Probing RNN Encoder-Decoder Generalization of Subregular Functions using Reduplication

Max Nelson
Department of Linguistics
University of Massachusetts Amherst
manelson@umass.edu

Hossep Dolatian
Department of Linguistics
Institute for Advanced Computational Science
Stony Brook University
hossep.dolatian@stonybrook.edu

Jonathan Rawski
Department of Linguistics
Institute for Advanced Computational Science
Stony Brook University
jonathan.rawski@stonybrook.edu

Brandon Prickett
Department of Linguistics
University of Massachusetts Amherst
bprickett@umass.edu

Abstract

This paper examines the generalization abilities of encoder-decoder networks on a class of subregular functions characteristic of natural language reduplication. We find that, for the simulations we run, attention is a necessary and sufficient mechanism for learning generalizable reduplication. We examine attention alignment to connect RNN computation to a class of 2-way transducers.

1 Introduction

Reduplication is a cross-linguistically common morphological process (Moravcsik, 1978; Rubino, 2005). It is estimated that total reduplication and partial reduplication occur in 85% and 75% of the world's languages, respectively (Rubino, 2013). Total reduplication places no bound on the size of the reduplicant while partial does.

1. (a) wanita → wanita∼wanita (Indonesian)
   'woman → women'
   (b) guyon → gu∼guyon (Sundanese)
   'to jest → to jest repeatedly'

Morphological and phonological processes are sufficiently characterized by the regular class of languages and functions, and effectively computed by finite-state transducers (FSTs) (Johnson, 1972; Kaplan and Kay, 1994; Koskenniemi, 1984; Roark and Sproat, 2007). In finite-state calculus, an FST can process the input string either once in one direction (1-way FST), or multiple times by going back and forth (2-way FST). 1-way FSTs compute *rational* functions, while 2-way FSTs are more expressive, computing *regular* functions (Engelfriet and Hoogeboom, 2001; Filiot and Reynier, 2016).[1] Most morphological and phonological processes are in fact restricted to subclasses of rational functions and their corresponding 1-way FSTs (Chandlee, 2014, 2017; Chandlee and Heinz, 2018). The exception is total reduplication, which is uncomputable by 1-way FSTs due to its unboundedness (Culy, 1985; Sproat, 1992). It needs the power of 2-way FSTs, and requires subclasses of the regular functions (Dolatian and Heinz, 2018b).

This paper uses these subregular functions that characterize reduplication to probe the learning and generalization capacities of Recurrent Neural Network (RNN) architectures. While given infinite computational power, RNNs can simulate Turing machines (Siegelmann, 2012), many RNN classes and their gating mechanisms are actually expressively equivalent to weighted finite-state acceptors (Rabusseau et al., 2019; Peng et al., 2018). Furthermore, growing evidence suggests that RNNs and other sequential networks practically function as subregular automata (Merrill, 2019; Weiss et al., 2018).

We extend these subregular characterizations to

[1] In the French literature on formal language theory, 1-way FSTs compute *rational functions*. In contrast, most work in American computer science calls this class the *regular functions*. We follow French conventions because we also discuss 2-way FSTs which compute *regular functions* in their system.

test encoder-decoder (ED; Sutskever et al., 2014) networks. We use a typology of reduplication patterns computed by subregular 2-way FSTs (Dolatian and Heinz, 2019) to probe the ability of the networks to learn patterns of varying complexity. Our results suggest that when adding attention (Bahdanau et al., 2014) to these models, not only do they successfully learn and generalize all of the attested reduplication patterns that we test, but the attention acts in an alignment suggestive of the subregular 2-way FSTs. In contrast, lack of attention prohibits learning of the functions, and the generalization is suggestive of 1-way FSTs. This provides a principled glimpse into the interpretability of these networks on well-understood computational grounds, motivated by linguistic insight (Rawski and Heinz, 2019).

The paper proceeds as follows. §2 overviews the computation and learnability of reduplication. Methods, results, and discussion are in §3,§4,§5, respectively. Conclusions are in §6.

2 Background

2.1 Computing reduplication

As stated, reduplication is characterized by different subclasses of regular functions and computed by their corresponding FSTs, forming the hierarchy shown in Figure 1. 1-way FSTs compute *rational* functions. They are widely used in computational linguistics and NLP (Roche and Schabes, 1997; Beesley and Karttunen, 2003; Roark and Sproat, 2007). 2-way FSTs are more powerful. They exactly compute *regular functions*, which mathematically correspond to string-to-string transductions using Monadic Second Order logic (Engelfriet and Hoogeboom, 2001), making them the functional counterpart of the regular languages (Büchi, 1960). They have mostly been used outside of NLP (Alur and Černý, 2011).

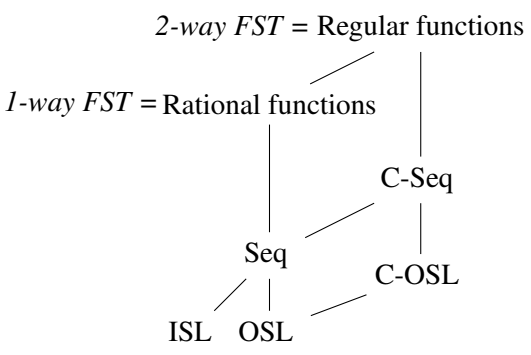

Figure 1: Hierarchy of subregular functions

When defined over a 1-way FST, all partial reduplicative functions are computable by Subsequential (Seq) functions (Chandlee and Heinz, 2012; Chandlee, 2017), which are computed by *deterministic* 1-way FSTs. Total reduplication is uncomputable by 1-way FSTs because there is no bound on the size of the reduplicant (Culy, 1985), so its output language is at least Mildly Context-Sensitive (Seki et al., 1991, 1993).

Over 2-way FSTs, both partial and total reduplication can be alternatively computed by a concatenation of subclasses of regular functions that are analogous to 1-way FST subclasses.[2] Almost all reduplicative processes, including total reduplication, are computed by Concatenated-Sequential (C-Seq) functions, which are concatenations of Seq functions (Dolatian and Heinz, 2018a,b). Most reduplication processes are sufficiently characterized by C-Seq functions because they can almost always be decomposed into two concatenated Seq functions: one to produce the reduplicant via truncation *Trunc(x)*, and one to produce an identical copy of the base *ID(x)*. Figure 2 shows such a division of a reduplicated word *gu∼guyon* (1b). [3] Figure 2 shows this division of a reduplicated word *gu∼guyon* (1b).

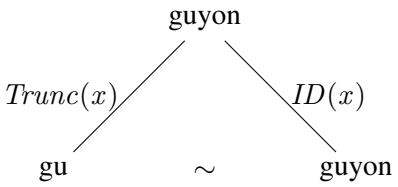

Figure 2: Initial-CV reduplication as a concatenation of subsequential functions.

Seq functions as 1-way FSTs and C-Seq functions as 2-way FSTs both compute partial reduplication, but differ in their *origin semantics* (Dolatian and Heinz, 2018b), the finite-state analog to alignment (Bojańczyk, 2014). Consider a function f, an FST T which computes f, and an input-output pair (x, y) such that $f(x) = y$. Given some substring y_j in y, the origin information of y_j with respect to T is the position x_i in x such that the

[2] See Alur et al. (2014) on the use of concatenation as a function combinator.

[3] Chandlee (2017) and Dolatian and Heinz (2018a)'s results are actually stronger. Over 1-way FSTs, most partial reduplicative processes are Input-Strictly Local (ISL) functions, a subclass of Seq functions. Over 2-way FSTs, most reduplicative processes are the concatenation of Output-Strictly Local (C-OSL) functions, a subclass of C-Seq.

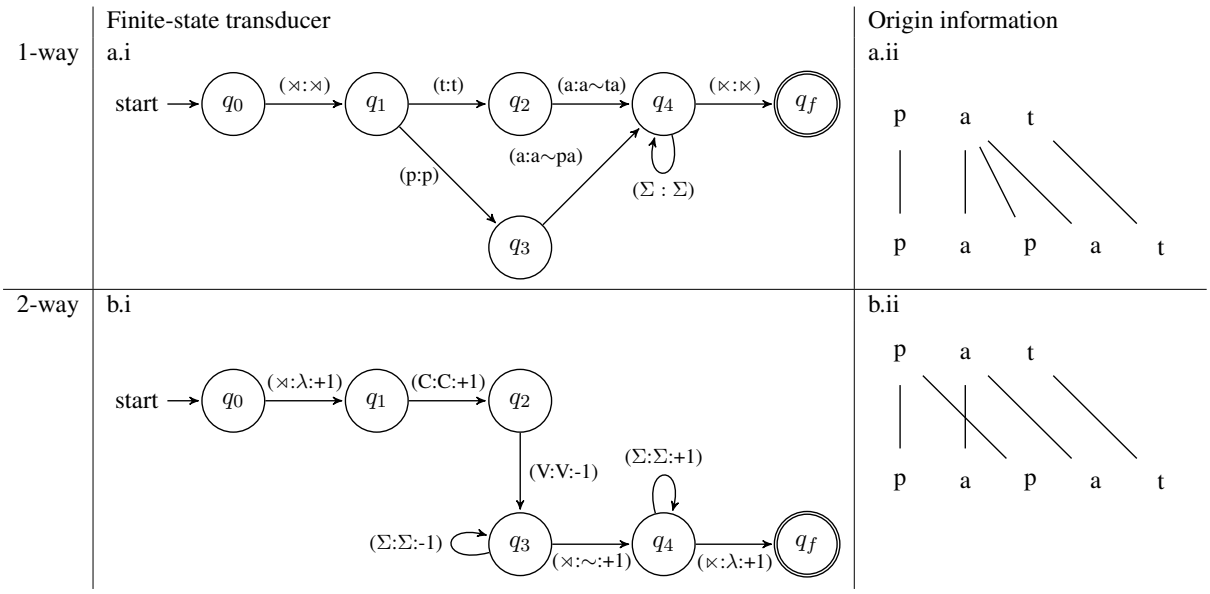

Figure 3: FSTs and origin information for initial-CV reduplication

FST's input-read head is in position x_i of the input x when the FST outputs the substring y_j.

To illustrate, consider initial-CV copying: $f(pat) = papat$. This function is computable by either the 1-way FST in Figure 3.a.i or the 2-way FST in Figure 3.b.i. The input is flanked by the end boundaries ⋊,⋉. The 1-way FST implicitly advances from left-to-right on the input string. The 2-way FST advances left-to-right via the explicit +1 direction parameter until it produces the first CV string (=the reduplicant). After that, it moves right-to-left via the -1 direction parameter and reaches the start boundary ⋊. It then advances left-to-right and outputs the base.[4] For the input-output pair $(pat, papat)$, the 1-way FST generates an 'alignment' or origin information such that the entire second copy '*pa*' is associated or generated from the vowel '*a*' in the input (Figure 3.a.ii). In contrast, the 2-way FST generates the alignment in Figure 3.b.ii where the second output '*p*' is associated with the input consonant '*p*'. The role of origin semantics and alignment acts as a diagnostic for understanding whether the neural networks we probe behave more like a 1-way or 2-way FST.

2.2 Learning reduplication

Chandlee et al. (2015) and Dolatian and Heinz (2018a) respectively show that ISL (Seq) and C-OSL (C-Seq) reduplicative processes are provably learnable by inducing their corresponding 1-way or 2-way FSTs in polynomial time and data. For Dolatian and Heinz (2018a), their proof relies on making the training data 'boundary enriched' with the reduplicative boundary symbol ∼, e.g. the training data for initial-CV reduplication is {(pat, pa∼pat), (mara, ma∼mara), etc.}. They hypothesize that learning without the boundary ∼ is tantamount to learning morpheme segmentation.

Gasser (1993) used simple RNNs to model reduplication and copying functions, finding that they could not properly learn reduplicative patterns. However, Prickett et al. (2018) found that ED networks, a class of RNNs that have performed well on a number of other morphological tasks (Cotterell et al., 2016; Kirov and Cotterell, 2018) could learn simple reduplicative patterns. These patterns used training data that did not represent a realistic language learning scenario, since all words had the same length and syllables were limited to a CV structure. We test the extent to which ED networks are capable of learning more realistic reduplicative functions. We find that vanilla EDs, like Prickett et al.'s, struggle to scale to realistic data, while EDs augmented with an attention mechanism easily acquire complex, natural-language-based reduplication patterns.

3 Methods

3.1 Data

We use a library of C-Seq transducers derived from the typology of natural language reduplication patterns (Dolatian and Heinz, 2019) to generate sets of input-output mappings which we use to

[4] See the appendix for more details on 2-way FSTs.

query several ED architectures.

The typology exhibits multiple parameters and distinctions. Already mentioned was the distinction between partial and total reduplication: copying a bounded substring of the input *gu∼guyon* (1b) vs. copying the entire potentially unbounded input *wanita* → *wanita∼wanita* (1a).

For partial reduplication, one subparameter is whether the reduplicant has a fixed size or a variable size that is still smaller than some fixed natural number. Fixed-sized partial reduplication is the most common pattern, e.g. initial CV-copying: *gu∼guyon* (1b) (Moravcsik, 1978; Rubino, 2005). One instantiation of variable-length partial reduplication is copying the initial foot (2(a)i) (Marantz, 1982), or syllable (2(b)i) (Haugen, 2005), which used to be unattested (Moravcsik, 1978). Another subparameter is whether the reduplicant is adjacent to the segments it copied (1b) or non-adjacent, i.e. wrong-sided (2c). Wrong-sided reduplication is controversial (Nelson, 2003) but attested (Riggle, 2004).

2. (a) i. (dimu)rU → dimu∼dimurU (Yidin)
'house' → 'houses'
ii. (gindal)ba → gindal∼gindalba
'lizard sp.' → 'lizards'
(b) i. vu.sa → vu∼vusa (Yaqui)
'awaken' → 'awaken (habitual)'
ii. vam.se → vam∼vamse
'hurry' → 'hurry (habitual)'
(c) qanga → qanga∼qan (Koryak)
'fire' → 'fire (absolute)'

Over 1-way FSTs, adjacent partial reduplication and foot/syllable copying are ISL while wrong-sided reduplication is Seq. Over 2-way FSTs, total reduplication and all the above partial reduplication functions are C-OSL, a subclass of C-Seq.[5]

We tested multiple patterns, including partial initial and wrong-sided reduplication of the first two syllables, total reduplication, and partial initial reduplication of the first two segments. For each pattern, the models are given base strings as input and trained to reproduce the base string along with its reduplicant (i.e. a right or left concatenated fully or partially copied form). For all patterns, 10,000 input-output pairs are generated, 7,000 of which are used to train the models while the remaining 3,000 are held out to test model generalization. For clarity the ∼ symbol is used throughout this paper to denote the boundary between a base and its reduplicant, however no such boundary is present in the model's training data.

[5]Foot and syllable copying are C-OSL if the input is marked by syllable/foot boundaries; otherwise they're C-Seq.

3.2 Models

Many ED networks were built and trained on the datasets described above. EDs are composed of a recurrent encoder, which sequentially processes an input string to yield a vector representation of the sequence in $\mathbb{R}^n$, and a recurrent decoder which takes the encoded representation of the input as a starting state and continues producing outputs until it produces a target stop symbol or reaches an experimenter-defined maximum length. The use of recurrent layers in both in the encoder and decoder allows EDs to map variable-length input sequences to variable-length output sequences, with no necessary relationship between the length of the input and target output (Sutskever et al., 2014).

Simple (SRNN) and gated (GRU) recurrence relations were tested as the encoder and decoder recurrent layers.[6] In SRNN layers the network's state at any timepoint, h_t, is dependent only on the input at that timepoint and the network's state at the previous timepoint (Elman, 1990).

$$h_t = tanh(W_x x_t + b_{ih} + W_h h_{t-1} + b_{hh}) \quad (1)$$

Consequently, in an SRNN there is only one path for the forward and backward propogation of information. This leads to potential problems for SRNNs in representing long-distance dependencies (Bengio et al., 1994) and problems with the backward flow of information during training (Hochreiter et al., 2001). GRU layers have a series of gates, called the reset r_t, update z_t, and new n_t gates, which create an alternative path of information flow (Cho et al., 2014), as shown in (2).

$$\begin{aligned} r_t &= \sigma(W_{ir} x_t + b_{ir} + W_{hr} h_{t-1} + b_{hr}) \\ z_t &= \sigma(W_{iz} x_t + b_{iz} + W_{hz} h_{t-1} + b_{hz}) \\ n_t &= tanh(W_{in} x_t + b_{in} + r_t \odot (W_{hn} h_{t-1} + b_{hn})) \\ h_t &= (1 - z_t) \odot n_t + z_t \odot h_{t-1} \end{aligned} \quad (2)$$

In a classic ED architecture, the encoded representation of the input is the only piece of infor-

[6]GRU layers have been shown to behave comparably to LSTMs, despite having fewer parameters (Chung et al., 2014). One difference between GRU and LSTM comes from (Weiss et al., 2018), who suggests that LSTMs are able to learn arbitrary $a^n b^n$ patterns while GRUs are not.

mation that is passed from the encoder to the decoder. This forces all necessary information in the input to be stored in this vector and preserved throughout the decoding process. In all experiments presented below, the target outputs consist of a concatenated reduplicant and base. Because the model must reproduce the base. it must preserve the identity of all phonemes in the input sequence. In order to test the ability of the model to learn the reduplicative function independent of its ability to store segment identities over arbitrarily long spans, a global weighted attention mechanism was incorporated into some of the models. This is a key point of departure from previous attempts to model reduplication with ED networks.

Attention allows the decoder to selectively attend to the hidden states of the encoder by learning a set of weights, W_{att}, which map the decoder's current state to a set of weights over timesteps in the input, and then concatenating the current decoder hidden state, h_t, the weighted combination of all encoder hidden states to yield a new current decoder state, h_{tt} (Bahdanau et al., 2014; Luong et al., 2015). This is illustrated in Equation 3, where E is a matrix of size *input length* $\times$ *hidden dimensionality* such that the ith row contains the encoder hidden state at timepoint i.

$$h_{tt} = \text{CAT}(h_t, \sigma(W_{att}h_t)^T E) \quad (3)$$

In this way, the decoder can pull information directly from the encoder by learning an alignment between the output and input representations.

The next section presents the results of training networks with either SRNN or GRU recurrent layers with and without an attention mechanism and then testing their ability to generalize the target pattern. All networks are trained to minimize phoneme level cross-entropy.

4 Results

In this section, we test ED networks on their ability to learn partial (§4.1,4.3) and total reduplication (4.2). Within partial reduplication, we test if they can learn adjacent reduplication vs. wrong-sided reduplication, and fixed-size vs. variable-length reduplication.

4.1 Partial reduplication

One simplifying assumption of previous work is that the reduplicant is a fixed-length substring of the base. This section tests the extent to which ED networks are able to learn reduplicative functions that copy a *variably* sized substring of the base in a way that is sensitive to linguistic structure which is not explicitly encoded in the training data.

Models were trained on initial and wrong-sided reduplication in which the reduplicant consisted of the first two-syllables in the word. Syllables were defined to be as onset-maximizing as possible and complex onsets and codas were included in the training data. This means that, for words with more than two syllables, the target reduplicant included everything between the left edge of the word and the right edge of the second vowel (initial: tasgatri→tasga∼tasgatri, wrong-sided: tasgatri→tasgatri∼tasgat). For words with only one or two vowels the reduplicant was the entire word (tasgat→tasgat∼tasgat). Due to the variable presence of onsets and codas, both simple and complex, reduplicants in these test cases vary in length between 2 and 10 phonemes, and may contain either 1 or 2 vowels.

In order for the model to learn this pattern, it must learn to identify which phonemes are consonants and which are vowels, must learn the syllabification rules, and must learn to handle the one-syllable exceptional case. Table (1) shows the generalization accuracy for the tested network architectures on datasets instantiating this pattern. As will be discussed in §4.3, the success of networks without attention is partially dependent on characteristics of the target language, namely the size of the language's segment inventory and permitted string lengths. To highlight these effects, results are reported from a representative *small* language, which has 10 unique phonemes and permits bases of between 3 and 9 segments, and a *large* language, which has 26 unique phonemes and permits bases of between 3 and 15 segments.

		Non-attention		Attention	
		Small	Large	Small	Large
Initial	SRNN	0.107	0.000	0.997	0.990
	GRU	0.787	0.234	1.000	1.000
wrong-sided	SRNN	0.001	0.000	0.995	0.994
	GRU	0.682	0.236	1.000	1.000

Table 1: Generalization accuracy by network type for all four languages that were tested.

The results suggest that the attention-based models are able to learn and generalize both initial and wrong-sided two-syllable reduplication patterns in a way that is robust to recurrence rela-

tion and language size. Non-attention GRU models show mild success in the small language, but seem heavily affected by language size, a result that will be explored thoroughly in §4.3. Non-attention RNN models are unable to learn the patterns in any of the simulations we ran.

The attention-based models are able to learn an alignment between the input and output that allows them to pull information directly from the input during decoding, sidestepping a potential information bottleneck at the encoded representation. To illustrate the alignment functions, an SRNN trained on two-syllable initial reduplication was used to make predictions about novel forms and the attention weights were stored. Figure (4) plots the attention weights for this model at every step in decoding for the three-syllable word *pastapo* and the two-syllable word *spaftof* ('<' and '>' represent start-of-sequence and end-of-sequence tokens, respectively).

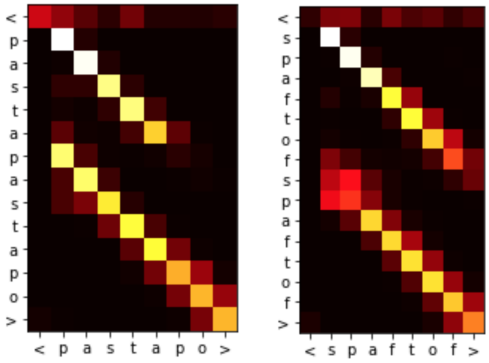

Figure 4: Attention weights over input (horizontal) at each time step of correct decoding of reduplicated form (vertical) for two-syllable initial reduplication of the words *pastapo* and *spaftof*. Darker squares indicate a lower weight on the alignment between two timesteps.

The attention weights confirm that the model learned an alignment between corresponding phonemes in the input and output. A single phoneme in the input has an output correspondent in both the base and reduplicant. These examples also illustrate the model's ability to i) identify the cut-off point for the reduplicant even when it is not explicitly marked and to ii) identify exceptional cases where the word is only two syllables and thus the reduplicant consists of material past the second vowel. In *pastapo* the model cuts off the reduplicant after the second vowel and in *spaftof* the model correctly includes the coda consonant because the word consists of only two syllables.

This section showed that attention-based models can learn initial and wrong-sided reduplication even when the pattern is complicated by sensitivity to linguistic structure that results in variable-length reduplicants. Once the network has learned enough structure to perform syllabification, the two-syllable *partial* reduplicative function is C-Seq. The next section examines the extent to which these networks learn *unbounded* copying, i.e. total reduplication.

4.2 Total reduplication

We test the ability of ED networks to learn and generalize total reduplication: *wanita* → wanita∼wanita (1a). As mentioned, total reduplication is not a rational function and is uncomputable with a 1-way FST, since there is no upper bound on the size of the copied string. However, it *is* a C-Seq function and computable by the corresponding 2-way FST. Total reduplication is thus a crucial test case for the RNN behavior.

As in §4.1, SRNN and GRU models with and without attention are trained on large and small languages where small languages have 10 phonemes and base lengths between 3 and 9 segments, and large languages have 26 phonemes and base lengths between 3 and 15 segments.

	Non-attention		Attention	
	Small	Large	Small	Large
SRNN	0.046	0.0	0.999	0.985
GRU	0.705	0.211	0.999	0.995

Table 2: Generalization accuracy by network type on both the large and small total reduplication patterns.

Table 2 shows the generalization accuracy for all network configurations. The results are nearly identical to those for the partial reduplication patterns in §4.1. Attention models can robustly learn the pattern, with negligible effects of recurrence relation or language size. Without attention, no model fully succeeds in generalizing the total reduplication pattern, with the best performance coming from the GRU on the small language.

These results show that attention-based models can learn a generalizable total reduplication function as well as they can learn partial reduplication functions. This means that attention-based ED network generalization does not distinguish between total and partial reduplication, despite glaring functional and automata-theoretic differences

in the functions themselves. This clearly suggests that an RNN architecture that can learn both functions necessarily computes a C-Seq function, which properly includes both processes. Furthermore, as discussed in §5, the interpretability of the corresponding FST characterization (2-way vs 1-way) and its origin semantics provides a direct computational link to the attention mechanism of these RNN architectures.

4.3 Alphabet size and string length effects

As shown so far, network architecture is not the only factor that influences a network's ability to learn a target reduplicative function. The composition of the target language, in terms of the number of segments in the language and the number of permitted string lengths, can have a dramatic effect on model behavior.

The effect of model architecture and language composition was investigated by testing the extent to which all network configurations could learn simple reduplication pattern while systematically varying the size of the segment inventory and permitted base lengths in the data. The reduplicative function chosen for these tests copied a fixed-window of two segments for initial reduplication: *guyon*→*gu*∼*guyon*. This was chosen because it is typologically well-attested (Moravcsik, 1978; Rubino, 2005, 2013) and also predicted to be the simplest reduplication pattern for the network to learn (since it is insensitive to linguistic structure and has a fixed-length reduplicant).

Data that followed this pattern was generated for languages with 10, 18, and 26 unique phonemes in their inventory and which permit bases to vary from 3 to between 5 and 10 segments. These results are shown in Figure (5).[7] The top panel shows the effect of alphabet size; string lengths are fixed between 3 and 8. The bottom panel, which shows the effect of string lengths; alphabet size is fixed at 26. The lines paralleling 1.0 in the top panel show that the ability of attention-based models to learn the target function is robust to alphabet size. The lines paralleling 1.0 in the bottom panel illustrate that attention-based models are similarly robust to string length variation.

In contrast, the non-attention models show large effects of alphabet size and string length. The non-attention SRNN shows very limited success. It is able to generalize with a very limited number of string lengths; but when maximum string length exceeds 7, it is no longer able to learn the target function at all. Consequently, the accuracy of the SRNN in the top panel, where maximum string length is fixed at 9, is stuck at 0.0 across all alphabet sizes.

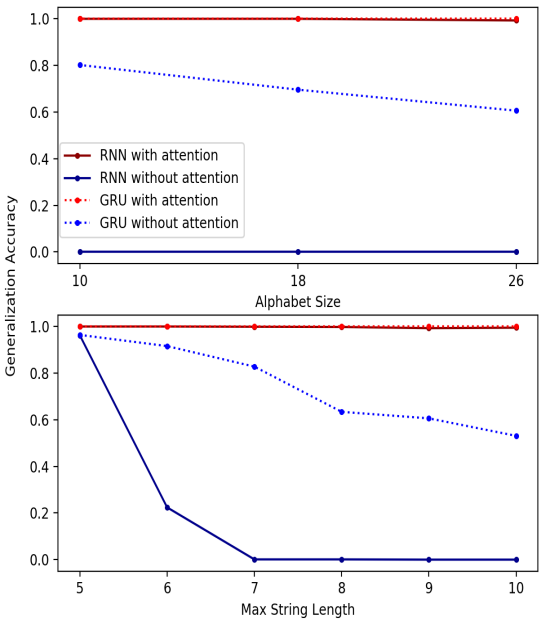

Figure 5: Effect on varying alphabet size and maximum string length, with minumum string length fixed at 3, on generalization accuracy.

The effects of both string length and alphabet size are also visible for the non-attention GRU. In the top panel, where maximum string length is fixed at 9, a decrease in generalization accuracy as a function of alphabet size is observed. The effect of maximum string length on the non-attention GRU is less dramatic than on the SRNN, but the GRU still displays a decrease from near ceiling accuracy with lengths between 3 and 5, to ~ 0.60 when lengths range between 3 and 10.

The sensitivity of non-attention SRNN and GRU models to alphabet size and string length are likely a result of the fact that these models are unable to directly reference the input during decoding and must pass all information through the encoder bottleneck. This hypothesis is strengthened by the fact that, without attention, the GRU performs much better than the SRNN. The GRU has extra gates between timepoints which aid in the long-distance preservation of information, mitigating the bottleneck problem to an extent. How-

[7] The reported results are from initial reduplication with a window size of two segments, however, wrong-sided reduplication and a larger window size of three were also tested with nearly identical results.

ever, while this assists the GRU network, it is not enough to make alphabet size and word length non-issues. The non-attention GRU is similar in architecture to the LSTM model of Prickett et al. (2018), with a slightly different training objective, suggesting that their model would similarly have difficulty scaling up.

The lack of a difference between the attention-based GRU and SRNN corroborates the idea that when this information bottleneck is not an issue both architectures are capable of learning generalizable reduplication.

5 Discussion

5.1 Origin semantics and alignment

As explained in §2.1, partial reduplication can be computed as a function with either 1-way or 2-way FSTs. However, the two finite-state algorithms differ in their origin semantics or alignment. The alignment difference is simulated by the attention-based RNNs. The alignments learned by attention-based models for partial reduplication in §4.1 and §4.3 are analogous to the origin semantics computed by the 2-way FST. We illustrate in Figure 6.

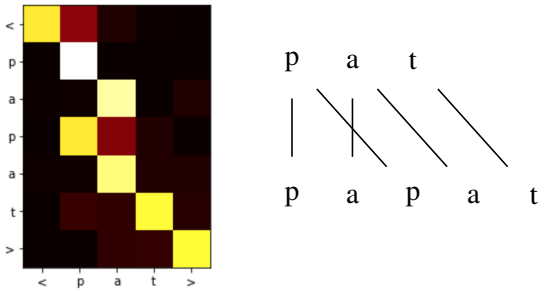

Figure 6: (left): Attention weights over input (horizontal) at each time step of correct decoding of reduplicated form (vertical) for the mapping *pat*→*pa*∼*pat*. Darker squares indicate a lower weight on the alignment between two timesteps. (right): Origin semantics of 2-way FST from Figure 3b.ii.

While both Seq and C-Seq functions sufficiently characterize partial reduplication, this 2-way-like alignment suggests that the RNNs are generalizing C-Seq functions (see Fig. 4 for other examples). This extends to total reduplication (§4.2) whose alignment when learned by the attention-based RNNs suggests the same origin information as 2-way FSTs. These results hint at the expressivity of the ED models, explicitly connecting their computations to the 2-way automata characterizing this subregular class.

5.2 Generality of copying mechanisms

The results suggest that the same general-purpose mechanism can be used to model both partial and total reduplication. The attention-based RNNs learned both processes with near-equal ease and generalizability and the same tools. This learning result fits well with reduplicative typology and theory. Partial and total reduplication are typologically and diachronically linked. If a language has partial reduplication, then it almost always has total reduplication, often because the former developed from the latter (Rubino, 2013). Because of this dependence, certain linguistic theories use the same mechanisms to generate both processes (Inkelas and Zoll, 2005).

Computationally, our result fits with the characterization of reduplication over 2-way FSTs (Dolatian and Heinz, 2018b) but not over 1-way FSTs (Chandlee et al., 2012). Because total reduplication cannot be modeled by a 1-way FSTs, some suggest that total and partial reduplication are ontologically different and should be computed with separate mechanisms (Roark and Sproat, 2007; Chandlee, 2017). In contrast, when computed over 2-way FSTs, both reduplicative processes fall under the *same* subclass of C-Seq functions.

5.3 Scaling problems

The results from §4.3 shows that attention-based RNNs could equally well learn a partial reduplication function regardless of alphabet size input size. In contrast, attention-less RNNs suffer. For an attention-less RNN, learning initial-CV copying with a small alphabet over smaller words is significantly easier then learning it with a larger alphabet over larger words. Their scaling difficulty is reminiscent of 1-way FST treatments of partial reduplication. To compute partial reduplication, 1-way FSTs can suffer a significant state explosion as alphabet size or reduplicant size increases. This is why some call 1-way FSTs 'burdensome models' for partial reduplication (Roark and Sproat, 2007, 54). 2-way FSTs do not suffer from state explosion (Dolatian and Heinz, 2018b).

6 Conclusions

We showed that RNN encoder-decoder networks with attention can learn partial and total redupli-

cation patterns. Non-attention models exhibited mixed success in learning generalizable reduplication functions in a way that was dependent on alphabet size and string length, suggesting that their failure is attributable to the information bottleneck between encoder and decoder rather than an inability to learn the target function. This corroborates the finding by Weiss et al. (2018) that recurrent networks' expressive power is restricted in practice, and shows the fruitfulness of using well-understood subregular classes to probe this expressivity.

References

Rajeev Alur, Adam Freilich, and Mukund Raghothaman. 2014. Regular combinators for string transformations. In Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, pages 9:1–9:10, New York, NY, USA. ACM.

Rajeev Alur and Pavol Černý. 2011. Streaming transducers for algorithmic verification of single-pass list-processing programs. In Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '11, pages 599–610, New York, NY, USA. ACM.

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

Kenneth Beesley and Lauri Karttunen. 2003. Finite-state morphology: Xerox tools and techniques. CSLI Publications.

Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. 1994. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2):157–166.

Mikołaj Bojańczyk. 2014. Transducers with origin information. In Automata, Languages, and Programming, pages 26–37, Berlin, Heidelberg. Springer.

J. Richard Büchi. 1960. Weak second-order arithmetic and finite automata. Mathematical Logic Quarterly, 6(1-6):66–92.

Jane Chandlee. 2014. Strictly Local Phonological Processes. Ph.D. thesis, University of Delaware, Newark, DE.

Jane Chandlee. 2017. Computational locality in morphological maps. Morphology, pages 1–43.

Jane Chandlee, Angeliki Athanasopoulou, and Jeffrey Heinz. 2012. Evidence for classifying metathesis patterns as subsequential. In The Proceedings of the 29th West Coast Conference on Formal Linguistics, pages 303–309, Somerville, MA. Cascillida Press.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2015. Output strictly local functions. In 14th Meeting on the Mathematics of Language, pages 112–125.

Jane Chandlee and Jeffrey Heinz. 2012. Bounded copying is subsequential: Implications for metathesis and reduplication. In Proceedings of the 12th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology, SIGMORPHON '12, pages 42–51, Montreal, Canada. Association for Computational Linguistics.

Jane Chandlee and Jeffrey Heinz. 2018. Strict locality and phonological maps. Linguistic Inquiry, 49(1):23–60.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR, abs/1412.3555.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, David Yarowsky, Jason Eisner, and Mans Hulden. 2016. The sigmorphon 2016 shared taskmorphological reinflection. In Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, pages 10–22.

Christopher Culy. 1985. The complexity of the vocabulary of Bambara. Linguistics and philosophy, 8:345–351.

Hossep Dolatian and Jeffrey Heinz. 2018a. Learning reduplication with 2-way finite-state transducers. In Proceedings of Machine Learning Research: International Conference on Grammatical Inference, volume 93 of Proceedings of Machine Learning Research, pages 67–80, Wroclaw, Poland.

Hossep Dolatian and Jeffrey Heinz. 2018b. Modeling reduplication with 2-way finite-state transducers. In Proceedings of the 15th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, Brussells, Belgium. Association for Computational Linguistics.

Hossep Dolatian and Jeffrey Heinz. 2019. Redtyp: A database of reduplication with computational models. In Proceedings of the Society for Computation in Linguistics, volume 2. Article 3.

Jeffrey L Elman. 1990. Finding structure in time. Cognitive science, 14(2):179–211.

Joost Engelfriet and Hendrik Jan Hoogeboom. 2001. MSO definable string transductions and two-way finite-state transducers. ACM Trans. Comput. Logic, 2(2):216–254.

Emmanuel Filiot and Pierre-Alain Reynier. 2016. Transducers, logic and algebra for functions of finite words. ACM SIGLOG News, 3(3):4–19.

Michael Gasser. 1993. Learning words in time: Towards a modular connectionist account of the acquisition of receptive morphology. Indiana University, Department of Computer Science.

Jason Haugen. 2005. Reduplicative allomorphy and language prehistory in Uto-Aztecan. In Bernhard Hurch, editor, Studies on reduplication, 28, pages 315–350. Walter de Gruyter, Berlin.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In John F Kolen and Stefan C Kremer, editors, A field guide to dynamical recurrent networks. John Wiley & Sons.

Sharon Inkelas and Cheryl Zoll. 2005. Reduplication: Doubling in Morphology. Cambridge University Press, Cambridge.

C Douglas Johnson. 1972. Formal aspects of phonological description. Mouton The Hague.

Ronald M Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. Computational linguistics, 20(3):331–378.

Christo Kirov and Ryan Cotterell. 2018. Recurrent neural networks in linguistic theory: Revisiting pinker and prince (1988) and the past tense debate. Transactions of the Association for Computational Linguistics, 6:651–665.

Kimmo Koskenniemi. 1984. A general computational model for word-form recognition and production. In Proceedings of the 10th international conference on Computational Linguistics, pages 178–181. Association for Computational Linguistics.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. CoRR, abs/1508.04025.

Alec Marantz. 1982. Re reduplication. Linguistic inquiry, 13(3):435–482.

William Merrill. 2019. Sequential neural networks as automata.

Edith Moravcsik. 1978. Reduplicative constructions. In Joseph Greenberg, editor, Universals of Human Language, volume 1, pages 297–334. Stanford University Press, Stanford, California.

Nicole Alice Nelson. 2003. Asymmetric anchoring. Ph.D. thesis, Rutgers University, New Brunswick, NJ.

Hao Peng, Roy Schwartz, Sam Thomson, and Noah A Smith. 2018. Rational recurrences. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 1203–1214.

Brandon Prickett, Aaron Traylor, and Joe Pater. 2018. Seq2seq models with dropout can learn generalizable reduplication. In Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology, pages 93–100.

Guillaume Rabusseau, Tianyu Li, and Doina Precup. 2019. Connecting weighted automata and recurrent neural networks through spectral learning. In AISTATS.

Jonathan Rawski and Jeffrey Heinz. 2019. No free lunch in linguistics or machine learning: Response to pater. Language, 94:1.

Jason Riggle. 2004. Nonlocal reduplication. In Proceedings of the 34th meeting of the North Eastern Einguistics Society. Graduate Linguistic Student Association, University of Massachusetts.

Brian Roark and Richard Sproat. 2007. Computational Approaches to Morphology and Syntax. Oxford University Press, Oxford.

Emmanuel Roche and Yves Schabes. 1997. Finite-state language processing. MIT press.

Carl Rubino. 2005. Reduplication: Form, function and distribution. In Studies on reduplication, pages 11–29. Mouton de Gruyter, Berlin.

Carl Rubino. 2013. Reduplication. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. Theoretical Computer Science, 88(2):191–229.

Hiroyuki Seki, Ryuichi Nakanishi, Yuichi Kaji, Sachiko Ando, and Tadao Kasami. 1993. Parallel multiple context-free grammars, finite-state translation systems, and polynomial-time recognizable subclasses of lexical-functional grammars. In Proceedings of the 31st annual meeting on Association for Computational Linguistics, pages 130–139. Association for Computational Linguistics.

Hava T Siegelmann. 2012. Neural networks and analog computation: beyond the Turing limit. Springer Science & Business Media.

Richard William Sproat. 1992. Morphology and computation. MIT press, Cambridge:MA.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. CoRR, abs/1409.3215.

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.

A Appendix

The definition and illustration for 2-way FSTs are taken from Dolatian and Heinz (2018b). We use $\rtimes$,$\ltimes$as the start and end boundaries.

3) **Definition:** A 2-way, deterministic FST is a six-tuple $(Q, \Sigma_{\ltimes}, \Gamma, q_0, F, \delta)$ such that:

Q is a finite set of states,
$\Sigma_{\ltimes} = \Sigma \cup \{\rtimes, \ltimes\}$ is the input alphabet,
Γ is the output alphabet,
$q_0 \in Q$ is the initial state,
$F \subseteq Q$ is the set of final states,
$\delta : Q \times \Sigma \rightarrow Q \times \Gamma^* \times D$ is the transition function where the direction $D = \{-1, 0, +1\}$.

For a survey on legitimate configurations in a 2-way FSTs, its computational properties, and complexity diagnostics, please see Dolatian and Heinz (2018b).

To illustrate 2-way FSTs, Figure 7 shows a 2-way FST for total reduplication. The 2-way operates by:

1. reading the input tape once from left to right in order to output the first copy,
2. going back to the start of the input tape by moving left until the start boundary $\rtimes$ is reached,
3. reading the input tape once more from left to right in order to output the second copy.

Specifically, this figure is interpreted as follows. The symbol Σ stands for any segment in the alphabet except for $\{\rtimes, \ltimes\}$. The arrow from q_1 to itself means this 2-way FST reads Σ, writes Σ, and advances the read head one step to the right on the input tape. The boundary symbol $\sim$ is a symbol in the output alphabet Γ, and is not necessary. We include it only for illustration.

We show an example derivation in Figure 8 for the input-output pair (*wanita*, *wanita*$\sim$*wanita*) (1a using the 2-way FST in Figure 7. The derivation shows the configurations of the computation for the input *wanita* and is step by step. Each tuple consists of four parts: *input string, output string, current state, transition*. In the *input string*, we underline the input symbol which FST will read next. The *output string* is what the 2-way FST has outputted up to that point. The symbol λ marks the empty string. The *current state* is what state the FST is currently in. The *transition* represents the used transition arc from input to output. In the first tuple, there is no transition arc used (N/A). But for other tuples, the form of the arc is:

$$\textit{input state} \xrightarrow[\text{direction}]{\text{input symbol:output string}} \textit{output state}$$

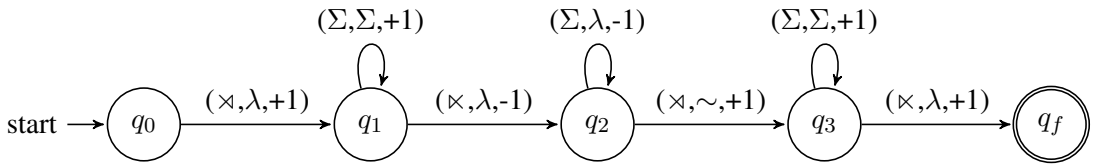

Figure 7: 2-way FST for total reduplication.

Outputting the first copy									
1.	(⋊wanita⋉,	λ,	q_0 ,	N/A)	9.	(⋊wanit̲a⋉,	wanita∼,	q_2,	$q_1 \xrightarrow[-1]{⋉:∼} q_2$)
2.	(⋊w̲anita⋉,	λ,	q_1,	$q_0 \xrightarrow[+1]{⋊:λ} q_1$)	10.	(⋊wanit̲a⋉,	wanita∼,	q_2,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)
3.	(⋊wa̲nita⋉,	w,	q_1,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)	11.	(⋊wani̲ta⋉,	wanita∼,	q_2,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)
4.	(⋊wan̲ita⋉,	wa,	q_1,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)	12.	(⋊wan̲ita⋉,	wanita∼,	q_2,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)
5.	(⋊wani̲ta⋉,	wan,	q_1,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)	13.	(⋊wa̲nita⋉,	wanita∼,	q_2,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)
6.	(⋊wanit̲a⋉,	wani,	q_1,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)	14.	(⋊w̲anita⋉,	wanita∼,	q_2,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)
7.	(⋊wanita̲⋉,	wanit,	q_1,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)	11.	(⋊̲wanita⋉,	wanita∼,	q_2,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)
8.	(⋊wanita⋉̲,	wanita,	q_1,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)					
Outputting the second copy									
12.	(⋊w̲anita⋉,	wanita∼,	q_3,	$q_2 \xrightarrow[+1]{⋊:λ} q_3$)	15.	(⋊wanit̲a⋉,	wanita∼wani,	q_3,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)
13.	(⋊wa̲nita⋉,	wanita∼w,	q_3,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)	15.	(⋊wanita̲⋉,	wanita∼wanit,	q_3,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)
14.	(⋊wan̲ita⋉,	wanita∼wa,	q_3,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)	16.	(⋊wanita⋉̲,	wanita∼wanita,	q_3,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)
14.	(⋊wani̲ta⋉,	wanita∼wan,	q_3,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)	17.	(⋊wanita⋉,	wanita∼wanita,	q_f,	$q_3 \xrightarrow[+1]{⋉:⋉} q_f$)

Figure 8: Derivation of *wanita→wanita∼wanita*.