniform distribution with a minimum value of 0 and a maximum value of 9. The number of input data to be inputted into the memorizer is N. For training, 1024 combinations of these N input data were generated and used in the batch learning method. During the learning of the memorizer, the value of t is changed from 1 to N, the memory vector m_t is calculated, and finally m_N is obtained. For m_0 at this time, random numbers generated from a uniform distribution following parameters with a minimum of -1 and a maximum of 1 were used. Since the softmax function was used in the output layer, the cross-entropy error function was used as the cost function. Parameter updates were performed when the memorization process for N data was completed, m_N was used as its output, each of the N keys was inputted into the recaller to output the value, and the error between the value and the correct data was calculated. Adam was used for parameter optimization.

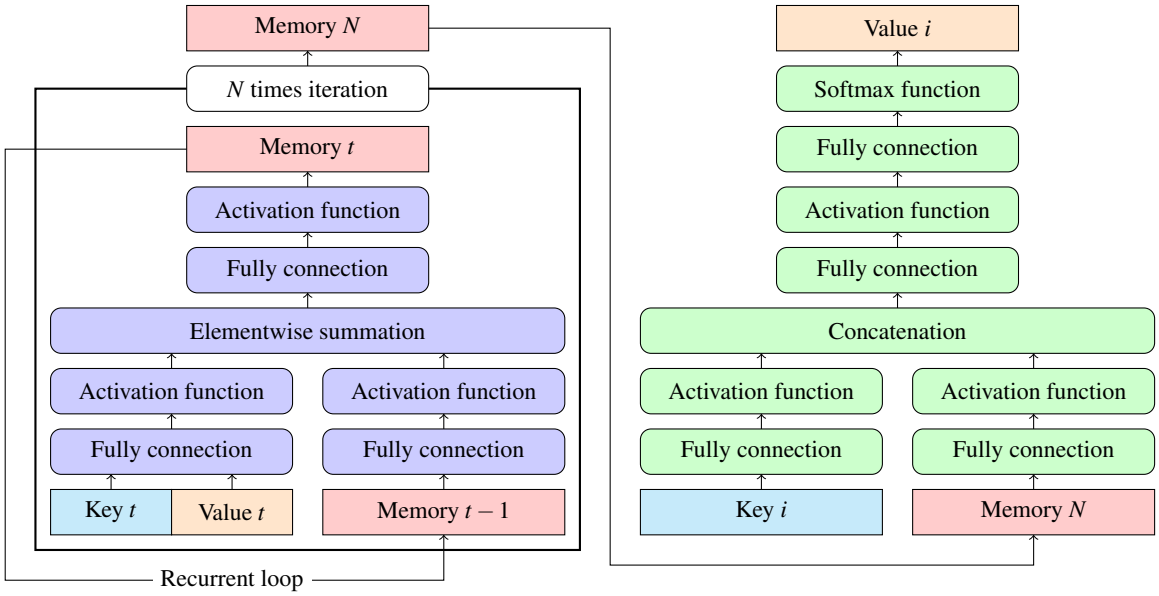

Fig. 3: Illustration of the memorizer–recaller network. The network on the left is the memorizer, which accepts N pairs of keys and values as input and outputs a memory vector. The recaller on the right accepts a single key and the memory vector as input and outputs the corresponding value. The rectangles in the figure represent data, and the rectangles with rounded corners represent some operation.

N	Epoch	Training accuracy	Validation accuracy
2	550	0.802	0.233
3	650	0.802	0.168
4	820	0.800	0.146
5	1000	0.801	0.139
6	1100	0.801	0.120
7	1510	0.800	0.118
8	1470	0.801	0.113
9	1720	0.800	0.111

Table 1: Results of training with a varying the number of input data N. Training was terminated when the accuracy for the training dataset reached 0.8. Epochs represent the learning time required to reach this point.

The results of the training are shown in Table 1. In the training, the training dataset and the validation dataset were randomly generated. In other words, there is no discernable rule or pattern that the machine learning model might discover in each of the data input into the model. Therefore, in order to derive the correct answer in the validation process, it is necessary to refer to the memory vector generated in the memorization process. Training was terminated when the accuracy in the training process reached 0.8 or more. Table 1 displays the number of epochs required until then and the accuracy in the validation dataset. As a result of the training, the accuracy in the training process reached 0.8, but on the other hand, the accuracy in the validation process was low, and we were not able to build a good model.

The reason why a good model could not be built through the training process is that the model overfit the training dataset and adapted excessively to it. In other words, this model learned the characteristics of the training dataset consisting of random elements, so it could not adapt to the validation dataset, which has no correlation and does not share any characteristics with the training dataset. Although the recaller should ideally refer to the memory vector, which is a condensed medium of the information inputted into the memorizer, to derive answers, it is merely attempting to output something based on the pattern of the randomly generated vector that it takes as input. In other words, in a learning task like this one, where the model is asked to predict output values corresponding to input values, it is thought to be impossible to incorporate the function of referencing the memory vector, as it would be sufficient for the model to output values corresponding to the input values as it goes through more learning iterations.

N	Epoch	Training accuracy	Validation accuracy
2	5580	0.800	0.800
3	9200	0.800	0.800
4	13920	0.800	0.800
5	19800	0.800	0.800
6	30600	0.800	0.800
7	53140	0.800	0.800
8	95280	0.800	0.800
9	500000	0.259	0.259

Table 2: Results of training with a varying the number of input data N, where the training dataset is randomly generated for each epoch. Training was terminated when the accuracy for the validation dataset reached 0.8. Epochs represent the learning time required to reach this point.

2.2.2 Learning with Random Value

What we want to implement in the memorizer–recaller network proposed in this study is a "way" of organizing input values, remembering them, and a "way" of searching for necessary information from memory and recalling it. In the general machine learning training methods conducted in the previous section, model learns the nature of the data and tries to understand the laws inherent in the data. A large language model that has intensively learned text can be considered to have discovered some rules for language generation. In contrast, what we want to construct in this study is a system that can remember and recall information regardless of the scale of learning data or the domain from which the learning data originates. The word "mnemonic" means a memory technique. The model we want to realize in this study is not a system to find the laws inherent in the data, but a system to learn how to remember. Therefore, we named this memorizer–recaller network mnemonist.

Therefore, in this study, to prevent the model from recognizing patterns inherent in the data and to enable the model to learn how to behave, we removed patterns from the learning data. Specifically, we probabilized k_i, the input value of the memorizer and recaller, and v_i, the input value of the memorizer and the correct value of the recaller. During the learning process, each of the 16 elements of k_i was randomly generated from a continuous uniform distribution with parameters from 0 to 9 for each epoch. The value of the only element of the value vector was also generated from a discrete uniform distribution with a minimum value of 0 and a maximum value of 9. Although we are using randomly generated data in the learning process here, the aim is that random data does not have a tendency. We are trying to make the memorizer–recaller network learn not to read data and learn the tendency of the data, but to learn how to remember data and how to retrieve information from the memory.

The results of learning using this learning method are shown in Table 2. In this learning process, we ended the learning when the validation accuracy reached 0.8. Generally, the accuracy of the validation dataset is lower than the accuracy of the training dataset, and if this is not the case, it is considered that there may be a suspicion of abnormal learning settings. However, since both the training dataset and the validation dataset in this learning method are random values, if the learning progresses normally in the training dataset, the accuracy in the validation dataset also improves at the same time, so this behavior is normal.

We conducted training while varying the number of data N to be memorized from 1 to 9. Up until N became 8, accuracy reached 0.8 both in the training dataset and the validation dataset. In other words, the trained memorizer–recaller was able to derive the correct answer in a validation dataset by learning from a training dataset, where both the training and validation dataset was randomly generated and has no patterns. On the other hand, when N was 9, despite training the model by the epochs of 500000, accuracy did not reach 0.8, and which meant the learning process did not proceed successfully. In other words, the memorizer–recaller constructed under these learning conditions can only memorize at most 8 pieces of information.

Next, as a demonstration of the memorizer–recaller's performance, we sequentially inputted the following 8 sets of key and value data in this order into the memorizer to generate a memory vector. Afterwards, we conducted a test using all the keys to see whether the values could be retrieved using the memory vector and the recaller. The integers shown in bold represent the values.

(1) $8.75, 7.90, 4.59, 0.500, 4.06, 0.180, 3.98, 8.82, 3.24, 4.33, 6.20, 7.92, 8.26, 1.95, 5.09, 7.79, \mathbf{7}$
(2) $3.46, 2.68, 0.510, 2.45, 4.30, 7.31, 4.32, 3.54, 7.52, 3.04, 5.83, 3.31, 8.61, 1.26, 7.83, 4.26, \mathbf{5}$
(3) $3.48, 8.12, 4.05, 5.52, 8.12, 0.890, 8.73, 5.88, 1.54, 3.22, 6.76, 5.47, 2.93, 0.350, 5.71, 8.63, \mathbf{3}$
(4) $5.88, 5.72, 8.96, 5.24, 3.73, 4.27, 5.61, 3.04, 6.07, 2.85, 7.01, 8.55, 5.96, 0.120, 5.61, 6.06, \mathbf{3}$
(5) $2.82, 8.69, 5.30, 5.94, 4.80, 2.07, 3.55, 5.57, 4.27, 4.23, 6.44, 2.59, 3.45, 6.74, 7.91, 0.930, \mathbf{5}$
(6) $7.21, 4.68, 6.11, 6.49, 5.24, 4.84, 6.83, 0.950, 4.26, 1.68, 6.63, 1.95, 1.22, 2.92, 1.35, 2.00, \mathbf{0}$
(7) $4.58, 8.25, 8.29, 0.750, 2.50, 0.0800, 7.58, 5.82, 7.57, 2.38, 3.58, 4.98, 1.48, 3.33, 1.32, 5.13, \mathbf{9}$

The number of input data	Accuracy
2	0.967
3	0.938
4	0.941
5	0.931
6	0.926
7	0.912
8	0.916
16	0.510
32	0.301
64	0.204
128	0.151
256	0.126

Table 3: The accuracy of the memorizer–recaller, which was trained by setting N to 8, when processing different numbers of input values.

(8) $6.33, 2.60, 3.90, 6.80, 3.56, 8.06, 5.75, 8.02, 6.12, 4.04, 8.81, 1.05, 6.90, 3.71, 6.08, 2.25, \mathbf{3}$

The results of this test, in which we wrote the output values of the recaller next to the colon, are as follows. In this case, the recaller was able to retrieve the correct value for all keys.

- $3.46, 2.68, 0.510, 2.45, 4.30, 7.31, 4.32, 3.54, 7.52, 3.04, 5.83, 3.31, 8.61, 1.26, 7.83, 4.26, \mathbf{5} : 5$
- $7.21, 4.68, 6.11, 6.49, 5.24, 4.84, 6.83, 0.950, 4.26, 1.68, 6.63, 1.95, 1.22, 2.92, 1.35, 2.00, \mathbf{0} : 0$
- $3.48, 8.12, 4.05, 5.52, 8.12, 0.890, 8.73, 5.88, 1.54, 3.22, 6.76, 5.47, 2.93, 0.350, 5.71, 8.63, \mathbf{3} : 3$
- $5.88, 5.72, 8.96, 5.24, 3.73, 4.27, 5.61, 3.04, 6.07, 2.85, 7.01, 8.55, 5.96, 0.120, 5.61, 6.06, \mathbf{3} : 3$
- $8.75, 7.90, 4.59, 0.500, 4.06, 0.180, 3.98, 8.82, 3.24, 4.33, 6.20, 7.92, 8.26, 1.95, 5.09, 7.79, \mathbf{7} : 7$
- $4.58, 8.25, 8.29, 0.750, 2.50, 0.0800, 7.58, 5.82, 7.57, 2.38, 3.58, 4.98, 1.48, 3.33, 1.32, 5.13, \mathbf{9} : 9$
- $6.33, 2.60, 3.90, 6.80, 3.56, 8.06, 5.75, 8.02, 6.12, 4.04, 8.81, 1.05, 6.90, 3.71, 6.08, 2.25, \mathbf{3} : 3$
- $2.82, 8.69, 5.30, 5.94, 4.80, 2.07, 3.55, 5.57, 4.27, 4.23, 6.44, 2.59, 3.45, 6.74, 7.91, 0.930, \mathbf{5} : 5$

Furthermore, the mean accuracy when repeating such a test 1024 times was 0.916. This indicates that even in the tests, the memorizer–recaller network has shown good performance, as intended, able to remember multiple pieces of information in a single memory vector, and to retrieve information from it.

Next, we checked how the mean accuracy changes when the memorizer–recaller, built for $N = 8$, is made to process various amounts of data. In this experiment, we computed mean accuracy by repeating such a test 1024 times. The results, as shown in Table 3, demonstrate an accuracy exceeding 0.9 up to the number of data used for training, which is 8, but the performance dramatically drops when the number of data processed is increased beyond that. This problem is a 10-class classification problem, so the accuracy when answering randomly would be 0.1. When the number of data to be memorized is set to 256, the accuracy became nearly the same as when answering randomly.

There was a tendency of way how the memorizer–recaller network retrieve correct answer. The recaller was able to produce the correct output for the last 8 pieces of information memorized by the memorizer, and was unable to recall the data provided to the memorizer that was to be memorized earlier. For instance, when the input data consisted of 16 items, the memorizer–recaller showed an accuracy of 0.510, which appears to greatly exceed the accuracy of 0.1 when predicting randomly. However, this was not because the memorizer–recaller could derive the correct answer with an accuracy of 0.510 uniformly for the data to be memorized. The value of 0.510 was calculated as the average of about 0.9 accuracy in deriving the correct answer for the last 8 pieces of data input to the memorizer, and about 0.1 accuracy in deriving the correct answer for the first 8 pieces of data.

By the new method of learning devised in this study, it was possible to build a system that can remember multiple pieces of information in a single vector, despite being a trained model, and can retrieve them, although it is a small amount of just 8 items. The memorizer–recaller network is similar to an encoder–decoder network like a transformer in terms of neural network structure, and the computational processing in the memorizer is an extension of ANNs or RNNs. However, by effectively utilizing random data in the learning process and not allowing the same input data to be learned twice, we were able to achieve the goal of making the artificial intelligence remember the procedure to solve problems, rather than having it learn patterns embedded in data. As a result, the memorizer–recaller network has become able to continue learning after the learning process, which distinguishes this network from existing artificial intelligence. On the other hand, the memory capacity is extremely small, and thus the memory vectors generated by the memorizer are far from being called permanent memory. To improve this, a breakthrough is needed, not just ad hoc measures like increasing the size of the network parameters.

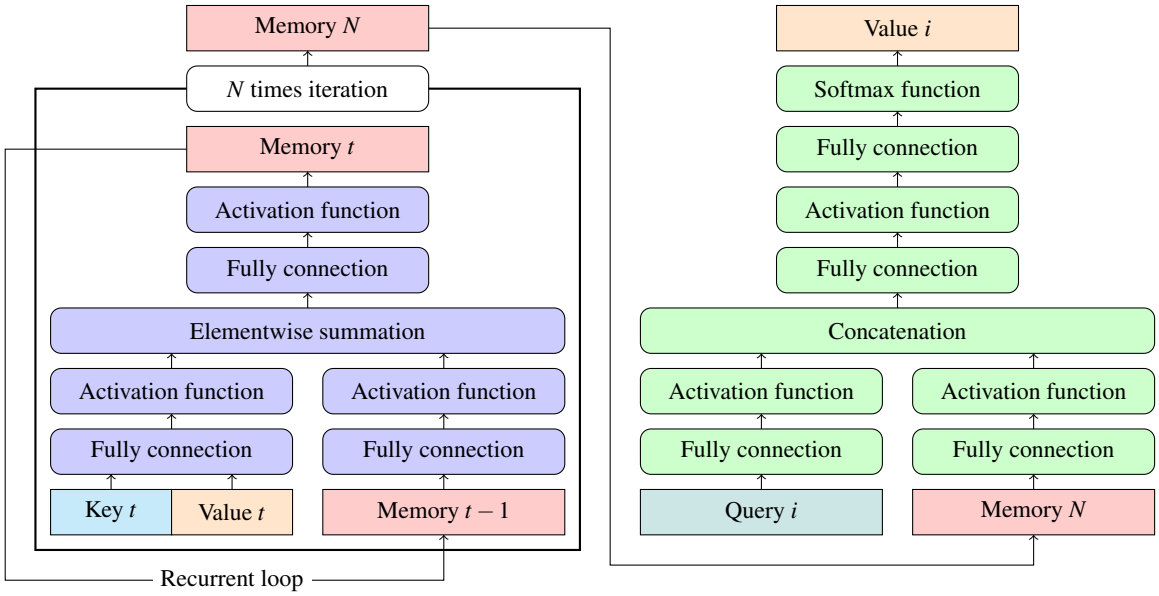

Fig. 4: Illustration of the network when generating a sorting algorithm with the memorizer–recaller. The network structure is the same as the previously mentioned memorizer–recaller network, but part of the input values of the recaller was changed to a query. In the figure, rectangles represent data, and rounded rectangles represent some operation.

2.3 Developing Sorting Algorithm

Next, we tried to see if a sorting algorithm could be generated by applying the memorizer–recaller network. In order to perform sorting, it is necessary for model to remember the information entered and determine its size. Since the memorizer–recaller is a system that can memorize input value information in a memory vector, we conducted this experiment, expecting that it could generate a sorting algorithm.

The structure of the network used for this experiment is as shown in Figure 4. Almost all of the structure is the same as the structure in Figure 3, but the input values of the recaller have changed from key to query. This query is an integer from 0 to $N-1$. Zero is a prompt to make the recaller output the smallest number among the N numbers input, one is a prompt to make the recaller output the second smallest number among the N numbers input, and $N-1$ is a prompt to make the recaller output the largest number among the N numbers input. By arranging these output results, you get the result of sorting the input values. Specifically, we used the following data as the input value of the memorizer. This is the case when N is 5. Also, the floating point numbers are the keys and the values we want to sort, and the bold characters are the values. The keys were generated from a uniform distribution with a minimum value of 0 and a maximum value of N.

(1) $3.35, \mathbf{A}$
(2) $1.05, \mathbf{B}$
(3) $0.640, \mathbf{C}$
(4) $1.58, \mathbf{D}$
(5) $1.82, \mathbf{E}$

For these input values, the recaller outputs as follows. The queries to the recaller in this case are 0, 1, 2, 3, 4 from the top to the bottom.

(1) C
(2) B
(3) D
(4) E
(5) A

The learning was conducted in the same way as described earlier. Learning was stopped when validation accuracy reached 0.95. By incrementally increasing the value of N from 2, we checked whether the learning could be completed normally. As results, it was possible for learning to complete normally within 500000 epochs up to N being 8. This number 8 was consistent with the results in the previous section. Although this value of N can vary slightly depending on the stopping condition of the learning, discussing whether this N value is 8, 7, or 9, etc. is meaningless, but it can be thought that about this extent of memory can only be retained by the memorizer–recaller network under current condition

of parameter size and architecture design.

Next, under this condition of N being 8, the constructed memorizer–recaller network was asked to solve sorting problems. An output was considered correct only when the order of the elements in the output values from the recaller perfectly matched the order of the elements in the input values. After performing the calculation 1024 times, the network was able to sort accurately in 893 trials. Therefore, the accuracy of this sorting algorithm was 0.872. The sorting algorithm generated in this study cannot provide a perfect answer and can only sort up to 8 numbers, but there is a possibility of achieving a better result by adjusting the network architecture of the memorizer–recaller, optimizing the parameter size, or making the learning termination condition more stringent.

As a result, as expected, it was possible to generate a sorting algorithm using the memorizer–recaller network. There are various types of sorting algorithms, but the worst time complexity of the comparison sorting method, which is the most standard sorting algorithm, is $O(N \log N)$ when the number of input numbers is N. In contrast, the computation complexity of the sorting algorithm generated using our memorizer–recaller is always $O(N)$. This experimental result reinforced the evidence that the memorizer–recaller network can hold input information in the memory vector, and also demonstrated its usefulness to apply actual problem.

3 Conclusion

In this study, we aimed to construct a system in which artificial intelligence can acquire knowledge even after the model has been trained, and we created a memorizer–recaller network. Furthermore, we attempted to construct a permanent memory system in which artificial intelligence can retain all sorts of information in the world. The memorizer–recaller system that we constructed in this study succeeded in holding separately inputted information in the memory vector of the memorizer and retrieving it by the recaller. We attempted to construct a sorting algorithm using this network, which also succeeded, demonstrating that the memorizer can correctly encode input information into the memory vector and that the recaller can appropriately use that information.

While reinforcement learning can be used to make models acquire procedures rather than learning patterns in data, this study did not use such a mechanism, and it is considered a new study that achieved this by using random numbers in the learning method. Random numbers are also used in diffusion models and generative adversarial networks (GANs). This study was inspired by those studies, but considering that diffusion models and GANs are indeed learning patterns in the learning data, it is considered different from those.

On the other hand, although the memorizer–recaller network was originally developed based on the motivation to allow artificial intelligence to hold an arbitrary amount of information, the actual amount of information that can be memorized by the network was only up to 8. Based on the results of preliminary experiments, it is expected that this number can be somewhat improved by increasing the parameter size of the network. However, this is not a fundamental solution. To improve this, some kind of breakthrough is considered necessary.

The fact that the memorizer–recaller network does not have sufficient memory may be related to catastrophic forgetting [5]. Originally, catastrophic forgetting occurs when you learn one task and another task. Since the information to be input to the memorizer in this study is not related to another task, it does not seem to be related to this phenomenon at first glance. However, as shown in the experimental results, the memorizer–recaller network has a nature of completely forgetting information before a certain point, so there is no doubt that some kind of forgetting is occurring. Several methods have been proposed to prevent catastrophic forgetting, including Elastic Weight Consolidation [6] and Deep Generative Reply [7]. We were unable to find a method that could be directly used in our developed memorizer–recaller network, but with some ingenuity, it may be possible to apply these methods developed so far. We plan to examine this in the future.

Although the results of this study only brought preliminary results, it is fundamentally different from the previous learning methods of neural networks and is considered groundbreaking. In the future, we will continue to develop with the expectation that it will become one of the basic methods for constructing artificial intelligence that can store any information in finite-sized data and freely utilize them.

Author Contributions

The concept of the study was conceived by KDY. All the authors conducted the analysis. All the authors made contributions to writing the manuscript. All authors approved the final version of the article.

Funding

This work was supported in part by the Top Global University Project from the Ministry of Education, Culture, Sports, Science, and Technology of Japan (MEXT).

REFERENCES

[1] Kazunori D Yamada, Samy Baladram, and Fangzhou Lin. Progress in research on implementing machine consciousness. *Interdisciplinary Information Sciences*, 2022.

[2] Braden Hancock, Antoine Bordes, Pierre-Emmanuel Mazare, and Jason Weston. Learning from dialogue after deployment: Feed yourself, chatbot! *Annual Meeting of the Association for Computational Linguistics*, 2019.

[3] John R. Searle. Minds, brains, and programs. *Behavioral and Brain Sciences*, 1980.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Conference on Neural Information Processing Systems*, 2017.

[5] Vinay Venkatesh Ramasesh, Ethan Dyer, and Maithra Raghu. Anatomy of catastrophic forgetting: Hidden representations and task semantics. *International Conference on Learning Representations*, 2021.

[6] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proc. of the national academy of sciences*, 2017.

[7] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Conference on Neural Information Processing Systems*, 2017.