Breaking Neural Reasoning Architectures With Metamorphic Relation-Based Adversarial Examples

Alvin Chan, *Graduate Student Member, IEEE*, Lei Ma, Felix Juefei-Xu, Yew-Soon Ong, *Fellow, IEEE*, Xiaofei Xie, Minhui Xue, *Member, IEEE*, and Yang Liu, *Senior Member, IEEE*

***Abstract*—The ability to read, reason, and infer lies at the heart of neural reasoning architectures. After all, the ability to perform logical reasoning over language remains a coveted goal of Artificial Intelligence. To this end, models such as the Turing-complete differentiable neural computer (DNC) boast of real logical reasoning capabilities, along with the ability to reason beyond simple surface-level matching. In this brief, we propose the first probe into DNC's logical reasoning capabilities with a focus on text-based question answering (QA). More concretely, we propose a conceptually simple but effective adversarial attack based on *metamorphic relations*. Our proposed adversarial attack reduces DNCs' state-of-the-art accuracy from 100% to 1.5% in the worst case, exposing weaknesses and susceptibilities in modern neural reasoning architectures. We further empirically explore possibilities to defend against such attacks and demonstrate the utility of our adversarial framework as a simple scalable method to improve model adversarial robustness.**

***Index Terms*—Adversarial examples, deep learning, differentiable neural computer (DNC), supervised learning.**

I. Introduction

Many modern neural readers are imbued with inductive biases that gear them toward reasoning capabilities. After all, the ability to perform logical reasoning across entities and facts is one of the many desirable properties that this class of models should possess. To this end, a recent line of work has focused on memory augmented neural architectures [1]–[6]. In particular, differentiable neural computer (DNC) [3], [7] is proposed to empower a model with complex logical reasoning capabilities by means of content addressing (i.e., attention) across an external differentiable memory store.

Leveraging observations that deep neural networks (DNNs) are vulnerable to adversarial examples [8]–[12], we present a conceptually simple but effective adversarial framework with two goals: 1) we show that simple adversarial examples can break the DNC's performance, reducing the state-of-the-art results from 100% to 1.5% in the worst case; 2) we empirically investigate two lines of defenses, showing the utility of our framework as a means to improve robustness.

Our model-agnostic framework is based on the key idea of *"Metamorphic Relations"* (MRs). By definition, *"programs"* that have several different inputs satisfying an MR should result in identical output. For instance, a simple MR for the mathematical sine function is $\sin(x+\pi) = \sin(x)$. In our context, this refers to injecting sentences that do not overwrite information from a passage and **preserve** the answer to a corresponding question; this is similar to how it is common for distractors to appear in a real-world text. For instance, adding sentences like "*Mary journeyed to the hallway.*" or "*Brian is a rhino.*" into the passage "*The office is south of the hallway. How do you go from the office to the hallway?*" should **not** change the correct answer "*north*". Unfortunately, our study reveals that the DNC model is susceptible to such simple attacks, possibly indicating a key weakness in this class of memory augmented models.

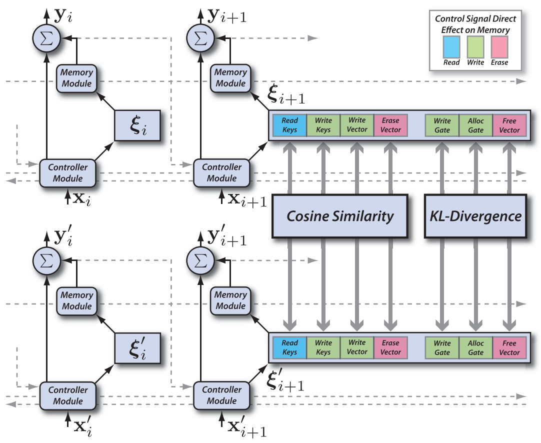

Fig. 1. (Top) Control signals from DNC with benign input can be compared with (Bottom) control signals from DNC with adversarial input.

The contributions of this brief are as follows.

1) We propose a conceptually simple but effective vulnerability detection framework **Pick-n-Plug**–an automated and scalable method that relies on MRs to generate grammatically correct adversarial examples in the natural language processing (NLP) question answering (QA) domain.
2) Analysis of DNC's control signals, as illustrated in Fig. 1, which shows that adversarial attacks disrupt the read, write, and erase functions of the DNC.
3) We perform an empirical analysis of a potential debugging approach, showing that using Pick-n-Plug in data augmentation procedure can enhance the model's generalization.

II. Background and Related Work

A. Adversarial Attack

Adversarial attacks were first studied in computer vision domains [8]. Carefully perturbed images, with changes imperceptible to humans, can easily fool DNNs. Since then, we have witnessed an arms' race between attackers [13] and defenders [14]. Although adversarial attack techniques are extensively studied in computer vision, there is limited work conducted in NLP domain. One challenge lies in the discrete nature of word inputs in NLP which makes the implementation of gradient-based perturbation methods

Manuscript received February 20, 2020; revised June 19, 2020 and March 7, 2021; accepted March 31, 2021. *(Corresponding author: Alvin Chan.)*

Alvin Chan, Yew-Soon Ong, and Yang Liu are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: guoweial001@e.ntu.edu.sg).

Lei Ma is with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 2R3, Canada.

Felix Juefei-Xu is with the Alibaba Group, Sunnyvale, CA 94085 USA.

Xiaofei Xie is with the Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka 819-0395, Japan.

Minhui Xue is with School of Computer Science, The University of Adelaide, Adelaide, SA 5005, Australia.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2021.3072166.

Digital Object Identifier 10.1109/TNNLS.2021.3072166

challenging. Different from adversarial attacks for images where small pixel changes are very unlikely to alter the correct class of an image, a change of word in a body of text may completely change its meaning under a particular NLP task or introduce grammatical errors.

For adversarial attacks in NLP [15], Jia [11] proposed to add distracting sentences to the original text; however, the grammatical correctness and the preservation of correct answer rely on manual checks, which makes such a method difficult to scale. Word substitution-based attacks [16]–[19] were also proposed, by changing words in the original text with synonyms. In [16], a portion of the substituted text is interpreted by humans to be a different class, highlighting the challenge of creating adversaries that avoid changing the meaning of the original text. There are also work [20] that perturb word embeddings during the training of DNNs to circumvent changing word inputs directly to improve robustness, but embedding perturbation is unlikely in a real-world attack since it is not an input layer. In this brief, we attempt to address these challenges with adversaries generated based on MRs. Our adversarial methods are scalable while minimizing the disruption of information from the original text for QA tasks.

Several defenses against adversarial attacks have emerged recently [21]–[24], such as those that use a cleansing model to neutralize adversarial perturbations from input [21]–[23] or, similar to our defense, augment training data [24], [25]. A related threat class poisoning attacks were also recently studied in NLP [26]. The main focus of this brief is to study the threat of MR-based adversarial attacks and possible defenses against them.

B. Differential Neural Computer (DNC)

A DNC is a DNN augmented with an external memory module in the form of a matrix $M \in \mathbb{R}^{\mathrm{N}\times\mathrm{W}}$ [3]. The DNN of DNC acts as the controller module, whose operations can be learned with gradient descent, while the external memory matrix serves as a module for data storage. The DNC's controller and memory module are like its CPU and RAM, respectively. The DNC's memory can be written to and accessed by the controller. At each input time-step, the DNC controller takes in an input vector $\mathbf{x}_t \in \mathbb{R}^{\mathrm{X}}$, and a set of read vectors $\boldsymbol{\mu}_{t-1} \in \mathbb{R}^{\mathrm{P}}$ from the previous time-step

$$(\mathbf{v}_t, \boldsymbol{\xi}_t) = \mathrm{Controller}([\mathbf{x}_t, \boldsymbol{\mu}_{t-1}], \theta_c) \tag{1}$$

θ_c is the controller's trainable weight parameters, $\mathbf{v}_t \in \mathbb{R}^{\mathrm{C}}$ is the controller output while ξ_t is a set of control signals.

The controller uses its write and read heads to manage the memory matrix. At each time-step, the controller's set of control signals $\boldsymbol{\xi}_t$ represents the operations of these heads. These control signals can be categorized into gates, keys, or vectors. Their values determine how, where, and what is being read, written, and erased from the memory matrix. A series of operations in the memory module with the control signals and its current memory matrix M_t erase and write new data and produce a concatenation of read vectors $\boldsymbol{\mu}_t \in \mathbb{R}^{\mathrm{P}}$

$$\boldsymbol{\mu}_t = \mathrm{MemoryModule}(\boldsymbol{\xi}_t, M_t). \tag{2}$$

The final output of the DNC is a sum of weighted controller output and weighted concatenation of read vectors from the memory module

$$\mathbf{y}_t = W_v \mathbf{v}_t + W_\mu \boldsymbol{\mu}_t + \mathbf{b}_t \tag{3}$$

where $W_v \in \mathbb{R}^{\mathrm{Y}\times\mathrm{C}}$, $W_\mu \in \mathbb{R}^{\mathrm{Y}\times\mathrm{P}}$ and $\mathbf{b}_t \in \mathbb{R}^{\mathrm{Y}}$ are trainable weights of the DNC.

C. bAbI Data Set

The DNC that we are evaluating has shown near-perfect performance on the bAbI data set which makes it a good candidate to evaluate DNC's vulnerability to attacks. The bAbI data set has 20 question & answer tasks of different themes to evaluate a range of logical reasoning capabilities [27]. Each task contains stories that are followed by one or more questions with answers that can be derived from the story. For example, the story "*The office is south of the hallway.*" followed by the question "*How do you go from the office to the hallway?*" with an answer "*north*" requires a model that learned path finding capability. In another task with example "*Mary journeyed to the hallway. Mary picked up the football. Where is the football?*" with an answer "*hallway*" demands the model to process two supporting facts. The performance of bAbI tasks is evaluated by word error rate (WER) which is the rate of incorrect answers over the total number of answers.

III. MR-Based Adversaries

In image classification, an adversarial example is a modified input x' with an imperceptible perturbation added to the original input x so that it fools the classifier f to change its originally correct prediction $y = f(x)$ to an incorrect prediction $y' = f(x')$, where $y' \neq y$.

A. Metamorphic Transformation

To generate adversarial examples which do not change the original answers to bAbI QA tasks, we drew inspiration from MRs. An example of MRs for *sine* function is $\sin(x+\pi) = \sin(x)$. MRs have been used to detect vulnerabilities in traditional software (e.g., C, C++ and Java) [28] and supervised classifiers [29]. Here, we define a metamorphic transformation (MT) T as a function that maps an input x to x' which satisfy a MR with f. More formally

$$x' = T(x) \qquad f(x) = f(x')$$

where x is the original input and x' is output of a MT of x.

In the example of *sine* function where $f(x) = \sin(x)$, a valid MT is $T(x) = x + \pi$. Similarly, in the QA tasks, the input x and x' can be generalized to X and X', i.e., $X' = T(X)$, $f(X) = f(X')$, where f is an oracle that is *always correct*, $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m]$, $X' = [\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_{m'}]$ and $\mathbf{x}_i, \mathbf{x}'_i \in \mathbf{R}^{\mathrm{X}}$ are one-hot vectors of words in the original and transformed adversarial input sequence, respectively. For any input X' generated by the MT on X, the answer $f(X')$ would remain unchanged from $f(X)$ under the oracle f.

Consider a deep learning model for a QA task as f' where its prediction of an input sequence X is $f'(X)$. If an adversarial input $X' = T(X)$ is generated with a MT T (i.e., $f(X) = f(X')$) such that $f'(X) \neq f'(X')$, this would be a successful adversarial example.

B. Pick-n-Plug

A MT can be composed of a series of n operator functions $g_1, g_2, \ldots, g_n$, such that: $T(X) = g_n(\ldots g_2(g_1(X))\ldots)$. We propose Pick-n-Plug which relies on MT $T_{\mathrm{pick-n-plug}}$ to generate adversarial examples. It consists of a pick operator g_{pick} that draws a set of adversarial sentences, $\mathbf{S}_{\mathrm{adv}}$, from a particular passage in a source task data ($\mathcal{D}_{\mathrm{source}}$) and plug operator g_{plug} that injects these sentences into a passage from another task (target task), *Story*, without changing the correct answers to its question, *Question*. The g_{pick} step to draw sentences can be a random search, as we have explored in our experiments, or other search methods. With a suitable source text data, $\mathcal{D}_{\mathrm{source}}$, where sentences do not disrupt the information from the target task, Pick-n-Plug can generate adversarial examples while ensuring grammatical correctness. Table I shows a Pick-n-Plug examples where task #19 is the target task with source task # 3. More formally

$$\begin{aligned} T_{\mathrm{pick-n-plug}}(X) &= g_{\mathrm{plug}}(g_{\mathrm{pick}}(X)) = X', \quad \text{where} \\ g_{\mathrm{pick}}(X) &= (X, [S_1, \ldots, S_k]) \\ g_{\mathrm{plug}}(X, [S_1, \ldots, S_k]) &= X' \end{aligned}$$

TABLE I

PICK-N-PLUG ADVERSARIAL EXAMPLE GENERATED BY INJECTING TASK #3 SENTENCES (RED) INTO A TASK #19 PASSAGE, AFTER THE STORY SEGMENT AND RIGHT BEFORE THE QUESTION SENTENCE. IN THE ORIGINAL PASSAGE (BLACK) WITHOUT ADDITIONAL TASK #3 SENTENCES, THE DNC MODEL PREDICTS THE ANSWER CORRECTLY (NORTH, NORTH)

Target Task: #19 (path finding), Source Task: #3 (three argument relations), Injection Position: before question
Passage: The office is south of the hallway. The office is north of the garden. The kitchen is east of the bedroom. The kitchen is north of the hallway. The bathroom is north of the kitchen. John traveled to the kitchen. Mary journeyed to the hallway. John went back to the bedroom. John went to the garden. How do you go from the office to the kitchen?   **Prediction:** south, east   **Answer:** north, north

```
Algorithm 1 Pick-n-Plug ($T_{\text{pick-n-plug}}$)
  Input: Original samples $X$ with story of length $l_{story}$, source
  task data $\mathcal{D}_{\text{source}}$, number of adversarial sentences $k$
  Initialize $\mathbf{S}_{adv} = \{\}$
  for each adversarial sentence do
      Sample $S_i \sim \mathcal{D}_{\text{source}}$
      $\mathbf{S}_{adv} \leftarrow \mathbf{S}_{adv} \cup S_i$
  if before_story then
      $X' \leftarrow [\mathbf{S}_{adv}; X]$
  else if before_question then
      $X' \leftarrow [X_{:l_{story}}; \mathbf{S}_{adv}; X_{(l_{story}+1):}]$

  return $X'$
```

where $[S_1, \ldots, S_k] \subseteq \mathbf{S}_{adv}$, $S_i = [\mathbf{x}_1^{S_i}, \mathbf{x}_2^{S_i}, \ldots, \mathbf{x}_j^{S_i}]$ is a sequence of word vectors in an adversarial sentence, k is the number of adversarial sentences picked from the source task. When $k = 0$, this recovers the original passage where no adversarial sentence is used. X' is an adversarial input as the final output of the Pick-n-Plug MT $T_{\text{pick-n-plug}}$, as summarized in Algorithm 1.

C. Pick-Permute-Plug

We also propose Pick-Permute-Plug with MT $T_{\text{pick-permute-plug}}$ to extend the adversarial capability of Pick-n-Plug and further probe DNC's control signals in § VI. In Pick-n-Plug, the diversity of adversarial injected sentences is restricted by the text of the source task. With an additional g_{permute} operator after picking sentences (g_{pick}) from a source task, words in a particular adversarial sentence can be permuted with "synonyms" to generate a much wider range of possible sentences. The Pick-Permute-Plug MT can be summarized as

$$\begin{aligned}T_{\text{pick-permute-plug}}(X) &= g_{\text{plug}}(g_{\text{permute}}(g_{\text{pick}}(X))) = X' \\ &\text{where} \quad g_{\text{pick}}(X) = (X, [S_1, \ldots, S_k]) \\ &g_{\text{permute}}(X, [S_1, \ldots, S_k]) = (X, [S'_1, \ldots, S'_k]) \\ &g_{\text{plug}}(X, [S'_1, \ldots, S'_k]) = X'\end{aligned}$$

and $S'_i = [\mathbf{x}_1^{S'_i}, \mathbf{x}_2^{S'_i}, \ldots, \mathbf{x}_j^{S'_i}]$ is an adversarial sentence from the permute step such that $S'_i \neq S_i$ if one or more of its words have been substituted with synonyms.

In the same example above, the word "*hallway*" in Sentence 2: "*Mary journeyed to the hallway.*" can be substituted with "*office*" under the g_{permute} operator to generate Sentence 3: "*Mary journeyed to the office.*", before injecting into the story. For a story containing Sentence 1, if the correct answer to the question: "*How do you go from the office to the hallway?*" is "*north*", adding Sentence 3 to the story should also not change the correct answer. Other words can also be selected to be permuted such as substituting the name word '*Mary*' in Sentence 3 with '*John*'. The added flexibility allows for more control of the target DNC's predictions and behaviors with a wider range of possible changes in the input sequence. In practice, the permute step could be executed by greedily permuting synonyms over the Pick-Permute-Plug process iteratively with respect to the DNC's output confidence, to induce prediction of a target output with high confidence.

IV. PERFORMANCE WITH MR-BASED ADVERSARIES

We investigate how DNC performs while facing adversarial examples generated with Pick-n-Plug methods under several factors such as position where sentences are added, source of added sentences, number of added sentences, and type of targeted task.

A. Experiment Setup

Similar to previous work [3], [7], the DNC was jointly trained on en-10k data subset of all 20 bAbI tasks. During training, the batch size is 32, the controller is bidirectional and has 172 hidden units in each direction. The memory unit has a total of 192 rows, each with a width of 64 and 4 read heads. The DNC trained with an root mean square prop (RMSprop) optimizer with a fixed learning rate of 3e-05 and momentum of 0.9.

We evaluate DNC's performance when faced with test data sets that are augmented with Pick-n-Plug, in a black-box manner where g_{pick} and g_{plug} are independent of the DNC's weights and predictions since adversarial sentences are sampled randomly from their source task.

B. Results and Discussion

The error rate of the DNC increases as more sentences are added to the original passage. For some cases where the number of added sentences are small, the disruptive effect of the position before the story is larger. When more sentences are added, the insertion of sentences just before questions results in higher error rates than the insertion at the start of the story (Table II).

Intuitively, the effect of adversarial sentences inserted at the beginning of the story can be thought as to mislead the DNC in the wrong direction from the start, focusing on details of the story that are not relevant in answering the question. In contrast, the adversarial sentences right before the question might cause the DNC to erase data in its memory that is important to correctly answer the question at the end, as a price of storing data from the relatively more recent adversarial sentences. This implies that, as the length of adversarial sentences increases, the effect of adversarial sentences in overwriting relevant data outweighs the effect of misleading the DNC's attention to a less relevant direction.

Adversarial sentences from source task #19 generally degrade the DNC's performance more than adversarial sentences from other 4 source tasks (#3, 15, 16, and 18), as shown in Table II. A possible reason is that the distracting strength of adversarial sentences lies in the amount of information they contain. In task #19, directional relationships between two locations are expressed in each sentence. This roughly represents a change to two entities' attribute per sentence. Furthermore, Pick-n-Plug examples are able to degrade the DNC's performance across all tasks (Tables II and IV).

TABLE II

WORD ERROR RATE OR WER (%) OF DNC ON TASK #3, #15, #16, #18, AND #19 WITH PICK-N-PLUG EXAMPLES ORIGINAL TEST WER FOR TASK #3, #15, #16, #18, AND #19 ARE 1.6%, 0%, 0%, 0.623%, AND 0%, RESPECTIVELY

Source	Position	# of Adversarial Sentences				Full Block
		1	2	3	4	
Target Task 3						
#15	before story	3.6	2.8	3.3	4.8	21.0
	before question	2.7	2.9	3.6	5.9	22.4
#16	before story	4.6	3.9	4.9	8.1	23.9
	before question	2.5	2.6	4.4	6.3	26
#18	before story	3.3	4.3	7	8.7	14.3
	before question	4.1	8.7	12.8	14.2	28.6
#19	before story	4.5	6.7	8.3	9.5	12.0
	before question	**6.4**	**10.5**	**15.2**	**20.4**	**26.5**
Target Task 15						
#3	before story	0.6	5	9.6	13.6	88.4
	before question	0.7	3.9	8.4	13.9	**98.5**
#16	before story	1.9	6.6	13.3	24.2	56.4
	before question	1.4	5.4	11.9	23.1	60.3
#18	before story	**4.6**	16.8	27.5	37.1	51.3
	before question	4.5	17.5	31.1	44.1	65.1
#19	before story	4.4	**22.4**	**40.9**	53.7	64.2
	before question	3.3	17.8	37.5	**55**	65.8
Target Task 16						
#3	before story	0.4	0.9	0.7	0.7	24.5
	before question	0.1	1.1	1.3	1.3	**94.2**
#15	before story	**2.8**	2	3.9	**7.7**	27.4
	before question	0.2	1	1.8	2.5	12.9
#18	before story	0.3	0.8	1.1	1.3	1.8
	before question	0.2	0.9	1.7	2.7	12
#19	before story	1.3	1.9	2.6	4.3	5.4
	before question	0.6	**2.5**	**4.1**	6.9	10.8
Target Task 18						
#3	before story	1.2	2.8	3.3	4	32.6
	before question	1	2.8	3.4	3.6	27.9
#15	before story	1.1	3	5.1	7.9	40.4
	before question	2	3.9	7.7	10.8	47.5
#16	before story	1.8	4.7	10.4	19.3	54.1
	before question	2.4	6.2	**16.4**	**24.9**	**63.7**
#19	before story	**4.6**	5.8	7.1	9.3	10.6
	before question	3.7	**6.7**	10.5	12	13.5
Target Task 19						
#3	before story	0.2	0.3	0.5	0.85	10.2
	before question	0.3	0.5	1.05	1.75	**51.4**
#15	before story	0.55	1.2	4	7.15	28.6
	before question	**0.9**	**2.95**	**8.55**	**14.8**	37.0
#16	before story	0.4	2.4	4.45	6.75	19.1
	before question	0.6	2.85	5.5	9.1	36.1
#18	before story	0.05	0.25	0.6	0.95	2.45
	before question	0.45	1.1	2.7	4.85	12.6

TABLE III

WORD ERROR RATE OR WER (%) OF DNC ON TASK #15 WITH PICK-N-PLUG AND WORD SUBSTITUTION ATTACK. PICK-N-PLUG ADVERSARIAL SENTENCES ARE ADDED BEFORE QUESTION. ORIGINAL TEST WER FOR TASK #15 IS 0%

Attack	# of Adv Sentences/Substitutions			
	1	2	3	4
Pick-n-Plug, Source #3	0.7	3.9	8.4	13.9
Pick-n-Plug, Source #19	3.3	17.8	37.5	55
Word Substitution	7.65	13.1	14.4	18

TABLE IV

WER (%) OF DNC ON REMAINING 15 BABI TASKS WITH PICK-N-PLUG EXAMPLES

	Target Task							
Source	1	2	4	5	6	7	8	9
#15	14	32.8	0.3	2.7	0.8	12.9	7.48	0.5
#16	45.2	32.3	0.6	7.9	14.7	11.8	15.4	6.8
clean	0.2	0.5	0	0.8	0.4	0.3	0.09	0

	Target Task						
Source	10	11	12	13	14	17	20
#15	2.4	20.5	31.7	47.5	2.7	21.8	50
#16	10.7	48.3	47.1	56.8	10.7	55.9	53.4
clean	0	0	0	0	0.2	0.7	0

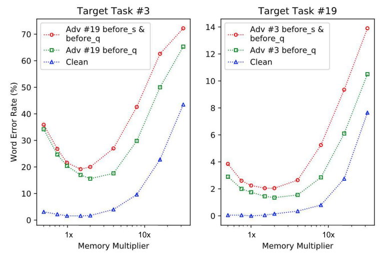

Fig. 2. (L) WER (%) of DNC with different sizes of augmented memory module.

C. Comparison With Word Substitution Attack

We also compare with a word substitution-based attacks, similar to [16], where words in the passage are substituted with synonyms to fool the DNC model. To adapt this substitution attack to Task #15, up to 4 unique name words in each test samples can be substituted with another random name in the word vocabulary. To ensure that the ground truth labels still hold for these test samples, we maintain the same name substitution throughout each sample. From Table III, we can see that the word substitution attacks increase the error rate of DNC with more word substitutions. At the maximum strength of 4 name substitution, we see error rate of 18% which is still lower than the strongest Pick-n-Plug scenario (55%), where 4 sentences are sourced from Task #19. This highlights the importance of investigating the vulnerability exposed by Pick-n-Plug, a previously unexplored attack.

V. ROLE OF DNC MEMORY MODULE

The augmented memory module of DNC facilitates storing information over long timescales. Since the adversarial sentences may deleteriously overwrite relevant stored information, a bigger size of the DNC's memory module might mitigate this effect by having more free space to write new information rather than overwriting important data. We study this approach by expanding the DNC's memory size.

A. Experiment Setup

We carry out experiments with Pick-n-Plug on DNC of memory size 0.5×, 0.75×, 1×, 2×, 4×, 8×, 16×, and 32× of the original 192 memory rows. The performance of DNC with these memory sizes is also evaluated on clean test data set as a baseline.

B. Results and Discussion

Fig. 2 shows that a bigger memory size plays a limited role in improving DNC's robustness when presented with examples from Pick-n-Plug. The error rates of DNC drop to a minimum (9% to 24% relative reduction in WER from the original memory size) in our experiments when memory size is either 2× or 1.5× of its original. However, as the memory size increases further, the DNC's error rates increase even past its original error rate at 1× memory. The error rate of the DNC stands at 15.6% at its best performance when task #3 passages are added with one block of 4 adversarial sentences before questions.

Its performance on clean test samples is also degraded (Fig. 2). This occurs earlier than the degradation of performance with adversarial examples in our experiments. A possible explanation to this might be that the DNC controller was overfitted to the original

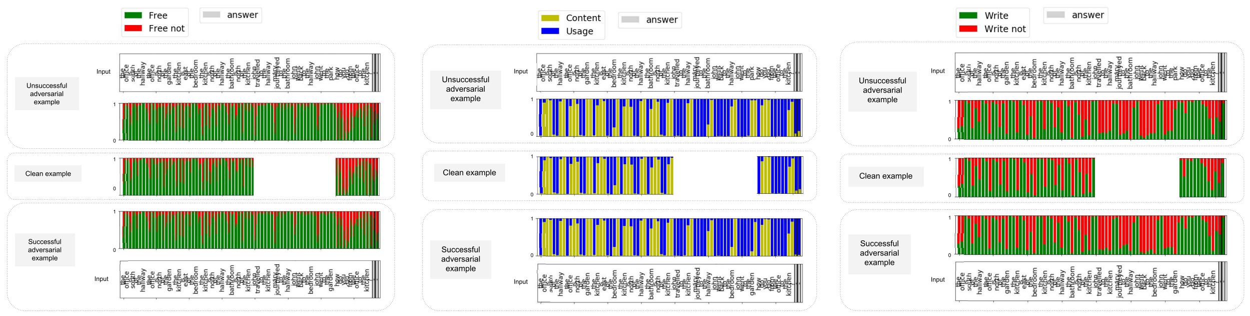

Fig. 3. (L-R) Free gate, allocation gate, and write gate values of DNC's read heads when the input sequence is (Top) an unsuccessful adversarial input, (Middle) a CE, and (Bottom) a successful adversarial input.

memory size and is unable to optimally handle memory modules of different sizes, even though they are compatible. This degradation due to the overfitting of DNC's controller to original memory size may outweigh the benefit of larger storage space, resulting in a maximum robust performance point close to the original memory size, 1.5× to 2× in our experiments. This also explains why a smaller memory that is closer to the original size can outperform a larger memory that is much further.

VI. Adversarial Effect on DNC Controller

We investigate the behaviors of DNC controller under different inputs, 1) clean input [clean example (CE)], 2) unsuccessful adversarial attack (UAA), and 3) successful adversarial attack (SAA) by probing its control signals. UAA is sampled from a Pick-n-Plug run on target task #19 that picks 4 adversarial sentences from source task #3, and inject them right before every question. To generate closely related versions of this UAA sample, we conduct a run of Pick-Permute-Plug where 4 location words in this UAA's adversarial sentences are permuted with a set of 8 synonyms in a brute force manner, while the rest of the adversarial sentences remains the same. Among these new adversarial examples which successfully caused the DNC to predict the incorrect answer, a SAA was sampled. These control signals comprise 3 gates, 2 keys, and 2 vectors. Gate values are scalar that range from 0 to 1, while keys and vectors are W-dimensional vectors with real values. For similarity comparison of 2 sequences of scalar values, like the gate values, we can use normalized Kullback–Leibler (KL)-divergence. We can use cosine similarity to compare vector-based control signals like the write/read keys and write/erase vectors.

Results & Discussion: For all pair-wise comparisons and for all of DNC's controller keys and vectors (see Table V), the mean cosine similarities at the story segment are significantly higher than the mean cosine similarities at the question segment. This indicates that the main adversarial effect in disrupting the DNC's controller keys and vectors emerge mainly at sections after the adversarial sentences are injected rather than before that.

When compared with CE, the cosine similarities of controller keys and vectors from SAA are lower than that from UAA, suggesting that the keys and vectors from a CE are perturbed more under a successful attack than an unsuccessful one. When comparing the DNC's controller keys and vectors under UAA and SAA, the cosine similarities between all of them-the write keys, read keys, write vectors, and erase vectors-are lowest in the adversarial segment. From Fig. 4, there are sharp drops in the similarities of the keys and vectors at the time-step where the input words are different. Their positions correspond to keywords ["*hallway*", "*bathroom*", "*hallway*", "*park*"] in the UAA and ["*kitchen*", "*hallway*", "*bedroom*", "*garden*"] in the SAA.

TABLE V
MEAN COSINE SIMILARITY BETWEEN DNC'S KEYS AND VECTORS WITH DIFFERENT INPUTS

		Story	Adversary	Question
Write Keys	CE-UAA	0.9969	-	0.9790
	CE-SAA	0.9962	-	**0.9627**
	UAA-SAA	0.9997	**0.8593**	0.9689
Read Keys	CE-UAA	0.9981	-	0.9749
	CE-SAA	0.9980	-	**0.9652**
	UAA-SAA	0.9999	**0.8417**	0.9737
Write Vectors	CE-UAA	0.9988	-	0.9799
	CE-SAA	0.9988	-	**0.9653**
	UAA-SAA	0.9999	**0.7400**	0.9761
Erase Vectors	CE-UAA	0.9996	-	0.9975
	CE-SAA	0.9995	-	**0.9942**
	UAA-SAA	1.000	**0.9660**	0.9957

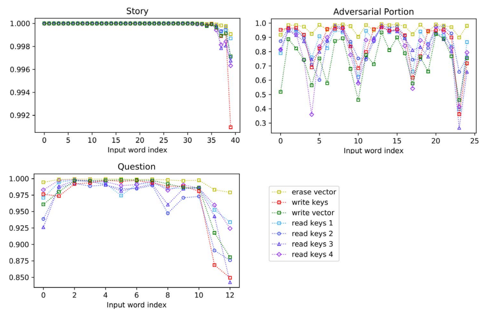

Fig. 4. Cosine similarity of DNC's keys and vectors, between when the DNC is presented with a UAA and when it is presented with a SAA. (Top Left) Cosine similarity values at the story. (Top Right) Adversarial portion. (Bottom) Question of a sample QA from task #19.

While cosine similarities are relatively stable in question segment, there is a sharp drop at the end where the answer is expected from the DNC. This suggests that the adversarial sentences have a latent effect on the keys and vectors which emerges in critical segments such as when information is retrieved to answer a question. These observations indicate that disruptions to these keys and vectors, which are involved in DNC's write, read, and erase operations, play a part of the overall adversarial effect from a successful attack. For gate values, we find no obvious difference between the patterns of the gate values when DNC is presented with a CE, UAA, and SAA from a general view (see Fig. 3).

When the KL-divergence is used to compare the gate values, significant patterns appear at the different segments of the input sequences. For all 3 types of gate values at the story segment (see Table VI), the KL-divergence of all pair-wise comparisons is significantly lower

TABLE VI

KL-Divergence of DNC Controller's Gate Values With Different Inputs

		Story	Adversary	Question
Free	CE-UAA	3.735E-05	-	0.002472
	CE-SAA	2.526E-05	-	0.02041
	UAA-SAA	2.212E-06	0.005523	0.02470
Alloc	CE-UAA	0.0003625	-	0.001112
	CE-SAA	0.0003089	-	0.01512
	UAA-SAA	3.730E-06	0.2279	0.02437
Write	CE-UAA	0.0003325	-	0.005531
	CE-SAA	0.0002664	-	0.007648
	UAA-SAA	1.369E-05	0.2408	0.001681

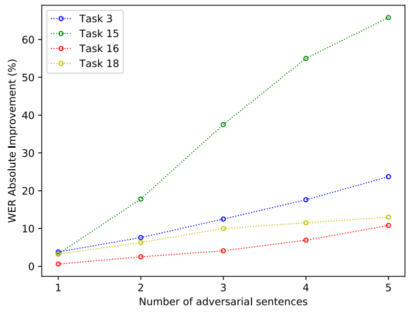

Fig. 5. Absolute WER reduction (%) of DNC on task #3, 15, 16, and 18 with Pick-n-Plug examples, showing that augmented training with Pick-n-Plug can mitigate vulnerabilities and generalize to examples with different numbers of adversarial sentences.

than the KL-divergence at the question segment, with 2 to 3 order of magnitude difference. This implies that the main adversarial effect on DNC's 3 gate values emerges after the injection of adversarial sentences, rather than before that. At the question segment, the KL-divergence of DNC's gate values from CE under SAA is higher than that under UAA for all the 3 gate types, indicating that the gate values are perturbed to a greater extent in a successful attack than an unsuccessful one.

VII. Robustness Enhancement With Pick-n-Plug

We use Pick-n-Plug to augment the training data set for adversarial training and found that this can mitigate the vulnerability and generalize beyond several key aspects.

A. Experiment Setup

We generate adversarial training data by using Pick-n-Plug to pick 3 adversarial sentences at each *pick* step from the training data set of task #19 and plugging them into training data sets of task #3, 15, 16, and 18 before questions. These 4 data sets were then combined with the original training data sets of 16 other tasks to form the training data.

B. Results and Discussion

Despite being only trained on adversarial examples that inject 3 adversarial sentences at each plug step, the adversarially trained DNC can resist adversarial examples with other numbers of adversarial sentences (Fig. 5), varying from 1 to 5.

Table VII shows that DNC trained on adversarial examples with a particular injection position can also resist examples with a different injection position. All our adversarially trained models have lower error rates than the nonadversarially trained model, regardless of their training and test injection positions.

TABLE VII

Mean WER (%) of DNC on Task #3, 15, 16, and 18 With Pick-n-Plug Examples, With Adversarial Sentences Injected at 3 Different Positions

Adv. Test Position	Adv. Training Position			w/o Adv. Training
	before_s	random	before_q	
before_s	0.633	0.725	0.775	14.7
random	0.606	0.612	0.831	2.11
before_q	0.775	1.63	0.8	16.8
clean	0.718	0.6	0.8	0.556

TABLE VIII

WER (%) of DNC When Tested With Pick-n-Plug Examples on 14 Tasks

	Target Task						
DNC Model	1	2	5	6	7	8	9
Adv. Trained	8.4	2.7	1	4.7	0.4	0.187	2.5
Baseline	24.8	32.5	5.2	6.7	11.1	22	4.7
	Target Task						
	10	11	12	13	14	17	20
Adv. Trained	2.4	5.1	13.1	20.1	2.8	17.5	14.1
Baseline	5.3	19.6	42.2	50.1	18.4	41.6	48.9

TABLE IX

WER (%) of DNC When Tested With Adversarial Examples Augmented With Sentences From Source Task #3, 15, 16, and 18

Source	DNC Model	Target Task			
		#3	#15	#16	#18
#3	Adv. Trained	-	0.5	0.2	1.3
	Baseline	-	14.2	1.5	4.2
#15	Adv. Trained	4.8	-	2.3	1.2
	Baseline	4.4	-	2.9	10.6
#16	Adv. Trained	7.1	6.6	-	15.2
	Baseline	8.1	24.5	-	26.2
#18	Adv. Trained	7.5	10.9	3.3	-
	Baseline	14.2	44.1	2.7	-

Even for target tasks whose training data were not adversarially augmented, the adversarially trained DNC model can resist examples on these tasks more effectively than the baseline model during test time (Table VIII). Target task #4 is omitted as it is not compatible with source task #19. No adversarial examples from these 14 tasks are used in the adversarial training phase, yet there is improved robustness in these tasks. This generalization to the other text and tasks which demand different capabilities highlights the practicality of our adversarial generation methods to improve robustness in other NLP applications.

For most of the cases after adversarial training, the DNC can resist adversarial examples more effectively even though these examples are generated from sources different from the one it was trained on (Table IX). With the possibility of adaptive adversaries and diversity of text content in the real world, the generality of robustness gained from our method accentuates their practical value.

VIII. Conclusion

We expose vulnerabilities in modern neural reasoning architectures with adversarial examples in text-based logical reasoning. Our adversarial examples are based on MRs, inspired by real-world text where sentences may contain nonoverwriting information. We show that a training procedure augmented with Pick-n-Plug can be an effective solution that generalizes beyond the number of additional sentences, positions, types of tasks, and sentences. We demonstrate that our method can be a valuable technique in discovering and mitigating these vulnerabilities.

References

[1] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," 2014, *arXiv:1410.5401*. [Online]. Available: http://arxiv.org/abs/1410.5401

[2] S. Sukhbaatar *et al.*, "End-to-end memory networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2440–2448.

[3] A. Graves *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, p. 471, 2016.

[4] M. Henaff, J. Weston, A. Szlam, A. Bordes, and Y. LeCun, "Tracking the world state with recurrent entity networks," 2016, *arXiv:1612.03969*. [Online]. Available: http://arxiv.org/abs/1612.03969

[5] E. Parisotto and R. Salakhutdinov, "Neural map: Structured memory for deep reinforcement learning," 2017, *arXiv:1702.08360*. [Online]. Available: http://arxiv.org/abs/1702.08360

[6] S. Back, S. Yu, S. R. Indurthi, J. Kim, and J. Choo, "MemoReader: Large-scale reading comprehension through neural memory controller," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2018, pp. 2131–2140.

[7] J. Franke, J. Niehues, and A. Waibel, "Robust and scalable differentiable neural computer for question answering," in *Proc. Workshop Mach. Reading Question Answering*, 2018, pp. 47–59.

[8] C. Szegedy *et al.*, "Intriguing properties of neural networks," 2013, *arXiv:1312.6199*. [Online]. Available: http://arxiv.org/abs/1312.6199

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. ICLR*, 2015, pp. 1–11.

[10] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2017, *arXiv:1706.06083*. [Online]. Available: http://arxiv.org/abs/1706.06083

[11] R. Jia and P. Liang, "Adversarial examples for evaluating reading comprehension systems," 2017, *arXiv:1707.07328*. [Online]. Available: https://arxiv.org/abs/1707.07328

[12] Y. Qin, N. Carlini, G. Cottrell, I. Goodfellow, and C. Raffel, "Imperceptible, robust, and targeted adversarial examples for automatic speech recognition," in *Proc. 36th Int. Conf. Mach. Learn.*, in Proceedings of Machine Learning Research, vol. 97, K. Chaudhuri and R. Salakhutdinov, Eds. MLResearchPress, 2019, pp. 5231–5240.

[13] T. Zheng, C. Chen, and K. Ren, "Distributionally adversarial attack," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 2253–2260. pp. .

[14] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. El Ghaoui, and M. I. Jordan, "Theoretically principled trade-off between robustness and accuracy," 2019, *arXiv:1901.08573*. [Online]. Available: http://arxiv.org/abs/1901.08573

[15] W. E. Zhang, Q. Z. Sheng, A. Alhazmi, and C. L. Li, "Adversarial attacks on deep-learning models in natural language processing: A survey," *ACM Trans. Intell. Syst. Technol.*, vol. 11, no. 3, pp. 1–41, 2020.

[16] M. Alzantot, Y. Sharma, A. Elgohary, B. Ho, M. B. Srivastava, and K. Chang, "Generating natural language adversarial examples," 2018, *arXiv:1804.07998*. [Online]. Available: https://arxiv.org/abs/1804.07998

[17] N. Papernot, P. McDaniel, A. Swami, and R. Harang, "Crafting adversarial input sequences for recurrent neural networks," in *Proc. MILCOM IEEE Mil. Commun. Conf.*, Nov. 2016, pp. 49–54.

[18] V. Kuleshov, S. Thakoor, T. Lau, and S. Ermon, "Adversarial examples for natural language classification problems," *OpenReview*, to be published.

[19] E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh, "Universal adversarial triggers for attacking and analyzing NLP," 2019, *arXiv:1908.07125*. [Online]. Available: http://arxiv.org/abs/1908.07125

[20] T. Miyato, A. M. Dai, and I. Goodfellow, "Adversarial training methods for semi-supervised text classification," 2016, *arXiv:1605.07725*. [Online]. Available: http://arxiv.org/abs/1605.07725

[21] X. Wang, H. Jin, and K. He, "Natural language adversarial attacks and defenses in word level," 2019, *arXiv:1909.06723*. [Online]. Available: http://arxiv.org/abs/1909.06723

[22] D. Pruthi, B. Dhingra, and Z. C. Lipton, "Combating adversarial misspellings with robust word recognition," 2019, *arXiv:1905.11268*. [Online]. Available: http://arxiv.org/abs/1905.11268

[23] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach, "Defense methods against adversarial examples for recurrent neural networks," 2019, *arXiv:1901.09963*. [Online]. Available: http://arxiv.org/abs/1901.09963

[24] Y. Zhou, X. Zheng, C.-J. Hsieh, K.-w. Chang, and X. Huang, "Defense against adversarial attacks in NLP via Dirichlet neighborhood ensemble," 2020, *arXiv:2006.11627*. [Online]. Available: http://arxiv.org/abs/2006.11627

[25] M. Sato, J. Suzuki, H. Shindo, and Y. Matsumoto, "Interpretable adversarial perturbation in input embedding space for text," 2018, *arXiv:1805.02917*. [Online]. Available: https://arxiv.org/abs/1805.02917

[26] A. Chan, Y. Tay, Y.-S. Ong, and A. Zhang, "Poison attacks against text datasets with conditional adversarially regularized autoencoder," 2020, *arXiv:2010.02684*. [Online]. Available: http://arxiv.org/abs/2010.02684

[27] J. Weston *et al.*, "Towards AI-complete question answering: A set of prerequisite toy tasks," 2015, *arXiv:1502.05698*. [Online]. Available: http://arxiv.org/abs/1502.05698

[28] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Dept. Comput. Sci., Hong Kong Univ. Sci. Technol., Hong Kong, Tech. Rep. HKUST-CS98-01 1998.

[29] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *J. Syst. Softw.*, vol. 84, no. 4, pp. 544–558, Apr. 2011.