Ph.D. DISSERTATION

Improving Rule-based Out-of-distribution Generalization Abilities of Sequence Generation Models

시퀀스생성모델의 규칙기반 분포외일반화능력 향상방법

BY

SE-GWANG KIM

AUGUST 2022

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Improving Rule-based Out-of-distribution Generalization Abilities of Sequence Generation Models

시퀀스생성모델의 규칙기반 분포외일반화능력 향상방법

BY

SE-GWANG KIM

AUGUST 2022

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Improving Rule-based Out-of-distribution Generalization Abilities of Sequence Generation Models

시퀀스생성모델의 규칙기반 분포외일반화능력 향상방법

지도교수 정 교 민
이 논문을 공학박사 학위논문으로 제출함

2022년 8월

서울대학교 대학원

전기 컴퓨터 공학부

김 세 광

김세광의 공학박사 학위 논문을 인준함

2022년 8월

위 원 장:	한 보 형
부위원장:	정 교 민
위 원:	황 승 원
위 원:	김 건 희
위 원:	임 성 수

Abstract

Developing human-level machines that can learn and extend rules is a long-standing challenge for the artificial intelligence community. Even though current deep learning models have proven remarkable performances over a wide range of applications, the models still struggle to apply learned rules to novel inputs that do not follow the training distribution. Such a lack of deep models' rule-based out-of-distribution generalization, i.e., *systematic generalization*, abilities limits many deep learning applications, especially about sequence generation tasks requiring logical reasoning, such as semantic parsing, or suffering from data scarcity, such as low-resource machine translation.

Therefore, this dissertation aims to measure and improve the systematic generalization abilities of deep learning sequence generation models. To measure the abilities of deep models, we propose number sequence prediction problems. We estimate deep learning models' computational powers by testing the models on our problems and comparing the models with Automata that can solve the problems. Then, to improve the systematic generalization abilities of deep models, we propose three frameworks. The first framework is devising a new input preprocessing module, called neural sequence-to-grid module. The module can learn how to segment and align sequence inputs into the grid inputs — more advantageous forms for learning and applying symbolic rules. We empirically show that a deep learning model taking the grid inputs can extend learned rules on symbolic reasoning tasks, including program code evaluations or bAbI tasks. The second framework is to train neural networks with structurally hinted examples. We make such examples by annotating the training targets with delimiter tokens representing the non-terminal nodes of the targets' parsing trees. We show the efficacy of our annotated targets, experimenting with instruction following tasks requiring compositional reasoning, and achieving substantial performance gains.

The last framework is to reformulate sequence generation tasks into classification-and-generation tasks using template retrieving and re-ranking with neural models. The templates, high-level sketches of target sequences, relieve the model's burdens of hard structural modeling and let the model focus on easy template realization. Experimental results show that our selected templates lead to substantial performance gains of deep learning models on four different semantic parsing tasks.

keywords: Deep Learning, Sequence Generation, Out-of-distribution Generalization
student number: 2016-20873

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Developing a human-level machine that can learn and apply rules has been the ultimate goal of the artificial intelligence community. Among human intelligence, the ability of rule-based out-of-distribution[1] (OoD) generalization, also known as *systematic generalization* [2, 3], is particularly remarkable. Thanks to the ability, humans can extend rules to the novel OoD test examples. For example, humans can extend addition rules; after learning addition rules from pairs of numbers up with small digits, e.g., `802+93`, humans can apply the rules to test pairs of numbers with much bigger digits, e.g., `131423+95410213`. As another example, humans can translate a novel utterance into its corresponding program in a compositional way; after understanding `animal.species=frog` $\wedge$ `animal.color=green`, as a composition of a template, e.g., $\{$*COND1*$\}\wedge\{$*COND2*$\}$, and basic components, e.g., $\{$*COND1*$\}$: `animal.species=frog` and $\{$*COND2*$\}$: `animal.color=green`, humans can translate the novel test utterance, e.g., "A white frog", consisting of rarely co-occurred words into the corresponding program, e.g., `animal.species=frog`$\wedge$`animal.color=white`. Note that although those test examples are OoD, they still follow certain rules found in the training examples, as conceptualized in Figure 1.1.

[1] Examples are OoD if they follow a distribution distinct from training. In contrast, examples that follow the training distribution are called in-distribution (ID).

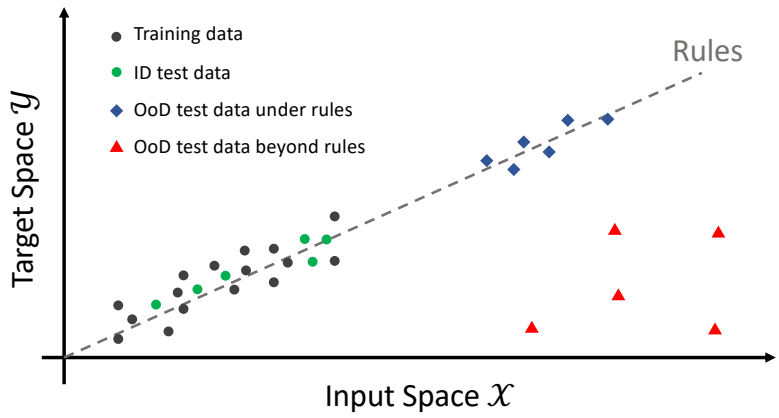

Figure 1.1: The concept of rule-based out-of-distribution generalization. Training data and in-distribution (ID) test data follow the same distribution. Out-of-distribution (OoD) test data do not follow the distribution. However, unlike OoD test data beyond rules, a model can handle OoD test data under rules, i.e., *systematic generalization*, if the model learns rules underlying the distribution.

Unfortunately, even the state-of-the-art deep learning models that have demonstrated impressive performances over a wide range of applications still struggle to achieve systematic generalization. Especially, this can be problematic in sequence generation applications as they often require handling OoD sequences [4, 5, 6, 7, 8]. For example, Natural Language Interfaces to Databases (NLIDB) [4] demands logical and algorithmic reasoning to predict the program queries from given natural language questions having novel combinations of entities. Furthermore, low-resource machine translation [5] also demands a systematic generalization ability to extend rules learned from scarce training examples.

To explore the poor systematic generalization of deep learning sequence generation models, a number of studies have defined systematic generalization tasks in various senses [9, 10, 11] and released new benchmark datasets [1, 7, 12, 13]. These

datasets commonly provide test sets that are OoD but obey rules governing the training examples. Therefore, the failures of deep models on these test sets imply that the models are unsuccessful in learning correct rules that can explain both the training and test examples. Also, many deep learning studies for achieving systematic generalization have been carried out from various perspectives. From the model architecture perspective, some researchers have developed new neural network modules that are advantageous for capturing inductive bias [14, 15]. From the data perspective, some other researchers have leveraged more diverse training data via data augmentation methods [9, 16, 17, 18] or pre-trained models [19] for relaxing OoD test set conditions. From the task formulation perspective, other researchers have focused on finding advantageous conditions, such as offering auxiliary information to models, and used them to reformulate systematic generalization tasks easier [20, 21]. However, despite the aforementioned studies, it still remains unclear how far current deep models can systematically generalize and how to lead the generalization with the deep models.

Therefore, this dissertation aims to measure and improve the rule-based out-of-distribution generalization abilities of deep learning sequence generation models.

The first part of the dissertation is about evaluating the systematic generalization abilities of current deep learning models. Inspired by number series tests to measure human intelligence, we propose number sequence prediction tasks to assess neural network models' computational powers for solving algorithmic problems. We define the complexity and difficulty of a number sequence prediction task with the structure of the smallest automaton that can generate the sequence. We suggest two types of number sequence prediction problems: the number-level and the digit-level problems. The number-level problems offer sequences as two-dimensional grids of digits and the digit-level problems provide a single-digit input per a time step. The complexity of a number-level sequence prediction can be defined with the depth of an equivalent combinatorial logic, and the complexity of a digit-level sequence prediction can be defined with an equivalent state automaton for the generation rule. Experiments with number-

level sequences suggest that CNN models are capable of learning the compound operations of sequence generation rules, but the depths of the compound operations are limited. For the digit-level problems, simple GRU and LSTM models can solve some problems with the complexity of finite state automata. Memory augmented models such as Stack-RNN, Attention, and Neural Turing Machines can solve the reverse-order task which has the complexity of a simple pushdown automaton. However, all of the above cannot solve general Fibonacci, Arithmetic, or Geometric sequence generation problems that represent the complexity of queue automata or Turing machines. The results show that our number sequence prediction problems effectively evaluate machine learning models' computational capabilities.

The remaining parts of the dissertation is to achieve deep learning models' systematic generalization through our various methods: (1) devising a new input preprocessing module, (2) training with structurally hinted examples, and (3) conditioning neural networks with templates obtained from retrieving and re-ranking.

First, we propose a new input preprocessing module to attack symbolic reasoning tasks, such as learning arithmetic operations and computer program evaluations, requiring rule-based generalization abilities. Specifically, our neural sequence-to-grid (seq2grid) module does the preprocessing by automatically segmenting and aligning an input sequence into a grid. Here, our module outputs the grid via a novel differentiable mapping, so that 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.

Second, to solve compositional generalization tasks requiring rule-based reason-

ing, we propose a framework to use structurally hinted training examples that are effective for standard seq2seq neural networks. As humans can compositionally understand a novel sentence by parsing it into known components like phrases and clauses, many compositional reasoning tasks are suggested to assess machine learning models. Among these tasks, the SCAN tasks are challenging for the standard deep learning models, such as RNN sequence-to-sequence models and Transformers, that show great success across many natural language processing tasks. Even though a long line of deep learning research has developed memory augmented neural networks aimed at the SCAN tasks, their generalities remain questionable for more complex and realistic applications where the standard seq2seq models are dominating. Hence, one needs to propose a method that helps the standard models to discover compositional rules. To this end, we propose a data augmentation technique using paring trees. Our technique annotates targets by inserting a new delimiter token in between them according to their parsing trees. For the training stage, the technique needs prior knowledge about the semantic or syntactic compositionality of the targets. On the other hand, for the test stage, the technique uses no such knowledge. Experiments show that our technique enables the standard models to achieve compositional generalization on the SCAN tasks. Furthermore, we validate our technique on a synthetic task and confirm strong performance gains of the standard models without using prior knowledge about semantic compositionality. As one way to infuse parsing tree information into sequences, our technique can be used for tasks with structured targets like program code generation tasks.

Lastly, to achieve compositional generalization of a neural semantic parser, we propose a new framework that offers selected templates to the parser via retrieving and re-ranking. While humans understand languages with compositional reasoning, the state-of-the-art neural networks struggle to grasp high-level representations, i.e., templates, and entities. It prevents the neural machines from handling instances with novel combinations of observed words. To overcome this issue, considerable studies have

been carried out to use a compositionally diverse utterance-program paired data, but they still require expensive data augmentation. Thus, it is essential to design a method to leverage cheap unpaired data that can be easily augmented. In this work, we propose a new framework that retrieves and re-ranks program templates from an augmented template pool. Specifically, we use a neural bi-encoder to retrieve candidate templates, and a cross-encoder to select the most relevant templates. The templates make a neural semantic parser easily predict the target programs. Empirically, our framework results in a strong compositional generalization of neural networks on widely used semantic parsing datasets such as ATIS, Advising, GeoQuery, and Scholar.

The remainder of the dissertation is organized as follows. Chapter 2 provides background about systematic generalization tasks with our definitions and examples and overview related work in terms of benchmark datasets and previously proposed techniques. Chapter 3 introduces our new task, called number sequence prediction, for evaluating the computational powers of neural networks. Chapter 4 describes our novel module that discovers latent alignments of input sequences, which in turn enables deep models systematically generalize. Chapter 5 explains our parsing tree annotation technique that infuses structural hints into targets. Chapter 6 introduces our classification-and-template-based approach to achieve a compositional generalization of a neural semantic parser. In the last chapter, we summarize our findings and suggest future research for achieving rule-based out-of-distribution generalization of deep learning models.

Chapter 2

Background

In this chapter, we first introduce the concept of rule-based out-of-distribution (OoD) generalization, i.e., systematic generalization, with examples. Then, we overview existing benchmark datasets for testing systematic generalization. Finally, we introduce three branches of previously proposed deep learning techniques.

2.1 Concept of Rule-based Out-of-distribution Generalization

2.1.1 Definition

Systematic generalization of a machine is to learn and apply rules such that the machine handles OoD test examples. The definition of these rules or OoD examples can vary depending on the tasks of interest [9, 10, 11]. Here, inspired by [9], we formalize a systematic generalization task for the sequence generation.

In broader strokes, we will define OoD test examples by considering two distinct distributions over one single sample space. Then, to guarantee that OoD examples obey rules governing the training set, we will consider a set of rule-extractions, mappings from the sample space to rule spaces. Finally, we will formulate the systematic

generalization in this setup.

More specifically, we first define the training and OoD test set. For a given sample space $\mathcal{S} = \mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ and $\mathcal{Y}$ are the sets of input and target sequences, we consider two distinct probabilities $\mathbb{P}_{\text{ID}}$ and $\mathbb{P}_{\text{OoD}}$ over $\mathcal{S}$. Then, we sample a training set D_{train} and test set D_{test} from $\mathbb{P}_{\text{ID}}$ and $\mathbb{P}_{\text{OoD}}$, respectively. By doing so, each test example $(x, y) \in D_{\text{test}}$ is OoD since the example does not follow the training distribution $\mathbb{P}_{\text{ID}}$.

Next, we make sure that two distributions $\mathbb{P}_{\text{ID}}$ and $\mathbb{P}_{\text{OoD}}$ have common underlying rules. To this end, we consider rule-extractions $r_1, \dots$, each of which is a measurable function from $\mathcal{S}$ into another measurable space $\mathcal{R}_i$, called a rule space. Then, for a rule-extraction r (=$r_1, \dots$), we give the input-target conditional probability constraint on pushforward measures $r_*\mathbb{P}_{\text{ID}}$ and $r_*\mathbb{P}_{\text{OoD}}$ as follows:

$$r_*\mathbb{P}_{\text{ID}}(Y|X) = r_*\mathbb{P}_{\text{OoD}}(Y'|X')$$

where (X, Y) and (X', Y') are input-target random variables from $r_*\mathbb{P}_{\text{ID}}$ and $r_*\mathbb{P}_{\text{OoD}}$. This guarantees that rules (=conditional probability) mapping from the input to the target are unchanged even if the two distributions are different. Other than that, we give no additional constraints about $\mathbb{P}_{\text{ID}}$ and $\mathbb{P}_{\text{OoD}}$ such as $\mathbb{P}_{\text{ID}}(X) = \mathbb{P}_{\text{OoD}}(X')$.

Finally, we are ready to define a systematic generalization task as follows:

Definition) Systematic Generalization Task. We are given a sample space $\mathcal{S} = \mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ and $\mathcal{Y}$ are sets of input and target sequences and measurable functions r_i $(i = 1, 2, \dots)$ mapping from $\mathcal{S}$ into a measurable space $\mathcal{R}_i$. Given two distinct probabilities $\mathbb{P}_{\text{ID}}$ and $\mathbb{P}_{\text{OoD}}$ over $\mathcal{S}$ such that $r_*\mathbb{P}_{\text{ID}}(Y|X) = r_*\mathbb{P}_{\text{OoD}}(Y'|X')$ for $(X, Y) \sim r_*\mathbb{P}_{\text{ID}}$, $(X', Y') \sim r_*\mathbb{P}_{\text{OoD}}$, and $r = r_1, \dots$, we sample the training set D_{train} and test set D_{test} from $\mathbb{P}_{\text{ID}}$ and $\mathbb{P}_{\text{OoD}}$, respectively. The *systematic generalization task* is that the machine predicts the target y' for the test input x' where $(x', y') \in D_{\text{test}}$ by using training examples of D_{train}.

Note that we are dealing with systematic generalization tasks in a seq2seq setup where both inputs and targets are sequences, so that the input and target spaces are

$\Sigma^* = \Sigma \cup \Sigma^2 \cup \cdots$ where Σ is a set of symbols or tokens.

2.1.2 Examples

To elaborate the aforementioned definition, it would be instructive to see some concrete examples. Bearing in mind that the following examples make OoD scenarios by assuming supports[1] of ID and OoD distribution are disjoint, e.g., differing the lengths of input or target sequences.

Copy Memory Task The copy memory task [22] is a test whether a machine can recall a sequence after a long time gap. Specifically, the input sequence x is given as

$$x = \overbrace{d_1 d_2 \cdots d_9 d_{10}}^{\text{sequence to recall}} \quad \overbrace{0 \cdots 0}^{T\text{-length blank}} \tag{2.1}$$

and the target sequence y is given as

$$y = d_1 \cdots d_{10}$$

where $d_1, \ldots d_n \sim \text{Unif}(\{1, \ldots, 9\})$. The time gap of the training examples is short, e.g., $T = 100$, whereas that of the test examples is long, e.g., $T = 1000$. This task can be explained by our formulation. Indeed, we have the sample space $\mathcal{S} = \Sigma^* \times \Sigma^*$ where $\Sigma = \{0, \ldots, 9\}$. The two distinct distributions are $\mathbb{P}_{\text{ID}} = f_*^{(100)}\text{Unif}((\Sigma - \{0\})^{10})$ and $\mathbb{P}_{\text{OoD}} = f_*^{(1000)}\text{Unif}((\Sigma - \{0\})^{10})$ where $f^{(T)} : \tilde{x} \mapsto (\tilde{x} \circ P, \tilde{x})$ with the concatenation $\circ$ and the T-length zeros P (Note that $\tilde{x} \circ P$ is in the form Eq. 2.1). Clearly, $\mathbb{P}_{\text{ID}} \neq \mathbb{P}_{\text{OoD}}$ as they have disjoint suppports. However, for a rule-extraction r that deletes all zero digits, we have $r_* \mathbb{P}_{\text{ID}} = r_* \mathbb{P}_{\text{OoD}}$; thus $r_* \mathbb{P}_{\text{ID}}(Y|X) = r_* \mathbb{P}_{\text{OoD}}(Y'|X') = 1$ if $X = Y (X' = Y')$ otherwise 0.

Addition Task The addition of two decimal numbers can be a systematic generalization task if we increase the number of digits in the test examples. Specifically, input

[1] The support of the probability measure P is the closure of $\{x : \mathbb{P}(x) > 0\}$.

sequence is given as

$$\overbrace{d_1 d_2 \cdots d_{T_1}}^{\text{1st term}} + \overbrace{d'_1 d'_2 \cdots d'_{T_2}}^{\text{2nd term}}$$

where $d_1 d_2 \cdots d_{T_1}$ and $d'_1 d'_2 \cdots d'_{T_2}$ are valid decimal numbers ($d_1, d'_1 \neq 0$) and the target sequence is given as the result of the decimal addition of two terms. The training examples are randomly sampled from short numbers, e.g., $T_1, T_2 \leq 6$, while the test examples are sampled from long numbers, e.g., $(T_1, T_2) = (7, 10)$. Obviously, two distributions $\mathbb{P}_{\text{ID}}$ and $\mathbb{P}_{\text{OoD}}$ have disjoint supports. To find the rule-extraction r, we first define a_i, the i-th digit addition under the number that is carried from the lower digit. For example, $a_2 : (12\mathbf{3}6 + \mathbf{7}9, 13\mathbf{1}5) \mapsto ((\mathbf{3}, \mathbf{7}, 1), (\mathbf{1}, 1))$. Here, the resulting input $(3, 7, 1)$ consists of 3 and 7 (=the 2nd digits of two terms) and the 1 (=the number carried from the 1st digit addition: 6+9=**1**5). The resulting target $(1, 1)$ consists of the first 1 (=the 2nd digit of the addition result) and the second 1 (=the number to be carried for the 3rd digit addition). Then, we define $r : (x, y) \mapsto \cup_{i=1,\ldots,} a_i(x, y)$. Under this rule-extraction, $r_* \mathbb{P}_{\text{ID}} \neq r_* \mathbb{P}_{\text{OoD}}$ (since the $T_1 \neq T_2$ for test examples) but we still have $r_* \mathbb{P}_{\text{ID}}(Y|X) = r_* \mathbb{P}_{\text{OoD}}(Y'|X')$. Note that each input component of the point in $r(x, y)$ entirely depends on x and so does for the target component and y; thus, X, Y (or X', Y') are well-defined. In fact, you can think of $r_* \mathbb{P}_{\text{ID}}(Y|X)$ as the *addition table*, whose column and row are the 1-digit numbers, with the numbers that are carried and to be carried.

2.2 Benchmark Datasets

A large number of studies has released new benchmark datasets for testing systematic generalization abilities of deep learning models. Especially, to construct new datasets, it is important to define what rules are tested and how these rules are applied to synthesize/define OoD test examples. In this point of view, we categorize existing datasets by required rules and explain their definitions of OoD test examples.

Symbolic Reasoning Tasks Symbolic reasoning, such as arithmetic or algorithmic reasoning, requires discovering the underlying rules of a data distribution. Depending on tasks and underlying rules, various symbolic reasoning tasks have been formulated as machine learning problems to examine the mathematical and systematic (rule-based) reasoning abilities of deep learning models.

In the most simplest forms, problems of learning various elementary algorithmic functions, such as copying/reversing/doubling of given sequences or recalling specific terms or entire sequences after a certain period of time steps, have been suggested [22, 23, 24]. In these problems, to check whether a machine indeed learns the underlying algorithmic rules, test sequences are given whose length is longer than observed during the training stage. Moreover, additions of unprecedented long binary numbers [23, 25, 24] are suggested too.

Beyond understanding one algorithmic rule, machines have been validated whether they can handle numbers in contexts given as natural languages or programming instructions. Algebraic word problems that requires to calculate the answer according to given natural language instruction, e.g., school-level math problems, are proposed [26, 27]. Tasks to evaluate program snippets according to program instructions like `for`-loop or `IF`-branching are introduced too [6]. These datasets commonly consist of examples with numbers so that OoD test examples are systematically defined by increasing used digits in examples.

Compositional Generalization Tasks Humans can understand natural language characterised by *compositionality* where constituents from lower levels are recursively combined with a grammar [2, 3]. Compositional reasoning, also known as algebraic capacity, is the key ability to generalize new structures with components observed during training. However, researchers have criticized that neural networks do not have the ability by testing them on datasets requiring various compositional rules.

Among these datasets, SCAN tasks [10], translations of command sequences into

corresponding action sequences, are de facto standard benchmarks due to their simple underlying rules and intuitive criteria for splitting training/test sets. For example, the SCAN length task offers training examples whose actions (=targets)' lengths ≤ 24 and then provides test examples whose actions lengths ≥ 24.

After that, a number of datasets has been released to further scrutinize what kinds of novel compositions the deep models struggle to understand. Beyond the simple human-designed training/test splitting criteria, a new automatic splitting method called distribution-based compositional assessment (DBCA) is proposed and applied to construct a new benchmark called CFQ [28]. DBCA defines atoms and compounds of examples, then measures the discrepancy in atoms' (compounds') distribution between the training and test sets. Here, compositional generalization naturally arises for the training and test sets if their distribution discrepancy in atoms is small whereas that in compounds is large. In the COGS dataset [13], parsing trees of English sentences come into play for splitting training and test sets. To test lexical or structural generalization, test examples are formed according to different sampling strategy; Sampling on unseen entities or parse tree structures is done for the test examples. In PCFG datasets [11], and these concepts were further summarized. In the context of semantic parsing, guaranteeing non-overlapping program templates between training and test programs [1] is also suggested too.

In Chapter 3, we propose number sequence prediction problems that require a model to predict the successive terms of number sequences whose initial terms are bigger than the ones in training examples. In particular, our problems provide further insights about systematic generalization abilities of neural networks in the lens of Automata theory. As the smallest automaton that can solve the given number sequence prediction problems is well-defined, we can directly compare computational powers between the deep models and the automaton.

2.3 Previous Approaches

In this section, we summarize various deep learning techniques for achieving compositional generalization into three branches: model-based, data-based, and task-based.

2.3.1 Model-based Approach

Designing new neural networks advantageous for capturing specific inductive bias can be a solution. For example, to process image data, Convolutional Neural Network (CNN) are specialized for grasping translation invariance. To capture the underlying inductive bias for systematic generalization, researchers have proposed following architectures.

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 writes it on the memory. For example, the Neural Turing Machine (NTM) [23], a de facto standard MANN, has differentiable external memory tapes that are analogous to recording tapes of Turing machine. These tapes can be read and written by controller networks such as perceptron networks or Long Short-Term Memory (LSTM). The NTM successfully handles out-of-distribution inputs in small algorithmic tasks like copy task, but it notoriously demands huge amount of computations for accessing the entire memory along time steps so that its scalability for more realistic tasks is questionable. To overcome the limited scalabilty, studies about improving NTM have been conducted [29, 30]. Also, different types of MANNs such as Stack-augmented RNN [25] for recognizing context free grammar or Memory network [31] for reasoning over synthetically generated story in natural language are introduced too. In Chapter 4, our preprocessed grid input can be seen as another representation of a se-

quential 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.

Neuro-symbolic Approach Another approach for achieving systematic generalization in symbolic reasoning problems is a neuro-symbolic approach that leverages a neural network that is trained to combine or choose given pre-defined domain-specific rules for solving task. To this end, memory-augmented neural networks (MANNs) and reinforcement learning techniques are often used.

To do symbolic reasoning over natural language or programming codes, Neural Programmer Interpreter (NPI) [32] and its recursion variant [33] have been proposed. To write compositional programs, these models generate sequential subprograms. Also, [34] proposes a reinforcement learning-based approach with structured parse-trees. Recently, Neural Symbolic Reader [35] trains models with weak supervision for generalization. In Chapter 4, our approach that uses automatic alignment without domain-specific knowledge is distinct from neural-symbolic approaches in that we require no pre-defined task-specific rule set for the alignment.

Especially, SCAN tasks have inspired many neuro-symbolic models as summarized in [19]. For example, NeSS [14] is a MANN equipped with a neural stack machine controlled by manually-defined instruction semantics. Among its instructions, $\texttt{CONCAT}_{\texttt{M}}$ or $\texttt{CONCAT}_{\texttt{S}}$ plays a crucial role to handle commands requiring repetitions like “around” or “twice”. Also, LANE [15] is another MANN consisting of the composer and the solver neural networks with memory. After the composer merges repetitive adjacent commands and yields analytic expression, the solver translates and records it on the memory. Note that the common bias behind both NeSS and LANE architectures is to capture repeating patterns of actions according to commands, resulting in perfect accuracy on all splits of the SCAN tasks. In Chapter 5, we infuse this bias by inserting delimiter tokens based on parsing trees rather than changing neural

network backbones.

2.3.2 Data-based Approach

Some researchers have leveraged diverse training data for achieving systematic generalization of deep learning models. The intuition behind this is that adding diverse training data makes out-of-distribution scenario be relaxed and become alike an in-distribution one. This approach is promising because it can be applied to standard models without changing their structures. We explain a few studies about this approach while focusing on how additional data are secured, e.g., borrowing from other domain data, synthesizing by handcrafted rules, or sampling from induced grammar.

Borrowing Data from Relevant Domain If there are data in other domain that shares the similar underlying rules with the domain of interest, it would be clever to use the data. For example, learning compositions in natural language could be helpful to solve compositional generation tasks. One can easily achieve this by using pre-trained transformers, i.e., T5 [36], that are widely used on a wide range of natural language tasks, such as machine translation, question and answering tasks, and natural language inference tasks [19]. This makes sense as pre-trained language models have already acquired general knowledge about natural language syntax and compositions. The idea of borrowing training data from relevant domains is also studied [37, 38]. However, all of these data-based approaches are not enough for training the standard seq2seq models to generalize on the other splits of the SCAN tasks like the length-split or the MCD-split. In Chapter 5, we present a data augmentation technique that works on those uncharted tasks.

Data Synthesis by Handcrafted Rules To secure compositionally diverse examples, one can design handcrafted rules to combine components of existing data. For example, to solve the jump-split SCAN task, GECA, a manual data augmentation method, [17], synthesizes new training examples by mixing given training examples'

components that have never been collocated. Also, [39] synthesizes more training examples about primitives usages within contexts by introducing hundreds of new primitives.

Data Sampling by Induced Grammar Researchers have attempted to directly discover rules underneath data via grammar induction. After a grammar is induced, one can readily sample new training examples from the induced grammar. [40, 41, 42] synthesized new training examples using training data' (quasi-) synchronous CFG induced by symbolic scaffolds. Data synthesis using neural parsers [9, 16] has been studied too. In Chapter 6, we use a rich enough template pool that requires data augmentations. Since we require unpaired templates which have a small variety than original programs (=targets), one can easily incorporate any of the aforementioned methods to our framework.

2.3.3 Task-based Approach

Given systematic generalization tasks, one can reformulate them under advantageous conditions leveraging auxiliary information. Intuitively, the task reformulation enables one to utilize this information that cannot be used in the original setup, leading to overall performance gains. Therefore, researchers have found available auxiliary information and used them to reformulate tasks, possibly by giving auxiliary supervision or entirely redefining the given inputs and targets.

Multi-task Learning with Auxiliary Targets Other than given input-target paired training examples, some researchers have tried to use readily available auxiliary information that can help to teach compositions of examples. One way to do so is to give supervision on input-target alignments. [8, 43] attain input-target alignment labels using off-the-shelf parsers and use them to train input-target attention outputs using cross-entropy loss. Also, [44] attaches an additional output head to a model and makes it predict the auxiliary depths of target tokens.

Reformulation of seq2seq Setup Other researchers have reformulated a given sequence transduction setup into another. [20] considers not only input-to-target mapping but also target-to-input mapping, i.e., back-translation, under the assumption that monolingual corpora of inputs and targets exist. By doing so, neural networks are successful at matching unpaired compositional novel input-target pairs. On the other hand, [45] considers a rule space, a collection of atomic input-target mapping rules, and then solves SCAN tasks by explicitly finding rules. Specifically, after defining a rule space with moderate complexity, new synthetic input-target pairs are sampled from the space. Then, a neural network trained to recover used rules from input-target pairs takes the input-target pairs of the original task and outputs our desired rules of the task. In Chapter 6, we reformulate the given a seq2seq task into a classification-and-template-augmented-generation task. By doing so, we can leverage a cheap and large template pool.

Chapter 3

Evaluating Computational Powers of Neural Networks with Number Sequence Prediction Problems

Well-defined machine learning tasks have been crucial for the researches. Major deep learning breakthroughs in the field of computer vision such as AlexNet [46], VGGNet [47] and ResNet [48] could not be possible without Imagenet dataset and challenges [49]. In the field of reinforcement learning, open platforms like MuJoCo [50] and Deepmind Lab [51] provide challenging environments for the studies. However, it is hard to find machine learning task suite for algorithmic reasoning although reasoning has always been a significant subject for many machine learning studies.

It is theoretically proven that carefully designed neural network models can simulate any Turing machine [52]. Hence, there have been studies applying neural network models to solve algorithmic tasks such as learning context-sensitive languages [53], solving graph questions [29], and composing low-level programs [54]. Also, there have been attempts to train neural networks with simple numerical rules such as copy, addition or multiplication [55, 56, 57, 58]. However, it has been unclear whether the proposed models express computational powers equivalent to Turing machines in practice. To provide a method to test the computational powers of neural network models, we propose a set of number sequence prediction problems designed to fit deep learning

methods.

A number sequence prediction problem is a kind of intelligence test for machine learning models inspired by number series tests, which are conventional methods to evaluate non-verbal human intelligence [59]. A typical number series test gives a sequence of numbers with a certain rule and requires a person to infer the rule and fill in the blanks. Similarly, a number sequence prediction problem requires a machine learning model to predict the following numbers from a given sequence. The numbers are represented as a sequence of digit symbols; hence the model has to learn discrete transition rules between the symbols such as carry or borrow rules.

To be specific, we suggest two types of number sequence prediction problems: the number-level problems and the digit-level problems. A number-level problem provides a two-dimensional grid of digits as an input where each row represents a multi-digit number. The target would be a grid of the same format filled with the following numbers. Solving a number-level problem is equivalent to constructing the combinatorial logic for the transition rules. On the other hand, a digit-level task provides a single digit as an input per each time step. A model needs to simulate a sequential state automaton to predict the outputs. The type of the state machine required can vary from a finite state machine to a Turing machine, depending on the generation rule of the sequence.

The number sequence prediction problems are good machine learning tasks for several reasons. First, typical deep learning models can easily fit into the problems. Generative models for 2D images can be directly applied to solve the number-level problems, and recurrent language models can fit into the digit-level problems after minimal modifications. Next, it is possible to define the complexity and difficulty of the problem. Like Kolmogorov complexity [60], we can define the complexity of a problem with the structure of the minimal automaton needs to be simulated. Finally, we can generate an arbitrarily large number of examples, which is hard for many machine learning tasks.

To empirically prove that the number sequence prediction problems can effectively

evaluate the computational capabilities of machine learning models, we conduct experiments with typical deep learning methods. We apply residual convolution neural network (CNN) [48] models for the number-level problems, and recurrent neural network (RNN) models with GRU [61] or LSTM [22] cells to the digit-level problems. We also augment RNN models with stack [55], external memory [58] and attention [62] which might help models solve more complex digit-level sequence prediction tasks. One-dimensional CNN models can be applied to digit-level sequences, but it is not equivalent to solving digit-level problems because for the CNN models the data needs to be given at the same time in parallel, losing the sequential nature of the problems. For each type of sequences, we measure the complexity of it by designing an automaton equivalent to the generation rule. In the experiments of the number-level problems, sequences are generated by various linear homogeneous recurrence relations. Since the digit transition rules of the relations can be implemented with combinatorial logic, we measure the complexity and the difficulty of a sequence from the width and the depth of the logic. Experiments show that CNN models are capable of learning the compound operations of number-level sequence generation rules but limited to certain complexity. Digit-level sequence prediction problems can be solved with state automata. Therefore, we define the complexity of a problem with the computing power of the automaton and choose sequences with complexities of finite state automata, pushdown automata, and linear bound Turing machines.

The contributions of this work are as follows:

- We propose a set of number sequence prediction problems for evaluating a machine learning model's algorithmic computing power.
- We define methods to measure complexities and difficulties of the problems based on the structure of automata to be simulated, which can predict the difficulty of training.
- Number-level sequence prediction experiments show that CNN models can sim-

ulate deep combinatorial logics up to certain depth.

- Digit-level sequence prediction tasks reveal that the computational powers of existing recurrent neural network models are limited to that of finite state automata or pushdown automata.

Overall, the set of our problems can be a well-defined method to verify whether a new machine learning architecture extends the computing power of previous models. There are some possible directions to extend the computational capabilities of neural network models. The first way is to apply training methods other than the typical methods we used in the experiments. For instance, reinforcement learning methods can be applied to the algorithmic tasks [63]. Next, non-backpropagation methods such as dynamic routing [64] might help neural network models learn more complex rules. Our number sequence prediction tasks would provide a well-defined basis for those possible future works.

3.1 Problem Definition

3.1.1 Number-level Sequence Prediction

Figure 3.1 illustrates a number-level sequence prediction problem. The model is given with an input sequence $A_1 \cdots A_n$ which is formatted as a two-dimensional grid with n rows. Each row corresponds to a term A_i which is a multi-digit number of l digits. A digit cell is a one-hot vector where the number of channels is equal to the base b of the digits. The target data $A_{n+1} \cdots A_{n+s}$ is a sequence of following numbers with the length shift s with the same data layout. In the experiments, we use sequence data of $n = 8$, $l = 8$ and $s = 4$. We denote this $\{0, 1\}^{l \times b}$ binary one-hot row tensor representation of a natural number A as $\langle A \rangle$.

We use various order-k homogeneous linear recurrence of the form $A_n = c_1 A_{n-1} + \cdots + c_k A_{n-k}$ with constant integer coefficients $c_1, \ldots, c_k$ to generate number sequences starting from randomly selected initial terms $A_1, \ldots, A_k$. For instance, $k =$

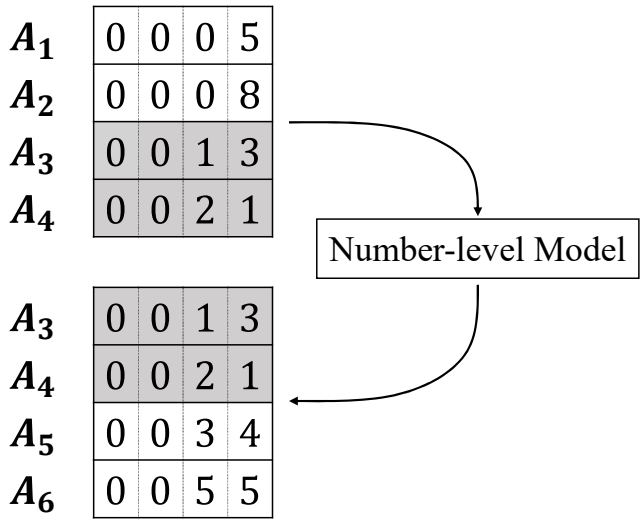

Figure 3.1: Input and target sequence examples of a number-level problem with the Fibonacci sequence. The number-level sequence example is with length $n = 4$, shift $s = 2$ and digit $l = 4$. A number in a cell is represented by an one-hot vector.

$2,\ c_1 = 1,\ c_2 = 1$ imply a general-Fibonacci sequence and $k = 2,\ c_1 = 2,\ c_2 = -1$ give an arithmetic progression. Likewise, a progression with arithmetic sequence as its difference whose recurrence is $A_n - A_{n-1} = A_{n-1} - A_{n-2} + c$ can be re-written in $A_n = 3A_{n-1} - 3A_{n-2} + A_{n-3}$. In the perspective of combinatorial logic, the generation rules of the sequences can be seen as k-ary operations of the binary tensors. For example, the generation rule of arithmetic sequences can be represented with a binary operation of $(\langle A\rangle, \langle B\rangle) \mapsto \langle 2A - B\rangle$. Since all inputs and outputs of an operation are binary, there exists a shortest disjunctive normal form (DNF) for the operation. We first define a combinatorial width of an operation with its disjunctive normal form, i.e. sum of minterms[1].

Definition) I. f the smallest DNF of a function $f : \{0,1\}^n \to \{0,1\}^m$ has $\Theta(w)$[2]

[1] A logical AND of literals in which each variable appears exactly once in true or complemented form [65]

[2] w is a function of input and output dimensions

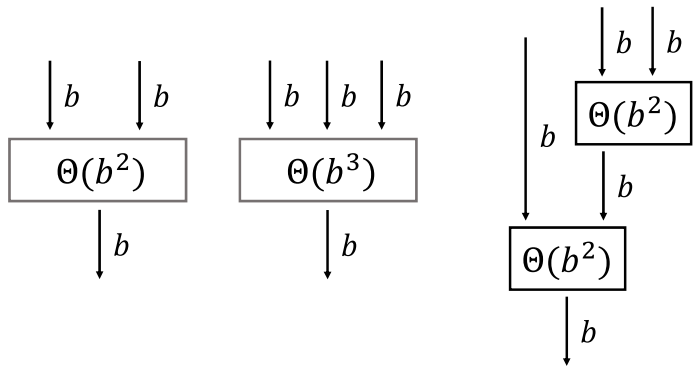

Figure 3.2: Conceptual schema of a binary operation (left), a ternary operation (middle) and an equivalent composition of two binary operations (right). The formulas represent combinatorial widths.

minterms, $\Theta(w)$ is called the **combinatorial width** of the function. If functions $f_1, \ldots, f_k$ have corresponding widths of $\Theta(w_1), \ldots, \Theta(w_k)$, the **compound width** of a composition $f_1 \circ \cdots \circ f_k$ is defined as $\Theta(w_1 + \cdots + w_k)$.

The decimal digit addition, for example, requires at least $\Theta(10^2)$ products since it has to memorize the consequences of all possible digit pair inputs. Therefore, the combinatorial width of a linear binary operation is $\Theta(b^2)$ where b is the base of the digits. Note that the compound width of a function is not unique. Consider a logical circuit for the ternary operation of $(\langle A\rangle, \langle B\rangle, \langle C\rangle) \mapsto \langle 2A - B + C\rangle$. As seen in Figure 3.2, the operation can be implemented with a single function of combinatorial width $\Theta(b^3)$, or a compound of two binary operations resulting in the compound width of $\Theta(b^2)$ in at the cost of a deeper data path. This depth of the path can define the complexity of the operation.

Definition) T. he **complexity** of a function $f : \{0,1\}^n \to \{0,1\}^m$ is the minimum number n of functions which make the compound width of $f_1 \circ \cdots \circ f_n = f$ the smallest. Such smallest compound width is called the **difficulty** of the function.

For example, the length of a row l is the complexity of the carry rule since the

carry digit of the most significant digit sequentially depends on all other digits. To eliminate the dependence on the dimensions, we ignore the carry or borrow rule while calculating a complexity. Since a logical product can be approximated with a neuron with a nonlinear activation, the difficulty should correspond to the number of neurons in the network. Also, since the complexity reflects the depth of a logical circuit, it should correspond to the number of layers in the network. Note that it is possible to compromise the width for the depth as seen in Figure 3.2. We expect deep neural networks to learn narrow but deeper representations.

3.1.2 Digit-level Sequence Prediction

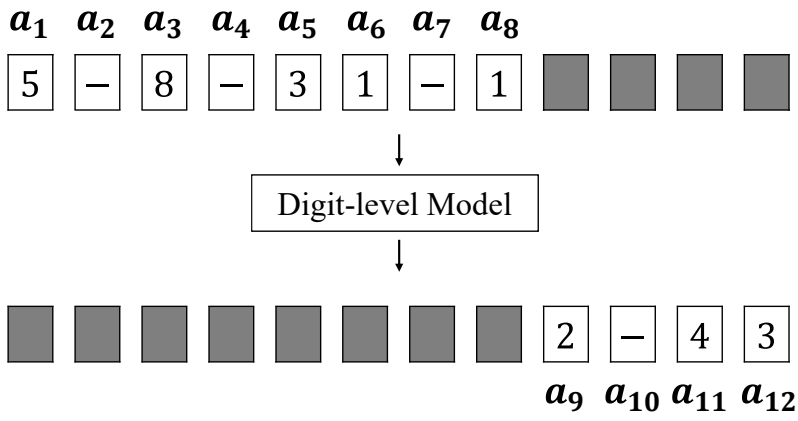

Figure 3.3: Input and target sequence examples of a digit-level problem with the Fibonacci sequence. The example is with $n = 8$ and $s = 4$. The order of the digits is little-endian (least significant digits first).

Figure 3.3 illustrates a digit-level sequence prediction problem. The model is given with sequential inputs of $a_1 \ldots a_n$, each of which is an integer number corresponds to a character. With the base of b, the numbers $0 \ldots b-1$ correspond to the digits. The second last number b is a blank, and the last number $b+1$ is a delimiter. After n inputs, we give delimiters as inputs for s time steps. The target sequence consists of n delimiters followed by $a_{n+1} \ldots a_{n+s}$. Because digit calculations must start from the

smallest digit, we order the digits in the little-endian order which is the reverse of the typical digit order. In the experiments, we use sequences of $n = 12$ and $s = 12$.

The sequential nature of the data makes it more difficult to solve the problems. Since the model has to retain information from the previous inputs, solving the problem is equivalent to modeling a sequential state automaton of the generation rule. The computing power of a state automaton falls into one of the four categories: finite state machine, pushdown automaton, linear bounded automaton, and Turing machine. All Turing machines are linearly bounded in the problems because the computation time is linearly bounded to the length of the sequence. Therefore, three levels of state automata are possible in the digit-level sequence prediction problems. We define the complexity of a sequence by the smallest state automaton required.

Definition) T. he **complexity** of a number sequence prediction problem is the complexity of a state automaton which can simulate the sequence generation rule with the smallest number of states. The **minimal grammar** of the sequences is the formal grammar can be recognized with the automaton.

To illustrate, we can think about the most straightforward sequence of number counting. If the numbers have at most l digits of base b, the counter can be implemented with $\Theta(lb)$ shift registers which can be translated to the same number of nondeterministic finite state automaton. Hence, the complexity of counting numbers is the complexity of finite state automata, and its minimal grammar is a regular grammar. In the experiments, we use progressions with a fixed difference because they can be understood as generalized forms of number counting sequences. Arithmetic, geometric and general-Fibonacci sequences can also be represented as digit-level sequences. The most straightforward automata capable of generating them are queue automata, which share the same computational powers with Turing machines. Since Turing machines must be linearly bounded in the digit-level problems, the minimal grammars of both arithmetic and general-Fibonacci sequences are context-sensitive grammars.

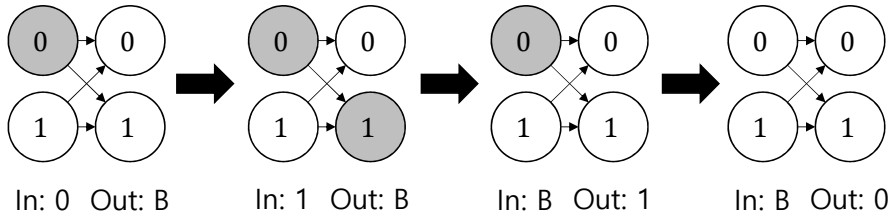

Figure 3.4: Nondeterministic finite state automaton that can solve reverse-order task with $n = 2$ and $b = 2$. Automata for fixed difference arithmetic sequence can be built in similar manners.

Between regular and context-sensitive languages, there are context-free languages which require pushdown automata. Palindromes are proper examples of context-free languages which cannot be expressed by lesser languages. Therefore, we add the experiment of a reverse-order task where the target sequence is the reverse of the input sequence. The input data consists of n random digits followed by n delimiters, and the target data is n delimiters followed by n digits, which is the reverse of the input sequence. If n is limited, it is possible to solve the reverse-order task with a finite state automaton as seen in Figure 3.4. Therefore, we train the models with $n = 1 \ldots 12$ and validate with $n = 16$ to force the complexity of the problem equivalent to a pushdown automaton.

3.2 Methods

3.2.1 Number-level Sequence Prediction Model

To solve number-level sequence prediction problems, we use a WaveNet-based model, as shown in Figure 3.5. Note that WaveNet model [66] is a generative model for sequential data. Since the data layout of number-level sequences is two-dimensional, we use 3×3 convolution kernels with dilation on the second dimension of the kernels where large receptive fields are necessary for the carry rules. Unlike WaveNet, we use

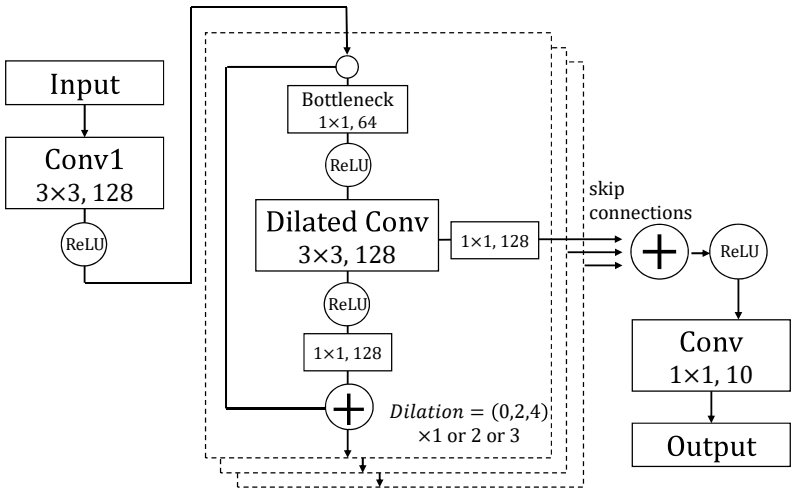

Figure 3.5: Schematic of number-level CNN models. The number of neurons in the convolution layers can be one of $64, 128, 192$. The residual blocks can be repeated once, twice or thrice, making 12, 21 or 30-layer CNN model.

ReLU activation because we empirically observe that the gate activation slows down the training speed but shows no improvement on the accuracy. The number of neurons per convolution layer can be 64, 128 or 192, which can correspond to the difficulty of a problem. Inspired by the bottleneck architectures of residual CNN [48], the first layer of each residual block has half the number of neurons. By stacking more residual blocks to the model, we can change the depth of the model. The base 12-layer model has three residual blocks of dilation (0,2,4), and they can be repeated to make 21-layer and 30-layer models. BatchNorm [67] and Dropout [68] methods are applied to all residual blocks.

3.2.2 Digit-level Sequence Prediction Models

To attack digit-level seqquence prediction problems, we use simple character-level RNN language models [69] with minimal modifications, as shown in the left side of Figure 3.6. LSTM [22], GRU [61], Stack-RNN [55] and Neural Turing Machine

(NTM) [58] are used for the recurrent modules in the middle. A Stack-RNN module uses a number of stacks equal to the base b, and an NTM module uses 4 read and write heads. A digit-level model with attention [62] follows the encoder-decoder architecture on the right side of Figure 3.6. The first half of an input sequence and the second half of a target sequence begun with the delimiter ($\langle$Go$\rangle$ symbol) are fed into the encoder and the decoder. We use both unidirectional and bidirectional LSTM modules for the models with attention. We set the number of neurons in all hidden layers to 128.

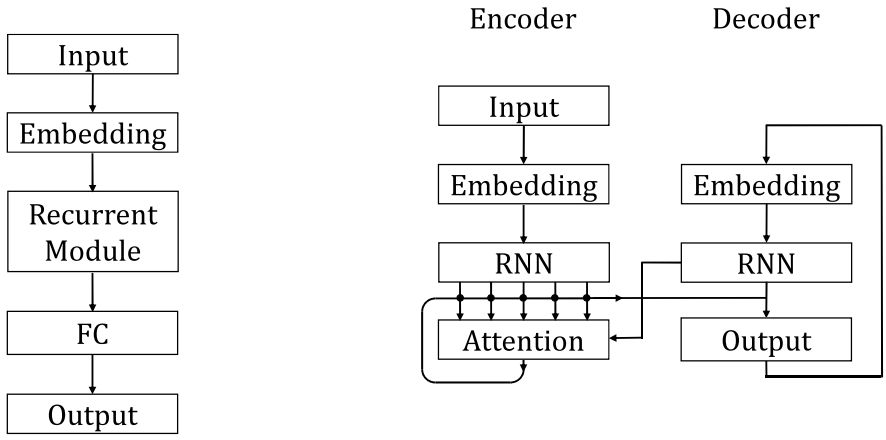

Figure 3.6: Schematics of digit-level neural network models. A recurrent module in a digit-level model can be either LSTM, GRU, Stack-RNN or Neural Turing Machine. Unlike other digit-level models, an attention model must follow the encoder-decoder structure which is illustrated on the right side.

3.3 Experimental Setup

3.3.1 Datasets

Number-level Sequence Prediction The objective of the experiments is to verify that complexity and difficulty of a number-level problem correspond to the depth and the parameter size of a CNN model. Total eight types of sequences are used in this part

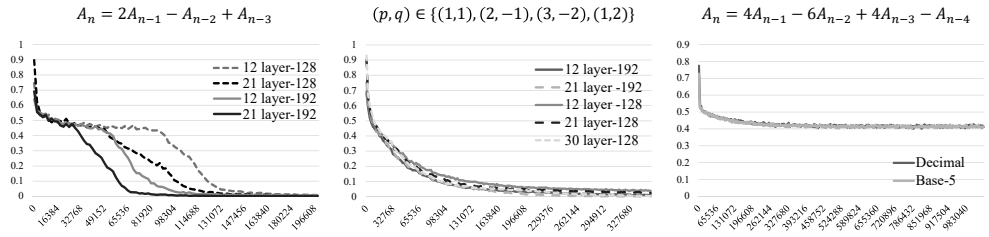

Figure 3.7: Validation error curves of the deep and wide models on the sequences generated by a ternary relation (left), the mixture of the four types of binary relations (middle) and the quaternary relation with different base digits (right). Y-axes are validation error rates, and the X-axes are training example counts. The value following the number of layers denotes the number of neurons in a convolution layer. The third experiment uses a 30-layer model with 128 neurons per layer.

of the experiments. First four types of the sequences have recurrence relations in the form of $A_n = pA_{n-1} + qA_{n-2}$ where $(p, q) \in \{(1, 1), (2, -1), (3, -2), (1, 2)\}$. These four sequences represent binary operations with the complexity of one. The fifth type of sequences has a relation of $A_n = A_{n-1} + A_{n-3}$ which represents a binary operation with the complexity of two because the model has to see through at least two layers to catch the relation between A_{n-1} and A_{n-3} with 3×3 convolution kernels. The sixth type of the sequences is a mixture of the first four types of the sequences. This is equivalent to building a ternary combinatorial logic with four times more width. For comparison, the seventh type of sequences is generated by a recurrence relation of $A_n = 2A_{n-1} - A_{n-2} + A_{n-3}$ which can be a compound of two binary operations. The last type sequence is a progression with a relation of $A_n = 4A_{n-1} - 6A_{n-2} + 4A_{n-3} - A_{n-4}$ whose general term can be calculated with a fourth-order polynomial. For the training data, the first k terms[3] of the sequences are chosen from $(0, 20000)$, while they are chosen from $(20000, 30000)$ in the validation dataset. We compare the learning curve patterns over various model configurations.

[3] k is the order of a recurrence relation

Digit-level Sequence Prediction The purpose of the digit-level sequence prediction experiments is to find complexity limits of the models. The first type of the sequences is a progression with a fixed difference, which can be understood as a variation of number counting sequences. We use the difference of 17 to observe the carry rules more often. The first term of a training data is chosen from the range of $(0, 9000)$, and that of a validation data is chosen from $(9000, 9900)$. In the second experiment, we use arithmetic sequences or general-Fibonacci sequences. The first two terms are chosen from the range of $(0, 4000)$ during the training and $(4000, 6000)$ for validations. Since it is impractical to build finite state automata for all cases, the model must simulate queue automata to solve the problems. The third experiment uses rounded geometric sequences with the relation $A_{n+1} = \lfloor 1.3A_n \rfloor$ where the first terms are randomly chosen from $(0, 4000)$ during the training and $(4000, 6000)$ for validations. The task also requires a smaller queue automaton since it has to remember only one previous number at a time. The last experiment tests the models with the reverse-order task, which has the complexity of a pushdown automaton. Since a reverse-order problem of fixed length can be solved by a finite automaton, we train the models with $n \in \{1 \dots 12\}$ and validate the models with $n = 16$ to force the models to learn a pushdown automaton.

3.3.2 Training and Validation

We follow the end-to-end training fashion. Thus the models have to learn the logical rules without any domain-specific prior knowledge. A batch of size 32 is randomly generated for each iteration by choosing the initial numbers and applying the generation rules. The space of all possible training sequences should be large enough to avoid overfitting. We evaluate the validation prediction error rate with a pre-defined validation dataset after every 32 iterations. We define a prediction error rate as a ratio of wrong predictions to the total predictions. The total predictions are counted as $l \times s = 32$ in number-level sequences and $s = 12$ in digit-level sequences. A prediction is determined by the digit channel with the maximum output value. Both number-level

and digit-level models are trained to minimize the cross-entropy loss function. The validation dataset is also randomly generated from the space outside of the training data space. For example, we choose the first two terms of number-level arithmetic sequences from the range of $(0, 20000)$ for the training dataset, but we choose them from the range of $(20000, 30000)$ for the validation dataset.

3.4 Results and Discussion

3.4.1 Number-level Sequence Prediction

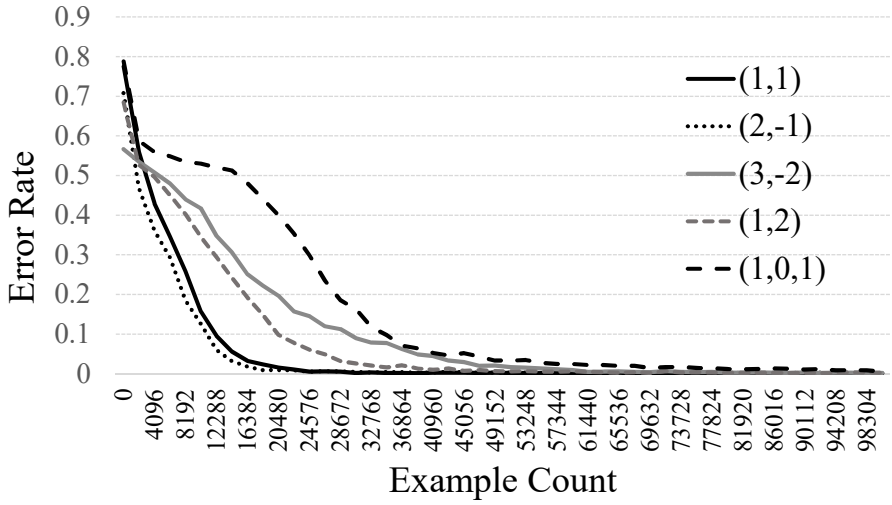

Figure 3.8: The learning curves of the 12-layer number-level model with 64 neurons on the five types of the basic sequences. (p, q) denotes the coefficients of binary operations $(A, B) \mapsto pA + qB$. $(1, 0, 1)$ denotes the relation of $A_n = A_{n-1} + A_{n-3}$.

Figure 3.8 and Figure 3.9 show the validation error curves and the error examples of a CNN number-level model during the training on the five types of sequences generated by the binary operations. Although the numbers of possible sequences exceed a hundred million, the model can achieve error rates near zero in less than a hundred

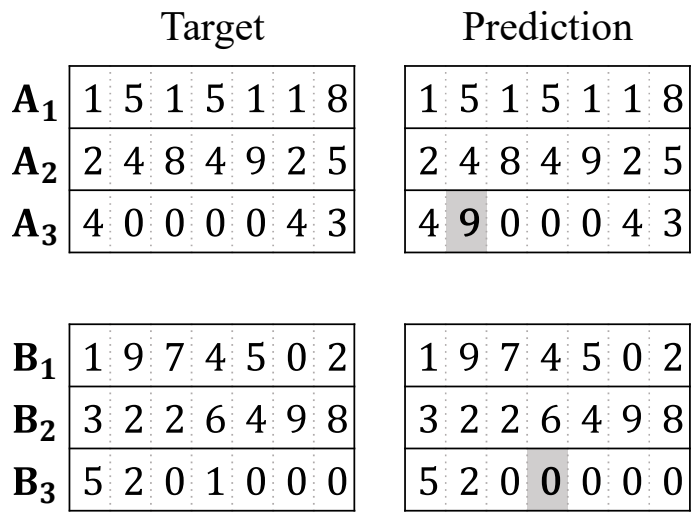

Figure 3.9: The error examples from number-level model trained with general-Fibonacci sequences. Shaded cells show the locations of the errors. The numbers are shown in little-endian order.

thousand examples. Since the validation data comes from the outside of the training data space, we can conclude that the model can learn the exact logic rules for the operations. The error examples show that it is hard to catch long-term carry rules, which is expected because the carry rules have complexities equal to $l = 8$. Deploying deeper models reduce the errors from those long-term carry rules, occasionally achieving a zero prediction error. The fifth sequence of rule $A_{n-1} + A_{n-3}$ shows a different learning curve pattern since 3×3 convolution kernels force the model to simulate the logic with the complexity of two. The results show that complexities of number-level sequence prediction problems can effectively predict the hardness of learning.

Figure 3.7 compares the learning curves of the models with various configurations and sequence data. The models successfully learn the rules from both the mixed set of primary sequences and the sequences generated by a ternary relation. However, the patterns of the learning curves are different. With the mixed set of primary sequences, the learning curves of the models show uniform convexity without a saddle point. Also,

Tasks	Reverse-order (training)	Geometric	Arithmetic	Fibonacci
LSTM	28.4% (1.2%)	79.4%	77.1%	80.5%
GRU	51.9% (0.9%)	69.0%	77.1%	79.3%
Attention(unidirectional)	42.0% (8.8%)	62.8%	77.0%	69.3%
Attention(bidirectional)	0.0% (0.0%)	51.0%	72.9%	60.9%
Stack-RNN	**0.0%** (0.0%)	64.1%	63.8%	69.4%
NTM	**0.0%** (0.0%)	57.1%	65.7%	68.1%

Table 3.1: Test error rates of the digit-level sequence prediction experiment. Identical training methods are applied to the models except the attention model. Parenthesized numbers in the reverse-order task column are training error rates with $n = 1 \ldots 12$.

there is no clear advantage of using deeper models. However, the learning curves with the sequences of a compound rule have saddle points, where we suspect the models find breakthroughs. Moreover, we can observe the advantages of using deeper models. Therefore, it can be concluded that deep learning models tend to learn complex but less difficult combinatorial logic, rather than the equivalent shallow but wide representations. Meanwhile, the last learning curves show that the CNN model finds it hard to learn the logic with the complexities more than three. The quaternary operator with base 5 has a smaller combinatorial width than a decimal ternary operator, but the model cannot learn the rule of the former.

3.4.2 Digit-level Sequence Prediction

Figure 3.10 shows that GRU and LSTM based models are capable of simulating finite state automata. Although the GRU model shows better performance than the LSTM model, both are not able to solve the problems that require queue or pushdown automata as seen in Table 3.1. Training error rates of GRU and LSTM models on reverse-order task converges around 0.01 suggesting that the models are capable of simulating

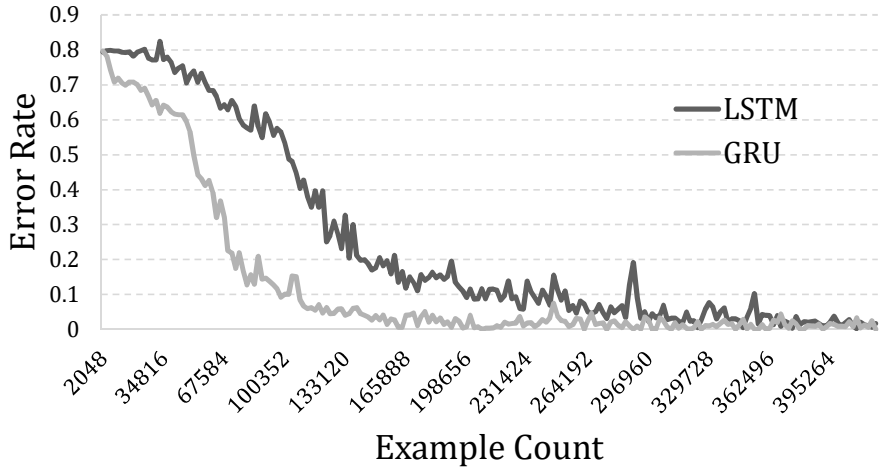

Figure 3.10: Validation error curves of LSTM and GRU digit-level sequence prediction models on the arithmetic sequences with fixed difference of 17.

Input	4	3	1	_	3	1	2													
Target							_	7	4	3	_	0	6	5	_	7	0	9	_	7	6	4
Output							_	3	4	3	_	5	6	5	_	3	0	9	_	9	5	4

Figure 3.11: Error examples from the digit-level LSTM model trained with general-Fibonacci sequences. The numbers are shown in little-endian order. Shaded cells show locations of the errors.

finite state automata for generating palindromes with a limited length. The error examples from the general-Fibonacci sequence prediction task in Figure 3.11 show the strategies of the models. The models remember relationships between the most significant digits, while relationships between the least significant digits are more critical for the digit computations. We can conclude that the computational powers of typical RNN models are limited to those of finite state automata if they are trained with typical training methods. Encoder-decoder model with attention, Stack-RNN and NTM models are capable of solving reverse-order task, but they are no better than typical RNN models in problems that require queue automata. The model with attention doesn't show significant differences if the model uses unidirectional LSTM. Using bidirectional LSTM seems to be crucial for simulating pushdown automata in the models with attention.

Chapter 4

Learning Symbolic Rules with Neural Sequence-to-grid Module

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 [70, 23, 71, 30, 6] and arithmetic addition [23, 25, 24, 72, 26, 73] 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 [74, 27, 75], complex mathematical equations [76], or programming snippets [6].

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 [7, 12, 26]. For instance, a model for the addition problem whose training inputs are a pair of numbers up to five dig-

its, 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 [77], 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 [78] and Transformer [79], rarely extend learned rules in that they are inclined to mimic the training data distribution [70, 12]. 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 [80] or has been limited to rudimentary logic such as copying sequences [70, 23, 71, 30, 6] and binary arithmetic [23, 25, 24]. 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 [81] 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 segmented 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 [25]. This mapping enables a joint training of our module and the neural network in an end-to-end fashion via a back-propagation.

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.

4.1 Motivation

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 4.1 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 [78] and Transformer [79], those in the grid-structured setup were fed to a ResNet-based CNN model [48]. As expected, Figure 4.2 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 non-trivial 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 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*.

4.2 Methods

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.

[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.

4.2.1 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 4.3. 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 [82].

4.2.2 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.

4.2.3 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 4.4, $G^{(t-1)}$ 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.

4.3 Experimental Setup

Implementation Details 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 [79], Universal Transformer (UT) [70] with dynamic halting[2], a LSTM seq2seq model (LSTM) [78], a LSTM seq2seq attention model with a bidirectional encoder (LSTM-Atten) [62] and a Relational Memory Core seq2seq model (RMC) [83]. 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.

4.3.1 Datasets

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

[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

(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 [81] 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, [26]. 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* [6] 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$`

bAbI QA Tasks Given as natural language with a small vocabulary of around 170, the bAbI QA tasks [75] 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 [75] 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 4.6. While previous work [70] uses sentence-level encodings, we use straightforward one-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 [84, 30]. 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.

4.3.2 Grid Decoders

CNN and ACNN Grid Decoders 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 [48] consisting of three stacks of 3-layer bottleneck building blocks of ResNet. Also, we implemented its attentional variant ACNN [85]; 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 4.5, 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 [86] with a learning rate $1e^{-3}$.

TextCNN Grid Decoder We chose a grid decoder as a variant of TextCNN [87]. 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 **??**. We used the ADAM optimizer to jointly train the seq2grid module and the TextCNN under a warm-up and decay learning rate scheme[3].

Table 4.1: 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)

	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**

4.4 Results and Discussion

4.4.1 Arithmetic and Algorithm Problems

Table 4.1 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 since it needs to align digit symbols on the grid

Table 4.2: 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.

	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

according to their scales. Indeed, Figure 4.7a shows that our module automatically finds such alignments that resemble the tailored grid of digits as shown in Figure 4.1.

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 $\star$ operations has non-linear time complexity, unlike addition or subtraction. Hence, we further investigate accuracy on snippets by those instructions as shown in Table 4.2. 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 4.8, the accuracy is fairly 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 $\star$ instructions on the OOD test set while the S2G-CNN does so on some examples as shown in Figure 4.8. These are surprising in that both the seq2grid module and the ACNN grid decoder do linear time computations in the input length.

4.4.2 bAbI QA Tasks

Our S2G-TextCNN outperforms sequential baseline models, such as the LSTM, the Transformer encoder, and the UT encoder, as shown in Table **??**. 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 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 **??**) indicates that the S2G-TextCNN consistently fails on the same set of tasks, as listed in Table 4.3. 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.

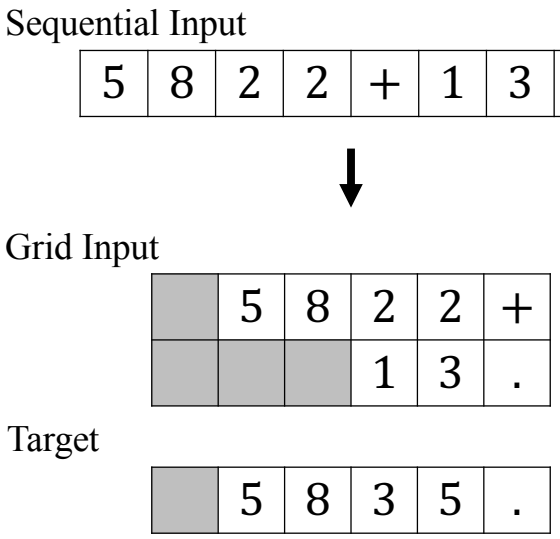

Figure 4.1: The illustration of the toy decimal addition problem. Each symbol is stored with its representation vector.

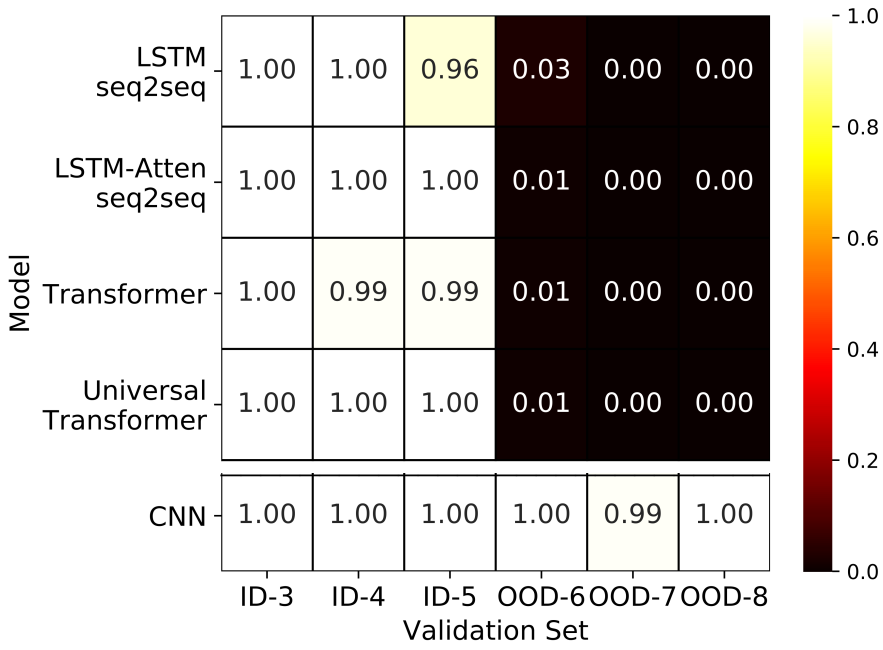

Figure 4.2: The validation accuracy results of the toy problem. Each column shows results from the k-digit set, where the three rightmost sets are OOD.

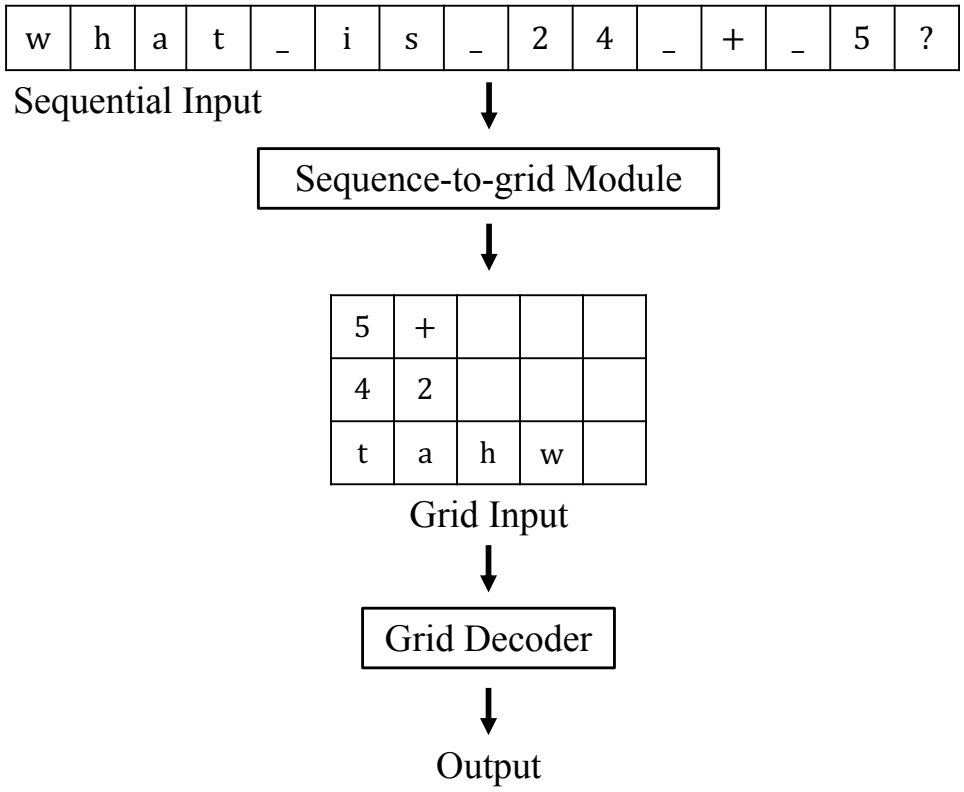

Figure 4.3: The sequence-input grid-output architecture.

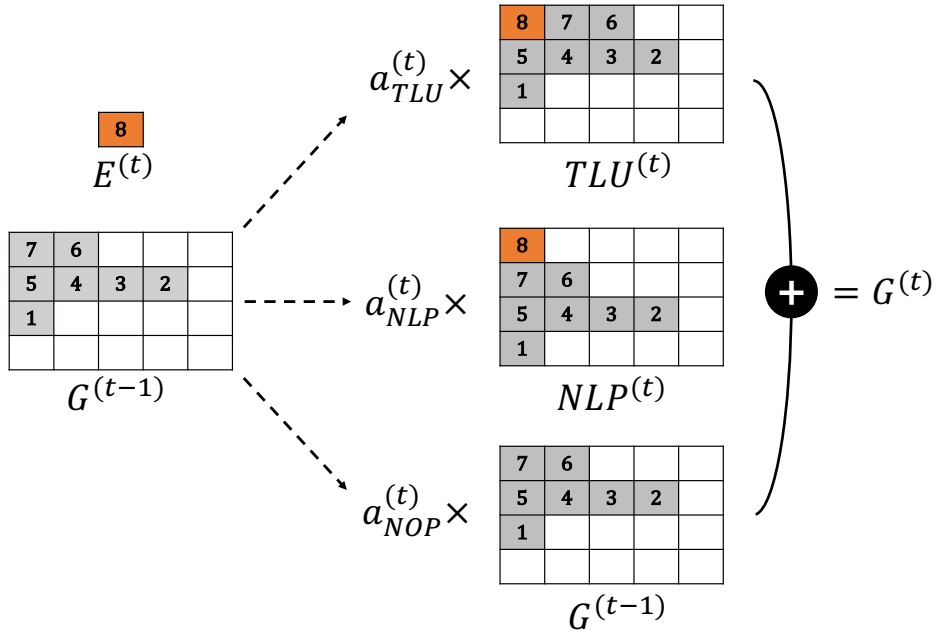

Figure 4.4: The nested list $G^{(t-1)}$ grows to $G^{(t)}$ by the action $a^{(t)} = (a_{TLU}^{(t)}, a_{NLP}^{(t)}, a_{NOP}^{(t)})$. $TLU^{(t)}$ and $NLP^{(t)}$ show outputs of *top-list-update* and *new-list-push* operations.

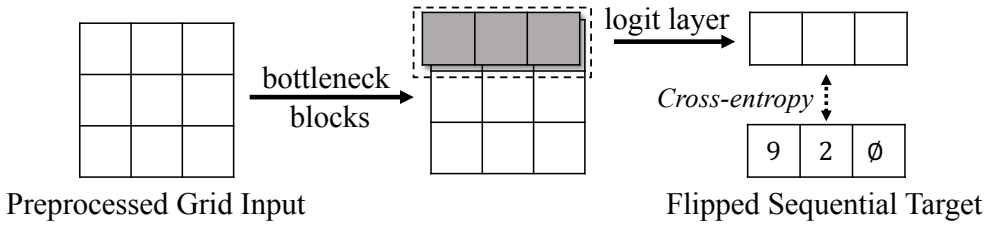

Figure 4.5: The grid decoder of S2G-CNN. Only the top list of the grid from bottleneck blocks is passed to the logit layer. The raw target `29` is flipped and padded to `92∅`.

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 4.6: Input examples of the bAbI QA tasks.

(a) Sequence (b) Program

Figure 4.7: 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.

```
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 4.8: Some OOD code snippets correctly answered by the best run of the S2G-CNN. Note that snippets contain `FOR` or $\star$ instruction requiring non-linear time complexity.

Table 4.3: 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.

		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

Chapter 5

Achieving Compositional Generalization via Parsing Tree Annotation

Humans can understand natural language by its compositionality [88, 89]. That is, even if a human reads a sentence written as novel combinations of known phrases or clauses, he or she can parse it into semantic or syntactic components.

To achieve artificial intelligence that possesses such ability, a large body of deep learning research [90, 91, 92, 15, 93, 82, 39, 14] has been carried out using the SCAN tasks [12], de facto standard compositional generalization problems. The SCAN dataset consists of finitely many commands, e.g., "run twice" and "walk right and run", and their corresponding target actions, e.g., "RUN RUN" and "RTURN WALK RUN". In particular, the SCAN dataset is divided into the training and the test set depending on specific compositionality of interest, and yields tasks like the jump-split task, the length-split task, and the MCD-split tasks. For example, in the training stage, the jump-split task shows commands like "run", "run twice", and "run after walk". However, it does not show "jump" with any context, such as "jump twice" or "jump twice after walk", except for "jump" itself. In the test stage, it asks those unobserved commands with "jump". As another example, the length-split task's test set suggests commands requiring longer actions than those that appear in the training set.

These SCAN tasks turn out to be extremely difficult to standard seq2seq deep learning models, such as RNN sequence-to-sequence (seq2seq) models [78, 62, 94] and self-attention based models [79, 95, 36]. This seems contradictory to recent advances, which are up to par with the level of human intelligence, in numerous natural language tasks including machine translation, natural language inference (NLI), and question and answering (QA). Moreover, studies about theoretical analysis of RNNs [96, 97] and self-attention [98, 99] have shown that their expressive powers are enough to capture the hierarchical structure of the SCAN tasks' language whose grammar allows only finitely many words.

Fortunately, it has been found that the standard models can solve the jump-split SCAN task with the help of pretraining [19] or data augmentation [17, 39]. This is possible since many additional examples apart from the given training ones enable the standard models to experience enough compositionality. However, none of these data-based approaches succeeds to make the standard models to resolve the other types of the SCAN tasks like the length-split or the MCD-splits[1].

To tackle those types of the SCAN tasks, there have been attempts to design new architectures largely deviated from the standard seq2seq architectures [14, 15]. These new architectures commonly exploit external memory allowing merging or concatenation operations aimed at the SCAN tasks' compositional rules such as "twice" or "thrice". However, such non-standard architectures with external memory for imposing task-specific inductive bias are only applicable to the SCAN or the SCAN-like tasks. Thus, they cannot be used to solve more complex and realistic applications where the standard models perform well.

In this work, to achieve compositional generalization with the standard seq2seq models, we suggest a novel data augmentation technique using parsing trees. The technique annotates each original target sequence by inserting a new delimiter token "<del>" in between the target for distinguishing its parsed components, as shown in

[1]The detailed description is deferred to Section 5.1

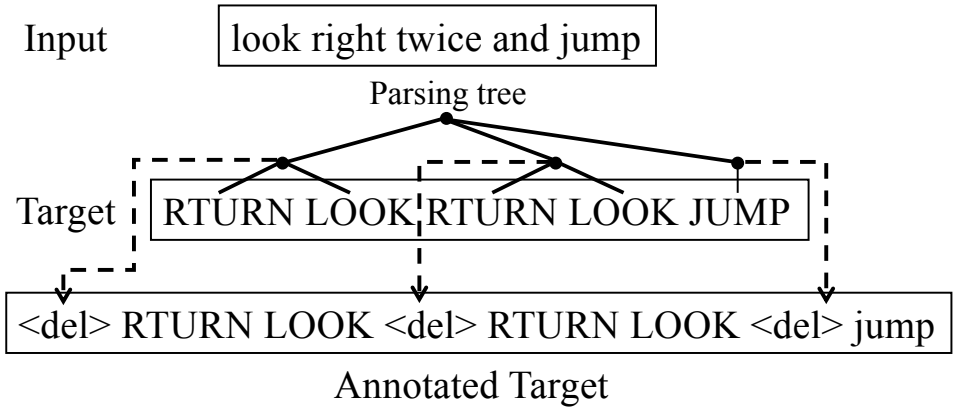

Figure 5.1: An example of applying our annotation technique. Delimiter tokens <del> indicate the beginnings of the parsed components obtained from the parsing tree.

Figure 5.1. For the training stage, the annotated targets are used instead of the original ones. Here, to obtain those parsed components, the technique uses prior knowledge about the original targets' semantic or syntactic compositionality. On the other hand, in the test stage, the technique does not need any such knowledge.

Empirically, we show that our technique enables the standard seq2seq models to achieve compositional generalization on the MCD-splits and the length-split of the SCAN dataset. We further validate our technique on a synthetic task and confirm the standard models' strong performance gains even without using prior knowledge about semantic compositionality. This shows our technique's applicability on more challenging compositional natural language tasks where syntactic parsing trees are readily available, such as programming code generation tasks [7].

5.1 Preliminaries: the SCAN Tasks

In this section, we introduce the SCAN tasks by split methods.

The goal of the SCAN (Simplified version of the CommAI Navigation) tasks [12]

Table 5.1: The ground-truth interpretation functions of the SCAN datasets. Here, double brackets $[\![\cdot]\!]$ denote the mappings from commands to actions (denoted by uppercase strings). Symbols x and u denote variables which are limited to primitives like "walk", "look", "run", and "jump". The linear order of movements denotes their temporal sequence.

$[\![\text{walk}]\!] = \text{WALK}$	$[\![\text{run}]\!] = \text{RUN}$
$[\![\text{look}]\!] = \text{LOOK}$	$[\![\text{jump}]\!] = \text{JUMP}$
$[\![x \text{ twice}]\!] = [\![x]\!][\![x]\!]$	$[\![x \text{ thrice}]\!] = [\![x]\!][\![x]\!][\![x]\!]$
$[\![x_1 \text{ and } x_2]\!] = [\![x_1]\!][\![x_2]\!]$	$[\![x_1 \text{ after } x_2]\!] = [\![x_2]\!][\![x_1]\!]$
$[\![\text{turn right}]\!] = \text{RTURN}$	$[\![\text{turn left}]\!] = \text{LTURN}$
$[\![u \text{ right}]\!] = \text{RTURN } [\![u]\!]$	$[\![u \text{ left}]\!] = \text{LTURN } [\![u]\!]$
$[\![\text{turn opposite right}]\!] = \text{RTURN RTURN}$	$[\![\text{turn opposite left}]\!] = \text{LTURN LTURN}$
$[\![u \text{ opposite right}]\!] = \text{RTURN RTURN } [\![u]\!]$	$[\![u \text{ opposite left}]\!] = \text{LTURN LTURN } [\![u]\!]$

$[\![\text{turn around right}]\!] = \text{RTURN RTURN RTURN RTURN}$

$[\![\text{turn around left}]\!] = \text{LTURN LTURN LTURN LTURN}$

$[\![u \text{ around right}]\!] = \text{LTURN } [\![u]\!] \text{ LTURN } [\![u]\!] \text{ LTURN } [\![u]\!] \text{ LTURN } [\![u]\!]$

$[\![u \text{ around right}]\!] = \text{RTURN } [\![u]\!] \text{ RTURN } [\![u]\!] \text{ RTURN } [\![u]\!] \text{ RTURN } [\![u]\!]$

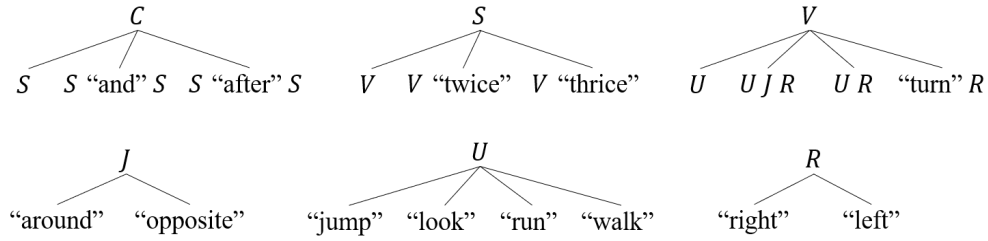

Figure 5.2: A grammar for commands

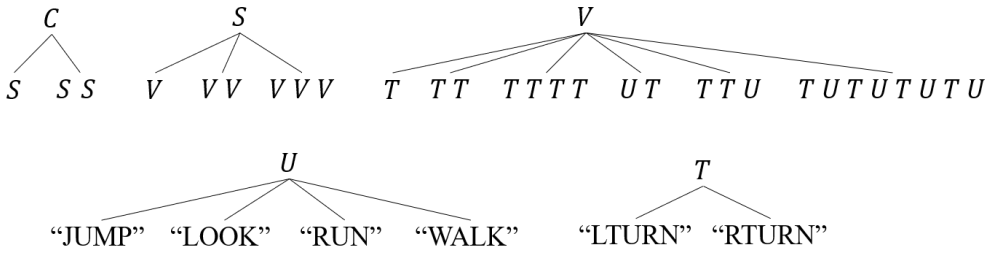

Figure 5.3: A grammar for actions

is to translate compositional navigation commands written in synthetic natural language into a sequence of actions. The inputs are commands, a total of 20,910, formed by a predefined grammar (Fig. 5.2) and the targets are actions that are the translation results of commands by the semantic interpretation mapping (Fig. 5.1). Depending on compositional generalization abilities to assess, split methods that divide all command-action pairs into the training or the test set are determined. Accordingly, specific tasks are defined as follows.

Random-split The training set is a random 80% subset of the total dataset and the test set is the remaining subset. Thus, this task is not for assessing compositional generalization ability but is used to test the given models' typical generalization abilities. Unlike other splits, the standard seq2seq models generalize well on the test set.

Jump-split The training set consists of all primitives, e.g., “jump”, “walk”, “run”, and “look”, and their composed commands, e.g., “run twice”, “walk opposite left and run twice”, except for composed commands of “jump”. The test set contains the remaining commands like “jump twice” and “jump after run”. Hence, to generalize on the test set, compositional understanding of “jump” along with other commands is necessary.

Length-split The training set has all 16,990 commands (81.3% of the total) requiring actions, i.e., targets, of lengths less than 24 and the test set has all remaining commands. Hence, the test set assesses the compositional generalization abilities about actions’ lengths.

MCD-split As the composition of commands can be explained by their grammar parsing trees, it is natural to consider a distribution over those trees’ subgraphs (compounds). To define the compound distribution of the dataset, DBCA (distribution-based compositionality assessment) method [7] captures the extent of how interesting a subgraph is within a parsing tree. Moreover, the method can serve as criteria to divide the SCAN dataset into the training and the test set, yielding three compositional generalization tasks, such as MCD1, MCD2, and MCD3. Whereas the training and test set of each task have similar distributions over nodes (atoms) of parsing trees, they have distinct distributions over subgraphs (compounds).

5.2 Motivation

In the next section, we explain the motivation of our annotation technique by pointing out that the SCAN dataset allows abundant many-to-one cases.

Many-to-one Cases Learning the compositions in the SCAN tasks can be regarded as discovering which part of the commands, e.g., the second “*run*” in the “run after

run" or the first "*run*" in the "*run* and run", corresponds to which part of the actions, e.g, the first "*RUN*" in the "*RUN* RUN". Hence, it is natural to assume that the presence of multiple commands corresponding to the same action sequence makes it harder to learn the compositions, i.e. many-to-one. In fact, such cases are fairly common in the SCAN dataset as its non-injective semantic interpretation function maps 20,910 commands to only 9,228 different actions. In the extreme case, the action sequence "RTURN RTURN RTURN RTURN RTURN RTURN" is the target of 19 different commands such as "turn around right and turn right twice", "turn opposite right thrice", and "turn right twice and turn opposite right twice".

5.3 Method

In the next section, we introduce our annotation technique using parsing trees to handle them.

Parsing Tree Annotation Technique We hypothesize that such abundant many-to-one cases confuse the standard seq2seq models to learn the compositionality. To reduce many-to-one cases, we annotate targets by inserting new delimiter tokens in between the actions according to the commands.

Specifically, our annotation technique can be described as follows. First, we induce a grammar for the target language, possibly via human parser or grammar induction heuristics [100] [101]. At this point, it is desirable to induce the grammar that can capture the compositionality of the target language with the minimum number of relations. Then, we obtain the parsing tree for each target sequence. In some cases, the induced grammar may allow multiple parsing trees, i.e., the grammar is ambiguous. To uniquely decide the parsing tree, we refer to the input sequence and the semantic interpretation function. Finally, we choose a non-terminal variable where a new delimiter token like "¡del¿" to be attached. We insert the token before every substring generated from the variable. These annotated targets, instead of original ones, are used for the

training. See Section 5.4.2, and Section 5.4.3 about specific implementations for the SCAN dataset and our synthetic dataset, respectively.

5.4 Experimental Setup

In this section, we describe the implementation details of the standard seq2seq models. Then, we introduce experimental details, such as specific implementations of our annotation technique, for the SCAN tasks and the multiplicative extension tasks.

5.4.1 Standard Seq2seq Models

We verified our technique with the standard seq2seq models: an LSTM seq2seq model (LSTM), a GRU seq2seq model (GRU), [78], and those with Bahdanau attention [62] (LSTM-Atten, GRU-Atten), a Transformer [79], and a T5 [36]. All the RNN seq2seq models had one layer with the hidden size 50 and the dropout rate 0.5. The Transformer and the T5 consisted of six layers with the hidden size 512 and eight attention heads. We used the ADAM optimizer [86] with a learning rate $1e^{-3}$ to train the RNN seq2seq models and the T5. As for the Transformer, we varied learning rates along the course of the training [79]. All models could fit in a single NVIDIA GTX 1080ti GPU. Our implementations[2] except for the T5 based on the open source library *tensor2tensor*[3] while we used *hugging face trasnformers*[4] for the T5.

5.4.2 The SCAN Tasks

Annotation Implementation For the SCAN tasks, we used the ground-truth interpretation function when (i) inducing grammar for the action language and (ii) obtaining the unique parsing tree for each action sequence.

[2]https://github.com/segwangkim/annotation-of-targets-using-parsing-trees

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

[4]https://github.com/huggingface/transformers

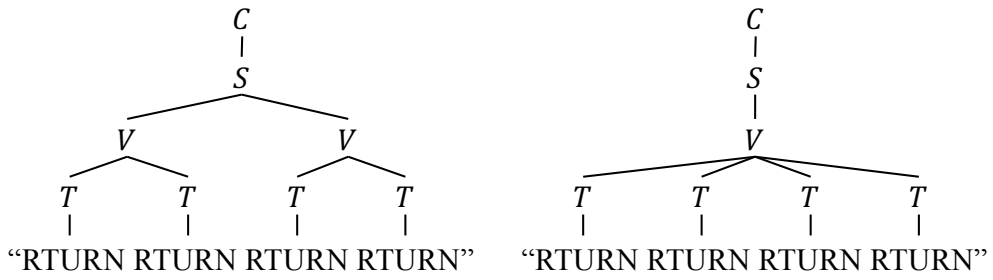

Figure 5.4: Two possible parsing trees for "RTURN RTURN RTURN RTURN". The left and right tree are come from commands "turn opposite right twice" and "turn around right", respectively.

We annotated the examples as follows. First, we manually induced a grammar for the action language, as shown in Fig. 5.3, to be similar to the given grammar of the command language. Then, we obtained the unique parsing tree for each action sequence according to its command's parsing tree and the ground truth interpretation function $[\![\cdot]\!]$. For example, the action sequence "RTURN RTURN RTURN RTURN" from "turn opposite right twice" and "turn around right" corresponded to the left and the right of Fig. 5.4, respectively. Finally, we inserted a new delimiter token "¡del¿" before every substring generated from the non-terminal variable V. For the aforementioned example, the former action sequence was annotated as "<del> RTURN RTURN <del> RTURN RTURN" while the later one was annotated as "<del> RTURN RTURN RTURN RTURN". This annotation process is summarized as Alg. 1.

Tokenization for T5 Whereas we fed all sequences word-by-word for the RNN seq2seq models and Transformer, we had no choice but to use the pretrained tokenizer coupled with the T5. Using the pretrained tokenizer could be problematic as the tokenizer took the raw actions and segmented them into components that capture

Algorithm 1 Annotation process for the SCAN dataset

Input: command sequence C.

def PTA(C): // stands for Parsing Tree Annotation

if "and" **in** C:

$C1 \leftarrow$ the command before "and" within C

$C2 \leftarrow$ the command after "and" within C

return PTA($C1$) + PTA($C2$)

if "after" **in** C:

$C1 \leftarrow$ the command before "after" within C

$C2 \leftarrow$ the command after "after" within C

return PTA($C2$) + PTA($C1$)

if "twice" **in** C:

$C \leftarrow$ the command before "twice" within C

return PTA(C) + PTA(C)

if "thrice" **in** C:

$C \leftarrow$ the command before "thrice" within C

return PTA(C) + PTA(C) + PTA(C)

return ["¡del¿"] + $[\![C]\!]$

Output: annotated action sequence PTA(C).

no semantics. For example, "I_TURN_LEFT"[5] was segmented into ("I", "_", "TUR", "N", "_", "LE", "FT"). Thus, compositional rules or linguistic semantics learned from the pretraining corpus became useless for the fine-tuning on the SCAN tasks. To resolve this issue, we preprocessed actions with straightforward modifications, e.g., "I_TURN_LEFT" to "lturn", before applying the tokenizer. By doing so, the pretrained tokenizer segmented actions more reasonably, e.g., "lturn" was segmented into ("l", "turn"). We denoted a T5 with the above tokenization method as T5*.

[5]The original SCAN datasets (https://github.com/brendenlake/SCAN.git) represents actions as snake upper case with leading "I" such as "I_TURN_LEFT", "I_LOOK", and "I_JUMP" unlike Fig. 5.3.

Table 5.2: The training and test datasets of the multiplicative extension tasks. The number of possible alphabets is fixed as 7 (“a” to “g”).

		n	k	min(d_i)	max(d_i)
	Training	7, 8, 9	3	0	7
	ID	7, 8, 9	3	0	7
Test	OOD-Easy	12	3	1	7
	OOD-Hard	18	3	1	7

5.4.3 Multiplicative Extension Tasks

Observe that we used prior knowledge about the ground-truth interpretation function and the the input and the target sequences’ grammars for the SCAN tasks. To validate our technique’s applicability under minimal prior knowledge, we further suggest a simple synthetic task that requires neither the interpretation function nor parsing trees of input sequences for applying our technique.

Task Definition The goal of multiplicative extension tasks is to translate a sequence of alphabet-number alternating terms into alphabet sequences. The input is given as:

$$a_1 d_1 a_2 d_2 \cdots a_k d_k$$

where a_i is an alphabet sampled from an alphabet set $\Sigma = \{\text{“a”}, \ldots, \text{“g”}\}$ without replacement and d_i is one-digit integer. The target under the ground-truth interpretation function f is given as:

$$f(a_1 d_1 a_2 d_2 \cdots a_k d_k) = \underbrace{a_1 \cdots a_1}_{d_1} \cdots \underbrace{a_k \cdots a_k}_{d_k}$$

Thus, this task tests multiplicative compositionality similar to that of the SCAN tasks, i.e., “twice” and “thrice”. We define training and test sets according to the maximum length of targets, $n = d_1 + \cdots d_k$, as shown in Table 5.2.

Note that this task shares no linguistic compositionality with natural language corpus so that there is no advantages from pretraining. Hence, we omit to test T5 at this task.

Annotation Implementation We used straightforward parsing trees that explain the target sequences. For given target $a_1 \cdots a_1 \cdots a_k \cdots a_k$ where each a_i repeats d_i times, we inserted a new delimiter token $s \notin \Sigma$, e.g., $s =$ "¡del¿", before the repetitions of the same a_i's, yielding $sa_1 \cdots a_1 sa_2 \cdots a_2 \cdots sa_k \cdots a_k$. This is natural as a grammar of the target language can be defined as $S \to V \mid VV \mid VVV \cdots$, $V \to T \mid TT \mid TTT \mid \cdots$ where S is a start symbol, V is a non-terminal symbol, and $V \in \Sigma$ is a terminal symbol. Note that this annotation was independent of parsing trees of inputs and the interpretation function f.

5.5 Results and Discussion

In this section, we empirically verify the effectiveness of our annotation technique in various aspects. First, we point out that our technique can reduce many-to-one cases in the SCAN tasks. Then, we present the efficacy on our technique to the compositional generalization tasks. Finally, we discuss about our technique in the view of Automata theory and the effect of a target tokenization method on the compositional generalization of the standard models.

5.5.1 Effects of Parsing Tree Annotation on Datasets

First, we analyze how much our technique reduces many-to-one cases of the SCAN dataset. To do so, let us formally describe the dataset as follows. Let $\mathcal{L}_C$ be the set of command sequences. Let $\mathcal{L}_A$ and $\mathcal{L}_A^\dagger$ be the sets of action sequences before and after applying our annotation technique, respectively. Accordingly, we also define $[\![\cdot]\!] : \mathcal{L}_C \to \mathcal{L}_A$ and $[\![\cdot]\!]^\dagger : \mathcal{L}_C \to \mathcal{L}_A^\dagger$, i.e. PTA in Alg. 1, as corresponding ground-truth interpretation functions.

Table 5.3: The sequence-level accuracy before and after applying our annotation technique (PTA) on SCAN tasks results (for 5 runs). T5* is a T5 with our tokenization method (refer to 4.5.2).

split		GRU	LSTM	GRU+Atten	LSTM+Atten	T5*	T5	Transformer
length	baseline	0.13 ± 0.06	0.14 ± 0.03	0.21 ± 0.02	0.15 ± 0.02	0.18 ± 0.02	0.17 ± 0.02	0.00 ± 0.00
	+PTA	0.96 ± 0.05	0.80 ± 0.15	0.99 ± 0.04	1.00 ± 0.00	0.94 ± 0.03	0.16 ± 0.08	0.00 ± 0.00
mcd1	baseline	0.19 ± 0.08	0.10 ± 0.02	0.32 ± 0.23	0.23 ± 0.04	0.07 ± 0.02	0.16 ± 0.10	0.03 ± 0.01
	+PTA	0.65 ± 0.07	0.71 ± 0.08	0.96 ± 0.08	0.94 ± 0.11	0.60 ± 0.12	0.80 ± 0.07	0.01 ± 0.01
mcd2	baseline	0.11 ± 0.09	0.09 ± 0.01	0.10 ± 0.03	0.11 ± 0.01	0.09 ± 0.02	0.10 ± 0.02	0.04 ± 0.01
	+PTA	0.18 ± 0.11	0.37 ± 0.31	0.52 ± 0.20	0.80 ± 0.15	0.28 ± 0.15	0.58 ± 0.10	0.00 ± 0.00
mcd3	baseline	0.19 ± 0.07	0.16 ± 0.05	0.23 ± 0.07	0.32 ± 0.05	0.08 ± 0.01	0.08 ± 0.01	0.05 ± 0.01
	+PTA	0.49 ± 0.12	0.58 ± 0.15	0.91 ± 0.14	0.76 ± 0.15	0.58 ± 0.12	0.58 ± 0.06	0.02 ± 0.01
mcd (mean)	baseline	0.16 ± 0.08	0.12 ± 0.03	0.22 ± 0.11	0.22 ± 0.03	0.08 ± 0.01	0.11 ± 0.04	0.04 ± 0.01
	+PTA	0.44 ± 0.10	0.55 ± 0.18	0.80 ± 0.14	0.83 ± 0.14	0.49 ± 0.13	0.65 ± 0.08	0.01 ± 0.01

Table 5.4: The sequence-level accuracy on the multiplicative extension task (for 5 runs).

		GRU	LSTM	GRU+Atten	LSTM+Atten	Transformer
ID	baseline	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.85 ± 0.21
	+PTA	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
OOD	baseline	0.03 ± 0.05	0.33 ± 0.14	0.00 ± 0.00	0.04 ± 0.07	0.00 ± 0.00
Easy	+PTA	0.28 ± 0.15	0.35 ± 0.10	0.98 ± 0.05	0.97 ± 0.02	0.00 ± 0.00
OOD	baseline	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Hard	+PTA	0.00 ± 0.00	0.00 ± 0.00	0.32 ± 0.21	0.17 ± 0.15	0.00 ± 0.00

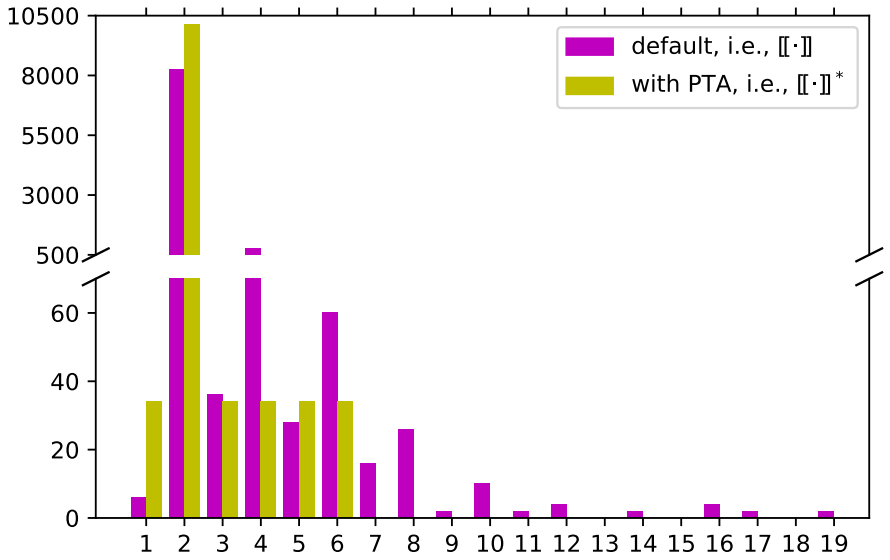

Figure 5.5: Histograms of many-to-one cases for $[\![\cdot]\!]$ and $[\![\cdot]\!]^*$. Each n of x-axis is the number of commands that are mapped to the same actions while its height is the frequency of such n-to-one cases.

To count many-to-one cases, for each interpretation function $f = [\![\cdot]\!]$ or $[\![\cdot]\!]^\dagger$, we first partitioned the domain $\mathcal{L}_C = \cup_{i=1}^{k} C_i$ by disjoint cells where each cell C_i is the set of commands that are mapped to the identical actions, i.e., $f(x) = f(y)\ \forall x, y \in C_i$. Then, according to the sizes of those cells, i.e., $|C_i|$, we counted the frequency and visualized it as a histogram as shown in Fig. 5.5.

While the total areas of all histograms under functions $f = [\![\cdot]\!]$ and $[\![\cdot]\!]^\dagger$ are identical as 20,910, the shapes of them are different. Note that the histogram for $[\![\cdot]\!]$ has a long tail whereas that for $[\![\cdot]\!]^\dagger$ has a short tail, indicating that one-to-many cases are significantly reduced by our annotation. From this result, we can conclude that our technique helps the standard models discover the training examples' compositionality significantly easier.

5.5.2 The SCAN and Multiplicative Extension Tasks

After applying our annotation technique, the SCAN length-split task is almost perfectly solved by the attentional RNN seq2seq models (GRU+Atten, LSTM+Atten) and the T5 with the manual tokenization (T5*) as shown in Table 5.3. Moreover, we also obtain huge performance gains at the multiplicative extension tasks using the attentional RNN seq2seq models as shown in Table 5.4. These imply that the models have sufficient expressive powers to handle test commands requiring longer target sequences. In other words, the standard seq2seq models fail on the length generalization tasks as they learn incorrect compositionality during the training stage.

Rather than the length generalization, our technique also induces huge performance gains for the SCAN MCD-split tasks as shown in Table 5.3. One may argue that the compound distribution discrepancy between the training and test set can be changed as our technique inserts delimiter tokens for the actions. Thus, our comparison between the results on the MCD-splits before and after applying our technique seems unfair. However, our comparison is still valid since the way of measuring the discrepancy only depends on parsing trees for commands, not actions.

Unfortunately, the Transformers still fail to generalize in all tasks regardless of our annotation. In other words, our technique is effective only for self-attention based models that have experienced sufficient linguistic compositionality from pretraining in advance.

5.5.3 Discussion

Automata-theoretic View One might think that all aforementioned results are achieved since the tasks become much easier after the ground-truth interpretation functions are modified as shown in Alg. 1. However, in the view of automata theory, our technique makes no difference in the level of difficulty. Indeed, both tasks before and after applying our annotation can be implemented by a Finite State Transducer (FST). To see this, note that FST can implement a rational relation between two regular languages. The

source language, e.g., $\mathcal{L}_C$, and the target language, e.g., $\mathcal{L}_A$ or $\mathcal{L}_A^\dagger$, have finitely many words, hence they are all regular languages. Moreover, the interpretation function, e.g., $[\![\cdot]\!]$ or $[\![\cdot]\!]^\dagger$ is a rational relation as the graph of the function is finite. Therefore, the difficulties of both tasks cannot be distinguished.

Tokenization Effects The performance discrepancies between the T5 and the T5* for the SCAN length-split and MCD-split tasks with our annotation are notable. In particular, as for the length-split, our annotation cannot induce any performance gain at all when the actions are naively tokenized without considering their semantics (T5).

Note that we can think of inserting delimiter tokens to the target sentences in the multiplicative extension tasks as another tokenization for the targets. This is because tokenization is transforming a string into a sequence while considering adjacent characters that co-occur frequently.

Henceforth, we can conclude that a tokenization method can significantly influence the compositionality that the standard models learn. To go further, at compositional generalization task whose targets follow strict grammar, it is beneficial to use tokenization based on the grammar. For example, code generation tasks have targets that can be represented as abstract syntax trees (AST). Since our technique annotates the target sequences with delimiters indicating specific non-terminal nodes of the targets' parsing trees, trees' compositionality can be infused into the sequences. Thus, we expect that our technique can promote the standard seq2seq model to achieve compositional generalization on those tasks.

Chapter 6

Achieving Compositional Generalization via Retrieving and Reranking Templates

Human compositional reasoning ability makes them capable of semantic parsing, translating a natural language utterance into a semantically identical machine-executable program. For example, human can understand a program, e.g., `animal.species=frog ∧ animal.color=green`, as a composition of a template, e.g., {*COND1*} ∧ {*COND2*}, and basic components, e.g., {*COND1*}: `animal.species=frog` and {*COND2*}: `animal.color=green`. Hence, human can compose a new program, e.g., `animal.species=frog ∧ animal.color=white`, even if the input utterance, e.g., "*A white frog*", has rarely co-occurred words [2, 3].

Even though the current deep learning models have demonstrated their exceptional performances on a wide range of natural language processing (NLP) tasks, they still struggle to do compositional reasoning [10, 26, 28]. Without resolving this issue, deep learning models cannot be used for real-world applications like Natural Language Interfaces to Databases (NLIDB) [4], low-resource machine translation [5], and higher-level perceptual reasoning (e.g., counting, position comparison) about visual information [102, 103].

To explore neural networks' compositional reasoning, studies have released com-

positionally challenging testbeds across diverse domains, including semantic parsing [13, 8, 1]. Other studies have been conducted to improve neural networks' compositional reasoning in various ways, such as developing specialized architectures [104, 105] or harnessing pre-computed auxiliary information [43, 106, 44, 1, 107]. In particular, studies using compositionally diverse paired data [108, 17, 16, 41, 9] recently have drawn attention, but they still require careful data augmentation. In contrast, unpaired data can be readily augmented; hence one needs to propose a method to leverage this cheap data.

In this work, we propose a new framework that leverages the compositionally diverse unpaired template pool via retrieving and re-ranking approaches. Our framework takes an utterance and selects its corresponding template from the pool even if the utterance is compositionally novel. The selected template leads a neural semantic parser, e.g., T5 [36], to achieve compositional generalization.

To select the template from the pool, we subsequently apply a bi-encoder and a cross-encoder. As shown in Figure 6.1, our bi-encoder, e.g., Dense Passage Retriever [109], retrieves a small number of candidate templates relevant to the input utterance from the template pool. Then, our cross-encoder, e.g., BERT re-ranker [110], re-ranks those candidates. Finally, we select the template of the highest re-ranking score.

Our framework has the following two advantages. First, we use a template pool that relieves the burden of utterance-program paired data augmentation. The augmentation methods demand to search over a combinatorially large sample space derived by the grammar of utterance-program pairs [9, 16, 41]. Instead, we use a pool of program templates that have relatively few grammar rules. Second, our framework uses templates as additional inputs, resulting in performance gains. Indeed, templates that are rough sketches of the target programs enable a semantic parser to easily generate novel programs, because the parser only needs to fill in the missing entities [111, 112]. Still, obtaining correct templates via *generation* can be tricky, but we tackle this issue with *retrieving* and *re-ranking*.

Experimental results show that our framework enables a neural network to achieve strong compositional reasoning on four widely used semantic parsing datasets like Advising, ATIS, GeoQuery, and Scholar. In particular, our bi-encoder successfully retrieves the ground-truth templates within 50 candidates, and our cross-encoder selects the ground-truth templates among those candidates. On compositional test sets over the four datasets, templates selected by our framework improve the average accuracy by 12.6 points compared to T5 baselines. Ours is the first retrieval-based approach to achieve compositional generalization.

6.1 Preliminaries

In this section, we formulate our task with an assumption and introduce datasets and their abstraction. Then, we provide some background on retrieval and template-based generation, which are the techniques used in this paper.

6.1.1 Task Formulation

Our goal is to map a natural language question (utterance) x into its corresponding program y_x. We have a training, validation, and test set (D_{train}, D_{dev}, and D_{test}) whose elements are utterance-program pairs (x, y_x). We also have a manually-defined abstraction $t: \ y_x \mapsto t(y_x)$ that maps a program into its template. Typically, we require that the programs that share a structural similarity collapse into one template, and the number of templates is smaller than that of programs, i.e., $|\{t(y) \ |(x, y) \in D\}| < |\{y \ |(x, y) \in D\}|$.

Sufficiently Large Template Pool We further assume that we have a template pool that is compositionally diverse enough to contain all templates of the training, validation and test set, i.e., $P_{\text{template}} = \{t(y_x) \mid (x, y_x) \in D_{\text{train}} \cup D_{\text{dev}} \cup D_{\text{test}}\}$. This is a plausible assumption because one can transform a cheap and large unpaired program

corpus into the template pool via abstraction or apply one of numerous data augmentation methods [40, 113, 17, 41, 9, 16, 18, 42].

6.1.2 Datasets

As a natural test-bed for compositional reasoning ability of neural semantic parsers, we use the following human-annotated text-to-SQL datasets[1] across four domains:

- **Advising**: User questions about US academic course information [8]
- **ATIS**: User questions for flight-booking task [114, 115]
- **GeoQuery**: User questions about US geography [116]
- **Scholar**: User questions about academic publications [117]

These datasets were initially arranged by [8] for testing compositional generalization. Unlike previously released datasets, the datasets provided challenging new splits, i.e., query-based splits, whose test sets consist of out-of-distribution (OoD) queries. In the sequel, [1] further re-arranged each of these datasets to have a balanced ratio of training/validation/test sets' sizes for a better comparison between an iid split and an OoD split (a.k.a. *Program split*). We test our framework on these newly balanced datasets.

Template Abstraction Figure 6.2 shows a manually defined abstraction we used for the datasets. This abstraction was suggested by [111] to define lossy intermediate representations. The abstraction substitutes the instance-specific entities, such as table names, alias-clauses, and clauses used in table-joining, with placeholders like `table` or `alias`.

[1] https://github.com/inbaroren/improving-compgen-in-semparse

Table 6.1: EM of test templates generated by T5 or top-ranked by DPR, Decomposable Attention (DA), transformer encoder (TE), and BERT.

		Advising	ATIS	GeoQuery	Scholar	Average
iid	T5	**94.6**	73.6	77.9	77.1	80.8
	DPR	90.1	**83.1**	**84.2**	86.7	86.0
	DA	86.8	68.4	76.8	85.7	79.4
	TE	54.3	33.1	18.9	42.9	37.3
	BERT	90.1	75.6	83.2	**89.5**	84.6
Prog.	T5	12.1	47.2	52.7	43.9	39.0
	DPR	44.2	49.1	72.5	55.1	55.2
	DA	5.1	33.0	31.9	55.1	31.3
	TE	1.2	12.6	2.2	14.3	7.6
	BERT	**56.0**	**69.2**	**81.3**	**71.4**	69.5

Data Statistics and Program Splits Data statistics can be found in Table **??**. Note that the program splits demand a parser to achieve compositional generalization. To see this, let us compare the numbers of templates in the training sets and pools. On a program split, the sum of numbers of templates in train/dev/test sets is nearly the size of the template pool that is a union of these sets (See Section 6.1.1). From this, we can see that those sets rarely share common templates, thus a parser needs to create test programs whose templates were almost never observed during training. In contrast, on an iid split, the size of the pool is only slightly bigger than the number of training templates, implying that there is no need for compositional reasoning.

6.1.3 Retrieval and Template-based Generation

Retrieval Retrieval is an essential technique that enables one to combine the parametric reasoning ability of neural networks and non-parametric relevant contexts of a

large corpus. Hence, text encoding research for retrieval has been carried out. There have been traditional keyword-based sparse encodings, such as TF-IDF or BM25 [118]. Recently, dense encodings, e.g., DPR [109] or REALM [119], that embed passages into a high-dimensional space by semantic distances have gained popularity. For example, they have been applied for a wide range of NLP tasks including open-domain QA [120, 121], knowledge-grounded dialogue [122], neural machine translation (NMT) [123]. In particular, [124] suggested a semantic parsing framework that retrieves multiple relevant utterance-program examples and uses them as additional inputs. In this work, we propose a neural semantic parsing framework for compositional generalization by selecting only one corresponding template among retrieved candidates.

Template-based Semantic Parsing Templates are rough sketches of target program structures, hence semantic parsing becomes easy when templates are provided. Therefore, extensive semantic parsing studies, such as coarse-to-fine decoding [112], hierarchical poset decoding [125], lossy intermediate representation [111], and abstract programs [126], have exploited templates to bridge a large structural gap between the input utterances and the target programs. Despite our framework that uses template-augmented inputs shares similarities with those studies, ours is distinct in that templates are obtained via retrieving and re-ranking, rather than generation.

6.2 Methods

In this section, we explain our retrieving and re-ranking framework step-by-step along with used models.

As shown in Figure 6.1, from the template pool, we want to choose the ground-truth template, e.g., `select table.lowest_point ···`. The chosen template and the input utterance are then fed to the generation model, e.g., T5 [36], for better predicting the target program. To select the desired template, a bi-encoder, e.g., DPR [109], retrieves candidates and a cross-encoder, e.g., BERT re-ranker [110], re-ranks

them.

Formally speaking, to predict the target program y_x from the input utterance x, we proceed with the following three steps: (1) using a bi-encoder, we retrieve relevant templates of x from the template pool P_{template}, yielding a set of candidate templates $P_x = \{t_1, \ldots, t_N\} \subset P_{\text{template}}$ (ideally, the ground-truth template $t(y_x)$ belongs to P_x); (2) using a cross-encoder, we re-rank candidate templates P_x and select the top-ranked template $t_x \in P_x$ (ideally, $t_x = t(y_x)$); (3) using a generation model, we predict the target program y from the template-augmented utterance input, i.e., x `[sep]` t_x, where `[sep]` is a separation token.

Advantages Our framework can secure the following two advantages. First, our usage of template-pool relieves the burden of utterance-program paired data augmentations. Essentially, data augmentation methods require humans, neural networks, or grammar induction algorithms to explore combinatorially large sample spaces derived from the programs' grammars. Hence, exploring smaller sample spaces derived from templates' simpler grammars is memory and computation effective to them. Second, we generate new programs from the template-augmented utterance, thus resulting in performance gains. Providing the target's high-level structure along with the input utterance has been proven effective for semantic parsers [112, 111]. Still, obtaining relevant templates can be an issue, which we circumvent by performing a retrieval and a re-ranking task. Template generation is possible as in [111], but our approach shows empirically superior performances (See Section 6.4.2).

6.2.1 Bi-encoder for Retrieval

Given utterance x, we want to retrieve a set P_x of N candidate templates $t_1, \ldots, t_N$ from the template pool P_{template} such that P_x contains the gold template $t(y_x)$ and the size of P_x is much smaller than P_{template}, i.e., $|P_x| \ll |P_{\text{template}}|$.

Model We use DPR [109] which follows a BERT-based bi-encoder architecture:

$$E_x = \text{BERT}_x(x), \quad E_t = \text{BERT}_t(t)$$

where E_x (E_t) is a contextualized high-dimensional embedding of the utterance x (template t) produced by a BERT encoder [95]. As the similarity between two embeddings E_x and E_t is measured by their inner product, solving a Maximum Inner Product Search (MIPS) problem enables us to retrieve a set of N (N=50) candidate templates P_x.

Our choice of neural dense encodings is apt. Indeed, traditional sparse encodings like BM25 or TF-IDF are not applicable to our case as utterances and templates are unlikely to share common words.

Training We fine-tune our BERT-initialized bi-encoder for adapting to templates that are not natural language sentences. We use the negative log likelihood of the positive template:

$$L(x, t^+, t_1^-, \ldots, t_M^-) = \\ -\log \frac{\exp(E_x \cdot E_{t^+})}{\exp(E_x \cdot E_{t^+}) + \sum_{m=1}^{M} \exp(E_x \cdot E_{t_m^-})}$$

where the positive template t^+ is the ground-truth template $t(y_x)$ and M (M=5) negative templates $t_1^-, \ldots, t_M^-$ are randomly selected from the template pool P_{template}. By doing so, the inner products between embeddings of corresponding pairs of utterances and templates get bigger, whereas those of irrelevant pairs get smaller.

6.2.2 Cross-encoder for Re-ranking

We further re-rank candidate templates of P_x in order to accurately select the most relevant template t_x ($=t(y_x)$, ideally) from P_x. As the bi-encoder does not simultaneously attend utterances and templates, we can expect improvements with a cross-encoder that performs full self-attention over utterance-template pairs.

Model We use a cross-encoder E_{cross}, such as a Decomposable Attention [127], a transformer encoder [43], a BERT re-ranker [110] followed by a final linear classifier and a softmax output layer. Cross encoders yield the similarity score $s_{x,t} \in [0, 1]$ between the utterance x and the candidate template t.

Training We train a cross-encoder with the standard cross-entropy loss:

$$L(x, t) = -I_{t=t(y_x)} \log s_{x,t} - (1 - I_{t=t(y_x)}) \log s_{x,t}$$

where $I_{a=b} = 1$ if $a = b$ else 0.

As for the negative pairs to the utterance x, we use almost correct but not exactly matched templates within P_x, i.e., (x, t) where $t \in P_x$ and $t \neq t(y_x)$. This causes the label imbalance situation; negative pairs are nearly N-times many. We simply resolve this issue by duplicating the positive pairs as many as negative pairs.

In the inference stage, for each test utterance x, we apply the cross-encoder and obtain similarity scores for all candidate pairs (x, t) where $t \in P_x$. Then, we select the template t_x that attains the top score.

6.2.3 Template-augmented Generation

Finally, we predict the target program from a template-augmented utterance.

Model We fine-tune a T5 [36] to learn the mapping x `[SEP]` $t(y_x) \mapsto y_x$ for $(x, y_x) \in D_{\text{train}}$. Applying RNN sequence-to-sequence (seq2seq) models is possible, but our preliminary experiments showed that seq2seq models mostly under-performed the T5. Thus, we opt out of seq2seq models for the generation.

Training The objective for the T5 fine-tuning is a conditional log-likelihood as follows:

$$L(\tilde{x}, y_x) = -\sum_k \log p(y_x^{(k)} | \tilde{x}, y_x^{(1)}, \ldots, y_x^{(k-1)})$$

where $\tilde{x} = x\,\texttt{[SEP]}\,t(y_x)$ is an augmented input, $y_x^{(k)}$ is the k-th token of y_x.

In the inference stage, we apply the T5 model to test utterances x with the template t_x selected by the aforementioned retrieving and re-ranking.

Our template-augmented generation is the same as the second step of lossy intermediate representation (IR) with the direct prediction by [111]. Unlike previous retrieval-based approaches that predict answers with all retrieved passages [128, 121, 120, 123, 124], we only use the most relevant passage (in our case, template) for the generation. We do so since (1) there is a unique ground-truth template for the given utterance and (2) previous studies [10, 28, 8] already have shown that neural networks are weak at re-combining basic components in a novel way. Also, note that no gradient from the T5 model flows to the retrieving or re-ranking models; Each training is performed independently.

6.3 Experimental Setup

In this section, we explain our experimental setups.

Our implementations[2] based on the open source library *transformers*[3]. Every experimental run fit into a single NVIDIA RTX 2080ti GPU.

6.3.1 Bi-encoders

As for the bi-encoder, we used memory-efficient GC-DPR [129] with the default configuration. We trained the bi-encoder for 40 epochs for each dataset and chose the optimal epoch for the retrieval based on the validation performances.

6.3.2 Cross-encoders

As for the cross-encoders, we implemented and trained three different models. First, we implemented a decomposable attention model [127] whose perceptrons have two

[2] TBD

[3] https://github.com/huggingface/transformers

layers of the hidden size 200. We trained the model with the dropout rate of 0.1 and the batch size of 64 using the Adagrad [130] optimizer with the learning rate of 0.025 for 32K steps. Second, we implemented a transformer encoder [43] which has two layers of the hidden size 128, the intermediate size 2048, and 16 attention heads. The model was trained by the Adam optimizer [86] with the learning rate $2e^{-5}$ for 20K steps. The batch size was 32, and the accumulation step was 16. Third, we used a BERT re-ranker [110]. Initialized by the configuration *bert-base-uncased*, the re-ranker was fine-tuned by the Adam optimizer with the learning rate $2e^{-5}$ for 16K steps. The batch size was set to 8. All models were saved at every quarter of the entire training step. For inference, we chose the best among them according to validation performances.

6.3.3 Generation Models

As for the program generation model, we chose T5 [36]. Initialized with the configuration *t5-base*, the T5 was fine-tuned by the Adam optimizer with the learning rate $1e^{-4}$ for 3,200 steps. The batch size was 1, and the accumulation step was 64. We ran five runs for each model, where each run took 2-6 hours depending on datasets. For evaluation, the greedy decoding scheme was used.

6.4 Results and Discussion

We evaluate retrieving, re-ranking, and generation models based on exact match (EM), that is, whether the predicted template (or program) and the target one are identical.

6.4.1 Retrieving

Our fine-tuned DPR succeeds in retrieving the corresponding templates to the input utterances within 50 candidates (Top-50) on both iid and program splits, as shown in Table 6.3. Especially, even on the ATIS dataset whose template pool is the largest (Refer to Table **??**), the Top-50 score is still solid. Not to mention that the DPR was

Table 6.2: Examples of top-3 retrieved templates (t_1, t_2, t_3) on program splits of test sets. The template t_i coincides with the ground-truth one is bold-faced.

Dataset: ADVISING	
x	number0 is worth how many credits ?
t_1	`select distinct table.total_gpa from alias where table.student_id = 1 ;`
t_2	`select distinct table.total_credit from alias where table.student_id = 1 ;`
t_3	`select count( distinct table.offering_id ) from alias where table.department = "department0" and table.number = number0 ;`
Dataset: ATIS	
x	what are all of the flights into and out of city_name0 's airport
t_1	`select distinct table.flight_id from alias where table.city_name = "city_name0" ;`
t_2	`select distinct table.city_code from alias where table.city_name = "city_name0" ;`
t_3	`select distinct table.flight_id from alias where table.city_name = "city_name0" and table.round_trip_cost is not null ;`
Dataset: GEOQUERY	
x	what is the average population of the us by state
t_1	`select sum( table.population ) from alias ;`
$\boldsymbol{t_2}$	`select avg ( table.population ) from alias ;`
t_3	`select table.population from alias where table.density = ( select max( table.density ) from alias ) ;`
Dataset: SCHOLAR	
x	conferences in year0
$\boldsymbol{t_1}$	`select distinct table.venueid from alias where table.year = year0 ;`
t_2	`select distinct table.paperid from alias where table.year = year0 ;`
t_3	`select distinct count( table.paperid ) from alias where table.journalid = "venuename0" and table.year = year0 ;`

successful for the hundreds of thousands size pool of text passages, scaling up our framework to the large template pool seems promising.

Table 6.2 shows some retrieved examples. Interestingly, we can observe that most of the Top-2 templates only differ in one word. For example, t_1 and t_2 in Advising examples (first row) are the same except for table column names: `table.total_gpa` and `table.total_credit`. These similarities indicate that our bi-encoder successfully captures semantic distances of templates in the dense space.

Unfortunately, it turns out that using the DPR is not good enough for selecting the correct templates. Even though the DPR gives the first rank to the ground-truth templates with high precision on the iid splits, the DPR misses almost half on the

Table 6.3: Top-1 & Top-50 retrieval accuracy on test sets, measured as whether the ground-truth template is in the top 1/50 retrieved templates.

		Advising	ATIS	GeoQuery	Scholar	Average
iid	Top-1	90.1	83.1	84.2	86.7	86.0
	Top-50	100.0	100.0	100.0	100.0	100.0
Prog.	Top-1	44.2	49.1	72.5	55.1	55.2
	Top-50	99.5	99.5	100.0	100.0	99.8

program splits. This degradation calls for a more careful template re-scoring, which we resolve with cross encoders.

6.4.2 Re-ranking

The BERT re-ranker is successful at raising gold templates' ranks to the top, as shown in Table 6.4. Unfortunately, neither the transformer encoder (TE) nor the decomposable attention model (DA) predicts better similarity scores than the DPR, possibly due to their insufficient expressivity. Note that the model sizes of TE (5M) and DA (0.5M) are far smaller than that of the BERT (110M).

By comparing the T5 and BERT performances, we can see that our framework is more robust for predicting compositionally novel templates. We can explain this performance gains from our task reformulation: solving two textual entailment classification tasks, i.e., scoring, instead of a hard generation task.

6.4.3 Program Prediction

Our framework leads the T5 models to achieve far better compositional generalization on the program splits, as shown in Table 6.5. Especially, on the program splits, our framework outperforms all other baselines and raises 12.6 accuracy points more on average than the T5 baselines (second row). Here, the T5 predicts programs directly

Table 6.4: EM of test templates generated by T5 or top-ranked by DPR, Decomposable Attention (DA), transformer encoder (TE), and BERT.

		Advising	ATIS	GeoQuery	Scholar	Average
iid	T5	**94.6**	73.6	77.9	77.1	80.8
	DPR	90.1	**83.1**	**84.2**	86.7	86.0
	DA	86.8	68.4	76.8	85.7	79.4
	TE	54.3	33.1	18.9	42.9	37.3
	BERT	90.1	75.6	83.2	**89.5**	84.6
Prog.	T5	12.1	47.2	52.7	43.9	39.0
	DPR	44.2	49.1	72.5	55.1	55.2
	DA	5.1	33.0	31.9	55.1	31.3
	TE	1.2	12.6	2.2	14.3	7.6
	BERT	**56.0**	**69.2**	**81.3**	**71.4**	69.5

from utterances. Not to mention that our framework does not degrade performances on the iid splits.

It would be instructive to see the correlation between the predicted template and program accuracy, as shown in Table 6.6. As expected, the more accurate templates we feed, the more accurate programs the T5 models predict. Therefore, we conclude that the well-selected templates by our framework improve accuracy in the program prediction step.

6.4.4 Discussion

Different Aspects of Compositional Generalization The novel templates come from the template pool rather than our generation models, and the generation is more likely used to plug in the entities observed during training to the novel templates. This leaves us with a somewhat unclear picture – can we truly say that the T5 models are

able to compositionally generalize with new templates? While there isn't yet a consensus about what compositional generalization is and how it could be evaluated, [11] suggested five distinct aspects of compositionality, where systematicity and productivity are the features that compositional generalization is mainly defined by [131, 2, 28]. While our result may not clearly fit into systematicity or productivity, it shows that compositionality can also be understood in terms of *substitutivity*, which is substituting a part of an expression with a synonym.

Possibility and Limitation Feeding ground-truth templates with utterances enables us to estimate our framework's upper bound accuracy, i.e., when we have an ideal ranker (third row in Table 6.5). On the program splits, generation with gold templates raises the accuracy up to $\times 1.5 \sim \times 5$ compared to the generation without templates, but the improved accuracy is still far from perfect.

Even so, the template-based approach is a stepping stone for developing a parser that can compositionally generalize. We think that a neural parser could learn how to instantiate a novel template by observing examples of filling in placeholders *locally* during training. However, a neural parser would struggle to learn how to re-combine structural constituents as it requires *global* reasoning. Therefore, future studies should focus on somewhat easy compositional template-instantiation then extend to the more challenging generalization.

Potentials for Using Cross-Domain Data One interesting point about templates is that one can define a program-to-template abstraction agnostic to domains. Note that templates of tested four datasets share strong structural similarities, as shown in Table 6.2. If we define more high-level abstraction for templates by anonymizing the table's column names such as `total_gpa` or `flight_id`, it would be feasible to further leverage training or augmented data from other domains. This opens a new possibility to apply our framework on achieving compositional generalization with cross-domain data [37, 38].

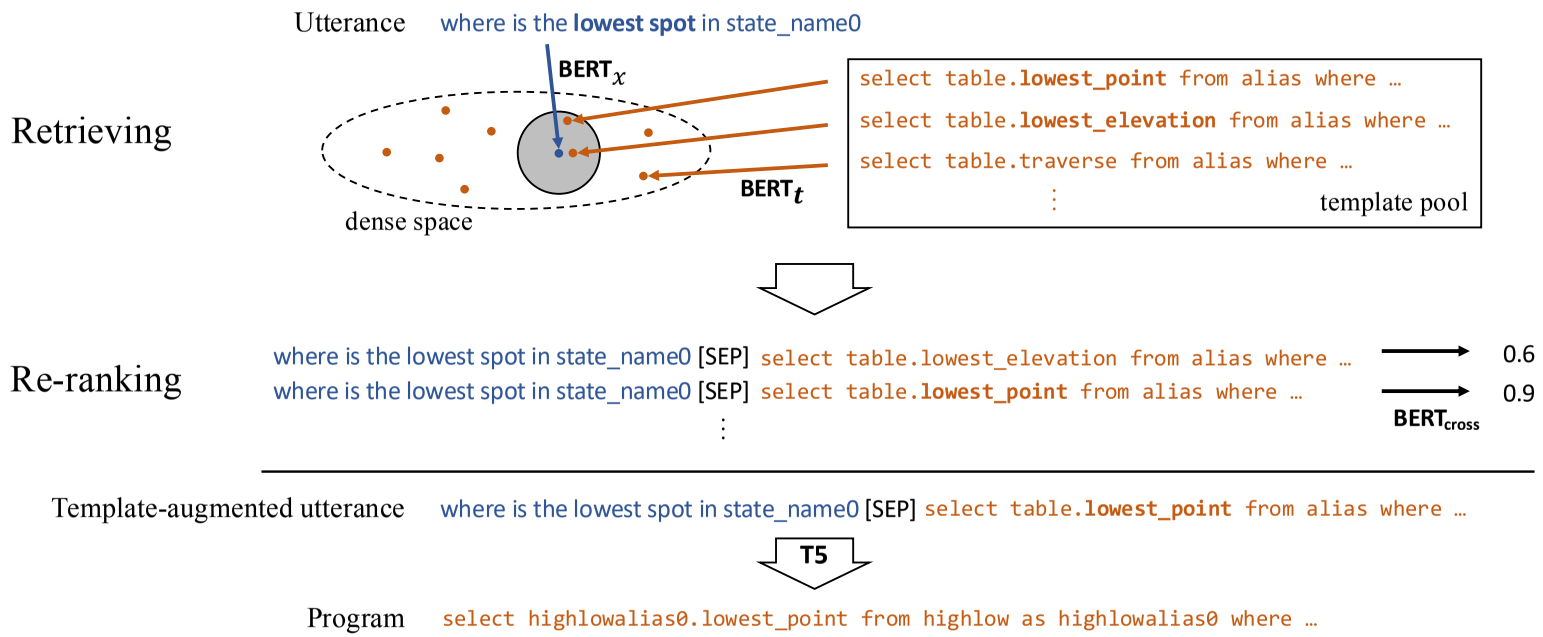

Figure 6.1: Our framework to achieve compositional generalization of a neural semantic parser. Given the input utterance, we first retrieve the nearest N templates on the dense space where the utterance and templates are embedded via a bi-encoder (BERT$_x$, BERT$_t$). Then, we re-rank retrieved N templates based on similarity scores by a cross-encoder BERT$_{\text{cross}}$. Finally, the utterance and the template of the highest re-ranking score are fed to T5 for generating the target program.

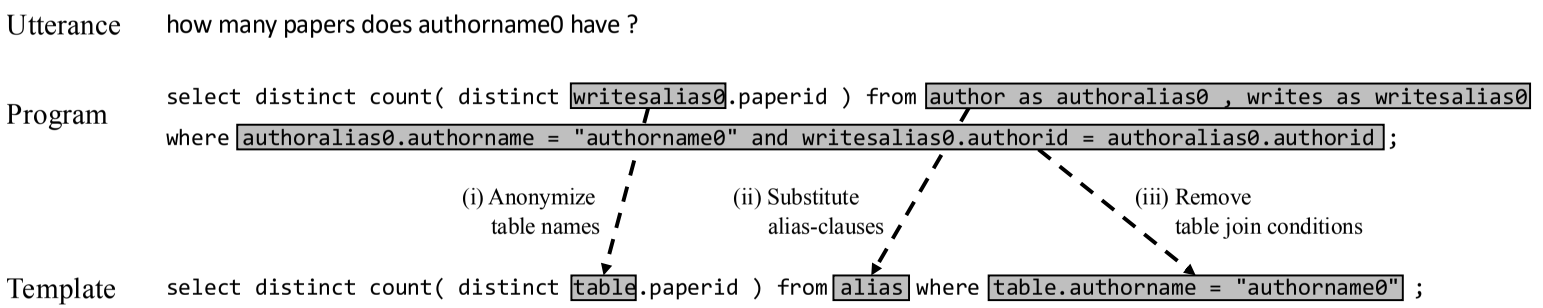

Figure 6.2: An example of an utterance, its program, and its template. The program is transformed to the template via a number of manually defined abstractions, such as (i), (ii), and (iii).

Table 6.5: EM of test programs for all models and all datasets, and in comparison with previous work: ♢ [1]. The input of the first two rows is an utterance, while that of the remaining rows is a template-augmented utterance.

	Advising		ATIS		GeoQuery		Scholar		Average	
	iid split	Prog. split	iid split	Prog. split	iid split	Prog. split	iid split	Prog. split	iid split	Prog. split
Seq2seq ♢	**92.4**	5.9	74.5	34.4	76.6	34.7	73.0	28.2	79.1	25.8
T5	91.7	7.3	**69.7**	34.3	**77.3**	47.9	70.5	30.4	77.3	30.0
+Template										
Gold	99.1	38.6	85.9	55.4	91.4	89.5	92.8	60.0	92.3	60.9
T5	93.9	5.1	68.9	31.1	77.7	50.3	72.8	32.9	78.3	29.8
DPR	89.6	8.1	72.9	23.8	76.6	66.4	81.9	36.5	**80.3**	33.7
DPR+DA	85.9	3.4	62.0	18.8	74.7	31.9	82.3	36.7	76.2	22.7
DPR+TE	53.9	0.0	31.3	7.3	18.9	1.1	41.9	10.0	36.5	4.6
DPR+BERT	89.8	**14.1**	66.4	**37.4**	76.6	**74.9**	**84.6**	**43.9**	79.3	**42.6**

Table 6.6: The Pearson correlation coefficient between the template accuracy and the program accuracy.

	Advising	ATIS	GeoQuery	Scholar	Average
iid	1.00	1.00	0.99	1.00	1.00
program	0.95	0.99	1.00	0.99	0.98

Chapter 7

Conclusion

7.1 Summary

In this dissertation, we explore deep learning models' possibilities for achieving rule-based out-of-distribution (OoD) generalization, i.e., *systematic generalization*, at sequence generation tasks. Whereas humans can handle OoD test examples that follow a distribution distinct to training, even state-of-the-art deep learning models struggle to do so. This can be problematic for applications requiring logical reasoning or huge data collection cost. Thus, we attempt to answer the following questions: (1) in what extent deep model can systematically generalize? (2) how to improve the systematic generalization abilities of deep models?

To answer the former question, in chapter 2, we define a sequence generation task requiring systematic generalization and summarize existing work about datasets and evaluation methods. In chapter 3, we assess the systematic generalization abilities of deep learning models with our proposed number sequence prediction problems. In particular, our evaluation methods for measuring deep models' computational powers give further insights by providing equally powerful automata.

To answer the latter question, we propose three different methods. In chapter 4, we propose a new model, called neural sequence-to-grid module. This module is ad-

vantageous to capture the latent alignments inside the input sequence by automatically segmenting the sequence into a grid, so that appending the module before CNN led the systematic generalization for arithmetic and algorithmic tasks. In chapter 5, we propose a new data augmentation method, called parsing tree annotation. Inserting delimiter tokens in between input sequences can hint the parse structure of the input, resulting in the improved systematic generalization of deep models of standard architecture. In chapter 6, we propose to reformulate a semantic parsing task into classification and generation tasks. By doing so, we can leverage a compostionally rich template pool, which in turn eases deep learning generation models' burden to model new target program structures for novel input utterances.

7.2 Suggestions for Future Research

In broader strokes, we explore three different approaches for improving the systematic generalization of deep models: model-based, data-based, and task-based. Each approach has its unique pros and cons; therefore we will suggest future work for these approaches, respectively.

Specialized Models for Systematic Generalization The pros of devising new specialized models include guaranteeing nearly perfect accuracy for target tasks [15, 14], giving insights to which inductive bias is critical [92], and motivating other new models [58]. However, the downsides of this approach are evident: not only designing new models is burdensome for researchers, but also the tasks these new models cover are relatively limited. Therefore, future research must be focused on capturing more general inductive bias that is not solely limited to target tasks or devising auxiliary modules that can incorporate with standard architecture models.

Data Augmentation for Systematic Generalization The most significant upside of data augmentation methods is that there is no need to deviate from the standard model

architecture. However, it still leaves the following important question: how to secure systematically diverse data? Borrowing from relevant datasets [19, 38] is cheap, but we cannot expect dramatic improvements due to the misalignment between the original data and the relevant data. On the other hand, applying grammar induction algorithms to data enables to synthesize new data or even directly output underlying rules [41, 9]. Still, this method has some downsides, such as data the algorithm can be applied to is limited or the algorithm can yield incorrectly induced grammar or synthesized examples. Therefore, if future research could address these minor downsides, it would be the ultimate solution for achieving systematic generalization of deep learning models.

Task Reformulation for Systematic Generalization Another promising research direction is redefining a given task in another form with favorable conditions. In fact, in many cases, tasks are not solely given by training input-target pairs. We may have incomplete data, such as unpaired corpora, or prior knowledge, such as domain-specific languages or expert rules. This calls for a new formulation of given tasks so that the newly formed task can leverage those extra resources most effectively [20, 45]. Since this approach is limited to neither one specific data, one model, nor one task definition, the approach opens big windows for many creative solutions.

Bibliography

[1] I. Oren *et al.*, "Improving compositional generalization in semantic parsing," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, (Online), pp. 2482–2495, 2020.

[2] J. Fodor *et al.*, "Connectionism and cognitive architecture: A critical analysis," *Cognition*, vol. 28, pp. 3–71, 1988.

[3] G. F. Marcus, *The algebraic mind: Integrating connectionism and cognitive science*. MIT press, 2003.

[4] R. C. A. Iacob *et al.*, "Neural approaches for natural language interfaces to databases: A survey," in *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 381–395, 2020.

[5] Y. Li *et al.*, "On compositional generalization of neural machine translation," in *ACL*, (Online), pp. 4767–4780, 2021.

[6] W. Zaremba and I. Sutskever, "Learning to execute," *arXiv preprint arXiv:1410.4615*, 2014.

[7] D. Keysers, N. Schärli, N. Scales, H. Buisman, D. Furrer, S. Kashubin, N. Momchev, D. Sinopalnikov, L. Stafiniak, T. Tihon, *et al.*, "Measuring compositional generalization: A comprehensive method on realistic data," *arXiv preprint arXiv:1912.09713*, 2019.

[8] C. Finegan-Dollak *et al.*, "Improving text-to-SQL evaluation methodology," in *ACL*, pp. 351–360, 2018.

[9] L. Qiu *et al.*, "Improving compositional generalization with latent structure and data augmentation," *arXiv preprint arXiv:2112.07610*, 2021.

[10] B. Lake *et al.*, "Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks," in *International Conference on Machine Learning*, pp. 2873–2882, 2018.

[11] D. Hupkes, V. Dankers, M. Mul, and E. Bruni, "Compositionality decomposed: how do neural networks generalise?," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pp. 5065–5069, 2021.

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

[13] N. Kim *et al.*, "COGS: A compositional generalization challenge based on semantic interpretation," in *EMNLP*, (Online), pp. 9087–9105, 2020.

[14] X. Chen, C. Liang, A. W. Yu, D. Song, and D. Zhou, "Compositional generalization via neural-symbolic stack machines," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[15] Q. Liu, S. An, J.-G. Lou, B. Chen, Z. Lin, Y. Gao, B. Zhou, N. Zheng, and D. Zhang, "Compositional generalization by learning analytical expressions," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[16] E. Akyürek *et al.*, "Learning to recombine and resample data for compositional generalization," in *International Conference on Learning Representations, ICLR*, 2021.

[17] J. Andreas, "Good-enough compositional data augmentation," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, (Online), pp. 7556–7566, Association for Computational Linguistics, July 2020.

[18] I. Oren *et al.*, "Finding needles in a haystack: Sampling structurally-diverse training sets from synthetic data for compositional generalization," in *ACL*, pp. 10793–10809, 2021.

[19] D. Furrer, M. van Zee, N. Scales, and N. Schärli, "Compositional generalization in semantic parsing: Pre-training vs. specialized architectures," *arXiv preprint arXiv:2007.08970*, 2020.

[20] Y. Guo, H. Zhu, Z. Lin, B. Chen, J.-G. Lou, and D. Zhang, "Revisiting iterative back-translation from the perspective of compositional generalization," *arXiv preprint arXiv:2012.04276*, 2020.

[21] J. Gordon, D. Lopez-Paz, M. Baroni, and D. Bouchacourt, "Permutation equivariant models for compositional generalization in language," in *International Conference on Learning Representations*, 2020.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[24] Ł. Kaiser and I. Sutskever, "Neural gpus learn algorithms," *arXiv preprint arXiv:1511.08228*, 2015.

[25] A. Joulin and T. Mikolov, "Inferring algorithmic patterns with stack-augmented recurrent nets," in *Advances in neural information processing systems*, pp. 190–198, 2015.

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

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

[28] D. Keysers *et al.*, "Measuring compositional generalization: A comprehensive method on realistic data," in *ICLR*, 2020.

[29] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, 2016.

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

[31] S. Sukhbaatar, J. Weston, R. Fergus, *et al.*, "End-to-end memory networks," in *Advances in neural information processing systems*, pp. 2440–2448, 2015.

[32] S. Reed and N. De Freitas, "Neural programmer-interpreters," *arXiv preprint arXiv:1511.06279*, 2015.

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

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

[35] X. Chen, C. Liang, A. W. Yu, D. Zhou, D. Song, and Q. V. Le, "Neural symbolic reader: Scalable integration of distributed and symbolic representations

for reading comprehension," in *International Conference on Learning Representations*, 2019.

[36] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[37] A. Suhr *et al.*, "Exploring unexplored generalization challenges for cross-database semantic parsing," in *ACL*, pp. 8372–8388, 2020.

[38] D. Tsarkov *et al.*, "*-cfq: Analyzing the scalability of machine learning on a compositional task," in *Thirty-Fifth AAAI Conference on Artificial Intelligence*, pp. 9949–9957, 2021.

[39] P. P. Kagitha, "Systematic generalization emerges in seq2seq models with variability in data," 2020.

[40] R. Jia *et al.*, "Data recombination for neural semantic parsing," in *ACL*, pp. 12–22, 2016.

[41] P. Shaw *et al.*, "Compositional generalization and natural language variation: Can a semantic parsing approach handle both?," in *ACL*, pp. 922–938, 2021.

[42] B. Wang *et al.*, "Learning to synthesize data for semantic parsing," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2760–2766, 2021.

[43] J. Kim *et al.*, "Improving compositional generalization in classification tasks via structure annotations," in *ACL*, (Online), pp. 637–645, 2021.

[44] Y. Jiang *et al.*, "Inducing transformer's compositional generalization ability via auxiliary sequence prediction tasks," in *Empirical Methods in Natural Language Processing, EMNLP*, pp. 6253–6265, 2021.

[45] M. Nye *et al.*, "Learning compositional rules via neural program synthesis," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 10832–10842, 2020.

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[47] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[49] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, Ieee, 2009.

[50] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033, IEEE, 2012.

[51] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, *et al.*, "Deepmind lab," *arXiv preprint arXiv:1612.03801*, 2016.

[52] H. T. Siegelmann and E. D. Sontag, “On the computational power of neural nets,” *Journal of computer and system sciences*, vol. 50, no. 1, pp. 132–150, 1995.

[53] F. A. Gers and E. Schmidhuber, “Lstm recurrent networks learn simple context-free and context-sensitive languages,” *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1333–1340, 2001.

[54] S. E. Reed and N. de Freitas, “Neural programmer-interpreters,” *CoRR*, vol. abs/1511.06279, 2015.

[55] A. Joulin and T. Mikolov, “Inferring algorithmic patterns with stack-augmented recurrent nets,” in *Advances in neural information processing systems*, pp. 190–198, 2015.

[56] Ł. Kaiser and I. Sutskever, “Neural gpus learn algorithms,” *arXiv preprint arXiv:1511.08228*, 2015.

[57] N. Kalchbrenner, I. Danihelka, and A. Graves, “Grid long short-term memory,” *arXiv preprint arXiv:1507.01526*, 2015.

[58] A. Graves, G. Wayne, and I. Danihelka, “Neural turing machines,” *CoRR*, vol. abs/1410.5401, 2014.

[59] B. A. Bracken and R. S. McCallum, *Universal nonverbal intelligence test*. Riverside Publishing Company, 1998.

[60] G. J. Chaitin, “Algorithmic information theory,” *IBM journal of research and development*, vol. 21, no. 4, pp. 350–359, 1977.

[61] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.

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

[63] W. Zaremba, T. Mikolov, A. Joulin, and R. Fergus, "Learning simple algorithms from examples," in *International Conference on Machine Learning*, pp. 421–429, 2016.

[64] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *CoRR*, vol. abs/1710.09829, 2017.

[65] R. H. Katz, *Contemporary Logic Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 2000.

[66] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016.

[67] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, pp. 448–456, 2015.

[68] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[69] A. Karpathy, "Multi-layer recurrent neural networks (lstm, gru, rnn) for character-level language models in torch." https://github.com/karpathy/char-rnn, 2015.

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

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

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

[73] A. Wangperawong, "Attending to mathematical language with transformers," *arXiv preprint arXiv:1812.02825*, 2018.

[74] J. Li, L. Wang, J. Zhang, Y. Wang, B. T. Dai, and D. Zhang, "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*, pp. 6162–6167, 2019.

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

[76] G. Lample and F. Charton, "Deep learning for symbolic mathematics," *arXiv preprint arXiv:1912.01412*, 2019.

[77] G. Marcus, "The algebraic mind," 2001.

[78] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

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

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

[81] H. Nam, S. Kim, and K. Jung, "Number sequence prediction problems for evaluating computational powers of neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4626–4633, 2019.

[82] J. Russin, J. Jo, R. O'Reilly, and Y. Bengio, "Compositional generalization by factorizing alignment and translation," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pp. 313–327, 2020.

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

[84] T. Munkhdalai, A. Sordoni, T. Wang, and A. Trischler, "Metalearned neural memory," in *Advances in Neural Information Processing Systems*, pp. 13331–13342, 2019.

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

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

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

[88] N. Chomsky, *Syntactic structures*. Walter de Gruyter, 2002.

[89] R. Montague, "Universal grammar," *Theoria*, vol. 36, no. 3, pp. 373–398, 1970.

[90] J. Bastings, M. Baroni, J. Weston, K. Cho, and D. Kiela, "Jump to better conclusions: SCAN both left and right," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, (Brussels, Belgium), pp. 47–55, Association for Computational Linguistics, Nov. 2018.

[91] R. Dessì and M. Baroni, "CNNs found to jump around more skillfully than RNNs: Compositional generalization in seq2seq convolutional networks," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, (Florence, Italy), pp. 3919–3923, Association for Computational Linguistics, July 2019.

[92] Y. Li, L. Zhao, J. Wang, and J. Hestness, "Compositional generalization for primitive substitutions," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, (Hong Kong, China), pp. 4293–4302, Association for Computational Linguistics, Nov. 2019.

[93] J. Loula, M. Baroni, and B. Lake, "Rearranging the familiar: Testing compositional generalization in recurrent networks," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, (Brussels, Belgium), pp. 108–114, Association for Computational Linguistics, Nov. 2018.

[94] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[95] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[96] K. Gulordava, P. Bojanowski, E. Grave, T. Linzen, and M. Baroni, "Colorless green recurrent networks dream hierarchically," *arXiv preprint arXiv:1803.11138*, 2018.

[97] T. Linzen, E. Dupoux, and Y. Goldberg, "Assessing the ability of LSTMs to learn syntax-sensitive dependencies," *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 521–535, 2016.

[98] M. Hahn, "Theoretical limitations of self-attention in neural sequence models," *arXiv preprint arXiv:1906.06755*, 2019.

[99] C. Yun, S. Bhojanapalli, A. S. Rawat, S. J. Reddi, and S. Kumar, "Are transformers universal approximators of sequence-to-sequence functions?," *arXiv preprint arXiv:1912.10077*, 2019.

[100] R. K. Ando and L. Lee, "Mostly-unsupervised statistical segmentation of japanese: Applications to kanji," in *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pp. 241–248, 2000.

[101] G. Carroll and E. Charniak, *Two experiments on learning probabilistic dependency grammars from corpora*. Department of Computer Science, Univ., 1992.

[102] J. Johnson *et al.*, "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2901–2910, 2017.

[103] Y.-L. Kuo *et al.*, "Compositional networks enable systematic generalization for grounded language understanding," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 216–226, 2021.

[104] X. Chen *et al.*, "Compositional generalization via neural-symbolic stack machines," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 1690–1701, 2020.

[105] Q. Liu *et al.*, "Compositional generalization by learning analytical expressions," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 11416–11427, 2020.

[106] P. Yin *et al.*, "Compositional generalization for neural semantic parsing via span-level supervised attention," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2810–2823, 2021.

[107] H. Zheng *et al.*, "Compositional generalization via semantic tagging," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 1022–1032, 2021.

[108] P. P. Kagitha, "Systematic generalization emerges in seq2seq models with variability in data," in *Akaike Technologies*, 2020.

[109] V. Karpukhin *et al.*, "Dense passage retrieval for open-domain question answering," in *EMNLP*, (Online), pp. 6769–6781, 2020.

[110] R. Nogueira *et al.*, "Passage re-ranking with bert," *arXiv preprint arXiv:1901.04085*, 2019.

[111] J. Herzig *et al.*, "Unlocking compositional generalization in pre-trained models using intermediate representations," *arXiv preprint arXiv:2104.07478*, 2021.

[112] L. Dong *et al.*, "Coarse-to-fine decoding for neural semantic parsing," in *ACL*, pp. 731–742, 2018.

[113] D. Chiang, "Hierarchical phrase-based translation," *computational linguistics*, vol. 33, no. 2, pp. 201–228, 2007.

[114] P. Price, "Evaluation of spoken language systems: The atis domain," in *Speech and Natural Language: Proceedings of a Workshop*, 1990.

[115] D. A. Dahl *et al.*, "Expanding the scope of the atis task: The atis-3 corpus," in *Human Language Technology: Proceedings of a Workshop*, 1994.

[116] J. M. Zelle *et al.*, "Learning to parse database queries using inductive logic programming," in *Proceedings of the national conference on artificial intelligence*, pp. 1050–1055, 1996.

[117] S. Iyer *et al.*, "Learning a neural semantic parser from user feedback," *arXiv preprint arXiv:1704.08760*, 2017.

[118] S. Robertson and H. Zaragoza, *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc, 2009.

[119] K. Guu *et al.*, "Realm: Retrieval-augmented language model pre-training," *arXiv preprint arXiv:2002.08909*, 2020.

[120] G. Izacard *et al.*, "Leveraging passage retrieval with generative models for open domain question answering," in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL*, pp. 874–880, 2021.

[121] P. S. H. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems*, 2020.

[122] K. Shuster *et al.*, "Retrieval augmentation reduces hallucination in conversation," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 3784–3803, 2021.

[123] D. Cai *et al.*, "Neural machine translation with monolingual translation memory," in *ACL*, (Online), pp. 7307–7318, 2021.

[124] P. Pasupat *et al.*, "Controllable semantic parsing via retrieval augmentation," in *EMNLP*, pp. 7683–7698, 2021.

[125] Y. Guo *et al.*, "Hierarchical poset decoding for compositional generalization in language," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 6913–6924, 2020.

[126] B. Wang *et al.*, "Learning semantic parsers from denotations with latent structured alignments and abstract programs," in *EMNLP*, pp. 3774–3785, 2019.

[127] A. Parikh *et al.*, "A decomposable attention model for natural language inference," in *EMNLP*, pp. 2249–2255, 2016.

[128] P. Yin *et al.*, "Reranking for neural semantic parsing," in *ACL*, pp. 4553–4559, 2019.

[129] L. Gao *et al.*, "Scaling deep contrastive learning batch size under memory limited setup," in *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP)*, (Online), pp. 316–321, 2021.

[130] J. Duchi *et al.*, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[131] N. Chomsky, *1965: Aspects of the theory of syntax. Cambridge*. MIT Press, 1965.

초 록

규칙을 학습하고 확장할 수 있는 인간 수준의 기계를 개발하는 것은 인공지능 커뮤니티의 오랜 과제이다. 현재의 딥 러닝 모델은 광범위한 응용 분야에서 놀라운 성능을 입증했지만, 모델은 여전히 학습분포를 따르지 않는 참신한 예제에 대해 학습된 규칙을 적용하는데 어려움을 겪고 있다. 이러한 규칙 기반 분포외일반화, 즉 체계적 일반화를 하지 못하는 딥러닝 모델은 응용이 제한되는데, 특히 시멘틱 파싱과 같은 논리적 추론이 필요하거나 저자원 기계 번역과 같은 데이터 부족에 시달리는 시퀀스 처리작업에 응용될 수 없다.

따라서 본 논문은 딥 러닝 텍스트 생성 모델의 체계적 일반화 능력을 측정하고 개선하는 것을 목표로 한다. 논문의 첫 번째 부분은 현재 딥 러닝 모델의 체계적 일반화 능력을 평가하는 것이다. 특히, 우리는 숫자 시퀀스 예측 문제를 설계하고 동등하게 표현되는 오토마타를 이용해 모델의 계산 능력을 측정한다. 논문의 나머지 부분은 딥 러닝 모델의 체계적 일반화를 달성하기 위한 다양한 프레임워크를 제안한다. 첫 번째 프레임워크는 신경 시퀀스 투 그리드 모듈이라는 새로운 입력 전처리 모듈을 고안하는 것이다. 모듈은 시퀀스 입력을 그리드 입력으로 분할하고 정렬하는 방법을 배울 수 있다. 즉, 기호 규칙을 학습하고 적용하는데 더 유리한 형태이다. 우리는 그리드 입력을 취하는 딥 러닝 모델이 프로그램 코드 평가 또는 bAbI 작업을 포함하여 상징적 추론 작업에 대해 학습된 규칙을 확장할 수 있음을 실험적으로 보여주었다. 두 번째 프레임워크는 구조적으로 암시된 예시로 신경망을 훈련시키는 것이다. 우리는 타겟의 구문 분석 트리의 비말단 노드를 나타내는 구분자 토큰으로 타겟에 주석을 단다. 우리는 구성 추론이 필

요한 작업에 실험하여 상당한 성능 향상을 달성함으로써 타겟에 주석을 다는 방법의 효과성을 보여주었다. 마지막 프레임워크는 선택된 템플릿을 신경 생성 모델에 제공하는 것이다. 대상 시퀀스의 높은 수준의 스케치인 템플릿은 모델이 어려운 구조 모델링을 해야하는 부담을 완화하고 모델이 쉬운 템플릿 실현에 집중할 수 있도록 한다. 저렴하고 큰 템플릿 풀에서 신경 모델로 검색하여 순위를 다시 매겨 템플릿을 선택한다. 실험 결과는 우리가 선택한 템플릿이 네 가지 다른 의미 시멘틱파싱에서 딥 러닝 모델의 성능을 크게 향상시킨다는 것을 보여준다.

주요어: 딥러닝, 시퀀스 생성, 분포외일반화
학번: 2016-20873