GUIDED SEQUENCE-TO-SEQUENCE LEARNING WITH EXTERNAL RULE MEMORY

Jiatao Gu[⋆] **Baotian Hu**[†] **Zhengdong Lu**[‡] **Hang Li**[‡] **Victor O.K. Li**[⋆]

[⋆]Department of Electrical and Electronic Engineering, The University of Hong Kong
{jiataogu, vli}@eee.hku.hk
[†]Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China
baotianchina@gmail.com
[‡]Huawei Noah's Ark Lab, Hong Kong
{lu.zhengdong, hangli.hl}@huawei.com

ABSTRACT

External memory has been proven to be essential for the success of neural network-based systems on many tasks, including Question-Answering, classification, machine translation and reasoning. In all those models the memory is used to store instance representations of multiple levels, analogous to "data" in the Von Neumann architecture of a computer, while the "instructions" are stored in the weights. In this paper, we however propose to use the memory for storing part of the instructions, and more specifically, the transformation rules in sequence-to-sequence learning tasks, in an external memory attached to a neural system. This memory can be accessed both by the neural network and by the human experts, hence serving as an interface for a novel learning paradigm where not only the instances but also the rule can be taught to the neural network. Our empirical study on a synthetic but challenging dataset verifies that our model is effective.

1 INTRODUCTION

Quite recently, external memory has been re-introduced to deep learning [Graves et al. (2014); Weston et al. (2014)], and is partially responsible for empirical success of neural network-based system on several challenging tasks, including but not limited to machine translation, Question-Answering, and reasoning. In those models, external memory is used mostly as an extension to the usual vectorial representation of instances, providing more flexibility not only in the format and the space limit of instance representation, but also in the way information is accessed. Unlike a computer, the instructions for the neural networks are essentially formed through a learning process in which instances and the desired outputs are given, and a learning algorithm (e.g., back-propagation with a certain discrepancy function) adjusts the weights.

In this paper, we propose the framework as "Guided Sequence-to-sequence (GUIDED-SEQ2SEQ) Learning" to use the memory for storing part of the instructions, in particular the transformation rules. As another contribution of this paper, we propose to use a hybrid addressing strategy for executing rules. To be more specific, the input sequences are represented as a mixture of distributed representation and symbol sequences saved in a short-term memory. This particular design of representation nicely combines the flexibility of the neural model and the rigour of the symbolic form.

Learning Objectives: We redefine the SEQ2SEQ problem by allowing the guidance of transformation rules. More specifically, adding a new rule consists of encoding it, putting it into rule memory, and letting the encoder-decoder learn the transformation specified in the rule. Once a new rule is added, it should be able to fetch this new rule from rule-memory when the system sees an applicable instance and execute it properly, while this ability is learned by supervised learning from examples.

2 MODEL OVERVIEW

Overall Architecture: Our model for GUIDED-SEQ2SEQ learning consists of an encoder, a decoder and an external rule memory, as illustrated in Figure 1. Given any instance, the encoder can interact with the rule memory to find the appropriate rule, and apply this rule to the input sequence to

form the representation in the short-term memory (STM). The STM has two sections, the distributed section for a input sequence representation, and the symbol section for saving the symbol substrings. For a certain substring in the symbol section, there will be a segment of memory allocated for the one hot representation for each symbol in it, and there is a certain indicate vector that can be recognised by the neural network. We have specifically designed Recurrent Neural Networks (RNNs) to generate this hybrid representation. In decoding, the decoder first reads the distributed section of the STM, which further triggers its reading of the symbol section, and finally generate the output sequence.

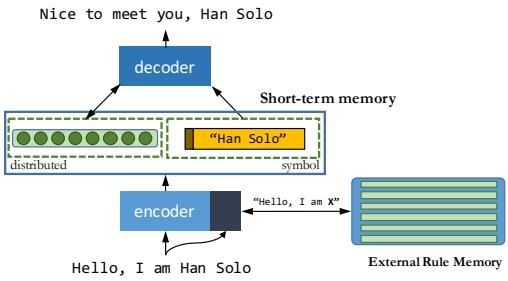

Figure 1: Given an input sequence, the system automatically accesses the external memory for the matched rule, and guides the sequence-to-sequence transformation on a hybrid representation stored in a short-term memory.

Rule-guided Execution: We utilise an explicit memory for storing rules, so the rules can be fetched and executed when needed. Introducing the external memory structure into a neural system has been recently explored such as [Weston et al. (2014); Sukhbaatar et al. (2015)]. We first encode the source parts of the rules using an RNN-encoder into fixed length vectors, and stack these vectors to form a memory matrix.

Given an instance I_{S}, we adopt another RNN-encoder to convert it to a vector i_{S}, for retrieving the corresponding rules in the memory, as illustrated in Figure 2. To make every step differentiable, reading is computed based on the soft attention mechanism as in [Bahdanau et al. (2014)]:

$$p_t = \text{softmax}\left(e(a_t, i_{\mathsf{S}})\right),\ t \in [1, n]; \quad c = \sum_{t=1}^{n} p_t c_t \tag{1}$$

where $e(\cdot)$ is the scoring function for the correspondence between instance input and rules, which is commonly formed as either inner product (INP) or a 2-layer network (DNN).

The readout vector c is used for an execution network. We model it by regarding the variables as pointers to locations storing the address of the replacing substring. The problem then becomes extracting the substring from the source instance, for which we adopt a structure similar to the Pointer Network (PtrNet) [Vinyals et al. (2015)]. In Figure 2, the input instance will be separately delivered to the external memory and the PtrNet, where the readout memory c will be used to encode together with the original input instance by another encoder, yielding an array of hidden states $\{s_1, ..., s_{T_{\mathsf{S}}}\}$. In the decoding phase, a specific RNN-decoder computes the hidden state h_t with the inputs x_t (typically the embedding of the previous predicted position y_{t-1}), and outputs a probability to point to the next position as follows,

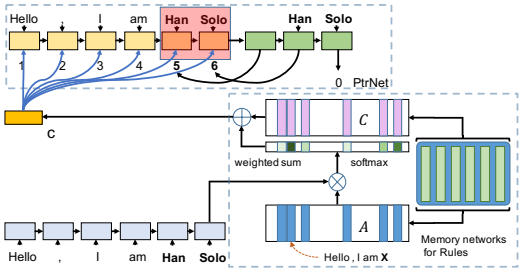

Figure 2: A framework of rule fetching and execution networks. The execution part is built with a Pointer Network.

$$P(R_{\mathsf{S}}, E|I_{\mathsf{S}}) = \prod_{t=1}^{T} P(y_t|x_{1:t}, I_{\mathsf{S}}); \quad P(y_t|x_{1:t}, I_{\mathsf{S}}) = \text{softmax}\left(e(s_{1:T_{\mathsf{S}}}, h_t)\right) \tag{2}$$

where T is the decoding length and the function $e(\cdot)$ is the scoring function for the correlation between current state and each input position. Generally, we compute it with a 2-layer neural network.

Hybrid Encoder-Decoder: Based on the execution results, any instance will be naturally transferred to its associated rule sequence as well as the extracted substring. Further, the rule sequence is pushed to the encoder of SEQ2SEQ learning to form a vectorial representation. On the other hand, analogous to the conventional SEQ2SEQ learning, a decoder produces the transformation results based on the hybrid STM in the GUIDED-SEQ2SEQ learning, which is also a hybrid. In detail, a standard RNN-decoder is utilised to predict the target rule sequence R_{T}, and the variables working as pointers are replaced directly with the memorised substrings for the target instance I_{T}.

Learning: Our proposed model decomposes the learning into two supervised learning sub-tasks:

First, use supervised learning of execution jointly with rule fetching and storing, with objectives:

$$\mathcal{L} = -\mathbb{E}_{I_{\mathsf{S}}}\left[\log P(R_{\mathsf{S}}|I_{\mathsf{S}};\mathcal{R})\right]; \quad R_{\mathsf{S}} = \begin{cases} \mathrm{T}(I_{\mathsf{S}}) & \textit{with rule} \\ I_{\mathsf{S}} & \textit{no rule} \end{cases} \tag{3}$$

where $\mathcal{R}$ is the rule set stored in the rule memory, and the function $\mathrm{T}(\cdot)$ is used to transfer instances to their corresponding rules. It is also worth noting that the learning described should be carried out from regular instances, for which no applicable rules are provided. To support this, the input instances for training the rules contain two categories, *With rule* (instances that has the associated rules in the rule memory) and *No rule* (otherwise). Moreover, adding an additional supervision on memory attention is efficient way to help directly guide the distribution for fetching correct rules.

Second, use supervised learning of transformation which is regarded as a standard encoder-decoder model. The transformation is similar to a conventional encoder-decoder model for machine translation or dialogue system, hence a similar learning algorithm to maximise the log-likelihood. Both the rules and instances (regardless of whatever there is an actual rule execution or not) are represented as the same form of hybrid representation in the STM, and directly make use of the decoded pointers to replace the symbols, fusing to the final output sequence.

3 EXPERIMENTS

Dataset Generation: To evaluate the new proposed framework, a synthetic dataset named "Simple Rules for SEQ2SEQ Learning (SRSS)" is created. The dataset contains symbols with a vocabulary of 1,000 words (with indices from 000 to 999), and two additional variables **X**, **Y**. We first generate prototype sequences by randomly choosing words from the vocabulary with length ranging from 5 to 20 words, and pair sequences as source and target one by one. Rules are obtained from the prototypes by replacing some words with the variables **X** or **Y** on both sides, and further yielding instances by replacing the same variables with arbitrary substrings. The substrings are randomly sampled within the length from 1 to 15, with examples given in Table 1(a). Although the SRSS dataset is designed for a general purpose of SEQ2SEQ learning, it has difficulty and scalability to extend to various real-world natural language applications.

Table 1: The example of the SRSS dataset (a), and the experimental results (b) showing the accuracy on unseen instances associated with old rules (learned for execution before) and newly added rules. "Ptr" means the rule execution part, and "EM" refers to the external rule memory. Results of rule-fetching (DNN, INP), with and without the supervision on memory attention (S) are reported.

(a) Examples for the generated SRSS dataset

Type	Examples	
	Source	Target
Proto.	$w^{(110)}w^{(332)}$	$w^{(111)}w^{(789)}$
Rule	$w^{(110)}\mathbf{X}$	$\mathbf{X}w^{(789)}$
Inst.	$w^{(110)}\boldsymbol{w}^{(233)}\boldsymbol{w}^{(233)}$	$\boldsymbol{w}^{(233)}\boldsymbol{w}^{(233)}w^{(789)}$
	$w^{(110)}\boldsymbol{w}^{(134)}\boldsymbol{w}^{(135)}$	$\boldsymbol{w}^{(134)}\boldsymbol{w}^{(135)}w^{(789)}$
	$w^{(110)}\boldsymbol{w}^{(783)}\boldsymbol{w}^{(001)}$	$\boldsymbol{w}^{(783)}\boldsymbol{w}^{(001)}w^{(789)}$

(b) Results for decoding accuracy

Approaches	old rules (%)		new rules(%)	
	with rule	no rule	with	no rule
Rule Enc/Dec	00.0	–	00.0	–
Ptr (without EM)	40.7	28.3	00.0	46.1
Ptr-EM (DNN)	47.8	22.9	00.1	57.8
Ptr-EM (DNN+S)	91.1	93.7	68.7	80.9
Ptr-EM (INP)	93.4	95.7	64.5	83.9
Ptr-EM (INP+S)	90.4	95.8	**69.1**	**84.5**

Training and Results: Following the algorithm of GUIDED-SEQ2SEQ learning, we separate the training into two networks where transformation can be efficiently learned on the selected rule set considering the number of rules is limited. To achieve this, we randomly generated 5000 different rules from the SRSS dataset and tuned the encoder-decoder carefully. Then, 90% of the selected rules are applied to train the rule execution, and for each rule 20 instances are generated.

In Table 1(b), the accuracy of both instances with rules in memory and pure instances are reported on different model variants. Two baselines, the standard encoder-decoder learned on the rule sets and an execution network with the memory inputs omitted. It is clear that although by execution with the external memory inputs, the new framework can be easily applied on newly added rules without learning any corresponding instances, fetching the correct rules from the external memory for execution is crucial. The model with inner product attention functions and additional supervision achieves the best scores for new rules. Moreover, as shown in the table, pure instances with no rules can also be processed properly, which implies that our model has the ability to dynamically choose to use or not to use rules for given instances, and generate correct decoding results.

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in Neural Information Processing Systems*, pp. 2431–2439, 2015.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pp. 2674–2682, 2015.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.