Learning Transductions to Test Systematic Compositionality

Josef Valvoda[ɘ] **Naomi Saphra**[β] **Jonathan Rawski**[ɕ]
Ryan Cotterell[ð] **Adina Williams**[ɛ]
[ɘ]University of Cambridge [β]New York University
[ɕ]San Jose State University [ð]ETH Zürich [ɛ]FAIR
jv406@cam.ac.uk nsaphra@nyu.edu jon.rawski@sjsu.edu
ryan.cotterell@inf.ethz.ch adinawilliams@fb.com

Abstract

Recombining known primitive concepts into larger novel combinations is a quintessentially human cognitive capability. Whether large neural models in NLP acquire this ability while learning from data is an open question. In this paper, we look at this problem from the perspective of formal languages. We use deterministic finite-state transducers to make an unbounded number of datasets with controllable properties governing compositionality. By randomly sampling over many transducers, we explore which of their properties (number of states, alphabet size, number of transitions etc.) contribute to learnability of a compositional relation by a neural network. In general, we find that the models either learn the relations completely or not at all. The key is transition coverage, setting a soft learnability limit at 400 examples per transition.

1 Introduction

Systematic compositionality is a hallmark of human language (Hockett, 1959; Chomsky, 1957; Montague, 1970; Partee, 1995). It is arguably a requirement for any model to call itself a model of language or to achieve human-like natural language understanding. For instance, English speakers reading the Jabberwocky (Carroll, 1871) comprehend that the adjective *slithy* composes with the plural noun *toves* to form the noun phrase *slithy toves*, without having a clear understanding of what either the word *slithy* or *toves*—let alone their composition—means. This ability allows humans to describe a complex environment using a limited number of primitives (in the case of language, words or morphemes). In cognitive science, whether neural networks' can acquire such behavior has been debated for over 30 years (Fodor and Pylyshyn, 1988; Marcus, 1998).

In recent work, researchers have analyzed sequence-to-sequence model performance on small, controlled datasets in order to draw conclusions about the inherent limitations of neural models to acquire systematic compositionality (Lake and Baroni, 2018; Hewitt et al., 2020; Hupkes et al., 2020; Dankers et al., 2021). For instance, Lake and Baroni (2018) cast doubt on neural models' ability to learn systematic behavior encoded in their toy SCAN dataset, but shortly thereafter, Bastings et al. (2018) demonstrate that an out-of-the-box sequence-to-sequence model could indeed master the task.

Rather than introducing another hand-crafted dataset, our approach combines elements from two linguistic traditions. On the one hand, we follow Montague's assertion that there is no important theoretical difference between natural and artificial languages (Montague, 1970), and use artificial languages, which are more tractable, to test for systematic compositionality. On the other, we take lessons from the field of grammatical inference (de la Higuera, 2010; Rawski and Heinz, 2019), and evaluate neural sequence-to-sequence models on many, automatically generated artificial languages sampled from particular classes of functions—as is standard practice at grammatical inference competitions (Balle et al., 2017).

In this paper, we study the class of string functions encoded by subsequential finite-state transducers (SFST), a restricted class of general finite-state transducers, which encode rational relations (Mohri, 1997). We sample arbitrary SFSTs to generate many different string-to-string datasets, and evaluate the behavior of neural sequence-to-sequence models when learning them. By controlling the properties of the SFSTs we sample from, we are able to make precise statements about the learnability of systematic phenomena.

Empirically, we find that neural sequence-to-sequence models are capable of learning SFSTs from finite data. Moreover, we observe an interesting tendency for neural models to either generalize correctly or to fail, with little middle ground. Our analysis suggests that an explanation for this behavior lies in whether or not the training data has sufficient coverage, i.e., every transition in a given transducer is crossed in a minimum number of training examples (≈ 400 on average in our experiments). We then turn to an existing hand-crafted dataset, namely the SCAN language, but approach it through the lens of the SFST that encodes it. Via a comparison with SCAN, we are able to identify that repeated sub-graphs in the transducer can have a positive effect on its learnability, relaxing the transition coverage requirement and thereby enabling learnability of larger SFSTs. Our results suggest that focusing on formal language theory, which allows for a controlled setting, could help future research on evaluating systematic compositionality.

2 Learning Finite-State Transductions

We argue that determining whether neural networks exhibit systematic compositionality is fundamentally equivalent to asking whether neural networks are capable of learning rule-based behavior.

2.1 Montague's Compositionality

Montague famously inspired the use of mathematical definitions of what it means to be compositional—specifically, a function is compositional if it is a homomorphism, or has structure preserving mapping between an input and an output algebra (Andreas, 2019). In the context of string-to-string maps, a function that preserves concatenation is a homomorphism. To make this notion more formal, let Σ and Γ, be a pair of alphabets. Making use of string concatenation $\circ$, we can construct monoids $(\Sigma^*, \circ)$ and $(\Gamma^*, \circ)$ where the identity element is the empty string ε. In this context, a function $f : \Sigma^* \to \Gamma^*$ is a homomorphism if it obeys the following two conditions:

1. $f(x \circ y) = f(x) \circ f(y)$ for all x, y in Σ^*
2. $f(\varepsilon) = \varepsilon$

As we will see, deterministic transducers naturally encode a homomorphism between two alphabets. Therefore, we test compositional ability of a neural network by training it to learn string-to-string maps encoded by a transducer.

2.2 Why Learn Finte-State Transductions?

Our study focuses on learning a particular kind of transductions. Specifically, we focus on restricted classes of regular relations, which are computed by finite-state transducers. We believe this is a natural starting point, since this class of grammars is mathematically well-studied, has provable learning guarantees, and has a long use history in linguistics and NLP (Mohri, 1997).

Finite-state transducers also encompass most previous work on systematic compositionality: many datasets, e.g., SCAN (Lake and Baroni, 2018) and gSCAN (Ruis et al., 2020), describe *finite* string relations and are, therefore, finite-state by definition. These handcrafted datasets have many advantages, like easy interpretability and domain specificity, since they directly encode particular relevant relationships, such as movement over a grid or specific linguistic phenomena. However, this realism pays the price of diminished robustness of any findings over such datasets (Rogers and Pullum, 2011). By removing the ability to simply adjust properties of the underlying function class, and the transducers which compute it, one loses the possibility to experiment more robustly over a whole class of functions, rather than just one instantiation of it.

Rather than manually designing individual datasets ourselves, we generate unboundedly many new datasets via randomly sampled SFSTs. This offers a principled view of the problem of learning artificial languages possessing certain properties, by simply varying properties of the class of transducers that generate them. Furthermore, as we will see in §5, one may view existing compositionality tasks as SFSTs which enables a deeper understanding of modeling results. Both specific artificial languages, such as the compositionality datasets mentioned above, and those randomly sampled from a particular function class such as the work presented in this paper, are worth studying. However, our approach has been missing from the systematic compositionality discourse.

2.3 Subsequential Finite-State Transducers

A **subsequential** finite-state transducer is a transducer that is deterministic with respect to the input tape. By construction, subsequential transducers are functional, i.e., the string-to-string relations they encode are functions. An alphabet Σ is a finite, non-empty set of symbols; let Σ^* be all strings over Σ, including λ, the empty string. A **subsequential finite-state transducer** (SFST) for an input alphabet Σ and an input–output alphabet Γ is a tuple $\mathcal{T} = \langle Q, q_0, Q_f, \delta, o, \omega \rangle$, where Q is the set of states, $q_0 \in Q$ is the (unique) start state, $Q_f \subseteq Q$ is the set of final states, $\delta : Q \times \Sigma \to Q$ is the

10 19 16 28 19 7 27 20	$\Rightarrow$	33 31 31 30 57 56 44
11 10 18 2 5 19 7 25 4 1 4	$\Rightarrow$	48 33 38 45 40 30 57 44 32
1 18 23 2 2 22 25 3 18 11 23 1 0 1	$\Rightarrow$	53 39 45 37 46 44 36 53 48 39 38 41
1 21 20 11 19 2 5 11 11 10 13 8	$\Rightarrow$	40 43 37 30 37 32 48 48 33 43 40

Table 1: Example of our inputs on the left and outputs on the right. Using numbers instead of words ensures the relationship between the input and output sequences requires knowing the underlining SFST and can't be a function of other factors, such as word similarity.

transition function, $o : Q \times \Sigma \to \Gamma^*$ is the output function, and $\omega : Q_f \to \Gamma^*$ is the final function. We define the reflexive, transitive closure of δ^* and o^* as $\delta^* : Q \times \Sigma^* \to Q$ and $o^* : Q \times \Sigma^* \to \Gamma^*$.

The semantics of a SFST is a transduction $t(\mathcal{T})$ defined as follows; let $\hat{t} = t(\mathcal{T})$. For $w \in \Sigma^*, \hat{t}(w) = uv$ where $o(q_0, w) = u$, and $\omega(q_f) = v$ if $\delta^*(q_0, w) = q_f$ for some $q_f \in Q_f; \hat{t}(w)$ is undefined otherwise. The class of **subsequential functions** are those describable with SFSTs, among other algebraic and logical characterisations (Bhaskar et al., 2020; Oncina et al., 1993). They are a subclass of the Regular relations, and a superclass of the finite relations.

Informally, for subsequential functions, any of the infinitely many possible input strings is classified as belonging to exactly one of finitely many regular languages. For any input element x of that string, the output u only depends on x and the regular stringset to which its preceding input string belongs. The key compositionality property of subsequential functions is finite lookahead; the transducer can only remember finitely much information about the input for any output symbol.

SFSTs as Homomorphisms. Now we discuss the exact sense, in which SFSTs are homomorphisms. Given a path in a SFST from $q \cdots q''$, for any state q' on this path, we have that $o(q \cdots q'') = o(q \cdots q') \circ o(q' \cdots q'')$, i.e., we can simply take the concatenation of the yield of the output path of the two subpaths. This will give us the same string as if we were to take the yield of the full path. By definition, an SFST therefore encodes a homomorphism.

2.4 Generating Random SFSTs

We follow these steps in generating the SFSTs used to create our datasets. We first generate random directed graphs from the following stochastic process. For a given a set of states Q with $|Q| = N$ and an input alphabet Σ, we sample a matrix $\mathbf{B}^{(\sigma)} \in \mathbb{B}^{N \times N}$ for each $\sigma \in \Sigma$ where $\mathbb{B} = \{0, 1\}$. During sampling, we additionally enforce the constraint that there be *at most* one non-zero entry in every row vector $\mathbf{b}_i^{(\sigma)}$. This constraint ensures that the resulting SFST is subsequential (or input deterministic) by construction. We sample uniform at random over this space of matrices that satisfy this constraint. In terms of the semantics, if the entry $b_{ij}^{(\sigma)} = 1$, it means our generated SFST has a transition from state $q_i \xrightarrow{\sigma} q_j$ with input symbol σ. Then, for every transition $q_i \xrightarrow{\sigma} q_j$ in our generated SFST, we sample its output symbol γ from a uniform distribution over the output alphabet Γ. This results in a transition $q_i \xrightarrow{\sigma/\gamma} q_j$. Finally, to get a canonical representation for particular SFSTs, we minimize it (Choffrut, 2003). Minimization also ensures that we have created thousands of *different and unique* SFSTs—i.e., that we are not doubly sampling.

A state in a finite-state transducer is *accessible* if there exists a path from the initial state to that state in the machine. A state is *co-accessible* if there is a path from it to a final state. To create interesting SFSTs, we additionally enforce both accessibility and co-accessibility. Testing for accessibility and co-accessiblity is easily achieved with a depth-first search. In order to ensure our sampled SFSTs satisfy these two constraints, we reject those samples that violate this constraint.

3 Experimental Methods

We evaluate whether neural sequence-to-sequence models can learn our generated SFSTs, and how they compare to the traditional symbolic learners.

3.1 Data

Here, we discuss the creation of the datasets we use to investigate the learnability of the regular relations. Our input alphabet consisted of a finite set of numbers; we set its size to a constant of 10. Half the time, we also add a special empty emission token λ (in our Python implementation,

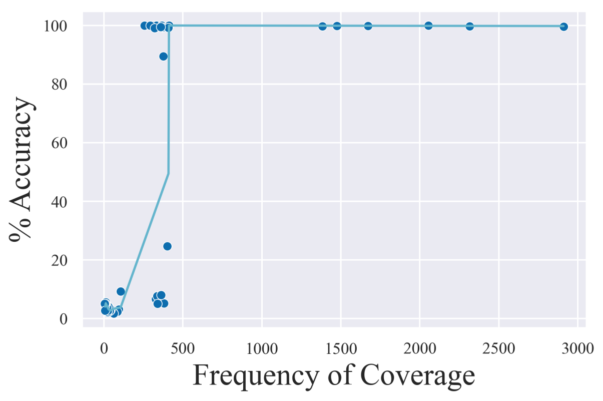

Figure 1: Average minimum frequency of a transition appearing in training data for learning a SFST is 400.

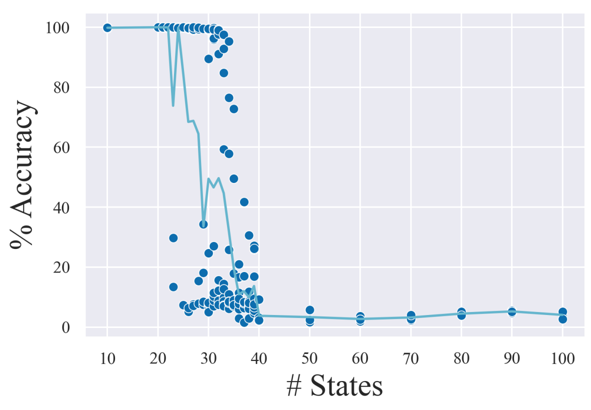

Figure 2: As the number of states in SFST increases, accuracy drops to nearly zero percent between 20 and 40 states.

-1) to the output alphabets to see if it affects the neural models learnability of the SFST.

We first generate 100 unique datasets using SFSTs where the number of states ranges from 10 to 100 in increments of 10. We further generate 100 unique datasets using SFSTs with states ranging from 21 to 39. In total, we experiment with 2000 unique datasets. However, using the sampling method, one can generate an arbitrary number of SFSTs and, concurrently, datasets, to various specifications as desired.

To generate the input–output pairs for our experiments, we randomly walk through the SFST. We define a parameter for the chance that a given state will emit a final symbol, and set it to 10%. This provides some diversity to the sequence length within computational limits of the GPU. Following Lake and Baroni (2018), the maximum number of steps is set to 50. We continue to walk the SFST until we collect $20,000$ unique input-output pairs. We further double the dataset size to $40,000$ for SFSTs with 21 to 39 states, since we observe the accuracy drop-off in this region. All datasets are randomly split 80–20 into training and test sets.

All particular states and transitions (i.e., single connections between precisely two states) are seen by the model during training. This means that coverage of both the states and transitions for all datasets is 100%. Recall though, that all sequences are unique, because the path through the states via the transitions is determined by random walk. Table 1 shows some example transductions.

3.2 Neural Sequence-to-Sequence Models

We train neural encoder-decoder models in the style of Bahdanau et al. (2015), using bi-directional LSTMs with a $\tanh$ activation function and a global attention mechanism. Attention allows the decoder to selectively "attend" to the encoder's hidden states by learning a set of weights. These weights map the decoder's current state to another set of weights for timesteps in the input. Attention then concatenates the current decoder hidden state, i.e. the weighted combination of all encoder hidden states, to yield a new current decoder state. This way, the decoder can pull information directly from the encoder by learning an alignment between the output and input representations.

We use the following hyperparameters; 200 dimensional encoder and decoder with 2 layers, maximum gradient clipping normalization of 5, dropout of 0.5 and batch size 64. Our alphabet tokens are embedded as 100 dimensional vectors. Just like Lake and Baroni (2018), we run the model for $100,000$ steps. Unlike them though, we rely on the neutral transducer code base from Wu and Cotterell (2019) which uses the Adam optimizer (Kingma and Ba, 2015) with the default learning rate of 0.001. When we report results for larger LSTM models, this is the effect of changing only the dimensions of the encoder and decoder, both to 300 dimensions, and keeping all the other hyperparameters the same as the smaller models.

3.3 OSTIA

The onward subsequential transducer inference algorithm (OSTIA; Oncina et al. 1993) inductively learns the class of subsequential functions from positive presentations of input-output strings, OSTIA works by first building a prefix-tree transducer of the training data, which is then transformed through a series of state-merging operations into the SFST encoding the function the data is drawn from. If a characteristic sample is contained in the learning data, OSTIA finds a correct transducer in polynomial (cubic) time. Since OSTIA is designed specifically with SFST in mind, we wanted to see how it compares to the neural models above and set a baseline with it. Unfortunately, OSTIA's cubic nature makes learning from large datasets like ours very time intensive. To speed up the process, we limit the number of samples we provide to OSTIA to 1000, only 10% to 5% of what the neural transducers obtain. This keeps OSTIA's run time roughly equivalent to its neural counterparts.

4 Results

Training over many datasets enables us to make the following observations about how different SFST properties affect the learnability of a compositional language.

Minimum Transition Coverage. Figure 1 reveals a threshold on the number of samples per transition required to comfortably learn the transducer: It is about 400 samples. This may seem unsurprising given neural networks' "notorious thirst" for data (Lake and Baroni, 2018). In the vast majority of cases, SFSTs that do not meet this threshold learn nothing. In the few cases where they manage to have non-zero accuracy, we observe that the earlier transitions in the SFST have attained reasonable coverage, leading us to believe that the coverage is the linchpin of learning SFSTs.

Number of States. Models' ability to learn datasets depends on the number of states in the SFST (see Figure 2). There is a sharp drop-off around 20 states, decreasing exponentially as states grow linearly, until 40 states. From there on, models only achieve low accuracy. Our findings offer an explanation of why training on shorter sequences impedes the learnability of a dataset. For example, in Lake and Baroni (2018)'s and Ruis et al. (2020)'s length based experiments, where the task is to generalize to commands with longer action sequence, we hypothesize that shorter sequences correlate with lower coverage. We find that, indeed, splitting the datasets along length drops the coverage compared to a random split. This suggests that splitting datasets by length of sequences has little to do with systematicity, but a lot to do with coverage. We discuss this in further detail in §5.

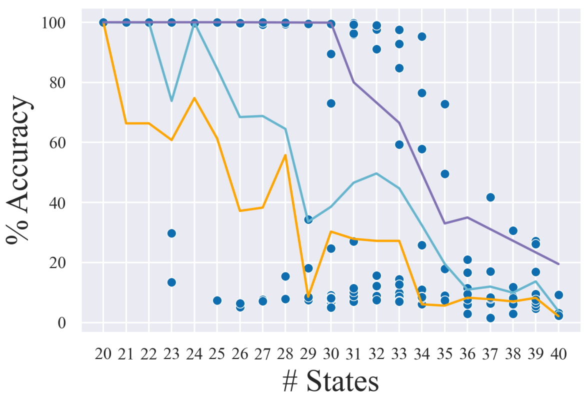

Figure 3: As SFST state size increases, average accuracy (blue line) decreases. Increasing the neural model's size improves accuracy (purple line). Empty string λ in output alphabet further reduces accuracy (yellow line). Similar behavior appears when plotting number of transitions instead of states.

Number of LSTM Hidden Units. We find coverage is only a soft limit. When we increase the memory size of the LSTM from 200 to 300 dimensions, the models can learn even those datasets that seemed previously unlearnable (see Figure 3, where the purple line is the average accuracy of 300 dimensional LSTM). Except the higher state coverage threshold, the trend these models follow is the same as for their lower dimensional counterparts.

Empty Transition. We explore another way of making a more difficult finite function: we allow the SFSTs to contain empty transitions λ and test whether languages generated by SFSTs with empty transitions are harder by measuring the average accuracy over SFSTs with and without them. We find a 12% performance drop on the test set: SFSTs without empty transitions achieve 47% average accuracy versus 35% for SFSTs with an empty string. We therefore conclude that the empty transition does make inducing a SFST empirically harder for the models (see Figure 3, where the blue line is average accuracy without empty strings, and the yellow line is average accuracy with them).

OSTIA is Slow. In terms of wallclock time, we find that a open-source implementation of OSTIA[1] does not scale to dataset sizes above 1000, for

[1] github.com/alenaks/OSTIA

which it takes roughly 5 hours on Nvidia Tesla V100 GPU to complete a run on one dataset, about the same time it takes to train our neural models. With 1000 samples, OSTIA fails to get accuracy above 1%. In contrast, the neural sequence-to-sequence models reach almost 100%. On the one hand, reducing the size of the training dataset disadvantages OSTIA. On the other hand, providing OSTIA with the full $20,000$ samples is unworkable, because it will run without converging for over 3 days on a single dataset. Given sufficient time and the function's characteristic sample, OSTIA *is* guaranteed to learn the SFSTs perfectly, but in practice, it is too slow to use.

5 Discussion: The Learnability of SCAN

The empirical learnability of SFSTs appears to hinge on the transition coverage. In this section we discuss the SCAN language and the experiments conducted on it. First, we argue the transition coverage can explain some of the results observed over SCAN. Second, we cast SCAN as a SFST to reason why it is learnable despite the parameters of the SFST that encodes it would suggest otherwise.

Lake and Baroni (2018) find that training models on a random split of the SCAN dataset dramatically outperforms models trained only on short commands, i.e. the lengthwise split. While the best random split model achieves 100% accuracy, the best model trained on the lengthwise split achieves only 21% accuracy. Furthermore, the accuracy drops from 80% to *zero* between the 24^{th} and 26^{th} token of the models output. A situation similar to what we observe in Fig. 3.

Lake and Baroni (2018) explore two possible sources of this error, specifically decoder search failures and early terminations. However, they admit that neither seems to fully explain the performance chasm. In light of our work, the likely reason for the performance drop-off comes down to the lack of sufficient transition coverage necessary to learn the particular transitions enabling longer commands. Since the longer action sequences require mastering repetition inducing commands: *twice*, *thrice*, *and*, and *after*, we hypothesise that a providing sufficient coverage of these commands would dissolve the difference in accuracy between the lenghtwise and random splits. In other words, we suspect the issue doesn't reflect a neural network's ability to learn systematic compositionality, but rather the lower coverage of SCAN commands which appear at the end of an input sequence.

We extend our analysis by encoding SCAN as a SFST and find that it far exceeds our maximum SFST size of 100 states—we estimate the full SFST for SCAN has $7,728$ states. With a finite dataset size of $20,000$ input–output pairs, SCAN should not be possible to learn with a high accuracy. However, unlike datasets of a similar size, it turns out that SCAN is nearly perfectly learnable in our (random-split) experiments. This naturally raises the question of what makes SCAN special. We find the answer within the structure of the SFST that encodes it. SCAN's input alphabet includes primitive commands such as *walk*, *turn*, *run*, *turn left* and *turn right* among others. Each primitive command has a single accept state. Figure 4, shows a simplified transducer encoding of the primitive *jump*. Although the SCAN SFST has over $7,000$ states, the majority of these states and the transitions between them are repeated. Qualitatively, SCAN has 23 states for encoding each primitive command connected by 50 transitions. While SCAN as a SFST would be very large, its core logic is not so complex. Two of SCAN's design features in particular explain this: reversal and repetition.

First, the transduction in SCAN reverses the order of commands so that *jump right* turns into `RTURN JUMP`. Because of this reversal, the SFST encoding of SCAN cannot have a single encoding for all SCAN primitives. Each must be encoded separately, since the only transition in SCAN that can return anything is the one leading to the final state. Recall Figure 4, which encodes the basic logic of a single primitive such as *jump*, ignoring *twice* and *thrice* commands. We need four of these blocks, each with 40 states, to encode the four SCAN primitives or just above what we find comfortably learnable in our datasets.

Second, SCAN also has commands that allow for repetitions of sequences: *twice*, *thrice*, *and*, and *after*, which additionally reverse the input strings for output. Adding those primitives in will triple the number of states transitioning to the end state. For example, from state 2 in Figure 4, which encodes *jump right* we will need two more states transitioning from it, encoding *jump right twice* and *jump right thrice*. Repetition therefore expands the number of states per SCAN primitive to 24, putting it far above our empirically learnable threshold.

This effect is exacerbated with the commands *and* and *after*, where for every primitive in SCAN,

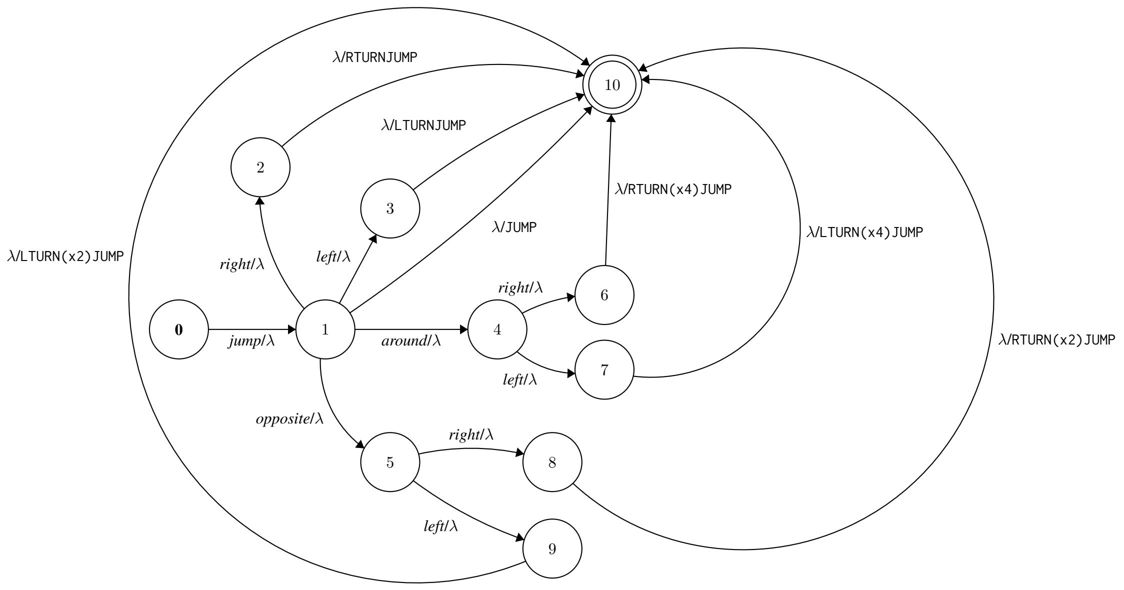

Figure 4: Core SFST block of SCAN encoding a primitive like *jump*. 0 is the start state and 10 is the accept state. Moving through states generates simple *jump* commands in SCAN dataset.

each transition connecting to the end state—10 in Figure 4—will have the whole SFST itself appended to its end, thereby multiplying the number of states each time. Since SCAN is learnable to a near 100% accuracy, we hypothesize that the LSTM must learn the repeated patterns between the different primitives, allowing models to learn SCAN despite it exceeding the ceiling of 40 states.

We lend some support to this hypothesis by including these special properties for random SFSTs in our datasets. We generate datasets with loops over the 20-state SFSTs, i.e., the largest feasibly learnable SFSTs. The loops add *twice* and *thrice* commands to each dataset, increasing the number of states in the SFST that generated them. We find no difference in learning these augmented SFSTs, demonstrating that sub-graph structures in the SFSTs indeed affect their learnability.

Therefore, we conclude that while SCAN's design contains many repeated blocks (i.e., sub-graphs), the ability of our neural models to learn a SFST depends on the complexity of the unique structure encoded by it, rather than these repetitions.[2] Viewing SCAN through the lens of automata thus reveals the properties that enable the learnability of the finite language it encodes.

However, the case of SCAN also reveals that to predict ease of learning requires a concept of complexity that is not reflected in SFST state counts: the duplicated sub-graph structures. We hope future work will use our method and explore different types of automata to identify classes that robustly encode learnability, bringing us closer to understanding how neural models learn such functions.

[2]Note that "blocks" are not visible to SFSTs. SFSTs have no way to encode similarity between their parts.

6 Related Work

We build on two strains of research: the work on systematic compositionality and on grammatical inference. This section contextualizes our work.

6.1 Compositionality Datasets

There is a growing number of artificial language datasets focused on systematic compositionality. Lake and Baroni (2018) introduced a SCAN dataset, made up of simple navigational text commands. The task is to translate the command in the simple natural language into sequences of actions. One successor to SCAN is the NACS dataset (Bastings et al., 2018), which is comparable to SCAN, but instead of mapping multiple input signals to a single duplicated output symbol (e.g., *walk twice* → `WALK WALK`), NACS does the opposite (`WALK WALK` → *walk twice*). NACS is both theoretically more difficult, and has been shown to be empirically more challenging (for both GRU and RNNs with attention, Bastings et al. 2018). The same inversion method applied to the transducers in our datasets may result in non-deterministic (and thus non-subsequential) transductions, since deterministic SFSTs are not closed under inverse (Mohri, 1997). Another successor is gSCAN (Ruis et al., 2020) which grounds SCAN-like commands in states of a grid world, closer to Mikolov et al. (2016)'s grid world grounding for their agents. gSCAN requires the agent to learn differences between sizes and colours of different geometric shapes and interact with them, by moving them around the grid world. Executing a gSCAN command is therefore much more difficult than to execute its SCAN counterpart. As Ruis et al. (2020) asserts, the gSCAN dataset removes artefacts in SCAN which are not central to the compositional generalization. They find that models perform worse on gSCAN than on SCAN.

Like the work above, we investigate systematic compositionality, but unlike the work above we generalise over the existing datasets by sampling from a well-defined class of patterns.

6.2 Grammatical Inference.

Grammatical inference studies the general behavior of learning algorithms over classes of functions generated by varieties of automata, rather than the specific behavior of specific algorithms on specific functions. Our work focuses on the learning of a restricted class of functions generated by a correspondingly restricted class of finite-state transducers. This allows us to synthesize our study of compositionality in neural models as rule-based inference by neural models, which we can restrict in principled ways. Finite-state machines generalise many techniques in NLP: probabilistic finite-state automata, hidden Markov models, Markov chains, n-grams, probabilistic suffix trees, deterministic stochastic probabilistic automata, weighted automata, and other syntactic objects which generate distributions over sets of possible infinite cardinality of strings, sequences, words, trees, and graphs (Vidal et al., 2005). Many grammatical inference studies of neural networks test them on samples drawn from some deterministic finite-state acceptor (DFA) (Cleeremans et al., 1989), and some then use quantization, clustering, or spectral algorithms to partition the RNN state space into an extractable DFA (Giles et al., 1992; Eyraud and Ayache, 2020) (see Jacobsson 2005 for a review).

Other inference experiments using neural nets attempt to see if the regular boundary can be empirically crossed. LSTMs (Hochreiter and Schmidhuber, 1997) can perform dynamic counting and variably learn simple counter languages such as some k-Dyck languages and $a^n b^n$ patterns (Weiss et al., 2018; Suzgun et al., 2019; Bhattamishra et al., 2020; Hewitt et al., 2020), which are generated by a finite-state machine with a counter on top (Schützenberger, 1962). In contrast, Avcu et al. (2017) show that LSTM and other RNN architectures often fail to learn long-distance dependencies drawn from simpler sub-regular language classes, even on large benchmarks (Mahalunkar and Kelleher, 2019). Nelson et al. (2020) study the inference of sequence-to-sequence networks, showing that RNN, LSTM, and GRU (Cho et al., 2014) systematically fail to learn a wide range of regular string copying functions generated from a family of two-way transducers, which characterize regular string-to-string functions. When augmented with attention, they reliably learn every function, and the attention history mirrors the derivations of the corresponding two-way transducers. These independently productive strands of work in compositionality and inference suggest that our work is a reasonable starting point for future interactions.

7 Conclusion

We study if sequence-to-sequence models are capable of learning systematic compositionality. To do this we sample arbitrary SFSTs and generate thousands of unique string-to-string datasets. By knowing the structure of the underlying SFSTs we can observe which of the underlying factors affect the learnability of this class of languages. This allows us to make precise statements about the learnability of systematic phenomena by a neural network.

We showed that neural models are indeed capable of learning generalized systematic behaviour. Moreover, we find that neural models tend to either generalize completely or fail miserably, with little middle ground. We pin down the underlying cause as transition coverage. With this in mind, we can now reason about possible cause for a pronounced performance difference between random and lengthwise split observed on the SCAN dataset. We observe that the cause here is likely not due to the lack of systematicity, but rather the lack of transition coverage. We find that formalising a human designed finite language as a SFST can reveal factors affecting its learnability which would otherwise remain hidden. Specifically, repeating sub-graphs in its SFST encoding can lead to learnability of SFSTs with a much higher number of states, and a corresponding lower transition coverage, than one would expect from an average dataset of similar number of states. Our formal language approach therefore not only allows for novel insights into the constrains for the neural models, but also a novel way of analysing the existing work.

References

Jacob Andreas. 2019. Measuring compositionality in representation learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Enes Avcu, Chihiro Shibata, and Jeffrey Heinz. 2017. Subregular complexity and deep learning. In *CLASP Papers in Computational Linguistics: Proceedings of the Conference on Logic and Machine Learning in Natural Language (LaML 2017), Gothenburg, 12 –13 June*, pages 20–33.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Borja Balle, Rémi Eyraud, Franco M. Luque, Ariadna Quattoni, and Sicco Verwer. 2017. Results of the sequence prediction challenge (SPiCe): a competition on learning the next symbol in a sequence. In *Proceedings of The 13th International Conference on Grammatical Inference*, volume 57 of *Proceedings of Machine Learning Research*, pages 132–136, Delft, The Netherlands. PMLR.

Jazmijn Bastings, Marco Baroni, Jason Weston, Kyunghyun Cho, and Douwe Kiela. 2018. Jump to better conclusions: SCAN both left and right. *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 47–55.

Siddharth Bhaskar, Jane Chandlee, Adam Jardine, and Christopher Oakden. 2020. Boolean monadic recursive schemes as a logical characterization of the subsequential functions. In *Language and Automata Theory and Applications - LATA 2020*, Lecture Notes in Computer Science, pages 157–169. Springer.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. On the practical ability of recurrent neural networks to recognize hierarchical languages. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1481–1494, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Lewis Carroll. 1871. *Through the Looking Glass*. Macmillan London.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Christian Choffrut. 2003. Minimizing subsequential transducers: a survey. *Theoretical Computer Science*, 292(1):131 – 143. Selected Papers in honor of Jean Berstel.

Noam Chomsky. 1957. *Syntactic Structures*. Walter de Gruyter.

Axel Cleeremans, David Servan-Schreiber, and James L. McClelland. 1989. Finite state automata and simple recurrent networks. *Neural Computation*, 1(3):372–381.

Verna Dankers, Elia Bruni, and Dieuwke Hupkes. 2021. The paradox of the compositionality of natural language: a neural machine translation case study. *arXiv preprint arXiv:2108.05885*.

Remi Eyraud and Stephane Ayache. 2020. Distillation of weighted automata from recurrent neural networks using a spectral approach. *arXiv preprint arXiv:2009.13101*.

Jerry A. Fodor and Zenon W. Pylyshyn. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71.

C. Lee Giles, Clifford B. Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. 1992. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405.

John Hewitt, Michael Hahn, Surya Ganguli, Percy Liang, and Christopher D. Manning. 2020. RNNs can generate bounded hierarchical languages with optimal memory. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1978–2010, Online. Association for Computational Linguistics.

Colin de la Higuera. 2010. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Charles F. Hockett. 1959. Animal “languages” and human language. *Human Biology*, 31(1):32–39.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality decomposed: How do neural networks generalise? (extended abstract). In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 5065–5069. International Joint Conferences on Artificial Intelligence Organization. Journal track.

Henrik Jacobsson. 2005. Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, 17(6):1223–1263.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2873–2882. PMLR.

Abhijit Mahalunkar and John Kelleher. 2019. Multi-element long distance dependencies: Using SPk languages to explore the characteristics of long-distance dependencies. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 34–43, Florence. Association for Computational Linguistics.

Gary F. Marcus. 1998. Rethinking eliminative connectionism. *Cognitive Psychology*, 37(3):243–282.

Tomas Mikolov, Armand Joulin, and Marco Baroni. 2016. A roadmap towards machine intelligence. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 29–61. Springer.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Richard Montague. 1970. Universal grammar. *Theoria*.

Max Nelson, Hossep Dolatian, Jonathan Rawski, and Brandon Prickett. 2020. Probing RNN encoder-decoder generalization of subregular functions using reduplication. In *Proceedings of the Society for Computation in Linguistics 2020*, pages 167–178, New York, New York. Association for Computational Linguistics.

José Oncina, Pedro García, and Enrique Vidal. 1993. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458.

Barbara Partee. 1995. Lexical semantics and compositionality. *An Invitation to Cognitive Science: Language*, 1:311–360.

Jonathan Rawski and Jeffrey Heinz. 2019. No free lunch in linguistics or machine learning: Response to pater. *Language*, 95(1):e125–e135.

James Rogers and Geoffrey K. Pullum. 2011. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20(3):329–342.

Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M Lake. 2020. A benchmark for systematic generalization in grounded language understanding. In *Advances in Neural Information Processing Systems 33 pre-proceedings (NeurIPS 2020)*.

Marcel P. Schützenberger. 1962. Finite counting automata. *Information and Control*, 5:91–107.

Mirac Suzgun, Yonatan Belinkov, Stuart Shieber, and Sebastian Gehrmann. 2019. LSTM networks can perform dynamic counting. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 44–54, Florence. Association for Computational Linguistics.

E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. 2005. Probabilistic finite-state machines - part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1013–1025.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision RNNs for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 740–745, Melbourne, Australia. Association for Computational Linguistics.

Shijie Wu and Ryan Cotterell. 2019. Exact hard monotonic attention for character-level transduction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1530–1537, Florence, Italy. Association for Computational Linguistics.