Formal and Empirical Studies of Counting Behaviour in ReLU RNNs

Nadine El-Naggar[1], Andrew Ryzhikov[2], Laure Daviaud[3], Pranava Madhyastha[1], and Tillman Weyde[1]

[1]City, University of London, UK
[2]University of East Anglia, UK
[3]University of Oxford, UK

Abstract

In recent years, the discussion about systematicity of neural network learning has gained renewed interest, in particular the formal analysis of neural network behaviour. In this paper, we investigate the capability of single-cell ReLU RNN models to demonstrate precise counting behaviour. Formally, we start by characterising the semi-Dyck-1 language and semi-Dyck-1 counter machine that can be implemented by a single Rectified Linear Unit (ReLU) cell. We define three Counter Indicator Conditions (CICs) on the weights of a ReLU cell and show that fulfilling these conditions is equivalent to accepting the semi-Dyck-1 language, i.e. to perform exact counting. Empirically, we study the ability of single-cell ReLU RNNs to learn to count by training and testing them on different datasets of Dyck-1 and semi-Dyck-1 strings. While networks that satisfy the CICs count exactly and thus correctly even on very long strings, the trained networks exhibit a wide range of results and never satisfy the CICs exactly. We investigate the effect of deviating from the CICs and find that configurations that fulfil the CICs are not at a minimum of the loss function in the most common setups. This is consistent with observations in previous research indicating that training ReLU networks for counting tasks often leads to poor results. We finally discuss implications of these results and possible avenues for improving network behaviour.

1 Introduction

Recurrent Neural Networks (RNNs) have been demonstrated to be Turing complete, and thus are theoretically capable of performing any computable task [Siegelmann and Sontag, 1992]. However, there is a long-standing debate on the ability of neural networks to learn and generalise systematically since Fodor and Pylyshyn [1988]. *Counting* is one systematic process that is fundamental to many computational systems that process sequential data. It enables the system to keep track of occurrences, identify patterns, and make predictions based on past observations. Counting is relevant for natural language processing tasks such as language modelling, speech recognition, and machine translation. Furthermore, it is a fundamental cognitive ability that also plays an important role in human language understanding and comprehension [Le Corre et al., 2006, Gelman and Gallistel, 2004] and mathematical reasoning.

For learning systems, the question is what constitutes counting, how it can be defined, and how we can determine if the learning of counting behaviour has been successful. This is often approached through formal languages, especially Dyck-1, and by framing counting as a classification or prediction task. While some formal aspects of RNNs have been studied, there are few specific results on counting available to our knowledge. Empirical studies on the ability of RNNs to count have addressed mainly LSTMs, while ReLU RNNs are under-explored. This may be due to ReLU RNNs being notoriously difficult to train. RNNs in general are relatively inefficient at training time, while the training of Transformer models [Vaswani et al., 2017] is easy to parallelise and scale, which has been exploited in large language models, such as LaMDA [Thoppilan et al., 2022] or ChatGPT/GPT-4 [OpenAI, 2023]. However, there has been significant recent progress in designing efficiently trainable RNNs that can offer efficient training and generation [Gu et al., 2022, Peng et al., 2023, Orvieto et al., 2023].

In this paper, we investigate the counting behaviour of ReLU RNNs by reduction to a single cell. We characterise the counting capability of a single ReLU RNN cell as a formal language, the semi-Dyck-1 language, with a grammar and an abstract counter machine. We identify three Counter Indicator Conditions (CICs) on the weights of a ReLU RNN, and prove that for correct counting behaviour of a single-cell ReLU RNN it is necessary and sufficient that the network fulfils these three CICs. We empirically validate that ReLU RNNs that fulfil the CICs indeed show the desired counting behaviour even for very long strings. However, our experiments also show that learning the CICs is challenging and we identify a misalignment between the two commonly used loss functions and the CICs as a potential cause.

Our main contributions in this paper are:

- A characterisation of the counting capabilities of ReLU RNNs with a grammar for the semi-Dyck-1 language and an abstract counter machine that accepts this language.
- The definition of three Counter Indicator Conditions on a single-cell ReLU RNN with a proof that fulfilling them is equivalent to the network accepting the semi-Dyck-1 language.
- An empirical evaluation of the behaviour of ReLU RNNs, showing that they do not learn to fulfil the Counter Indicator Conditions in a common training setup and identifying causes for this result.

2 Related Work

RNNs are a powerful computational model, capable of computing any function if appropriately configured, and Turing complete with both sigmoid and ReLU activation functions [Siegelmann and Sontag, 1992, Chen et al., 2018]. However, systematicity in NN learning has been debated for over 30 years [see Fodor and Pylyshyn, 1988, Marcus et al., 1999, Lake and Baroni, 2018, inter alia]. In this debate, mostly an intuitive notion of systematicity has been used. Often examples of datasets or benchmarks have been used to capture the desired capabilities, such as the SCAN task [Lake and Baroni, 2018] and more recently the extensive BIG-Bench for large language models [Srivastava et al., 2022].

A different approach to defining systematic behaviour is the use of formal languages. The Chomsky hierarchy [Chomsky, 1957] has regular languages on its first level that are accepted by finite state automata (FSA) [Sipser, 1996, de la Higuera, 2010]. The next level, context-free languages, that are defined by a context-free grammar, can be accepted by a stack machine. While FSA are insufficient for infinite counting because of their finite state space, stack machines are more than is needed for simple counting functionality. Dyck languages [Rozenberg and Salomaa, 1997, Hopcroft and Ullman, 1969, Chomsky, 1956, Duchon, 2000] are context-free languages which consist of strings of well balanced brackets where an opening bracket is pushed onto the stack and a closing bracket pops the stack. In the case of some simple context-free languages such as Dyck-1 and $a^n b^n$, a stack can be reduced to a single counter, i.e. we store the number of items on the stack rather than the items themselves. These automata can be cast as Minsky machines [Minsky, 1967], which are monosymbolic pushdown automata. More general abstract counter machines have been defined by Fischer et al. [1968] and later adaptations followed by Merrill [2020].

Gers and Schmidhuber [2001] already train LSTMs to predict the next token in the context-free language $a^n b^n$ and the context-sensitive language $a^n b^n c^n$. There is recently a renewed interest in the abilities of RNNs to count and accept different formal languages. Weiss et al. [2018] show that RNNs with squashing activation functions, such as Elman RNNs and GRUs, do not have the capacity to count indefinitely with finite precision activation values, while other RNN models, such as LSTMs and ReLU RNNs, do. In terms of theoretical studies, Chen et al. [2018] provide extensive theoretical work on ReLU RNNs, but do not address counting, while Merrill [2019] does address counting, but not for ReLU RNNs.

Empirically, Karpathy et al. [2015], Bernardy [2018], and Skachkova et al. [2018] evaluate the prediction of brackets in natural and artificial strings with LSTMs. RNNs that incorporate some form of stack have been designed and evaluated by Mali et al. [2021], Joulin and Mikolov [2015], Grefenstette et al. [2015], Suzgun et al. [2019b], Hao et al. [2018], Mali et al. [2019], and Das et al. [1992]. However, for ReLU RNNs there are few results available. Beyond predicting brackets, the generalisation to longer strings has been studied to a limited extent by Suzgun et al. [2019a], and Weiss et al. [2018]

already found that no models exhibit perfect long-term generalisation. El-Naggar et al. [2022] explore the extent to which RNNs generalise Dyck-1 acceptance to very long strings and observe that ReLU models show high variability of learning effect. Lan et al. [2022] replace backpropagation with a genetic algorithm and a minimum description length target, which improves long-term generalisation. They provide proofs that two resulting ReLUs accept specific languages ($a^n b^n, a^n b^{2n}$) for arbitrary length strings.

The general question under which conditions ReLUs can count exactly and whether they can learn to count using backpropagation has not yet been addressed to our knowledge. The published reports of poor learning outcomes of ReLU RNNs with backpropagation on counting tasks suggest there is a specific problem that deserves studying.

3 Formalising Counting Behaviour in ReLU RNNs

In this section, we ask under which conditions a single-cell ReLU Recurrent Neural Network (ReLU RNN) can count, or more formally under which conditions a single-cell ReLU RNN accepts the semi-Dyck-1 language. The semi-Dyck-1 language is a variant of the Dyck-1 language (which is the language of well-formed bracket strings), adapted to the limitations of a single-cell ReLU RNN. In Theorem 8, we give a set of conditions that are necessary and sufficient for this to happen. The definitions used in the following are inspired by the notions of the General Counter Machine and Real-Time Language Acceptance presented in Merrill [2020].

3.1 Counter Machines and semi-Dyck-1 Language

We first define a particular variant of a counter machine and demonstrate its links with the semi-Dyck-1 language. This specific counter machine aligns with the computational constraints of a ReLU cell, which is incapable of producing *negative* activation values.

Definition 1 (Stateless Incremental Non-Negative 1-Counter Machine (SINC))**.** *A Stateless Incremental Non-Negative 1-Counter Machine (SINC) is a tuple* (Σ, u) *where* Σ *is a finite alphabet and* u *is a counter update function:*

$$u : \Sigma \to \{-1, +1\}$$

where $+1$ *(resp.* -1*) denote the function* $f(x) := x + 1$ *(resp.* $f(x) := x - 1$ *if* $x > 0$ *and* $f(0) = 0$*) defined on* $x \in \mathbb{N}$*.*

Definition 2 (Computations of SINC)**.** *A configuration of a SINC is a non-negative integer value (the value in the counter). A computation on input* $x = x_1 \cdots x_n$ *where* $x_i \in \Sigma$ *for all* $1 \le i \le n$ *is a sequence of configurations:*

$$c_0 \to c_1 \to \cdots \to c_n$$

such that $c_0 = 0$ *is the initial configuration, and for all* $1 \le i \le n$, $c_i = u(x_i)(c_{i-1})$*. The value* c_n *is called the output on* x *of the SINC.*

An alphabet Σ is a finite set of symbols, called tokens and a string on Σ is a sequence of tokens. In this paper we only consider finite strings, so finite sequences of tokens. The language accepted by a SINC is defined as the set of input strings that have output 0. In the following, we use the bracket characters $\langle$ and $\rangle$ to denote the characters in our alphabet, while other brackets have their usual meaning.

Note that a SINC is a particular instance of one-state pushdown automata with a one-token stack-alphabet, or can be seen as a one-state one-counter Minsky machine with an input alphabet.

Definition 3 (semi-Dyck-1 counter)**.** *A SINC* (Σ, u) *is said to be a semi-Dyck-1 counter if* $\Sigma = \{\langle, \rangle\}$, $u(\langle) = +1$ *and* $u(\rangle) = -1$*.*

The Dyck-1 language is the language of well-formed strings of brackets, e.g $\langle\langle\langle\rangle\rangle\langle\rangle\rangle$. It is defined on the alphabet $\Sigma = \{\langle, \rangle\}$ and generated by the context-free grammar $s \to \varepsilon | \langle s \rangle s$, where ε denotes the empty string.

Definition 4 (semi-Dyck-1 language)**.** *We define the semi-Dyck-1 language as generated by the context-free grammar* $s \to \varepsilon | \langle s \rangle s | s \rangle s$*.*

This is the language of strings constructed from strings of the Dyck-1 language in which closing brackets can be inserted at any point.

Proposition 5. *A semi-Dyck-1 counter accepts the semi-Dyck-1 language.*

Proof. Consider a semi-Dyck-1 counter (Σ, u) with $\Sigma = \{\langle, \rangle\}$. Let us first prove that any string in the semi-Dyck-1 language is accepted. Let x be a string in the semi-Dyck-1 language. By definition, $u(\langle) = +1$ and $u(\rangle) = -1$, which means that the counter is incremented by 1 everytime an opening bracket is read and decreased by 1 when a closing bracket is read, unless the counter value is 0, in which case, it remains 0. It is easy to see that the output on x is 0. This can be formally proved by induction on the grammar generating the semi-Dyck-1 language. Indeed,

- The output on ε is 0;
- Suppose now that s and s' are strings in the semi-Dyck-1 language with output 0 and $x = \langle s\rangle s'$. Then, on input x, first the counter is incremented by 1 after the first opening bracket. Then after reading s its value is either 1 (if s is in Dyck-1) or 0 (if s has at least one more closing bracket than opening ones) by induction hypothesis on s. In any case, after reading the closing bracket the counter has value 0, and after reading s', has again value 0, by induction hypothesis on s', so the output on x is 0;
- Suppose now that s and s' are strings in the semi-Dyck-1 language with output 0 and $x = s\rangle s'$. Then the counter is at 0 after reading s by induction hypothesis, remains at 0 after reading the closing bracket and is hence also at 0 after reading s' by induction hypothesis. So the output on x is 0.

On the other hand, we shall demonstrate that any accepted string belongs to the semi-Dyck-1 language. Consider an input string, x, which is accepted by the semi-Dyck-1 counter. We prove by induction that the output of x represents the quantity of opening brackets that remain unmatched by a closing bracket later on in the string.

- This is clearly true for ε.
- Suppose $x = x'\langle$ with the output on x' being the number of opening brackets that are not matched later on in x' by a closing bracket. Then, by definition of the semi-Dyck-1 counter, the output on x is the output on x' incremented by 1, and hence the induction hypothesis is satisfied for x.
- Suppose $x = x'\rangle$ with the output on x' being the number of opening brackets that are not matched later on in x' by a closing bracket. If this value is 0, then the same goes for x and the induction is satisfied. If this value is positive, then it is decremented by 1 for x, which also closes an opening bracket in x' so the induction is also satisfied.

This is adequate to conclude the proof, as the output of x equals zero, signifying that every opening bracket in x is subsequently matched by a closing bracket. Consequently, x is a member of the semi-Dyck-1 language. □

3.2 ReLU Recurrent Neural Networks (ReLU RNNs) as Counters

We define here formally a ReLU Recurrent Neural Network (ReLU RNN) and state the conditions that we will prove to be necessary and sufficient for it to behave as a semi-Dyck-1 counter.

Definition 6 (ReLU Recurrent Neural Network (ReLU RNN)). *A single-cell ReLU RNN is a tuple $(\Sigma, n, u, W, U, W_b)$ where Σ is a finite alphabet, n is a positive integer, u is a mapping from Σ to $\mathbb{R}^n$, W is a vector in $\mathbb{R}^n$ and U and W_b are real numbers. W, U, and W_b are called the weights of the ReLU RNN. A ReLU RNN takes as input a string $x = x_1 \cdots x_n$ on Σ, considering a token per timestep. An output activation function is computed at each timestep and defined as follows: $h_0 = 0$ and for all $t = 1, \ldots, n$,*

$$h_t = max(0, Wu(x_t) + Uh_{t-1} + W_b)$$

The product $Wu(x_t)$ is to be understood as the scalar product of two vectors in $\mathbb{R}^n$. The output of the ReLU RNN on input x is h_n.

The language accepted by a ReLU RNN is defined as the set of input strings having output 0. We now fix $\Sigma = \{\langle, \rangle\}$, and given a ReLU RNN $\mathcal{R} = (\Sigma, n, u, W, U, W_b)$, we define the following reals:

$$a_{\mathcal{R}} = Wu(\langle) + W_b \quad \text{and} \quad b_{\mathcal{R}} = Wu(\rangle) + W_b$$

Note that with this definition, the update function becomes $h_t = max(0, a_{\mathcal{R}} + Uh_{t-1})$ if $x_t = \langle$ and $h_t = max(0, b_{\mathcal{R}} + Uh_{t-1})$ if $x_t = \rangle$.

Definition 7 (Counting ReLU Recurrent Neural Network). *A single-cell ReLU RNN $\mathcal{R} = (\Sigma, n, u, W, U, W_b)$ is said to be Counting if it satisfies the three following conditions:*

1. $a_{\mathcal{R}} = -b_{\mathcal{R}}$
2. $a_{\mathcal{R}} > 0$
3. $U = 1$

We call these conditions *Counter Indicator Conditions (CICs)*. We now state our main result below.

Theorem 8. *For all single-cell ReLU RNNs $\mathcal{R}$, the three following assertions are equivalent:*

- *$\mathcal{R}$ is Counting,*
- *$\mathcal{R}$ accepts the semi-Dyck-1 language,*
- *$\mathcal{R}$ accepts the same language as a semi-Dyck-1 counter.*

First, we give two lemmas that will be used in the proof of Theorem 8.

Lemma 9. *For all ReLU RNNs $\mathcal{R}$ on $\Sigma = \{\langle, \rangle\}$ accepting the semi-Dyck-1 language, for all positive integers n and non-negative integers m, the output of $\mathcal{R}$ on input $\langle^n\rangle^m$ is:*

$$\begin{array}{ll} a_{\mathcal{R}}(1 + U + \cdots + U^{n-1}) & \text{if } m = 0 \\ a_{\mathcal{R}}U^m(1 + U + \cdots + U^{n-1}) + b_{\mathcal{R}}(1 + U + \cdots + U^{m-1}) & \text{if } n > m \geq 1 \\ \max(0, (a_{\mathcal{R}}U^n + b_{\mathcal{R}})(1 + U + \cdots + U^{n-1})) & \text{if } n = m \end{array}$$

Proof. The first item is proved by induction on n. For $n = 1$, the output on $\langle$ is $\max(0, a_{\mathcal{R}})$ which is $a_{\mathcal{R}}$ since $\langle$ is not in the semi-Dyck-1 language. Suppose now that this is true for some positive integer n. By definition of $\mathcal{R}$ and induction hypothesis, the output on $\langle^{n+1}$ is $\max(0, a_{\mathcal{R}} + U(a_{\mathcal{R}}(1+U+\cdots+U^{n-1})))$, which is $\max(0, a_{\mathcal{R}}(1 + U + \cdots + U^n))$. Since $\langle^{n+1}$ is not in the semi-Dyck-1 language, this has to be positive so the output is $a_{\mathcal{R}}(1 + U + \cdots + U^n)$.

The second item is proved by induction on m, considering the induction hypothesis is true for all $n > m$. For $m = 1$, for any $n > 1$, the output on $\langle^n\rangle$ is deduced from the previous item and is $\max(0, b_{\mathcal{R}} + Ua_{\mathcal{R}}(1+U+\cdots+U^{n-1}))$. This has to be positive so the output is $a_{\mathcal{R}}U(1+U+\cdots+U^{n-1}) + b_{\mathcal{R}}$. Suppose now that this is true for some positive integer m, and let $n > m + 1$. Then, by induction hypothesis, on input $\langle^n\rangle^{m+1}$, the output is $\max(0, b_{\mathcal{R}} + U(a_{\mathcal{R}}U^m(1+U+\cdots+U^{n-1}) + b_{\mathcal{R}}(1+U+\cdots+U^{m-1})))$, which is $a_{\mathcal{R}}U^{m+1}(1 + U + \cdots + U^{n-1}) + b_{\mathcal{R}}(1 + U + \cdots + U^m)$ since this has to be positive.

The third item is directly derived from the second and first ones used on input $\langle^n\rangle^{n-1}$'. If $n = 1$, the output on $\langle\rangle$ is $\max(0, a_{\mathcal{R}}U + b_{\mathcal{R}})$. Otherwise, for all positive integers $n > 1$, the output on $\langle^n\rangle^n$' is $\max(0, b_{\mathcal{R}} + U(a_{\mathcal{R}}U^{n-1}(1 + U + \cdots + U^{n-1}) + b_{\mathcal{R}}(1 + U + \cdots + U^{n-2}))$, which is $\max(0, (a_{\mathcal{R}}U^n + b_{\mathcal{R}})(1 + U + \cdots + U^{n-1}))$. □

Lemma 10. *For all strings x on alphabet $\Sigma = \{\langle, \rangle\}$, the output value of a semi-Dyck-1 counter on x is n for some integer n if and only if the output in a Counting ReLU RNN $\mathcal{R}$ on x is $a_{\mathcal{R}}n$.*

Proof. This follows from the definition of a Counting ReLU RNN. The activation function is now $h_t = max(0, a_{\mathcal{R}} + h_{t-1})$ if $x_t = \langle$ and $h_t = max(0, -a_{\mathcal{R}} + h_{t-1})$ if $x_t = \rangle$ for some $a_{\mathcal{R}} > 0$. In other words, the activation function is incremented by $a_{\mathcal{R}}$ everytime an opening bracket is read and decreased by $a_{\mathcal{R}}$ when a closing bracket is read, unless the value is 0, in which case, it remains 0. This describes exactly the computation of the counter in a semi-Dyck-1 counter, except that the increment and decrement are by 1 instead of $a_{\mathcal{R}}$. □

Proof of Theorem 8. The two last items are equivalent by Proposition 5.

We prove first that if $\mathcal{R}$ accepts the semi-Dyck-1 language, then it is Counting, i.e. it satisfies the three conditions from Definition 7. On input $x = \langle$, the output is $\max(0, a_{\mathcal{R}})$. Since x is not in the semi-Dyck-1 language, this output has to be positive, hence $a_{\mathcal{R}} > 0$. On input $x = \rangle$, the output is $\max(0, b_{\mathcal{R}})$. Since x is in the semi-Dyck-1 language, this output has to be 0, hence $b_{\mathcal{R}} \leq 0$. We prove now that $U = 1$, using Lemma 9.

- On input $\langle\langle$, the output is $a_{\mathcal{R}}(1 + U)$. This has to be positive, hence $U > -1$, since $a_{\mathcal{R}} > 0$.
- For all positive integers n, on input $\langle^n\rangle^n$, the output is $\max(0, (a_{\mathcal{R}}U^n + b_{\mathcal{R}})(1 + U + \cdots + U^{n-1}))$ and has to be non-positive. Since $U > -1$, then $(1 + U + \cdots + U^{n-1}) \geq 0$, hence $(a_{\mathcal{R}}U^n + b_{\mathcal{R}}) \leq 0$ for all n. Since $a_{\mathcal{R}} > 0$, $b_{\mathcal{R}} \leq 0$ and $\lim_{n\to\infty} U^n = \infty$ if $U > 1$, then necessarily $U \leq 1$.
- For all integers $n \geq 2$, on input $\langle^n\rangle^{n-1}$', the output is $\alpha_n = a_{\mathcal{R}}U^{n-1}(1 + U + \cdots + U^{n-1}) + b_{\mathcal{R}}(1 + U + \cdots + U^{n-2})$ and has to be positive. If $-1 < U < 1$, then $\lim_{n\to\infty}(1 + U + \cdots + U^{n-1}) = \lim_{n\to\infty}(1+U+\cdots+U^{n-2}) = (1-U)^{-1}$ and $\lim_{n\to\infty} U^{n-1} = 0$, hence $\lim_{n\to\infty} \alpha_n = b_{\mathcal{R}}(1-U)^{-1} \leq 0$. This contradicts the fact that α_n is strictly positive for all n. Then we obtain that $U = 1$.

Finally, with $U = 1$, using input $x = \langle\rangle$, the output is $\max(0, a_{\mathcal{R}} + b_{\mathcal{R}})$ and is 0, so $a_{\mathcal{R}} + b_{\mathcal{R}} \leq 0$. For all positive integers n, on input $\langle^n\rangle^{n-1}$, the output is $na_{\mathcal{R}} + (n-1)b_{\mathcal{R}} > 0$, and hence $a_{\mathcal{R}} + ((n-1)/n)b_{\mathcal{R}} > 0$. Since $\lim_{n\to\infty} a_{\mathcal{R}} + ((n-1)/n)b_{\mathcal{R}} = a_{\mathcal{R}} + b_{\mathcal{R}}$, then $a_{\mathcal{R}} + b_{\mathcal{R}} \geq 0$. We finally get $a_{\mathcal{R}} = -b_{\mathcal{R}}$.

We finally prove that if $\mathcal{R}$ is Counting, it accepts the same language as a semi-Dyck-1 counter. This is immediate by Lemma 10. A string is accepted by $\mathcal{R}$ if and only if its output is 0 and if and only if the output of a semi-Dyck-1 counter is 0. □

4 Experiments

Having identified the CICs that determine whether a ReLU RNN is Counting, we study the relationship between the exact and approximate fulfilment of the CICs and empirical behaviour of a ReLU RNN. We also investigate if training ReLU RNNs leads to them reaching or approximating the CICs and evaluate their counting behaviour empirically. Our general setup is similar to that of Gers and Schmidhuber [2001], Weiss et al. [2018], and Suzgun et al. [2019a].

4.1 Datasets and Metrics

To test the counting behaviour of our models, we use ten disjoint datasets, with their characteristics shown in Table 1. All Dyck-1 strings, except those in the Zigzag dataset, are generated in the same manner as Suzgun et al. [2019a], using a probabilistic Dyck-1 grammar. The Dyck-1 Zigzag dataset is created in the same way as in El-Naggar et al. [2022]. The Zigzag Dyck-1 strings consist of repetitions of j opening brackets followed by j closing brackets, where $j = \{10, 20, 25, 50, 100, 125, 200, 250, 500, 1000\}$. All datasets consist of strings that are each a valid string in their respective languages as a whole.

In our datasets, there are two labels at every timestep, as introduced by Gers and Schmidhuber [2001], Weiss et al. [2018], Suzgun et al. [2019a]. For Dyck-1 the labels indicate which next tokens, $\langle$ or $\rangle$, would be possible in the language, i.e. the string at the current timestep concatenated with the indicated token would be a prefix for a valid Dyck-1 string. An opening bracket can occur at any point in a Dyck-1 string, therefore the corresponding label 1 is always 1, but a closing bracket cannot occur at a point when there are no excess opening brackets, therefore label 2, for a closing bracket, is 0 iff there are no excess opening brackets, i.e. the current string is a valid Dyck-1 string. Therefore, this task can also be viewed as a classification task for Dyck-1 validity. Label 1 is obviously redundant in this setting, but we include it to ensure comparability with the literature.

The semi-Dyck-1 datasets are created from the Dyck-1 datasets by replacing every opening bracket with a closing one with probability 0.5. In order to generate strings of odd length, for half the strings we either add a closing bracket at a random position or remove a randomly chosen opening bracket, each with probability 0.5. In this way, the average string length is maintained.

The labels are set to ensure compatibility with the previous experiments and the literature. At every timestep we set the label 1 to 1 and label 2 to 0 if the string up to this point is a valid semi-Dyck-1 string, analogous to the Dyck-1 datasets. For semi-Dyck-1, the interpretation of output neurons as

Table 1: The datasets in our experiments, with different string structures and lengths, each in two versions: Dyck-1 (D) and semi-Dyck-1 (S). We report for all datasets the size (number of strings), lengths of the strings, percentage of valid strings (i.e. count value of 0), the mean and maximal counter value per string averaged over each dataset.

			D valid	counter		S valid	counter	
Type	**size**	**lengths**	**%**	**mean**	**max**	**%**	**mean**	**max**
Training	10,000	2–50	5.5	4.1	8.5	65.3	0.6	2.8
Validation	5,000	2–50	5.2	4.3	8.9	65.3	0.6	2.9
Long	5,000	52–100	2.4	6.9	14.3	64.3	0.6	3.8
Zigzag	10	2,000	1.2	114.0	228.0	53.9	3.0	24.2
Very Long	100	1,000	0.2	27.2	56.1	63.2	0.7	7.4

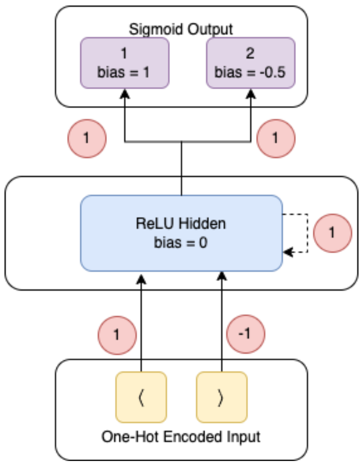


Figure 1: ReLU Model with a configuration satisfying the Counter Indicator Conditions.

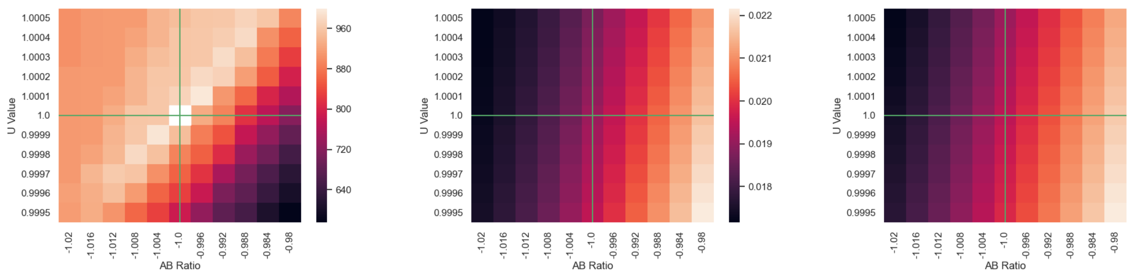
indicating what next token is allowed in the language is not valid in our encoding (we can always add an opening or closing bracket and the string is still a prefix to a valid semi-Dyck-1 string).

This way of changing the datasets to contain strings that are in semi-Dyck-1 but not Dyck-1 has several side effects. It changes the overall proportion of closing brackets from 0.5 to just over 0.75. The class ratio between valid and invalid strings is much more balanced, as can be seen in Table 1. The counter values that are needed for processing the strings also changed to much lower values, especially for the Zigzag and the Very Long datasets. Higher values require the model to count more precisely as any deviations from $U = 1$ are multiplied by the counter values and deviations from $a = -b$ accumulate. The semi-Dyck-1 dataset is therefore easier to process and better model performance in tests can be expected.

4.2 Experimental Setup and Evaluation

For our experiments, we use a ReLU RNN with a single hidden neuron as shown in Figure 1. Output neuron 1 has always target value 1 and neuron 2 indicates whether the string up to the current time is valid in the respective language. Although output neuron 1 has no role in the classification, it is still included in the loss calculation and optimisation, for compatibility as mentioned above. The output neurons have a sigmoid activation function (see Appendix A for the definition). We experiment with Binary Cross Entropy (BCE) loss and Mean Squared Error (MSE) loss (see Appendix A for the definitions). The combination of a sigmoid activation with MSE loss is an unusual combination which does not have a probabilistic interpretation like the commonly used cross-entropy, but it is what was used by Gers and Schmidhuber [2001], Weiss et al. [2018], and Suzgun et al. [2019a] and is retained here for comparability.

We report average Accuracy (number of strings classified correctly at every timestep) as a classification metric, as well as First Point of Failure (FPF), which reflects the exact counting capability



(a) FPF (b) MSE (c) BCE

Figure 2: Heatmaps showing the FPF values on the Dyck-1 Very Long dataset and MSE and BCE loss on the Dyck-1 Validation dataset for models with a correct configuration and with deviations. The thin green lines represent the CIC values for the a/b ratio and U value, and the intersection between the green lines is the point of a correct model. For FPF, the value in the centre of graph (a) is undefined, as no failures occurred for the correct model. It can be seen that the lowest MSE and BCE loss values are not located at the position of the correct model configurations. For a larger version of these heatmaps see Appendix C.

better. The FPF is the first point at which a model fails when a string is processed. For each model the average FPF value over the Very Long dataset is reported. FPF tests the generalisation abilities of models for very long strings, where fully correct processing becomes increasingly rare for imperfect models, so that accuracy ceases to differentiate models well.

4.3 Validating the Counting Model

For the correct model configuration, we use input weight vector $[1, -1]$, which, together with a hidden neuron bias of 0, leads to $a = 1$ and $b = -1$, which satisfies the CICs 1 and 2. We also use a recurrent weight $U = 1$, satisfying the CIC 3. The output bias is 1 for output neuron 1 and -0.5 for output neuron 2. We use a threshold of 0.5 for the final classification when calculating accuracy and FPF. As shown in Table 3, the correct model achieves perfect results on all datasets.

4.4 Effect of Deviation from the Correct Model

We systematically vary the a and U values from correct weights to study the effect of the deviation from CIC values on the MSE validation loss, BCE validation loss and FPF. For the U deviations, we use increments of 0.0001 between 0.9995 and 1.0005 for the U weight. Similarly, we use increments of 0.004 between 0.98 and 1.02 for the a weight. For these model variations, the MSE loss and BCE loss calculated on the Dyck-1 Validation dataset, and the FPF on the Dyck-1 Very Long dataset are shown in Figure 2. We observe that the highest FPF value occurs where the correct model is located. However, within our test grid, the lowest MSE or BCE losses do not occur at the correct a/b ratio or U value. There are different values of a and b possible for any given a/b ratio value. If we vary the biases proportional to the value of b (or b) for a given a/b ratio the only change to the ReLU activation value is a multiplication by a constant factor, as is easy to show by induction. Thus the only change of the relative magnitudes of network outputs is caused by the sigmoid output activation (for MSE), which is a monotonic function. We have plotted additional heatmaps in Appendix C, illustrating the very minor changes of the loss for different values of b. Thus, minimising the MSE or BCE loss will likely not converge to a model that fulfils the CICs and counts correctly.

4.5 Training ReLUs to Count

We train models in different configurations: using MSE (M/) and BCE (B/) loss with Randomly Initialised (RI), and Correctly Initialised weights with and without Bias in the ReLU (CB and CI, respectively). We train models for 30 epochs, with the Adam optimiser [Kingma and Ba, 2014], using

Table 2: Numbers of trained and converged models for Dyck-1 and semi-Dyck-1 experiments. The model names are as in Table 3.

	Dyck-1						Semi-Dyck-1					
	M/RI	**M/CI**	**M/CB**	**B/RI**	**B/CI**	**B/CB**	**M/RI**	**M/CI**	**M/CB**	**B/RI**	**B/CI**	**B/CB**
Trained	35	10	10	10	10	10	25	16	16	16	16	16
Conv	12	10	8	6	10	7	5	16	14	1	16	14

Table 3: Classification performance of various models trained and tested on Dyck-1 and semi-Dyck-1, and models with correct weights. Models are trained with MSE (M/) and BCE (B/) loss. Accuracy in percent and FPF values are given as mean (minimum/maximum) over runs after training models with Random Initialisation (RI), Correct Initialisation - without or with trainable ReLU bias (CI and CB). For B/RI trained on semi-Dyck-1, only one model converged, therefore there are no (min/max) values. An FPF value '-' indicates that the model did not fail on the Very Long dataset.

Mod	**Train**	**Validation**	**Long**	**Zigzag**	**V. Long**	**FPF**
Models with correct weights, Testing: Dyck-1 (*), semi-Dyck-1 (**)						
(*)	100	100	100	100	100	-
(**)	100	100	100	100	100	-
(a) Training: Dyck-1, Testing: Dyck-1						
M/RI	91.6 (48.7/100)	90.5 (43.1/100)	62.4 (3.74/100)	20.0 (0.0/40.0)	0.4 (0.0/4.0)	846.4 (528.2/979.3)
M/CI	95.2 (71.4/100)	94.4 (67.3/100)	76.1 (10.9/100)	25.0 (0.0/50.0)	1.2 (0.0/8.0)	905.3 (862.1/987.7)
M/CB	85.4 (39.4/100)	83.6 (33.2/100)	42.9 (0.9/98.1)	13.8 (0.0/30.0)	0.0 (0.0/0.0)	757.1 (432.0/911.9)
B/RI	97.7 (86.4/100)	97.3 (83.8/100)	83.9 (12.9/100)	23.3 (10.0/40.0)	8.3 (0.0/50.0)	848.4 (305.1/995.8)
B/CI	100 (100/100)	100 (100/100)	95.5 (60.3/100)	29.0 (10.0/60.0)	2.9 (0.0/29.0)	843.8 (553.0/992.4)
B/CB	91.9 (65.2/100)	90.4 (59.5/100)	72.2(3.9/100)	21.4 (10.0/40.0)	0.0 (0.0/0.0)	703.3 (257.4/959.9)
(b) Training: Dyck-1, Testing: semi-Dyck-1						
M/RI	100 (100/100)	100 (99.9/100)	100 (99.6/100)	44.2 (20.0/90.0)	99.0 (88.0/100)	992.3 (907.6/-)
M/CI	100 (100/100)	100 (99.9/100)	100 (99.7/100)	53.0 (20.0/100)	98.0 (88.0/100)	984.9 (907.6/-)
M/CB	99.9 (99.4/100)	99.9 (99.2/100)	99.5 (97.0/100)	99.9 (99.2/100)	90.8 (56.0/100)	938.3 (723.2/-)
B/RI	100 (100/100)	100 (100/100)	100 (99.9/100)	66.7 (20.0/90.0)	99.3 (96.0/100)	995.1 (970.8/-)
B/CI	90.0 (0.0/100)	90.0 (0.0/100)	90.0 (0.0/100)	49.0 (0.0/100)	90.0 (0.0/100)	900.3 (2.6/-)
B/CB	100 (100/100)	100 (100/100)	99.8 (99.3/100)	48.6 (20.0/90.0)	96.0 (84.0/100)	974.4 (892.3/-)
(c) Training: semi-Dyck-1, Testing: Dyck-1						
M/RI	99.4 (97.8/100)	99.2 (97.1/100)	73.3 (33.1/100)	22.0 (10.0/40.0)	3.2 (0.0/16.0)	724.7 (461.3/948.9)
M/CI	98.3 (87.5/100)	97.9 (85.6/100)	71.8 (25.8/100)	18.8 (10.0/70.0)	5.9 (0.0/94.0)	815.6 (436.2/999.5)
M/CB	100 (99.9/100)	99.9 (99.8/100)	90.3 (79.0/100)	25.0 (10.0/30.0)	6.8 (0.0/25.0)	893.3 (730.2/989.0)
B/RI	100	100	99.1	0.0	0.0	957.7
B/CI	90.8 (4.4/100)	90.2 (2.7/100)	68.9 (0.0/100)	16.4 (0.0/70.0)	7.1 (0.0/100)	788.2 (55.1/-)
B/CB	81.5 (0.0/100)	80.3 (0.0/100)	50.2 (0.0/100)	5.6 (0.0/30.0)	0.0 (0.0/0.0)	698.7 (0/985.0)
(d) Training: semi-Dyck-1, Testing: semi-Dyck-1						
M/RI	100 (100/100)	100 (100/100)	100 (100/100)	46.0 (30.0/90.0)	100 (100/100)	- (-/-)
M/CI	100 (100/100)	100 (100/100)	100 (100/100)	42.5 (20.0/100)	100 (100/100)	- (-/-)
M/CB	100 (100/100)	100 (100/100)	100 (100/100)	50.0 (20.0/100)	99.7 (98.0/100)	995.8 (970.8/-)
B/RI	100	100	100	50.0	100	-
B/CI	100 (100/100)	100 (100/100)	100 (100/100)	68.1 (20.0/100)	99.7 (95.0/100)	996.0 (936.5/-)
B/CB	100 (100/100)	100 (100/100)	100 (100/100)	60.7 (30.0/100)	100 (100/100)	- (-/-)

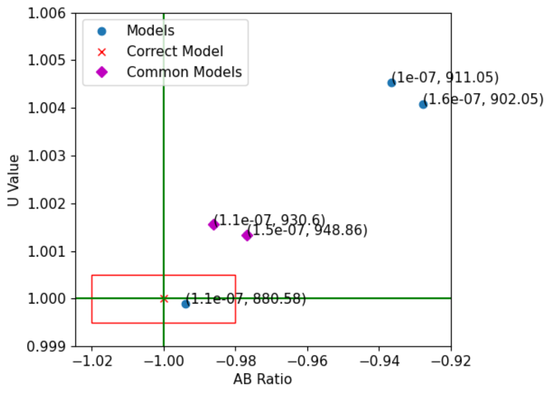
a learning rate of 0.01. We train models on the Dyck-1 dataset and on the semi-Dyck-1 dataset, and test each variant on both Dyck-1 and semi-Dyck-1 versions of the datasets.

We trained different models for different numbers of runs and in many runs the models did not converge to a loss value substantially below their initial state throughout the training. Therefore, we only include models that have converged in our results. The number of trained and converged models for each configuration is shown in Table 2. We select the models with the lowest validation loss for each run.

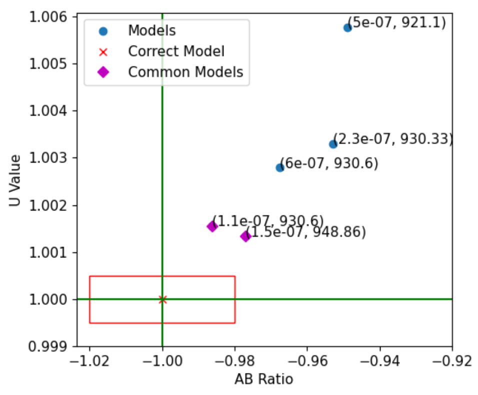
We report the accuracy for the converged models on all datasets and FPF values on the Very Long dataset in Table 3. All models show a large variation in the results. The largest differences can be observed between testing on the Dyck-1 and the semi-Dyck-1 datasets ((a) and (c) vs (b) and (d)), which was expected, given the difference in counter values discussed in section 4.1. The results differ also between the models trained on different datasets ((a) and (b) vs (c) and (d)), but unexpectedly the models trained on the less demanding semi-Dyck-1 dataset perform mostly better than the models trained on Dyck-1. However, fewer models converge when training on semi-Dyck-1 from random initialisation. Models trained with BCE loss perform broadly similar to the models trained with MSE loss.

Models trained from correct initialisation (*/C*) do not retain the correct weight configuration and their performance after training is often worse than that of the correct models, especially with a trainable bias (*/CB). This is consistent with our observation that the correct weights are not at the minimum of the loss (Section 4.4) and with the the findings of El-Naggar et al. [2022] that training from correct weights with a sigmoid output activation function results in unlearning of correct weights. However, the models that are correctly initialised tend to converge better (see Table 2).

Overall, no trained models show perfect results in the tests, i.e. they do not learn the correct weights that satisfy the CICs. For the models trained on Dyck-1 strings, we show plots of the a/b and U values of the 5 M/RI models with the lowest validation loss and the 5 M/RI models with the highest FPF in Figure 3. More distribution plots can be found in Appendix D. The plots show that the all models deviate from the correct values, and the deviations of the a/b ratio and the U value tends to be positive.



(a) Average Validation Loss



(b) Average FPF

Figure 3: The best 5 M/RI models trained and tested on Dyck-1 selected (a) by average validation loss and (b) by FPF on the Very Long dataset. The red box represents the area corresponding to the heatmaps in Figure 2. The correct model position is at the intersection of the green lines. Two models present in both sets are marked with pink diamonds. For all models, the validation loss and FPF values are shown next to the markers.

5 Discussion and Conclusions

We have formalised the counting mechanism of a ReLU RNN cell with the semi-Dyck-1 language and corresponding abstract counter machines that account for the absence of negative activation values in ReLUs. On this basis, we established three Counter Indicator Conditions (CICs) on the ReLU weights, which are necessary and sufficient for exhibiting correct counting behaviour. ReLU RNN cells that satisfy the CICs can count exactly, allowing for generalisation to strings of arbitrary length and arbitrarily great counter values (numeric representation permitting). We have empirically validated that a single-cell ReLU RNN that satisfies the CICs does indeed count correctly and accept the semi-Dyck-1 language, even on very long strings. However, our results also indicate that the mean squared error and the binary cross entropy as loss functions for training ReLU RNNs do not train the ReLU RNN to satisfy the CICs so that the trained models fail on very long strings.

Another observation was that ReLU RNN training often fails to converge. In earlier work [El-Naggar et al., 2022], we observed that LSTMs are trainable with reliable convergence on this type of data, suggesting that they possess a more suitable inductive bias for counting. However, we also observed that LSTMs tend to systematically underestimate counts, unlike ReLU RNNs which do not exhibit this issue. Therefore, it would be useful to better understand the interaction between stable convergence and long-term counting success and the loss function.

We believe that our empirical results in this paper offer relevant insights into some of the difficulties encountered during the training of ReLU RNNs, as highlighted by Weiss et al. [2018]. On the other hand, the theoretical framework of semi-Dyck-1 language and CICs and our finding of misalignment of the loss functions can be used to design new training methods and network architectures that take into account these conditions to enable effective learning of exact counting in ReLU RNN networks. These can include heterogeneous models that include discrete counter modules or specific regularisations that stabilise convergence and create an inductive bias towards exact counting.

Acknowledgment

We thank the anonymous reviewers for their comments and constructive feedback. Laure Daviaud has been supported for this work by the EPSRC grant EP/T018313/1. Andrew Ryzhikov is partially supported by the EPSRC grant EP/T018313/1 and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 852769, ARiAT).

References

Jean-Philippe Bernardy. Can recurrent neural networks learn nested recursion? *Linguistic Issues in Language Technology*, 16, 2018. URL `https://aclanthology.org/2018.lilt-16.1`.

Yining Chen, Sorcha Gilroy, Andreas Maletti, Jonathan May, and Kevin Knight. Recurrent neural networks as weighted language recognizers. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2261–2271. Association for Computational Linguistics, 2018. doi: 10.18653/v1/n18-1205. URL `https://doi.org/10.18653/v1/n18-1205`.

Noam Chomsky. Three models for the description of language. *IRE Trans. Inf. Theory*, 2(3):113–124, 1956. doi: 10.1109/TIT.1956.1056813. URL `https://doi.org/10.1109/TIT.1956.1056813`.

Noam Chomsky. *Syntactic Structures.* Mouton, 1957.

Sreerupa Das, C. Lee Giles, and Guo-Zheng Sun. Using prior knowledge in a NNPDA to learn context-free languages. In Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 65–72. Morgan Kaufmann, 1992. URL `http://papers.nips.cc/paper/587-using-prior-knowledge-in-a-nnpda-to-learn-context-free-languages`.

Colin de la Higuera. *Grammatical inference: learning automata and grammars.* Cambridge University Press, 2010.

Philippe Duchon. On the enumeration and generation of generalized Dyck words. *Discret. Math.*, 225(1-3):121–135, 2000. doi: 10.1016/S0012-365X(00)00150-3. URL `https://doi.org/10.1016/S0012-365X(00)00150-3`.

Nadine El-Naggar, Pranava Madhyastha, and Tillman Weyde. Experiments in learning Dyck-1 languages with recurrent neural networks. In Alan Bundy and Denis Mareschal, editors, *Proceedings of the 3rd Human-Like Computing Workshop (HLC 2022) co-located with the 2nd International Joint*

Conference on Learning and Reasoning (IJCLR 2022), Windsor, United Kingdom, September 28-30th, 2022, volume 3227 of *CEUR Workshop Proceedings*, pages 24–28. CEUR-WS.org, 2022. URL `http://ceur-ws.org/Vol-3227/El-Naggar.PP5.pdf`.

Nadine El-Naggar, Pranava Madhyastha, and Tillman Weyde. Exploring the long-term generalization of counting behavior in RNNs. In *I Can't Believe It's Not Better Workshop: Understanding Deep Learning Through Empirical Falsification*, 2022.

Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Counter machines and counter languages. *Math. Syst. Theory*, 2(3):265–283, 1968. doi: 10.1007/BF01694011. URL `https://doi.org/10.1007/BF01694011`.

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

Rochel Gelman and Charles Randy Gallistel. Language and the origin of numerical concepts. *Science*, 306(5695):441–443, 2004.

Felix A. Gers and Jürgen Schmidhuber. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Trans. Neural Networks*, 12(6):1333–1340, 2001. doi: 10.1109/72.963769. URL `https://doi.org/10.1109/72.963769`.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Advances in neural information processing systems*, pages 1828–1836, 2015.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=uYLFoz1vlAC`.

Yiding Hao, William Merrill, Dana Angluin, Robert Frank, Noah Amsel, Andrew Benz, and Simon Mendelsohn. Context-free transductions with neural stacks. In Tal Linzen, Grzegorz Chrupala, and Afra Alishahi, editors, *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pages 306–315. Association for Computational Linguistics, 2018. doi: 10.18653/v1/w18-5433. URL `https://doi.org/10.18653/v1/w18-5433`.

John E. Hopcroft and Jeffrey D. Ullman. *Formal languages and their relation to automata*. Addison-Wesley series in computer science and information processing. Addison-Wesley, 1969. ISBN 0201029839. URL `https://www.worldcat.org/oclc/00005012`.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *Advances in neural information processing systems*, 28, 2015.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2879–2888. PMLR, 2018. URL `http://proceedings.mlr.press/v80/lake18a.html`.

Nur Geffen Lan, Michal Geyer, Emmanuel Chemla, and Roni Katzir. Minimum description length recurrent neural networks. *Trans. Assoc. Comput. Linguistics*, 10:785–799, 2022. URL `https://transacl.org/ojs/index.php/tacl/article/view/3649`.

Mathieu Le Corre, Gretchen Van de Walle, Elizabeth M Brannon, and Susan Carey. Re-visiting the competence/performance debate in the acquisition of the counting principles. *Cognitive psychology*, 52(2):130–169, 2006.

Ankur Arjun Mali, Alexander Ororbia, and C. Lee Giles. The neural state pushdown automata. *CoRR*, abs/1909.05233, 2019. URL `http://arxiv.org/abs/1909.05233`.

Ankur Arjun Mali, Alexander Ororbia, Daniel Kifer, and C. Lee Giles. Recognizing long grammatical sequences using recurrent networks augmented with an external differentiable stack. In Jane Chandlee, Rémi Eyraud, Jeff Heinz, Adam Jardine, and Menno van Zaanen, editors, *Proceedings of the 15th International Conference on Grammatical Inference, 23-27 August 2021, Virtual Event*, volume 153 of *Proceedings of Machine Learning Research*, pages 130–153. PMLR, 2021. URL `https://proceedings.mlr.press/v153/mali21a.html`.

Gary F Marcus, Sugumaran Vijayan, S Bandi Rao, and Peter M Vishton. Rule learning by seven-month-old infants. *Science*, 283(5398):77–80, 1999.

William Merrill. Sequential neural networks as automata. *CoRR*, abs/1906.01615, 2019. URL `http://arxiv.org/abs/1906.01615`.

William Merrill. On the linguistic capacity of real-time counter automata. *CoRR*, abs/2004.06866, 2020. URL `https://arxiv.org/abs/2004.06866`.

Marvin Lee Minsky. *Computation.* Prentice-Hall Englewood Cliffs, 1967.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/arXiv.2303.08774. URL `https://doi.org/10.48550/arXiv.2303.08774`.

Antonio Orvieto, Samuel L. Smith, Albert Gu, Anushan Fernando, Çaglar Gülçehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. *CoRR*, abs/2303.06349, 2023. doi: 10.48550/arXiv.2303.06349. URL `https://doi.org/10.48550/arXiv.2303.06349`.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran G. V., Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartlomiej Koptyra, Hayden Lau, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Xiangru Tang, Bolun Wang, Johan S. Wind, Stanislaw Wozniak, Ruichong Zhang, Zhenyuan Zhang, Qihang Zhao, Peng Zhou, Jian Zhu, and Rui-Jie Zhu. RWKV: reinventing rnns for the transformer era. *CoRR*, abs/2305.13048, 2023. doi: 10.48550/arXiv.2305.13048. URL `https://doi.org/10.48550/arXiv.2305.13048`.

Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages, Volume 1: Word, Language, Grammar.* Springer, 1997. ISBN 978-3-642-63863-3. doi: 10.1007/978-3-642-59136-5. URL `https://doi.org/10.1007/978-3-642-59136-5`.

Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 440–449, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi: 10.1145/130385.130432. URL `https://doi.org/10.1145/130385.130432`.

Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

Natalia Skachkova, Thomas Alexander Trost, and Dietrich Klakow. Closing brackets with recurrent neural networks. In Tal Linzen, Grzegorz Chrupala, and Afra Alishahi, editors, *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pages 232–239. Association for Computational Linguistics, 2018. doi: 10.18653/v1/w18-5425. URL `https://doi.org/10.18653/v1/w18-5425`.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M Shieber. LSTM networks can perform dynamic counting. *arXiv preprint arXiv:1906.03648*, 2019a.

Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M Shieber. Memory-augmented recurrent neural networks can learn generalized Dyck languages. *arXiv preprint arXiv:1911.03329*, 2019b.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Kathleen S. Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Agüera y Arcas, Claire Cui, Marian Croak, Ed H. Chi, and Quoc Le. Lamda: Language models for dialog applications. *CoRR*, abs/2201.08239, 2022. URL `https://arxiv.org/abs/2201.08239`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision RNNs for language recognition. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pages 740–745. Association for Computational Linguistics, 2018. doi: 10.18653/v1/P18-2117. URL `https://aclanthology.org/P18-2117/`.

A Model Output and Loss Functions

Definition 11 (Model Output Calculation). *The model's output layer performs the following calculation:*

$$Y = \sigma(W_Y h_t + b_Y),$$

where Y is the output W_Y is the output weight, h_t is the output of the ReLU hidden layer, and b_Y is the output bias. σ is the logistic sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

where e is Euler's number.

Definition 12 (Mean Squared Error MSE Loss (MSE)). *Mean Squared Error loss is calculated as follows:*

$$MSELoss = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

where n is the length of the string, Y_i is the predicted output value and $\hat{Y}_i$ is the target output value at timestep i.

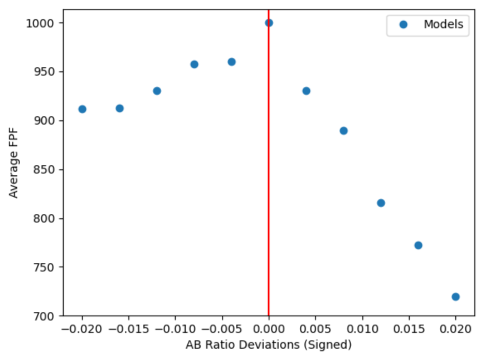
Definition 13 (Binary Cross Entropy Loss (BCE)). *Binary Cross Entropy Loss is calculated using the following equation:*

$$BCELoss = -\sum_{n=i}^{c}\hat{Y}_i \log(Y_i) + (1-\hat{Y}_i)\log(1-Y)$$

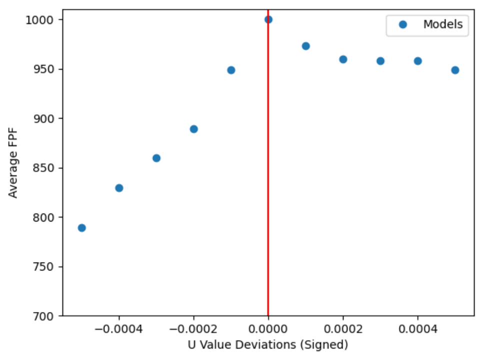
where Y is the predicted output, $\hat{Y}$ is the target output and c is the number of classes.

B Deviation from CICs - FPF and Loss Plots

See Figure 4 and Figure 5.

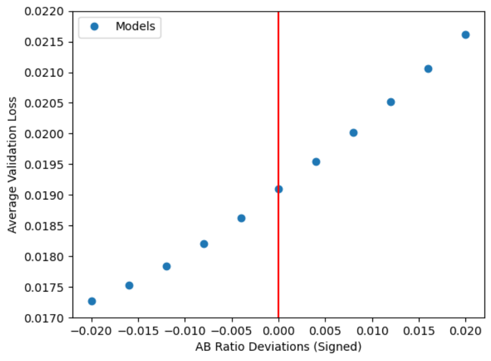


(a) a/b Deviations and FPF

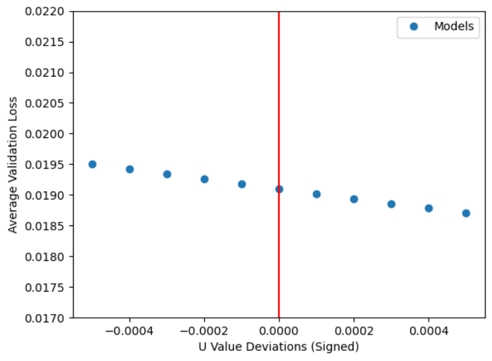


(b) U Deviations and FPF

Figure 4: Effect of Deviations from the CICs on the First point of Failure (maximum is 1000).



(a) a/b Deviations and Validation Loss

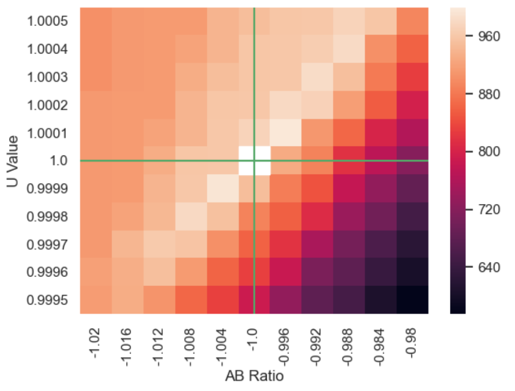


(b) U Deviations and Validation Loss

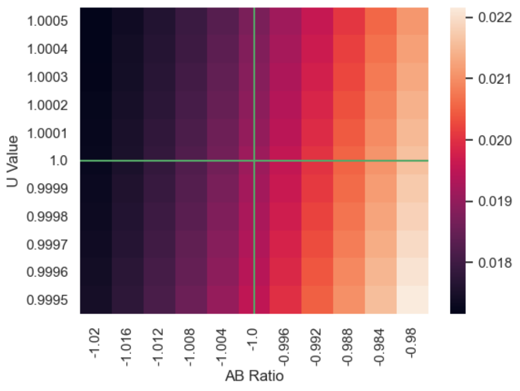
Figure 5: Effect of Deviations in CICs on the Validation Loss

C Deviation from CICs - FPF and Loss Heatmaps.

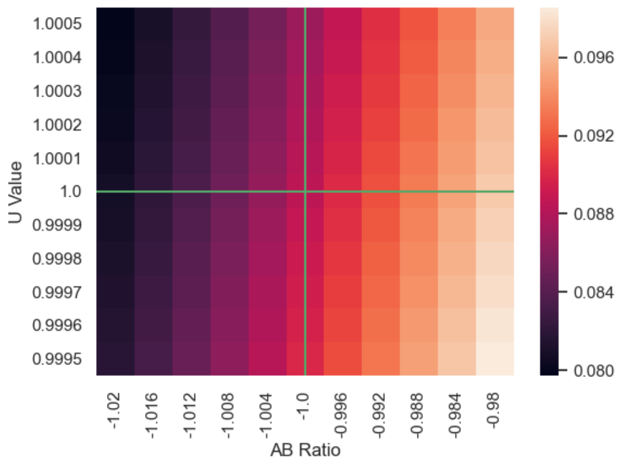
See Figure 6 and Figure 7.



(a) FPF



(b) MSE



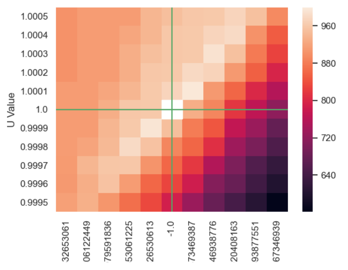
(c) BCE

Figure 6: Heatmaps showing the FPF values on the Dyck-1 Very Long dataset and MSE and BCE loss on the Dyck-1 Validation dataset for models with a correct configuration and with deviations. The thin green lines represent the CIC values for the a/b ratio and U value, and the intersection between the green lines is the point of a correct model. For FPF, the value in the centre of graph (a) is undefined, as no failures occurred for the correct model. It can be seen that the lowest MSE and BCE loss values are not located at the position of the correct model configurations.

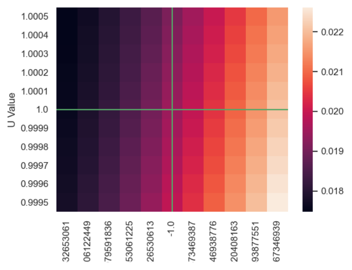
D Distribution of CICs in Models Trained from Random Initialisation

We inspect the 12 models successfully trained on Dyck-1 from random initialisation with MSE loss and extract the U value and a/b ratio. We verify that $a > 0$ and plot the distribution of the a/b ratio and U and Euclidean distance between the observed $[a/b, U]$ and the correct $[-1, 1]$ in Figure 8. The models do not reach the correct combination of values. The U value distribution has a clear peak at 1. The a/b ratio has a broader distribution with the mean not at -1, but slightly above.

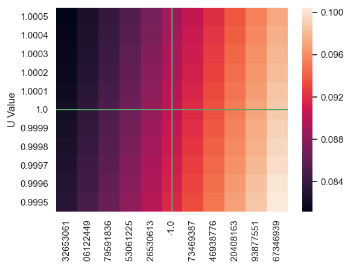
b=0.98



(a) FPF

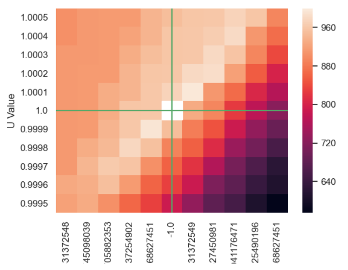


(b) MSE

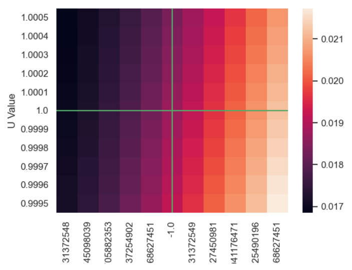


(c) BCE

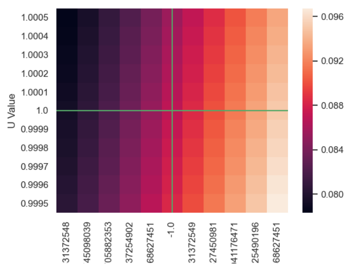
b=1.02



(d) FPF

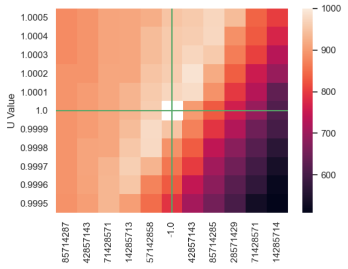


(e) MSE

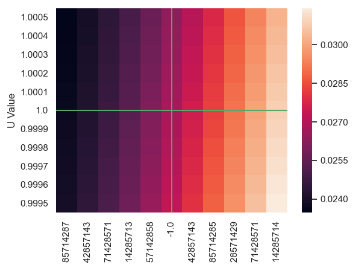


(f) BCE

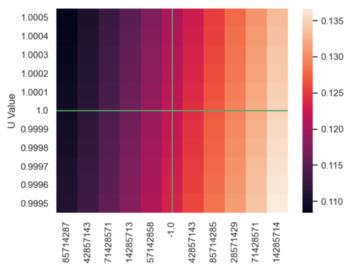
b=0.7



(g) FPF

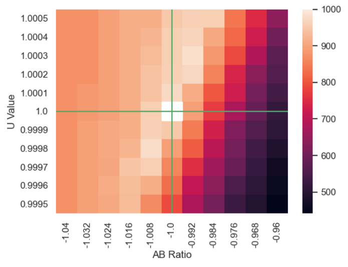


(h) MSE

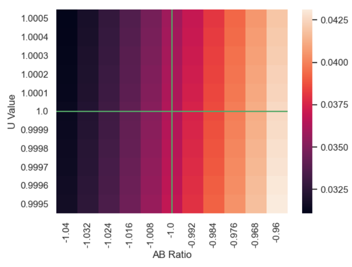


(i) BCE

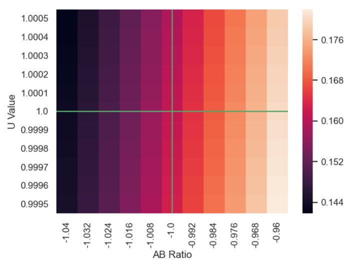
b=0.5



(j) FPF



(k) MSE

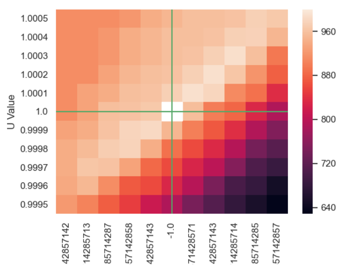


(l) BCE

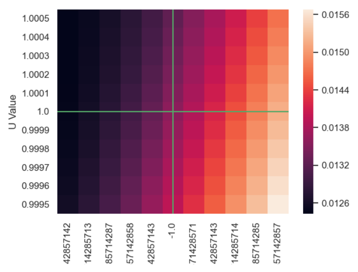
E Testing on Dyck-1 Strings for semi-Dyck-1 Acceptance

The Dyck-1 datasets contain only valid Dyck-1 strings with balanced opening and closing brackets. The prefix sequences $(x_0, \ldots, x_k$, where $k < n)$ are also evaluated, where we have excess opening brackets or balanced brackets. The case of excess closing brackets does not occur in these datasets. Dyck-1 acceptance (where the excess closing brackets are invalid) or semi-Dyck-1 acceptance (where the results of excess closing brackets are valid) are therefore not fully covered by these datasets. However, in the case of a single-cell ReLU RNN $\mathcal{R}$, if $b_{\mathcal{R}} \leq 0$ and $h_t = 0$ for a balanced bracket string s with t tokens, then for $s_{t+1} = \rangle$, we have $h_{t+1} = 0$ by Definition 6. If $\mathcal{R}$ is a semi-Dyck-1 counter for excess open and

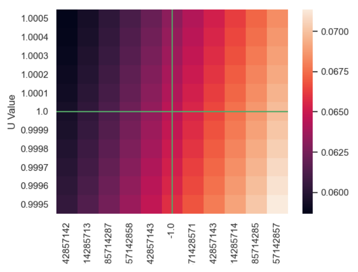
b=1.4



(m) FPF

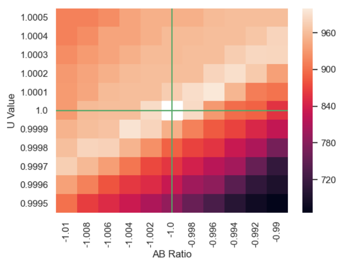


(n) MSE

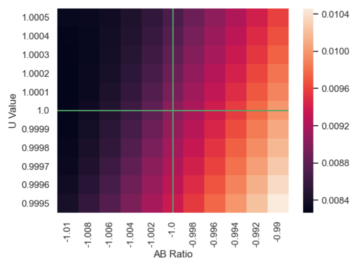


(o) BCE

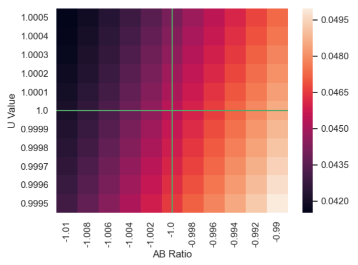
b=2



(p) FPF



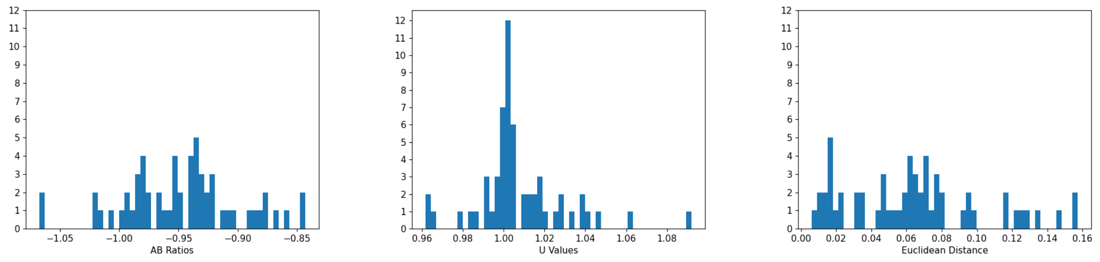
(q) MSE



(r) BCE

Figure 7: Heatmaps showing the FPF values on the Dyck-1 Very Long dataset and MSE and BCE loss on the Dyck-1 Validation dataset for models with a correct configuration and with deviations. The thin green lines represent the CIC values for the a/b ratio and U value, and the intersection between the green lines is the point of a correct model. It can be seen that the lowest MSE and BCE loss values are not located at the position of the correct model configurations.

for balanced bracket strings, it has to have $b \leq 0$, following the same argument as in the proof of Theorem 8. Therefore, it is sufficient to test excess opening and balanced bracket strings to test if a ReLU RNN accepts the semi-Dyck-1 language.



(a) $\boldsymbol{a}/\boldsymbol{b}$ Ratio (b) Recurrent Weight $\boldsymbol{U}$ (c) Euclidean Distance

Figure 8: Distributions of the CICs over the 12 converged M/RI Dyck-1 models.