Neural Sequence-to-grid Module for Learning Symbolic Rules

Segwang Kim, [1] **Hyoungwook Nam,** [2] **Joonyoung Kim,** [1] **Kyomin Jung**[1]
[1] Department of Electrical and Computer Engineering, Seoul National University, Seoul, Republic of Korea
[2] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA
{ksk5693, kimjymcl, kjung}@snu.ac.kr, hn5@illinois.edu

Abstract

Logical reasoning tasks over symbols, such as learning arithmetic operations and computer program evaluations, have become challenges to deep learning. In particular, even state-of-the-art neural networks fail to achieve *out-of-distribution* (OOD) generalization of symbolic reasoning tasks, whereas humans can easily extend learned symbolic rules. To resolve this difficulty, we propose a neural sequence-to-grid (seq2grid) module, an input preprocessor that automatically segments and aligns an input sequence into a grid. As our module outputs a grid via a novel differentiable mapping, any neural network structure taking a grid input, such as ResNet or TextCNN, can be jointly trained with our module in an end-to-end fashion. Extensive experiments show that neural networks having our module as an input preprocessor achieve OOD generalization on various arithmetic and algorithmic problems including number sequence prediction problems, algebraic word problems, and computer program evaluation problems while other state-of-the-art sequence transduction models cannot. Moreover, we verify that our module enhances TextCNN to solve the bAbI QA tasks without external memory.

Introduction

Symbolic reasoning tasks such as learning arithmetic operations or evaluating computer programs offer solid standards for validating the logical inference abilities of deep learning models. Among machine learning tasks, symbolic reasoning problems are apt for testing mathematical, algorithmic, and systematic reasoning as they have strict rules mapping a given input to a well-defined unique target. In particular, a large body of works on deep learning has considered sequence transduction problems for symbolic reasoning. Some symbolic problems such as copying sequences (Dehghani et al. 2018; Graves, Wayne, and Danihelka 2014; Grefenstette et al. 2015; Rae et al. 2016; Zaremba and Sutskever 2014) and arithmetic addition (Graves, Wayne, and Danihelka 2014; Joulin and Mikolov 2015; Kaiser and Sutskever 2015; Kalchbrenner, Danihelka, and Graves 2015; Saxton et al. 2019; Wangperawong 2018) can be solved after understanding simple rules regardless of the inputs. Others demand a deep learning model to discover necessary rules and apply them depending on inputs given as natural language words (Li et al. 2019; Wang, Liu, and Shi 2017; Weston et al. 2015), complex mathematical equations (Lample and Charton 2019), or programming snippets (Zaremba and Sutskever 2014).

Among them, symbolic reasoning problems can test whether a trained deep learning model can systematically extend rules to *out-of-distribution* (OOD) data that follow a distinct distribution from the training data (Keysers et al. 2019; Lake and Baroni 2017; Saxton et al. 2019). For instance, a model for the addition problem whose training inputs are a pair of numbers up to five digits, say `5872+13`, can face an OOD input of a pair of two 6-digit numbers upon the testing phase, e.g., `641436+135321`. Human intelligence with *algebraic mind* can naturally extend learned rules (Marcus 2003), yet it is non-trivial to equip deep learning models for sequence transduction problems to handle OOD generalization.

However, it has been found that popular sequence transduction neural networks, such as LSTM seq2seq model (Sutskever, Vinyals, and Le 2014) and Transformer (Vaswani et al. 2017), rarely extend learned rules in that they are inclined to mimic the training data distribution (Dehghani et al. 2018; Lake and Baroni 2017). There have been significant initial efforts to improve a model's abilities to extend learned rules. However, their success has been dependent on the direct use of numerical values (Trask et al. 2018) or has been limited to rudimentary logic such as copying sequences (Dehghani et al. 2018; Graves, Wayne, and Danihelka 2014; Grefenstette et al. 2015; Rae et al. 2016; Zaremba and Sutskever 2014) and binary arithmetic (Graves, Wayne, and Danihelka 2014; Joulin and Mikolov 2015; Kaiser and Sutskever 2015). Furthermore, OOD generalization on symbolic problems for complex or context-dependent logic forms such as decimal arithmetic, algebraic word problems, computer program evaluation problems has not been tackled. Our objective is to fill this gap and design a module that helps neural networks to achieve OOD generalization in these problems.

One observation from a previous study (Nam, Kim, and Jung 2019) is that typical sequence transduction neural networks cannot process OOD instances of number sequence prediction problems, such as predicting a Fibonacci sequence. However, when an input sequence is manually seg-

mented and aligned into a grid of digits, a CNN can easily process OOD instances. This means providing the aligned grid input enables to exploit inductive bias by the convolution's local and parallel computation. The grid, however, must be handcrafted in the study, which is inapplicable for general sequence transduction tasks. Overcoming this limitation requires a new input preprocessing module that automatically aligns an input sequence into a grid without supervision for the alignment.

In this work, we propose a neural sequence-to-grid (seq2grid) module, an input preprocessor that learns how to segment and align an input sequence into a grid. The grid syntactically aligned by our module is then semantically decoded by a neural network. In particular, our module produces a grid by a novel differentiable mapping called nested list operation inspired by Stack RNN (Joulin and Mikolov 2015). This mapping enables a joint training of our module and the neural network in an end-to-end fashion via a backpropagation.

Experimental results show that ResNets with our seq2grid module achieve OOD generalization on various arithmetic and algorithmic reasoning problems, such as number sequence prediction problems, algebraic word problems, and computer program evaluation problems. These are nearly impossible for other contemporary sequence-to-sequence models including LSTM seq2seq models and Transformer-based models. Specifically, we find that the seq2grid can infuse an input context into a grid so that doing arithmetic under linguistic instructions or selecting the true branch of if/else statements in code snippets become possible. Further, we demonstrate that the seq2grid module can enhance TextCNN to solve the bAbI QA tasks without the help of external memory. From all the aforementioned problems, we verify the generality of the seq2grid module in that it automatically preprocesses the sequential input into the grid input in a data-driven way.

Motivation for Sequence-to-grid Method

To demonstrate the benefits of the sequence-to-grid preprocessing method for symbolic reasoning tasks, we devise a toy decimal addition problem in two different setups: sequential and grid-structured. Figure 1 illustrates how the problem is defined in both setups and shows why alignment on a grid makes it easier. If the lengths of the numbers increase, the temporal distances between corresponding digits, e.g., `2` and `3`, also increase in the sequential setup. However, the spatial distances between them remain constant in the grid-structured setup since they are *manually* aligned according to their digits. Therefore, we can expect that the local and parallel nature of convolution will extend the rule to longer inputs, while sequence transduction models will struggle to handle the increased distances.

To see this, we trained deep learning models[1] using numbers up to five digits and validate on six separate validation sets, each of which contains only k-digit ($k = 3, \ldots, 8$) numbers. Hence, the validation results from the former three sets tested *in-distribution* (ID) generalization, whereas the latter three tested OOD generalization. While the input and the target in the sequential setup were sequentially fed to sequence transduction models such as LSTM seq2seq model (Sutskever, Vinyals, and Le 2014) and Transformer (Vaswani et al. 2017), those in the grid-structured setup were fed to a ResNet-based CNN model (He et al. 2016). As expected, Figure 2 shows that extending the addition rule to OOD validation sets is easy in the grid-structured setup, whereas it is extremely difficult in the sequential setup.

Therefore, providing aligned grid input for local and parallel computation can be key to achieving OOD generalization. However, manual preprocessing that aligns an input sequence into a grid is impossible for most symbolic problems. For instance, in computer program evaluation problems, symbols within the code snippet can represent not only integers but also programming instructions so that it is nontrivial to manually align those symbols on the grid. Likewise, in the bAbI QA tasks, questions and stories given as natural language have no ground-truth alignment which we

[1]The models had the same configurations used in arithmetic and algorithmic problems (refer to experiments) except for the CNN that was the grid decoder of the S2G-CNN.

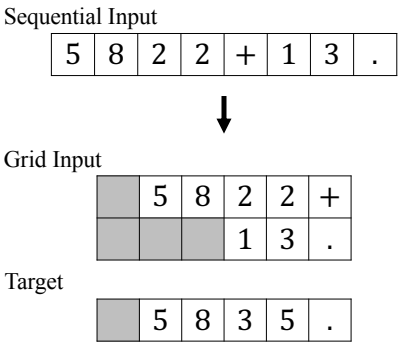

Figure 1: The illustration of the toy decimal addition problem. Each symbol is stored with its representation vector.

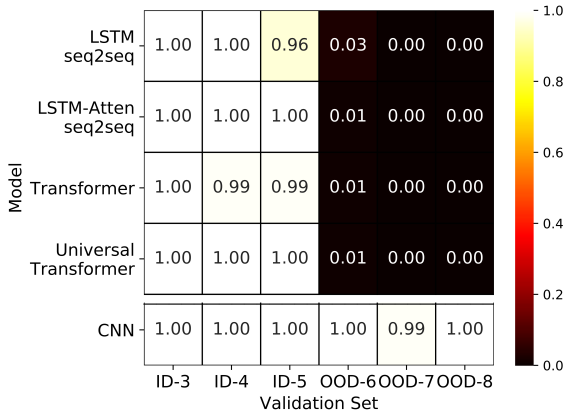

Figure 2: The validation accuracy results of the toy problem. Each column shows results from the k-digit set, where the three rightmost sets are OOD.

can exploit for preprocessing in advance. Accordingly, we need a data-driven preprocessing method that automatically aligns an input symbol sequence into a grid for general symbolic tasks. We implement it by designing a sequence-to-grid module executing novel *nested list operations*.

Related Work

Symbolic Reasoning Tasks Symbolic reasoning requires discovering the underlying rules of a data distribution rather than mimicking data patterns. Hence, there have been studies to formulate symbolic reasoning tasks in machine learning problems to examine the mathematical and systematic (rule-based) reasoning abilities of deep learning models. Additions of unprecedented long binary numbers (Graves, Wayne, and Danihelka 2014; Joulin and Mikolov 2015; Kaiser and Sutskever 2015) or Number sequence prediction problems (Nam, Kim, and Jung 2019) are studied. Also, school-level math problems including algebraic word problems are unified (Saxton et al. 2019). Evaluating program snippets (Zaremba and Sutskever 2014) further requires algorithmic abilities. Besides extending mathematical rules, systematic generalization abilities in synthetic natural language tasks are tested by the bAbI QA tasks (Weston et al. 2015) or the SCAN problems (Lake and Baroni 2017).

Memory Augmented Neural Network Storing all input information into external memory and querying over it is one way to tackle symbolic reasoning tasks. Such neural networks, also known as memory augmented neural networks (MANN), vary according to their memory structures and controllers. Here, the memory controller is a neural network that reads an input symbol and its external memory, encodes the symbol, and write it on the memory. After the incipient MANNs like Memory network (Sukhbaatar et al. 2015) were introduced, studies about implementing Automata with differentiable tape as neural networks (Graves et al. 2016; Rae et al. 2016; Joulin and Mikolov 2015; Grefenstette et al. 2015) has been carried out. We emphasize that our preprocessed grid input can be seen as another representation of a sequential input rather than a memory used in MANNs; the RNN encoder of our module does not read the grid and the symbol embedding is directly written to the grid rather than passed through neural network layers.

Neural-Symbolic Learning Another approach for OOD generalization in symbolic problems is Neural-Symbolic-approach that integrates the connectionist and the symbolist paradigms. Neural Programmer Interpreter (NPI) (Reed and De Freitas 2015) and its recursion variant (Cai, Shin, and Song 2017) have been proposed for solving compositional programs through sequential subprograms. Also, (Chen, Liu, and Song 2017) proposes a reinforcement learning-based approach with structured parse-trees. Recently, Neural Symbolic Reader (Chen et al. 2019) trains models with weak supervision for generalization. However, our approach via automatic alignment without domain-specific knowledge is distinct from neural-symbolic approaches which require all of the supervision for sequential sub-operations.

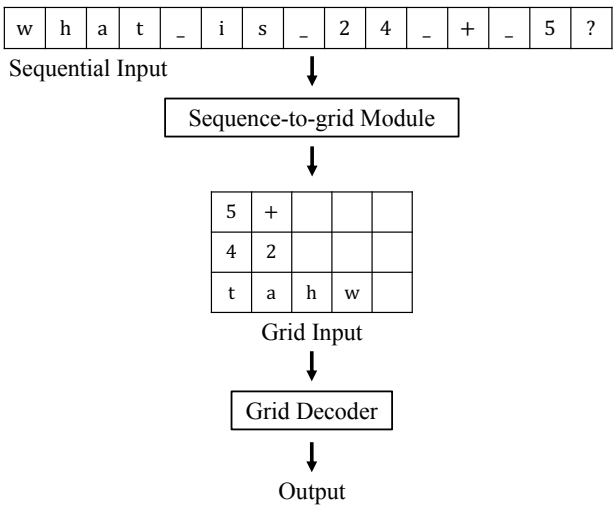

Figure 3: The sequence-input grid-output architecture.

Method

In this section, we first describe a sequence-input grid-output architecture consisting of a neural sequence-to-grid (seq2grid) module and a grid decoder. Then, we introduce how the seq2grid module preprocesses a sequence input as a grid input. Finally, we explain nested list operations that are executed by the seq2grid module.

Sequence-input Grid-output Architecture

The key idea of the sequence-to-grid method is to decouple symbolic reasoning into two steps: automatically aligning an input sequence into a grid, and doing semantic computations over the grid. Hence, we propose the sequence-input grid-output architecture consisting of a seq2grid module and a grid decoder as shown in Figure 3. The seq2grid module preprocesses a sequential input into a grid input. The grid decoder, a neural network that can handle two-dimensional inputs, predicts the target from the grid input. Practically, we choose the grid decoder like ResNet or TextCNN according to problems. Note that our approach that separates the syntactic (=alignment) and semantic processing is similar to the syntactic attention (Russin et al. 2020).

Neural Sequence-to-grid Module

The main challenge for implementing the seq2grid module is that the grid must be formed via differentiable mappings to ensure an end-to-end training. To do so, we design the seq2grid module with an RNN encoder that gives an action sequence for differentiable nested list operations.

Formally, the seq2grid module works as follows. First, for an input sequence given as symbol embeddings $E^{(t)} \in \mathbb{R}^h$ where $t = 1, \ldots, T$, the RNN encoder maps $(E^{(t)}, r^{(t-1)})$ into $r^{(t)} \in \mathbb{R}^h$. Then, a dense layer followed by a softmax layer computes an action: $r^{(t)} \mapsto a^{(t)} \in \mathbb{R}^3$. Next, starting from the zero-initialized grid $G^{(0)} \in (\mathbb{R}^h)^{W \times H}$, a series

of nested list operations sequentially push the input symbol $E^{(t)}$ into the previous grid $G^{(t-1)}$ in different extents under the action $a^{(t)}$. As a result, we obtain the grid input $G^{(T)} \in (\mathbb{R}^h)^{W \times H}$ that will be fed through a grid decoder. Note that all aforementioned mappings are differentiable including nested list operations which we will explain below.

Nested List Operations

To understand how the nested list operations work, we first regard the grid $G \in (\mathbb{R}^h)^{H \times W}$ as a *nested list* consisting of H lists of W slots, where each slot is a vector of dimension h. We denote the i-th list as $G_i \in (\mathbb{R}^h)^W$ where G_1 is the top list. Likewise, the j-th slot vector in the i-th list is denoted as $G_{i,j} \in \mathbb{R}^h$ where $G_{i,1}$ is the leftmost slot of the i-th list.

Now, we define a differetiable map that pushes the input symbol $E^{(t)} \in \mathbb{R}^h$ into the grid under the action $a^{(t)} \in \mathbb{R}^3$. Here, each component of $a^{(t)} = (a^{(t)}_{TLU}, a^{(t)}_{NLP}, a^{(t)}_{NOP})$ is the probability of performing one of three nested list operations: *top-list-update* ($a^{(t)}_{TLU}$), *new-list-push* ($a^{(t)}_{NLP}$), and *no-op* ($a^{(t)}_{NOP}$). As shown in Figure 4, $G^{(t-1)}$ with $(E^{(t)}, a^{(t)})$ grows to $G^{(t)}$:

$$G^{(t)} = a^{(t)}_{TLU} TLU^{(t)} + a^{(t)}_{NLP} NLP^{(t)} + a^{(t)}_{NOP} G^{(t-1)},$$

where $TLU^{(t)} \in (\mathbb{R}^h)^{H \times W}$ is defined as

$$\begin{aligned} TLU^{(t)}_{1,1} &= E^{(t)}, \\ TLU^{(t)}_{1,j} &= G^{(t-1)}_{1,j-1} \qquad \text{for } j > 1, \\ TLU^{(t)}_{i} &= G^{(t-1)}_{i} \qquad \text{for } i > 1, \end{aligned}$$

and $NLP^{(t)} \in (\mathbb{R}^h)^{H \times W}$ is defined as

$$\begin{aligned} NLP^{(t)}_{1} &= (E^{(t)}, E_\emptyset, \ldots, E_\emptyset), \\ NLP^{(t)}_{i} &= G^{(t-1)}_{i-1} \qquad \text{for } i > 1. \end{aligned}$$

Here, $E_\emptyset := \mathbf{0} \in \mathbb{R}^h$ is the empty symbol $\emptyset$. Accordingly, the zero-initialized grid $G^{(0)} = (E_\emptyset, \ldots, E_\emptyset)$ grows to the final grid $G^{(T)}$ as time goes. By doing so, we "preprocess" the input sequence into the grid input in that each slot of $G^{(T)}$ holds nothing but a weighted sum of input symbols.

Experimental Setup

We evaluated the seq2grid module on symbolic problems whose targets are given as sequences or single labels. To this end, we built neural network models, such as S2G-CNN and S2G-TextCNN, that followed the sequence-input grid-output architecture by varying the grid decoder according to target modalities of problems. Refer to each problem section for our grid decoder choices and their training losses.

We compared our models with five baselines: Transformer (Vaswani et al. 2017), Universal Transformer (UT) (Dehghani et al. 2018) with dynamic halting[2], a LSTM seq2seq model (LSTM) (Sutskever, Vinyals, and Le 2014), a LSTM seq2seq attention model with a bidirectional encoder (LSTM-Atten) (Bahdanau, Cho, and Bengio 2014) and a Relational Memory Core seq2seq model (RMC) (Santoro et al. 2018). The Transformer and the UT consisted of two layers with the hidden size 128 and four attention heads. The LSTM, the LSTM-Atten, and the RMC had three layers with the hidden size 1024, 512, and 512 each.

We determined configurations of our models by hyperparameter sweeping for each problem. Our implementations[3] based on the open source library `tensor2tensor`[4] contain detailed training schemes and hyperparameters of our models and the baselines. All models could fit in a single NVIDIA GTX 1080ti GPU.

The next three sections follow the same organization to illustrate the experiments and their results. First, we introduce a set of symbolic reasoning tasks by describing the layouts of the inputs and the targets. Next, we describe our grid decoder architecture to solve such problems, which is combined with the seq2grid module to follow the sequence-input grid-output architecture. Finally, we analyze the experimental results and discuss their implications.

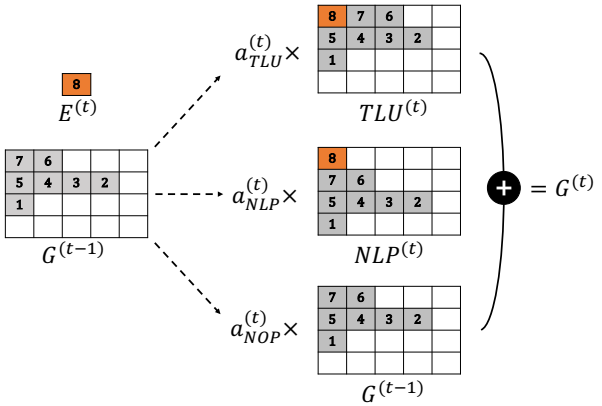

Figure 4: The nested list $G^{(t-1)}$ grows to $G^{(t)}$ by the action $a^{(t)} = (a^{(t)}_{TLU}, a^{(t)}_{NLP}, a^{(t)}_{NOP})$. $TLU^{(t)}$ and $NLP^{(t)}$ show outputs of *top-list-update* and *new-list-push* operations.

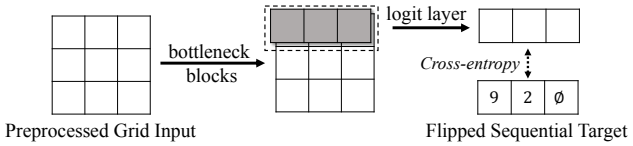

Figure 5: The grid decoder of S2G-CNN. Only the top list of the grid from bottleneck blocks is passed to the logit layer. The raw target `29` is flipped and padded to `92`$\emptyset$.

Arithmetic and Algorithmic Problems

Arithmetic and algorithmic problems are useful to test abilities to extend rules on longer inputs since the input contains

[2]The UT can take different ponder time for each position.

[3]https://github.com/SegwangKim/neural-seq2grid-module

[4]https://github.com/tensorflow/tensor2tensor

digits. We test our models on three different arithmetic and algorithmic inference problems. Each problem consists of a training set and two test sets randomly sampled from distributions controlled by difficulty parameters. Two test sets represent in-distribution data (ID) and OOD data (OOD). Difficulty parameters of the training set can be overlapped with those of the ID test set, but instances of the two sets are strictly separated by their hash codes. The training set of all problems contains 1M random examples and the two test sets contain 10K examples each. We tokenize all inputs and targets by characters and decimal digits. We score the output by sequence-level accuracy, i.e., whether the output entirely matches the target sequence. For convenience, we denote $\langle$EOS$\rangle$ as `$`.

Number Sequence Prediction As the name suggests, the goal of the number sequence prediction problem (Nam, Kim, and Jung 2019) is to predict the next term of an integer sequence. After randomly choosing three initial terms, we generate a sequence via the recursion $a_n = 2a_{n-1} - a_{n-2} + a_{n-3}$ which progresses the sequence up to the n^{th} term. The input is the first n terms $a_0, \ldots, a_{n-1}$ and the target is the last term a_n. The difficulty of the instance is parameterized by the maximum number of digits of the initial terms $a_0, \ldots, a_{k-1}$, i.e., *length*, and the total number of input integer terms n, i.e., *#terms*. Those two difficulty parameters, *length* and *#terms*, vary (1-4, 4-6), (4, 4-6), and (6, 10-12) for the training set, the ID test set, and the OOD test set, respectively. The input and the target of a training example are as follows.

Input: `7008 -205 4 7221$`
Target: `14233$`

Algebraic Word Problem To test the arithmetic abilities under linguistic instructions, we choose algebraic word problems, i.e., add-or-sub word, (Saxton et al. 2019). The difficulty of the problems is controlled by *entropy*, the number of digits within a question. Here, we make two differences from the original dataset. First, we only allow integers whereas floating-point numbers can appear originally. Second, our *entropy* is the total number of digits in the input, whereas the original entropy is the maximum number of digits that input can have. Our *entropy* varies 16-20, 16-20, and 32-40 for the training set, the ID test set, and the OOD test set, respectively. In the OOD test, we also impose every integer to be of length above 16 to guarantee that it is longer than any integers in the training set. The input and the target of a training example are as follows.

Input: `What is -784518 take away 7323?$`
Target: `-791841$`

Computer Program Evaluation Predicting the execution results of programs requires algorithmic reasoning such as doing arithmetic operations or following programming instructions like variable assignments, branches, and loops. We use *mixed strategy* (Zaremba and Sutskever 2014) to generate the training data with *nesting* 2 and *length* 5. For the ID test set and the OOD test set, *nesting* and *length* are set to be (2, 5) and (2, 7), respectively. The input, a random Python snippet, and the target, the execution result, of a training example are as follows.

Input:
```
j=891
for x in range(11):j-=878
print((368 if 821<874 else j))$
```
Target: `368$`

	Sequence		Add-or-sub		Program	
	ID	OOD	ID	OOD	ID	OOD
Baselines						
LSTM	0.21	0.00	0.99	0.00	0.25	0.07
LSTM-Atten	0.68	0.00	**1.00**	0.00	0.37	0.01
RMC	0.01	0.00	0.99	0.00	0.33	0.01
Transformer	0.97	0.00	0.97	0.00	0.37	0.00
UT	**1.00**	0.00	**1.00**	0.00	**0.62**	0.00
Ours						
S2G-CNN	0.96	**0.99**	0.98	0.53	0.51	0.33
S2G-ACNN	0.90	0.92	0.96	**0.55**	0.44	**0.35**

Table 1: Best sequence-level accuracy (out of 5 runs) on number sequence prediction problems (sequence), algebraic word problems (Add-or-sub), and computer program evaluation problems (Program)

Grid Decoder

For solving arithmetic and algorithmic problems with digits, it is desirable to choose a grid decoder that can do local and parallel computation. Therefore, we implemented a CNN (He et al. 2016) consisting of three stacks of 3-layer bottleneck building blocks of ResNet. Also, we implemented its attentional variant ACNN (Ramachandran et al. 2019); every 3×3 convolution of the CNN was substituted with a stand-alone self-attention convolution. We used 3×25-sized grids from the seq2grid module having 3-layered GRU encoder of hidden size 128 for both decoders. As shown in Figure 5, we measured cross-entropy loss between the flipped-and-padded target and the output from the logit layer. Here, the loss for empty symbol $\emptyset$ was included as we read out logits backward in the inference stage. We jointly trained the seq2grid module and the CNN (ACNN) by the ADAM optimizer (Kingma and Ba 2014) with a learning rate $1e^{-3}$.

Results

Table 1 shows that our models, S2G-CNN and S2G-ACNN, can generalize on OOD test sets. In particular, both grid decoders achieve similar OOD generalization, implying that feeding the grid input via our seq2grid module can be beneficial to any decoder that can do local and parallel computations. On the other hand, all baselines catastrophically fail at the OOD test sets although they seemingly perform well on the ID test set. This shows that extending rules to longer numbers via sequential processing is extremely difficult.

As for the number sequence prediction problems, their OOD test results serve as unit tests for the seq2grid module

(a) Sequence (b) Program

Figure 6: Visualizations of preprocessed grid inputs of (a) number sequence prediction problems and (b) computer program evaluation problems. The top and the bottom row correspond to S2G-CNN and S2G-ACNN, respectively.

	instruction	ID	OOD
LSTM-Atten	IF-ELSE	0.46	0.26
	FOR	0.06	0.03
	*	0.07	0.04
UT	IF-ELSE	**0.81**	0.01
	FOR	**0.38**	0.00
	*	**0.52**	0.00
S2G-CNN	IF-ELSE	0.73	**0.57**
	FOR	0.20	0.09
	*	0.25	0.14

Table 2: Accuracy by instruction types of the best runs on the computer program evaluation problems. For example, the S2G-CNN correctly answers 73% of all ID snippets containing IF-ELSE instructions.

since it needs to align digit symbols on the grid according to their scales. Indeed, Figure 6a shows that our module automatically finds such alignments that resemble the tailored grid of digits as shown in Figure 1.

For the algebraic word problems, they require context-dependent arithmetic unlike number sequence prediction problems using the fixed progression rules. In particular, linguistic instructions like `add` or `take away` indicates how to add/subtract given two numbers in a specific order. Since our grid decoders apply the fixed convolutional filters over the grid, linguistic instructions must be reflected in the grid input beforehand for doing context-dependent arithmetic. This shows that our seq2grid module can infuse the instruction information into the grid input.

For the computer program evaluation problems, predicting the output of a code snippet demands an understanding of algorithmic rules like branching mechanisms or for-loop given as programming instructions IF-ELSE or FOR. Also, computing * operations has non-linear time complexity, unlike addition or subtraction. Hence, we further investigate accuracy on snippets by those instructions as shown in Table 2. For the OOD snippets containing IF-ELSE instructions, our S2G-CNN achieves 57% accuracy for them. Considering that they can contain other instructions besides branching one as shown in Figure 7, the accuracy is fairly

```
print((11*7288719))
```

```
print(((6110039 if 7327755<3501784 else
1005398)*11))
```

```
b=6367476
for x in range(19):b-=9082877
print((3569363 if 7448172<9420320 else b))
```

```
e=(450693 if 4556818<2999168 else 3618338)
for x in range(10):e-=4489485
print(e)
```

Figure 7: Some OOD code snippets correctly answered by the best run of the S2G-CNN. Note that snippets contain FOR or * instruction requiring non-linear time complexity.

Task 2. two-supporting-facts

⟨CLS⟩ Where is the apple ? ⟨SEP⟩ Mary journeyed to the garden . Sandra got the football there . Mary picked up the apple there . Mary dropped the apple .

Task 17. basic-deduction

⟨CLS⟩ What is gertrude afraid of ? ⟨SEP⟩ Wolves are afraid of sheep . Gertrude is a wolf . Winona is a wolf . Sheep are afraid of mice . Mice are afraid of cats . Cats are afraid of sheep . Emily is a cat . Jessica is a wolf .

Task 19. path-finding

⟨CLS⟩ How do you go from the garden to the office ? ⟨SEP⟩ The kitchen is west of the office . The office is north of the hallway . The garden is east of the bathroom . The garden is south of the hallway . The bedroom is east of the hallway .

Figure 8: Input examples of the bAbI QA tasks.

high. For the non-linear operations, the S2G-ACNN shows little understanding compared with the UT on the ID test set. However, the UT fails to extend rules of FOR and * instructions on the OOD test set while the S2G-CNN does so on some examples as shown in Figure 7. These are surprising in that both the seq2grid module and the ACNN grid decoder do linear time computations in the input length.

bAbI QA Tasks

Given as natural language with a small vocabulary of around 170, the bAbI QA tasks (Weston et al. 2015) test 20 types of simple reasoning abilities such as *counting, induction, deduction,* and *path-finding*. A problem instance consists of a story, a question, and the answer. Here, the story contains supporting sentences about the answer and distractors that are irrelevant sentences to the answer. We formulate the bAbI QA tasks (Weston et al. 2015) in a sequence classification setup such that an input is a concatenation of ⟨CLS⟩ token, a question, ⟨SEP⟩ token, and a story as shown in Figure 8. While previous work (Dehghani et al. 2018) uses sentence-level encodings, we use straightforward one-

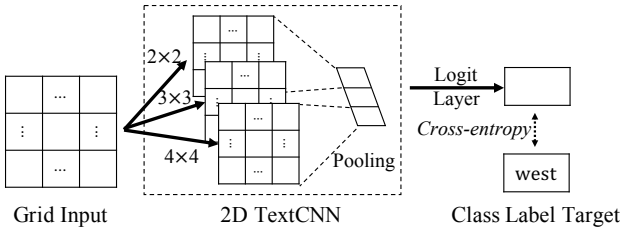

Figure 9: The grid decoder of S2G-TextCNN.

	#params	Error	#Failed tasks
Baselines[5]			
LSTM	25.6M	24.9 ± 5.8	12.1 ± 3.7
Transformer	0.5M	33.1 ± 1.7	18.9 ± 0.3
UT	0.5M	26.8 ± 6.0	15.0 ± 4.0
TextCNN	0.2M	37.8 ± 0.4	19.0 ± 0.0
Ours			
S2G-TextCNN	0.8M	10.8 ± 0.8	6.0 ± 0.0

Table 3: Error and #Failed tasks ($> 5\%$ error) on the bAbI QA 10k joint tasks (for 10 runs).

hot word-level encodings. This setup yields the increase of the average input length from 13.6 to 78.9, which in turn requires to handle much longer dependencies. Hence, solving the bAbI tasks under word-level encodings is much harder than those under sentence-level encodings. State-of-the-art models deal with longer dependencies via augmenting neural networks with external memory (Munkhdalai et al. 2019; Rae et al. 2016). However, we will show that the seq2grid module can enhance a simple neural network like TextCNN to effectively solve the word-level bAbI tasks, even in the absence of a complex and expensive memory structure.

Grid Decoder

We chose a grid decoder as a variant of TextCNN (Kim 2014). After the seq2grid module having 2-layered GRU encoders of hidden size 128 gave the 4×8-sized grid input, our TextCNN predicted the label by applying $k\times k$-CNNs ($k = 2, 3, 4$), max-pooling, and dropout with the rate 0.4 as shown in the Figure 9. We used the ADAM optimizer to jointly train the seq2grid module and the TextCNN under a warm-up and decay learning rate scheme[3].

Results

Our S2G-TextCNN outperforms sequential baseline models, such as the LSTM, the Transformer encoder, and the UT encoder, as shown in Table 3. Note that we fed word-level inputs that require doing reasoning over distant symbols, i.e., the average length of inputs is 78.9, and we used the grid that has only 32 ($= 4\times8$) slots. From these setups, we can conclude that our module can compress long inputs into grid

[5]We use only encoders of baselines used in arithmetic and algorithmic problems. As for the logit, LSTM uses the last hidden state while others use the hidden one corresponding to ⟨CLS⟩ token.

		Baselines		Ours
Task	#supps	LSTM	UT	S2G-TextCNN
1: single-supporting-fact	1.0	0.0	0.0	0.0
2: two-supporting-facts	2.0	47.4	55.0	31.2
3: three-supporting-facts	3.0	45.9	67.9	31.5
4: two-arg-relations	1.0	0.1	0.0	0.0
5: three-arg-relations	1.0	0.8	5.5	1.0
6: yes-no-questions	1.0	0.5	0.1	0.0
7: counting	2.3	1.8	4.0	0.0
8: lists-sets	1.9	0.2	2.3	1.8
9: simple-negation	1.0	0.0	0.0	0.0
10: indefinite-knowledge	1.0	0.3	0.0	0.0
11: basic-coreference	2.0	0.0	0.1	0.0
12: conjunction	1.0	0.0	0.0	0.0
13: compound-coreference	2.0	0.0	0.0	0.0
14: time-reasoning	2.0	20.6	4.4	7.3
15: basic-deduction	2.0	34.8	18.5	0.0
16: basic-induction	3.0	52.1	53.6	51.7
17: positional-reasoning	2.0	41.1	41.0	31.4
18: size-reasoning	2.0	8.6	9.1	3.8
19: path-finding	2.0	90.9	79.1	35.1
20: agents-motivations	1.0	1.8	1.4	0.0
Mean error (%)		17.3	17.1	9.7
#Failed tasks		8	8	6

Table 4: Task-wise errors on the bAbI QA 10k joint tasks for the best runs. #supps is the average number of supporting sentences in the story.

inputs while selecting only necessary words along story arcs. Moreover, the compression is effective in terms of the number of parameters. Indeed, the GRU encoder inside our module is much smaller than the LSTM but enough to provide grid inputs to our grid decoder for solving the bAbI tasks.

We highlight that our seq2grid module, not the TextCNN decoder, leads to the superior performance of our model. Since the attempt to use the usual TextCNN alone fails at almost all tasks, the dramatic performance gain by the aid of the seq2grid module is somewhat surprising.

We further analyze errors by tasks to see the possibility and the limitation of our sequence-to-grid method. The zero variance in the number of failed tasks (Table 3) indicates that the S2G-TextCNN consistently fails on the same set of tasks, as listed in Table 4. Those failed tasks including *two-supporting-facts, positional reasoning,* and *path-finding* seem reasonably difficult for our models in that all of them require more than one supporting sentence for the reasoning.

Conclusion

We introduced a neural sequence-to-grid (seq2grid) module which automatically segments and aligns an sequential input into a grid. Our module was used as an input preprocessor for a neural network that took a grid input. In particular, our module executed our novel nested list operations, ensuring an end-to-end joint training with the neural network. Empirically, our module enhanced neural networks in various symbolic reasoning tasks.

Acknowledgements

The authors appreciate Hyunkyung Bae for assistance with experiments. K. Jung is with ASRI and ECE, Seoul National University, Korea. This work was supported by Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFCIT1902-06.

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

Cai, J.; Shin, R.; and Song, D. 2017. Making neural programming architectures generalize via recursion. *arXiv preprint arXiv:1704.06611* .

Chen, X.; Liang, C.; Yu, A. W.; Zhou, D.; Song, D.; and Le, Q. V. 2019. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *International Conference on Learning Representations*.

Chen, X.; Liu, C.; and Song, D. 2017. Towards synthesizing complex programs from input-output examples. *arXiv preprint arXiv:1706.01284* .

Dehghani, M.; Gouws, S.; Vinyals, O.; Uszkoreit, J.; and Kaiser, Ł. 2018. Universal transformers. *arXiv preprint arXiv:1807.03819* .

Graves, A.; Wayne, G.; and Danihelka, I. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401* .

Graves, A.; Wayne, G.; Reynolds, M.; Harley, T.; Danihelka, I.; Grabska-Barwińska, A.; Colmenarejo, S. G.; Grefenstette, E.; Ramalho, T.; Agapiou, J.; et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature* 538(7626): 471–476.

Grefenstette, E.; Hermann, K. M.; Suleyman, M.; and Blunsom, P. 2015. Learning to transduce with unbounded memory. In *Advances in neural information processing systems*, 1828–1836.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Joulin, A.; and Mikolov, T. 2015. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in neural information processing systems*, 190–198.

Kaiser, Ł.; and Sutskever, I. 2015. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228* .

Kalchbrenner, N.; Danihelka, I.; and Graves, A. 2015. Grid long short-term memory. *arXiv preprint arXiv:1507.01526* .

Keysers, D.; Schärli, N.; Scales, N.; Buisman, H.; Furrer, D.; Kashubin, S.; Momchev, N.; Sinopalnikov, D.; Stafiniak, L.; Tihon, T.; et al. 2019. Measuring Compositional Generalization: A Comprehensive Method on Realistic Data. *arXiv preprint arXiv:1912.09713* .

Kim, Y. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* .

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Lake, B. M.; and Baroni, M. 2017. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. *arXiv preprint arXiv:1711.00350* .

Lample, G.; and Charton, F. 2019. Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412* .

Li, J.; Wang, L.; Zhang, J.; Wang, Y.; Dai, B. T.; and Zhang, D. 2019. Modeling Intra-Relation in Math Word Problems with Different Functional Multi-Head Attentions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 6162–6167.

Marcus, G. 2003. *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. MIT Press.

Munkhdalai, T.; Sordoni, A.; Wang, T.; and Trischler, A. 2019. Metalearned neural memory. In *Advances in Neural Information Processing Systems*, 13331–13342.

Nam, H.; Kim, S.; and Jung, K. 2019. Number Sequence Prediction Problems for Evaluating Computational Powers of Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 4626–4633.

Rae, J.; Hunt, J. J.; Danihelka, I.; Harley, T.; Senior, A. W.; Wayne, G.; Graves, A.; and Lillicrap, T. 2016. Scaling memory-augmented neural networks with sparse reads and writes. In *Advances in Neural Information Processing Systems*, 3621–3629.

Ramachandran, P.; Parmar, N.; Vaswani, A.; Bello, I.; Levskaya, A.; and Shlens, J. 2019. Stand-alone self-attention in vision models. *arXiv preprint arXiv:1906.05909* .

Reed, S.; and De Freitas, N. 2015. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279* .

Russin, J.; Jo, J.; O'Reilly, R.; and Bengio, Y. 2020. Compositional Generalization by Factorizing Alignment and Translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, 313–327.

Santoro, A.; Faulkner, R.; Raposo, D.; Rae, J.; Chrzanowski, M.; Weber, T.; Wierstra, D.; Vinyals, O.; Pascanu, R.; and Lillicrap, T. 2018. Relational recurrent neural networks. In *Advances in Neural Information Processing Systems*, 7299–7310.

Saxton, D.; Grefenstette, E.; Hill, F.; and Kohli, P. 2019. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557* .

Sukhbaatar, S.; Weston, J.; Fergus, R.; et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*, 2440–2448.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Trask, A.; Hill, F.; Reed, S. E.; Rae, J.; Dyer, C.; and Blunsom, P. 2018. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*, 8035–8044.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Wang, Y.; Liu, X.; and Shi, S. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 845–854.

Wangperawong, A. 2018. Attending to Mathematical Language with Transformers. *arXiv preprint arXiv:1812.02825* .

Weston, J.; Bordes, A.; Chopra, S.; Rush, A. M.; van Merriënboer, B.; Joulin, A.; and Mikolov, T. 2015. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698* .

Zaremba, W.; and Sutskever, I. 2014. Learning to execute. *arXiv preprint arXiv:1410.4615* .