Robustness of Differentiable Neural Computer Using Limited Retention Vector-based Memory Deallocation in Language Model

Donghyun Lee, Hosung Park, Soonshin Seo, Hyunsoo Son, Gyujin Kim, and Ji-Hwan Kim*
Department of Computer Science and Engineering, Sogang University
35 Baekbeom-ro, Mapo-gu, Seoul, 04107, Republic of Korea
[e-mail: {redizard, hosungpark, ssseo, sonhyunsoo, kgh3620, kimjihwan}@sogang.ac.kr]
*Corresponding author: Ji-Hwan Kim

Received December 31, 2020; revised February 20, 2021; accepted March 2, 2021; published March 31, 2021

Abstract

Recurrent neural network (RNN) architectures have been used for language modeling (LM) tasks that require learning long-range word or character sequences. However, the RNN architecture is still suffered from unstable gradients on long-range sequences. To address the issue of long-range sequences, an attention mechanism has been used, showing state-of-the-art (SOTA) performance in all LM tasks. A differentiable neural computer (DNC) is a deep learning architecture using an attention mechanism. The DNC architecture is a neural network augmented with a content-addressable external memory. However, in the write operation, some information unrelated to the input word remains in memory. Moreover, DNCs have been found to perform poorly with low numbers of weight parameters. Therefore, we propose a robust memory deallocation method using a limited retention vector. The limited retention vector determines whether the network increases or decreases its usage of information in external memory according to a threshold. We experimentally evaluate the robustness of a DNC implementing the proposed approach according to the size of the controller and external memory on the enwik8 LM task. When we decreased the number of weight parameters by 32.47%, the proposed DNC showed a low bits-per-character (BPC) degradation of 4.30%, demonstrating the effectiveness of our approach in language modeling tasks.

Keywords: Differentiable Neural Computer (DNC), Language Model (LM), Memory Deallocation, Retention Vector, Robustness

This research was funded by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2020R1F1A1076562).

http://doi.org/10.3837/tiis.2021.03.002 **ISSN : 1976-7277**

1. Introduction

A language model (LM) calculates the probability of the current word or character with a previous word or character sequence [1]. For the sequence $S = (s_1, s_2, \dots, s_n)$, the probability of the LM is denoted as $P(S)$.

$$P(S) = (s_1, s_2, \dots, s_n), \tag{1}$$

where n is the sequence length of S. When the number of elements in the word or character history $(s_1, s_2, \dots, s_{n-1})$ increases, it is difficult to estimate the probability for the current word or character s_n, because the word or character history will not appear in the text. Therefore, Markov assumption is applied to the LM to estimate $P(s_n|s_1, s_2, \dots, s_{n-1})$. The sequence length affects to the current word or character s_n is $(n-1)$.

For modelling LMs, recurrent neural network (RNN) architectures have shown high performance, because a recurrent hidden layer maintain context of long-range word sequences. Especially, a long short-term memory (LSTM) has been successful in modelling context dependent information [2]. The LSTM is a hidden node consists of memory cells and three gates that maintain context dependent information over time. The LSTM has been proposed to prevent vanishing and exploding gradients problems associated with RNN architectures. However, previous works showed that the LSTM are still suffered from unstable gradients on long-range sequences consist of more than 100-200 words or characters. To address this issue, training tricks of RNN architecture, such as gradient clipping or layer-wise normalization, are used.

For training deep learning models with long-range sequences, an attention mechanism has been used. The attention mechanism is an effective method for selecting important information on longer sequences. It supports deep learning models to obtain information of how models attend to different blocks of input sequences [3]. One of the most widely used attention-based model is the Transformer, and it outperforms LSTM-based LMs [4]. The Transformer is composed of an encoding component and decoding component. The encoding component is a stack of encoder blocks and the decoding component is a stack of decoder blocks. The encoder's input sequence is used as an input of a self-attention layer. This layer allows the encoder to address context in the input sequence. An output of the self-attention layer is used as input of a feed-forward neural network. In the decoder block, an encoder-decoder attention layer is added. The encoder-decoder attention layer supports the decoder focus on appropriate contexts in the input sequence. However, the Transformer splits long-range input sequence into fixed size chunks. It causes the context fragmentation problem [5].

Another attention-based deep learning models are a differentiable neural computer (DNC). The DNC architecture is a neural network augmented with a content-addressable external memory [6]. In the DNC, a neural network is a controller, and a $N \times M$ matrix is the external memory (N is the number of vectors in the external memory and M is a dimension of vectors in the external memory). The memory attention mechanism of the DNC decides where information is stored in the external memory and maintains the order of sequences through a temporal link. In the experiments, the DNC showed a performance improvement on the bAbI question-answering task, graph traversal, and block puzzle problems. Recently, the DNC-based LM have been proposed. In previous works, the DNC showed perplexity (PPL) to 98.6 on the word-level Penn Treebank (PTB) dataset [7]. It exhibited better performance than deep neural network (DNN) and LSTM-based LMs, but lower performance than the Transformer-based LM.

However, in the write operation, some information unrelated to the input word remains in memory [8]. For memory deallocation, an erase vector is multiplied with the external memory at the previous time-step, and then, the current external memory is generated by adding deallocated external memory to a multiplication of an attention vector and input vector. In the write operation, if a *i*-th selected address of the memory is deallocated, the *i*-th index of the erase vector should be 1. The vanilla DNC generates the erase vector with sigmoid functions. It causes that the element of the erase vector can only be 1 when an input of the sigmoid function is infinity. Also, the DNC architecture have been shown performance degradation with low numbers of weight parameters in the write operation, because of poor memory deallocation.

In this paper, we present a robust memory deallocation method using a retention vector. The retention vector determines whether the DNC increases or decreases its usage of information in external memory. It indicates a retention ratio for information at each memory address. In the vanilla DNC, the retention vector is only used to generate the attention vector in the write operation. Therefore, while the retention vector is updated, information of the currently accessed memory address is not deallocated with the retention vector and remain in the external memory at all time-steps although this information is not related to current input sequence. It causes low robustness of memory deallocation in the write operation. Our proposed method selects the minimum of elements in the retention vector. The selected minimum element is converted to 0 using a threshold value generated from a threshold gate. Next, the previous external memory is multiplied by the proposed retention vector.

We evaluated the proposed method on enwik8 benchmark LM task. First, we evaluated performance of the DNC using the proposed method compared with the vanilla DNC and Transformer. Second, we evaluated robustness of the proposed approach according to the size of the controller and external memory on the enwik8 LM task.

The rest of our paper is organized as follows. Section 2 reviews previous works on LMs using deep learning models. Section 3 describes the write operation of the vanilla DNC architecture. Section 4 explains our proposed method in more detail. Section 4 describes experimental environments and results, and Section 5 concludes our paper.

2. Previous Works

In the RNN-based LM, to estimate the current word or character, a recurrent hidden layer is used to serve the role of memory through recurrence [9]. The memory ability of the recurrent hidden layer can learn from the first word or character to the *n*-th word or character. However, the recurrent hidden layer trained in the previous time-step is gradually distorted by newly input sequence as the time-step progresses [10]. The RNN is updated with summation of information stored in the recurrent hidden layer in every time-step and current input word or character. As a result, although the RNN-based LM can store a short sequence containing short-term context information, it is difficult to store long-range sequence including long-term context information.

For modelling long-term context information, the LSTM has been adopted to the RNN-based LM [11]. The LSTM has memory cells, so it can store context information in every time-step [12]. Memory cells of the LSTM are updated as follows:

$$mc_t = fg_t mc_{t-1} + ig_t(W_{x,mc}x_t + W_{ho,mc}ho_{t-1} + b_{mc}), \quad (2)$$

where mc_t is a value of the memory cell at time-step t, mc_t is a value of the memory cell at time-step $(t-1)$, ig_t is a value of an input gate at time-step t, fg_t is a value of an forget gate at time-step t, ho_{t-1} is a vector of a hidden layer at time-step $(t-1)$, $W_{x,mc}$ is a weight matrix between an input vector x_t and the memory cell at time-step t, $W_{ho,mc}$ is a weight matrix between the hidden layer and the memory cell at time $(t-1)$, and b_{mc} is a value of bias for the memory cell. Depending on the context, the forget gate retains or deletes the value of mc_{t-1} which stores the previous input sequence from time-step 0 to $(t-1)$. The LSTM achieved a higher performance than the RNN in LM tasks, but previous works showed that the LSTM are still suffered from unstable gradients on long-range sequences consist of more than 200 words or characters [13]. Although training tricks [14], such as gradient clipping or layer-wise normalization, are used, the LSTM cannot maintain long-term context information.

The attention mechanism has been proposed to model long-term context information in LM tasks. The attention mechanism allows the deep learning model to focus on relevant information of an input sequence [15, 16]. The Transformer is the most common attention-based deep learning model [4]. The Transformer is composed of an encoding component and decoding component. The encoding component is a stack of encoder blocks and the decoding component is a stack of decoder blocks. The encoder's input sequence is used as an input of a self-attention layer. This layer allows the encoder to address context in the input sequence [17]. An output of the self-attention layer is used as input of a feed-forward neural network. In the decoder block, an encoder-decoder attention layer is added. The encoder-decoder attention layer supports the decoder focus on appropriate contexts in the input sequence [18].

However, the Transformer splits long-range input sequence into fixed size chunks. It causes the context fragmentation problem. Previous works have been proposed to overcome the context fragmentation problem. Bidirectional encoder representations from Transformers (BERT) uses the encoder component of the Transformer to attend to bi-directional context information in a pre-training stage [19, 20]. In addition, a next sentence prediction method is adopted to BERT. This method predicts whether the generated sequence is the actual next sequence of the first sequence [21]. A generative pre-trained Transformer 2 (GPT-2) uses the decoder component of the Transformer [22, 23]. To model the GPT-2, an unsupervised pre-training method using a LM and supervised fine-tuning method are adopted [24]. In [25], a generative pre-trained Transformer 3 (GPT-3) was proposed. The GPT-3 is an autoregressive model. It is composed of 175 billion trainable parameters. In the experiments, the GPT-3 generated human-like text and showed the best performance when it uses an in-context few-shot learning method.

3. Differentiable Neural Computer

3.1 Structural Overview of Differentiable Neural Computer

Another attention-based deep learning models are a differentiable neural computer (DNC). **Fig. 1** is a structural overview of the DNC architecture, which consists of the controller D and external memory EM. An input vector x_t and R read vectors r_{t-1}^{i} at time-step $(t-1)$ are concatenated. A concatenated vector is used as inputs of the controller D. Two output vectors of the controller are generated: 1) a controller output vector do_t and 2) an interface vector I_t. The controller output vector is equal to the output of a hidden layer in the deep learning model [26]. The interface vector I_t determines the memory address accessed at time-step t to perform the read and write operation [27]. The DNC performs the write operation, so the converted concatenated vector is written on the external memory. The external memory is an $N \times M$

matrix, where N is the number of vectors in the external memory and M is the dimension of vectors in the external memory. After the write operation, the DNC performs the read operation. In the read operation, read vectors r_t^i are generated with the attention mechanism. These vectors are projected into a dimension of deep learning model T. The output vector of T is added to the controller output vector and then projected into a final output vector y_t.

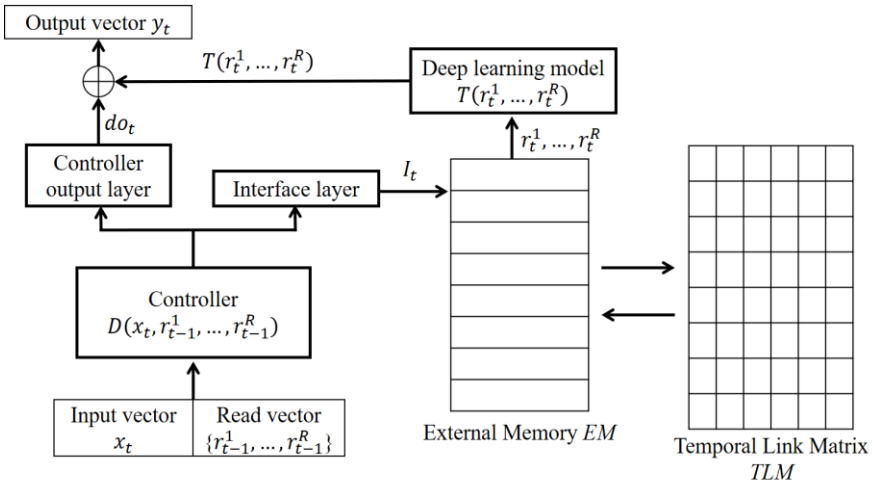


Fig. 1. Overview of DNC

In the read operation, a read attention vector $\omega_t^{r,i}$ is used to generate read vectors. The i-th read vector r_t^i at time t is defined as (3).

$$r_t^i = EM_t^{\mathrm{T}} \omega_t^{r,i}, \tag{3}$$

where $\omega_t^{r,i}$ is the read attention vector and EM_t^{T} is a transposed external memory. A dimension of $\omega_t^{r,i}$ is N, so a dimension of r_t^i is M. In the write operation, a write attention vector $\omega_t^{r,i}$ determines the memory address and information storage ratio. Unlike the read operation, the write operation also requires an erase vector e_t ($\in [0,1]^M$) and converted concatenation vector v_t ($\in \mathrm{R}^M$) at time-step t. The erase vector determines the ratio at which information is erased, before v_t is stored in the external memory at time-step t. e_t and v_t are element of the interface vector.

$$EM_t = EM_{t-1} \circ (ZM - \omega_t^w e_t^T) + \omega_t^w v_t^T, \tag{4}$$

where EM_t and EM_{t-1} are the external memory at time-step t and $(t-1)$, respectively. ZM is a matrix of the same size as that of EM_t. All elements of ZM are 1. $\circ$ is the element-wise product. e_t^T is the transposed erase vector. v_t^T is the transposed converted concatenation vector. ω_t^w is the write attention vector. The dimension of ω_t^w is N.

3.2 Write Operation of Differentiable Neural Computer

To determine ω_t^w in (4), the DNC uses both a content-based addressing method and dynamic memory allocation method [28, 29]. A combination of the following three cases is performed with two methods: 1) when there is no write operation at time-step t, 2) when ω_t^w is generated

by the dynamic memory allocation method, and 3) when ω_t^w is generated by the content-based addressing. It can be shown as (5).

$$\omega_t^w = g_t^w[g_t^a a_t + (1 - g_t^a)C(M_{t-1}, k_t^w, \beta_t^w)], \tag{5}$$

where g_t^w ($\in [0,1]$) is a write gate at time-step t, g_t^a ($\in [0,1]$) is an allocation gate at time-step t, a_t ($\in R^N$) is an allocation weighting vector at time-step t for the dynamic memory allocation, and $C(M_{t-1}, k_t^w, \beta_t^w)$ ($\in R^N$) is a content-based addressing vector. The write gate g_t^w determines the degree to which the write operation is performed or not. It is an element of the interface vector. The allocation gate g_t^a is used as an interpolation factor between the dynamic memory allocation method and content-based addressing. It is also an element of the interface vector.

The allocation weighting vector a_t determines the degree to which memory addresses are allocated [30, 31]. Assume that the external memory consists of 5 memory addresses and a first memory address can be allocated, then $a_t = [1; 0; 0; 0; 0]$. To generate a_t, a usage vector u_t ($\in [0,1]^N$) is calculated. u_t determines whether to increase or decrease usage of information stored in the *i*-th external memory address. If usage of information stored in the *i*-th memory address is increased, it signifies that read operations will be performed to the *i*-th memory address at later time-steps when the write operation is performed to *i*-th memory address at time-step $(t-1)$. In contrast, if usage of information stored in the *i*-th memory address is decreased, there were no write operations performed to *i*-th memory address at time-step $(t-1)$. It also means that the degree to which write operations are performed to *i*-th memory address in layer time-steps is gradually reduced and the converted concatenation vector is stored in *i*-th memory address if the controller reads information from the *i*-th memory address.

The usage vector u_t is generated by a retention vector ψ_t ($\in [0,1]^N$). ψ_t determines a retention degree of information stored in each memory address [32]. If $\psi_t[i] = 0$, it implies that information stored in the *i*-th external memory address will not be maintained. In the DNC, information that is read recently cannot be used to perform read operations because this information is the previous information. If $\psi_t[i] = 1$, it implies that information stored in the *i*-th external memory address have to be maintained because this information is not read by the DNC, recently. The usage vector uses ψ_t to select the external memory address and shows the lowest usage in memory deallocation. ψ_t is defined as (6)

$$\psi_t = \prod_{i=1}^{R}(1 - fg_t^i w_{t-1}^{r,i}), \tag{6}$$

where $w_{t-1}^{r,i}$ is the read attention vector at time-step $(t-1)$, fg_t^i ($\in [0,1]$) is a free gate, and R is the number of read attention vectors. The free gate is an element of the interface vector. It determines to guarantee the degree that information in the external memory can be maintained after read operations are performed. Therefore, u_t using ψ_t is defined as (7).

$$u_t = (u_{t-1} + \omega_{t-1}^w - u_{t-1} \circ \omega_{t-1}^w) \circ \psi_t, \tag{7}$$

where u_{t-1} is the usage vector at time-step $(t-1)$, ω_{t-1}^w is the write attention vector at time-step $(t-1)$, and ψ_t is the retention vector at time-step t. u_{t-1}, ω_{t-1}^w, and ψ_t are real-value vectors ($\in [0,1]^N$).

Finally, the allocation weighting vector a_t is defined as (8).

$$a_t[\phi_t[j]] = (1 - u_t[\phi_t[j]]) \prod_{i=1}^{j-1} u_t[\phi_t[i]], \tag{8}$$

where ϕ_t ($\in \mathrm{R}^N$) is a free list at time t, u_t is the usage vector at time $(t-1)$, and a_t is the allocation weighting vector at time t. It defined as ascending order of u_t from an index of u_t. The index of u_t sorted in ascending order using each element of u_t, because a_t sequentially accesses the index of u_t sorted by ϕ_t, in which the first index of u_t is considered to be the memory address with the smallest usage. For example, if $u_t = [0; 0.9; 0.4; 0.7]$, then $\phi_t = [0; 2; 3; 1]$.

4. Differentiable Neural Computer Using Limited Retention Vector-based Memory Deallocation

In the write operation, the retention vector ψ_t only affects to the usage vector u_t, but not the external memory. That is, while a vector of the external memory currently accessed by the updated usage vector, the vector of the currently accessed memory address is not deallocated and remain. Therefore, in the read operation, the vector has not been deallocated can be accessed and affected to read vectors. As an example, assume that the DNC is trained on the question-answering task and a story "John is at the playground. There is a soccer ball at Bob's house. John picks up the soccer ball." is given. The question is "Who picked up the soccer ball?". In this example, because Bob's house also has information "soccer ball," the sentence "There is a soccer ball at Bob's house" is unnecessary information that must be deleted from the external memory. If the DNC does not properly perform memory deallocation, it may provide the wrong answer.

Previous works have been proposed to solve the above problem with the retention vector. Their proposed method is that the external memory at time-step $(t-1)$ is multiplied by the retention vector. It is defined as (9).

$$EM_t = EM_{t-1} \circ \psi_t 1^T \circ (ZM - \omega_t^w e_t^T) + \omega_t^w v_t^T, \tag{9}$$

where EM_t and EM_{t-1} are the external memory at time-step t and $(t-1)$, respectively. ψ_t is the retention vector at time-step t, 1^T is the transposed vector. All elements of 1^T are 1. ZM is a matrix of the same size as that of EM_t. All elements of ZM are 1. ω_t^w is the write attention vector at time-step t, e_t^T is the transposed erase vector. v_t^T is the transposed converted concatenation vector. $\psi_t 1^T$ is defined as the same matrix as the external memory; thus, all elements of EM_{t-1} are multiplied to ψ_t. However, despite some information stored in the external memory is no longer needed, their method cannot definitely deallocate unnecessary information from the external memory. To definitely deallocate unnecessary information, an element of the retention vector has to be 0.

To address this issue, we proposed limited retention vector-based memory deallocation (LRV-DNC). The proposed method converts the minimum of elements in the retention vector to 0 according to a threshold. The pseudo algorithm for the proposed method is illustrated in **Fig. 2**. The detailed explanation of **Fig. 2** is as follows:

1. Search the minimum of elements in the retention vector. The proposed method selects multiple elements if they are minimum.

2. If the output value of a forget gate is less than the threshold, then the proposed method converts selected elements of the retention vector to 0. If not, it does not convert.

3. The retention vector generated from 2 is multiplied to the external memory at time-step $(t-1)$

In the stage 2, the forget gate $g_t^{\psi_t}$ ($\in [0,1]$) is an element of the interface vector. $g_t^{\psi_t}$ determines a decision boundary to which the minimum of elements in the retention vector is converted to 0 or not according to the threshold T ($\in [0,1]$). The threshold T is defined as handicraft value, so it has to be decided before training the DNC. The limitation of our proposed DNC architecture is as follows: 1) To find the minimum of elements in the retention vector, search algorithms are needed. We use a binary search algorithm. However, in the worst case, the time complexity of the binary search algorithm is $O(N)$ (N is the number of elements in the retention vector). 2) The threshold T is defined as handicraft value. Therefore, if pre-trained DNC architecture is used to another LM task, T has to be modified.

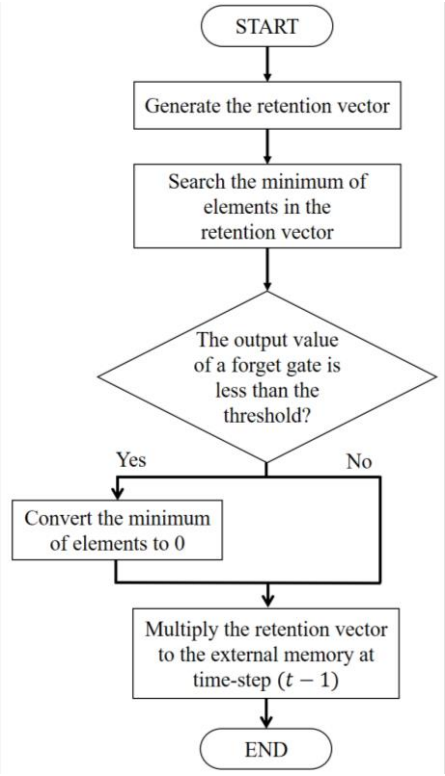


Fig. 2. Pseudo algorithm for the proposed memory deallocation

5. Experiments and Discussion

The proposed LRV-DNC was evaluated on the enwik8 LM task. In Section 5.1, we describe experimental environment. In Section 5.2, we compare the proposed LRV-DNC-based LM with the Transformer and vanilla DNC. In Section 5.3, we evaluated robustness of the LRV-DNC-based LM.

5.1 Experimental Environment

The enwik8 LM task dataset consists of 100M characters. Its domain is unprocessed Wikipedia data. We split the dataset into 90, 5, and 5M characters, respectively, for the train, validation, and test data. The number of characters in the enwik8 LM task was 206, so we used a 1-of-$|V|$ vector as an input of all LMs. Each experiment was performed as 3-fold validation for justifying our experiments.

We used bits-per-character (BPC) as an evaluation metric. BPC is the average number of bits (a unit of entropy) for encoding a character. In the experiments, BPC is defined as $loss/\log(2)$. A server specification was as follows: a 2.20GHz Intel Xeon Gold 5120 CPU and two Nvidia V100 GPUs.

5.2 The enwik8 LM Task

5.2.1 Experimental Setup

The baseline was the LSTM-based LM from [33]. The Transformer-based LM from [5] was considered as the SOTA model. We implemented the vanilla DNC-based LM with PyTorch. The controller is the LSTM. In the controller, the number of LSTM hidden layers was 3, and the number of LSTM hidden nodes was 1024. The number of vectors in the external memory was 128. The dimension of each vector in the external memory was 256. The learning rate initialized at 1×10^{-3}. To reduce the learning rate, a scheduler with a plateau objective function was used. The weight decay was 1×10^{-7}. The length of the back-propagation to time (BPTT) was 120. The batch size was 20.

We also implemented the previous memory deallocation method using the retention vector (RV-DNC) and our proposed LRV-DNC-based LM with PyTorch. Hyper-parameters of the RV-DNC-based LM and LRV-DNC-based LM are equal to that of the vanilla DNC-based LM. In addition, in LRV-DNC-based LM, the threshold was 0.5.

5.2.2 Experimental Results

Table 1 shows the BPC results of the LRV-DNC-based LMs. We evaluated the BPC results of the proposed DNC-based LM according to the number of read vectors. The LRV-DNC-based LM using 4 read vectors showed the best BPC results in the DNC-based LMs, with BPC of 1.3847. Although the BPC result of the Transformer-based LM was a SOTA result (BPC was 1.1120), our proposed method showed a relative improvement of 0.47% compared with the vanilla DNC in terms of BPC. In addition, our LRV-DNC-based LM showed a relative improvement of 0.27% compared with the previous RV-DNC-based LM.

Table 1. BPC results of the LRV-DNC-based LMs

Model	No. of weight parameters	No. of read vectors	BPC	
			Validation	Test
LSTM	18.1 M	-	-	1.4610
Transformer	44.1 M	-	-	**1.1120**
Vanilla DNC	42.5 M	4	1.3959	1.3912
RV-DNC	42.5 M	4	1.3892	1.3885
LRV-DNC	38.9 M	1	1.3911	1.3908
	40.1 M	2	1.3882	1.3879
	41.4 M	3	1.3860	1.3856
	42.5 M	4	1.3851	**1.3847**

The discussion for **Table 1** is as follows: (1) We found that BPC was better when the LRV-DNC-based LM used more read vectors. The LRV-DNC-based LM using 4 read vectors showed a relative improvement of 0.44% compared with the LRV-DNC-based LM using a one read vector in terms of BPC. (2) Although the vanilla DNC and RV-DNC-based LM used 4 read vectors, the LRV-DNC-based LM showed the best BPC result. It showed a relative improvement of 0.27-0.47% compared with other DNC-based LMs. It means that our proposed DNC outperformed to previous DNC architecture in the LM task.

5.3 Robustness of LRV-DNC-based LM

5.3.1 Experimental Setup

We also used the LSTM-based LM as the baseline, and the Transformer-based LM as the SOTA model. The hyper-parameters of vanilla DNC and LRV-DNC-based LMs were equal to Section 5.2.1. In experiments for robustness evaluation, two types of experiments were performed for analyzing robustness: 1) the size of the external memory and 2) the size of the controller.

5.3.2 Experimental Results

Table 2 and **Table 3** shows the BPC results of the vanilla DNC-based LMs according to the different size of the external memory and controller, respectively. In **Table 2**, although the size of the external memory is decreased, the vanilla DNC-based LM using 32×32 external memory demonstrated a relative degradation of 1.66 % compared with the vanilla DNC-based LM using 128×256 external memory. In addition, in **Table 3**, although the size of the controller is decreased, the vanilla DNC-based LM using 32×32 external memory demonstrated a relative degradation of 4.56 % compared with the vanilla DNC-based LM using 128×256 external memory.

Table 2. BPC results of the vanilla DNC-based LMs according to the different size of the external memory on the enwik8 LM task

Model	No. of weight parameters	No. of read vectors	No. of vectors in the external memory	Dim. of vectors in the external memory	BPC	
					Val	Test
LSTM	18.1 M	-			-	1.4610
Transformer	44.1 M	-			-	**1.1120**
Vanilla DNC	30.5 M	1	32	32	1.4192	1.4181
	30.8 M	2			1.4188	1.4179

	31.1 M	3			1.4175	1.4172
	31.5 M	4			1.4169	1.4163
	30.9 M	1	64	64	1.4151	1.4147
	31.5 M	2			1.4135	1.4128
	32.2 M	3			1.4130	1.4123
	32.8 M	4			1.4126	1.4119
	31.7 M	1	128	128	1.4123	1.4117
	32.9 M	2			1.4121	1.4112
	34.2 M	3			1.4124	1.4115
	35.3 M	4			1.4097	1.4098
	42.5 M	4	128	256	**1.3959**	**1.3912**

The discussion for **Table 2** and **Table 3** is as follows: (1) In **Table 2**, the vanilla DNC using 2 read vectors showed higher performance than that of the vanilla DNC using 3 read vectors when the size of the external memory is 128×128. However, when the vanilla DNC used 32×32 and 64×64 size of the external memory, we found that BPC was better when the vanilla DNC-based LM used more read vectors. (2) Despite the number of weight parameters in the vanilla DNC-based LM is smaller than that of the LSTM-based LM in **Table 3**, the vanilla DNC-based LM showed higher performance than the LSTM-based LM. It means that the DNC architecture can be train with long-range context, although the LSTM cannot train with long-range context.

Table 3. BPC results of the vanilla DNC-based LMs according to the different size of the controller on the enwik8 LM task

Model	No. of weight parameters	No. of read vectors	No. of vectors in the external memory	Dim. of vectors in the external memory	BPC	
					Val	Test
LSTM	18.1 M	-			-	1.4610
Transformer	44.1 M	-			-	**1.1120**
Vanilla DNC	13.8 M	1	32	32	1.4595	1.4588
	14.1 M	2			1.4585	1.4582
	14.4 M	3			1.4579	1.4570
	14.7 M	4			1.4563	1.4567
	14.2 M	1	64	64	1.4556	1.4553
	14.8 M	2			1.4547	1.4544
	15.4 M	3			1.4536	1.4535
	16.0 M	4			1.4531	1.4532
	15.0 M	1	128	128	1.4525	1.4518
	16.2 M	2			1.4517	1.4511
	17.4 M	3			1.4504	1.4500
	18.6 M	4			1.4479	1.4481
	42.5 M	4	128	256	**1.3959**	**1.3912**

Table 4 and **Table 5** shows the BPC results of the proposed LRV-DNC-based LMs according to the different size of the external memory and controller, respectively. In **Table 4**, although the size of the external memory is decreased, the LRV-DNC-based LM using 32×32 size of the external memory demonstrated a relative degradation of 1.48 % compared with the LRV-DNC-based LM using 128×256 size of the external memory. In addition, in **Table 5**, although the size of the controller is decreased, the LRV-DNC-based LM using 32×32 size of

the external memory demonstrated a relative degradation of 4.30 % compared with the LRV-DNC-based LM using 128×256 size of the external memory.

The discussion for **Table 4** and **Table 5** is as follows: (1) A degradation rate of the LRV-DNC-based LM using 32×32 size of the external memory was lower than that of the vanilla DNC-based LM using 32×32 size of the external memory. Although both DNC-based LMs had same number of weight parameters, the proposed LRV-DNC was more robust than the vanilla DNC. Therefore, the vanilla DNC was improved with the proposed LRV method as an aspect of robustness. (2) In **Table 4**, we found that BPC was better when the LRV-DNC-based LM used more read vectors. This result was differed to **Table 2**. (3) Despite the number of weight parameters in the LRV-DNC-based LM is smaller than that of the LSTM-based LM in **Table 5**, the vanilla DNC-based LM showed higher performance than the LSTM-based LM. It means that the proposed LRV-DNC architecture can be train with long-range context, although the LSTM cannot train with long-range context, although the LRV-DNC showed lower performance than the Transformer.

Table 4. BPC results of the LRV-DNC-based LMs according to the different size of the external memory on the enwik8 LM task

Model	No. of weight parameters	No. of read vectors	No. of vectors in the external memory	Dim. of vectors in the external memory	BPC	
					Val	Test
LSTM	18.1 M	-			-	1.4610
Transformer	44.1 M	-			-	**1.1120**
LRV-DNC	30.5 M	1	32	32	1.4056	1.4050
	30.8 M	2			1.4034	1.4029
	31.1 M	3			1.4011	1.4003
	31.5 M	4			1.3997	1.3992
	30.9 M	1	64	64	1.4015	1.4018
	31.5 M	2			1.4003	1.4001
	32.2 M	3			1.3990	1.3997
	32.8 M	4			1.3982	1.3979
	31.7 M	1	128	128	1.3997	1.4005
	32.9 M	2			1.3984	1.3985
	34.2 M	3			1.3965	1.3972
	35.3 M	4			1.3942	1.3935
	42.5 M	4	128	256	**1.3851**	**1.3847**

Table 5. BPC results of the LRV-DNC-based LMs according to the different size of the controller on the enwik8 LM task

Model	No. of weight parameters	No. of read vectors	No. of vectors in the external memory	Dim. of vectors in the external memory	BPC	
					Val	Test
LSTM	18.1 M	-			-	1.4610
Transformer	44.1 M	-			-	**1.1120**
LRV-DNC	13.8 M	1	32	32	1.4446	1.4439
	14.1 M	2			1.4438	1.4442
	14.4 M	3			1.4413	1.4419
	14.7 M	4			1.4391	1.4383
	14.2 M	1	64	64	1.4395	1.4397

	14.8 M	2			1.4387	1.4382
	15.4 M	3			1.4362	1.4360
	16.0 M	4			1.4351	1.4362
	15.0 M	1	128	128	1.4375	1.4370
	16.2 M	2			1.4359	1.4362
	17.4 M	3			1.4345	1.4338
	18.6 M	4			1.4321	1.4316
	42.5 M	4	128	256	**1.3851**	**1.3847**

6. Conclusion

The vanilla DNC architecture has drawback in memory deallocation, because the vector of the currently accessed external memory address is not deallocated definitely. Therefore, in the read operation, the vector has not been deallocated can be accessed. It affects to read vectors, and causes performance degradation. To solve this issue, we proposed the memory deallocation method using the limited retention vector (LRV-DNC). The proposed method searches the minimum elements in the retention vector. These elements are converted to 0 if the output value of the forget gate is less than the handcraft threshold. The modified retention vector is multiplied to the external memory at time-step $(t-1)$. In the experiments, the LRV-DNC-based LM showed a relative improvement of 0.27-0.47% compared with other DNC-based LMs, although the Transformer-based LM showed the SOTA results. In addition, the LRV-DNC-based LM showed a relative degradation of 1.48-4.30%. These results were more robustness to the vanilla DNC-based LM. Therefore, the vanilla DNC can have higher robustness if it uses our proposed LRV method.

References

[1] S. Mani, S. V. Gothe, S. Ghosh, A. K. Mishra, P. Kulshreshtha, M. Bhargavi, and M. Jumaran, "Real-time optimized n-gram for mobile devices," in *Proc. of the 13th International Conference on Semantic Computing*, pp. 87-92, 2019. Article (CrossRef Link)

[2] R. Mu and X. Zeng, "A review of deep learning research," *KSII Transactions on Internet and Information Systems*, vol. 13, no. 4, pp. 1738-1764, Apr. 2019. Article (CrossRef Link)

[3] F. Lin, X. Ma, Y. Chen, J. Zhou, and B. Liu, "PC-SAN: Pretraining-based contextual self-attention model for topic essay generation," *KSII Transactions on Internet and Information Systems*, vol. 14, no. 8, pp. 3168-3186, Aug. 2020. Article (CrossRef Link)

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, T. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, pp. 1-11, 2017. Article (CrossRef Link)

[5] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," in *Proc. of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2928-2988, 2019. Article (CrossRef Link)

[6] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska, and S. Colmenarejo, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, pp. 471-476, Oct. 2016. Article (CrossRef Link)

[7] W. Luo and F. Yu, "Recurrent highway networks with grouped auxiliary memory," *IEEE Access*, vol. 7, pp. 182037-182049, Dec. 2019. Article (CrossRef Link)

[8] R. Csordas and J. Schmidhuber, "Improving differentiable neural computers through memory masking, de-allocation, and link distribution sharpness control," in *Proc. of International Conference on Learning Representations*, pp. 7299-7310, 2019. Article (CrossRef Link)

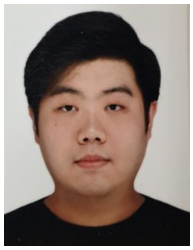
[9] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. of the 11th Annual Conference of the International Speech Communication Association*, pp. 1045-1048, 2010. Article (CrossRef Link)
[10] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. of the 30th International Conference on Machine Learning*, vol. 28, no. 3, pp. 1310-1318, 2013. Article (CrossRef Link)
[11] E. Arisoy, A. Sethy, B. Ramabhadran, and S. Chen, "Bidirectional recurrent neural network language models for automatic speech recognition," in *Proc. of 2015 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 5421-5425, 2015. Article (CrossRef Link)
[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computing*, vol. 9, no. 8, pp. 1735-1780, Nov. 1997. Article (CrossRef Link)
[13] U. Khandelwal, H. He, P. Qi, and D. Jurafsky, "Sharp nearby, fuzzy far away: How neural language models use context," in *Proc. of the 56th Annual Workshops of the Association for Computational Linguistics*, pp. 284-294, 2018. Article (CrossRef Link)
[14] Y. Zhang, X. Wang, and H. Tang, "An improved Elman neural network with piecewise weighted gradient for time series prediction," *Neurocomputing*, vol. 359, pp. 199-208, Sep. 2019. Article (CrossRef Link)
[15] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *Proc. of the 32nd International Conference on Machine Learning*, vol. 37, pp. 2048-2057, 2015. Article (CrossRef Link)
[16] Y. Belinkov and J. Glass, "Analysis methods in neural language processing: A survey," *Transactions of Association for Computational Linguistics*, vol. 7, pp. 49-72, Mar. 2019. Article (CrossRef Link)
[17] R. Al-Rfou, D. Choe, N. Constant, M. Guo, and L. Jones, "Character-level language modeling with deeper self-attention," in *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 1, pp. 3159-3166, 2019. Article (CrossRef Link)
[18] S. Duan, H. Zhao, J. Zhou, and R. Wang, "Syntax-aware transformer encoder for neural machine translation," in *Proc. of 2019 International Conference on Asian Language Processing*, pp. 396-401, 2019. Article (CrossRef Link)
[19] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, vol. 1, pp. 4171-4186, 2019. Article (CrossRef Link)
[20] J. Yang, M. Wang, H. Zhou, C. Zhao, W. Zhang, Y. Yu, and L. Li, "Towards making the most of BERT in neural machine translation," in *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 5, pp. 9378-9385, 2020. Article (CrossRef Link)
[21] W. Shi and V. Demberg, "Next sentence prediction helps implicit discourse relation classification within and across domains," in *Proc. of 2019 Conference on Empirical Methods in Natural Language Processing*, pp. 5790-5796, 2019. Article (CrossRef Link)
[22] Y. Qu, P. Liu, W. Song, L. Liu, and M. Cheng, "A text generation and prediction system: pre-training on new corpora using BERT and GPT-2," in *Proc. of the 10th International Conference on Electronics Information and Emergency Communication*, pp. 323-326, 2020. Article (CrossRef Link)
[23] W. Ko and J. Li, "Assessing discourse relations in language generation from GPT-2," in *Proc. of the 13th International Conference on Natural Language Generation*, pp. 52-59, 2020. Article (CrossRef Link)
[24] L. Floridi and M. Chiriatti, "GPT-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681-694, Nov. 2020. Article (CrossRef Link)
[25] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, and A. Neelakanta, "Language models are few-shot learners," in *Proc. of Conference on Neural Information Processing Systems*, pp. 1-25, 2020. Article (CrossRef Link)

[26] J. Kim, I. Choi, and M. Lee, "Context aware video caption generation with consecutive differentiable neural computer," *Electronics*, vol. 9, no. 7, pp. 1-15, July 2020. Article (CrossRef Link)
[27] T. Park, I. Choi, and M. Lee, "Distributed memory based self-supervised differentiable neural computer," *arXiv:2007.10637*, 2020. Article (CrossRef Link)
[28] M. Rasekh and F. Safi-Esfahani, "EDNC: Evolving differentiable neural computers," *Neurocomputing*, vol. 412, pp. 514-542, Oct. 2020. Article (CrossRef Link)
[29] R. Sharma, A. Kumar, D. Meena, and S. Pushp, "Employing differentiable neural computers for image captioning and neural machine translation," *Procedia Computer Science*, vol. 173, pp. 234-244, July 2020. Article (CrossRef Link)
[30] Y. Ming, D. Pelsusi, C. Fang, M. Prasad, Y. Wang, D. Wu, and C. T. Lin, "EEG data analysis with stacked differentiable neural computers," *Neural Computing and Applications*, vol. 32, pp. 7611-7621, June 2018. Article (CrossRef Link)
[31] A. Mufti, S. Penkov, and S. Ramamoorthy, "Iterative model-based reinforcement learning using simulations in the differentiable neural computer," *arXiv:1906.07248*, 2019. Article (CrossRef Link)
[32] C. Yin, J. Tang, Z. Xu, and Y. Wang, "Memory augmented deep recurrent neural network for video question answering," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3159-3167, Sep. 2020. Article (CrossRef Link)
[33] A. Mujika, F. Meier, and A. Steger, "Fast-slow recurrent neural networks," in *Proc. of the 31st Annual Conference on Neural Information Processing Systems*, pp. 5917-5926, 2017. Article (CrossRef Link)

Donghyun Lee received his B.E. degree in Computer Science and Engineering from Sogang University, Republic of Korea in 2013 and his Ph.D. degree in Computer Science and Engineering from Sogang University. He is currently pursuing a Post-Doc. researcher in the Computer Science and Engineering Department, Sogang University. His research interests include speech recognition and spoken multimedia content search.

Hosung Park received his B.E. degree in Computer Science and Engineering from Handong Global University in 2016. He also received his M.E. degree in Computer Science and Engineering from Sogang University in 2018. He is currently pursuing a Ph.D. degree in Computer Science and Engineering at Sogang University. His research interests include speech recognition and spoken multimedia content.

Soonshin Seo received his B.A. degree in Linguistics and B.E. degree in Computer Science and Engineering from Hankuk University of Foreign Studies in 2018. He is currently pursuing a Ph.D. degree in Computer Science and Engineering at Sogang University. His research interests include speaker recognition and spoken multimedia content search.



Hyunsoo Son received his B.E. degree in Computer Science and Engineering from Sogang University in 2019. He is currently pursuing a M.E. degree in Computer Science and Engineering at Sogang University. His research interests include speech recognition and spoken multimedia content search.

Gyujin Kim received his B.E. degree in Computer Science and Engineering from Sogang University in 2019. He is currently pursuing a M.E. degree in Computer Science and Engineering at Sogang University. His research interests include speech recognition.

Ji-Hwan Kim received the B.E. and M.E. degrees in Computer Science from KAIST (Korea Advanced Institute of Science and Technology) in 1996 and 1998 respectively and Ph.D. degree in Engineering from the University of Cambridge in 2001. From 2001 to 2007, he was a chief research engineer and a senior research engineer in LG Electronics Institute of Technology, where he was engaged in development of speech recognizers for mobile devices. In 2004, he was a visiting scientist in MIT Media Lab. Since 2007, he has been a faculty member in the Department of Computer Science and Engineering, Sogang University. Currently, he is a full professor. His research interests include spoken multimedia content search, speech recognition for embedded systems and dialogue understanding.