

# Benchmarking Neural Network Generalization for Grammar Induction

Nur Lan<sup>1,2</sup>, Emmanuel Chemla<sup>1</sup>, Roni Katzir<sup>2</sup>

<sup>1</sup>Ecole Normale Supérieure

<sup>2</sup>Tel Aviv University

{nur.lan, emmanuel.chemla}@ens.psl.eu

rkatzir@tauex.tau.ac.il

## Abstract

How well do neural networks generalize? Even for grammar induction tasks, where the target generalization is fully known, previous works have left the question open, testing very limited ranges beyond the training set and using different success criteria. We provide a measure of neural network generalization based on fully specified formal languages. Given a model and a formal grammar, the method assigns a generalization score representing how well a model generalizes to unseen samples in inverse relation to the amount of data it was trained on. The benchmark includes languages such as  $a^n b^n$ ,  $a^n b^n c^n$ ,  $a^n b^m c^{n+m}$ , and Dyck-1 and 2. We evaluate selected architectures using the benchmark and find that networks trained with a Minimum Description Length objective (MDL) generalize better and using less data than networks trained using standard loss functions. The benchmark is available at <https://github.com/taucmpling/bliss>.

## 1 Introduction

The extent to which artificial neural networks (ANNs) generalize beyond their training data is an open research question. In this work we approach this question from the perspective of grammar induction, that is, the learning of a formal grammar from a finite (often small) sample from the (typically infinite) language of that grammar. In order to succeed in this task, a model must strike a balance between fitting the training data and generalizing to a potentially infinite set of unseen strings. Humans tested on such tasks show systematic generalization from small sets of examples (Fitch and Hauser, 2004, Malassis et al., 2020).

While a range of ANN architectures have been shown to reach approximations for formal languages, the quality of this approximation remains an open matter, as we show below. Here we build on previous probes of ANN generalization for grammar induction and introduce a unified and

general way to assess this capability, for a given pair of a learning model and a corpus drawn from a formal language. Our main contributions are:

1. **A benchmark for formal language learning.** The benchmark relies on a method for quantifying ANN generalization for formal languages, including probabilistic languages. The method assigns an index score representing a model’s generalization performance in inverse relation to the size of the training data. We introduce the method and provide concrete datasets for well-studied formal languages.
2. **An evaluation of selected architectures.** We test recurrent neural networks (RNNs) of the Long-Short Term Memory type (LSTM; Hochreiter and Schmidhuber, 1997); Memory-augmented RNNs (MARNN; Suzgun et al., 2019b;) and an RNN variant which replaces the traditional gradient-based training regime with an objective that optimizes the model’s Minimum Description Length (MDLRNN; Lan et al., 2022).

We find that equipping ANNs with memory devices such as differentiable stacks helps generalization, but generalization remains partial and slow. At the same time, training with MDL leads in some of the test cases that we examined to potentially perfect generalization with significantly less data. In other cases, training with MDL did not result in successful generalization, possibly because the optimization procedure we used for the architecture search failed to find the global optimum under the MDL objective function.

## 2 Background

Learning formal languages has long been used to probe various aspects of ANN performance. These most often include inquiries about: (i) ANNs’ ability to generalize beyond their training data, and

Language	Paper	Model	Metric	Training size	Max train $n$	Max test $n$
$a^n b^n$	GS'01	LSTM	$M_{cat'}$	16,000	30	1,000
	JM'15	Stack-RNN	$M_{det}$	20 <sup>†</sup>	19	60
	WGY'18	LSTM	$Bin$	100 <sup>†</sup>	100	256
	LGCK'22	MDLRNN	$M_{det}$	500	22	$\infty$
$a^n b^n c^n$	GS'01	LSTM	$M_{cat'}$	51,000	40	500
	JM'15	Stack-RNN	$M_{det}$	20 <sup>†</sup>	19	60
	WGY'18	LSTM	$Bin$	50 <sup>†</sup>	50	100
	LGCK'22	MDLRNN	$M_{det}$	500	22	$\infty$
Dyck-1	SGBS'19a	LSTM	$M_{cat'}$	10,000	50	100
	SGBS'19b	MARNN	$M_{cat'}$	5,000	50	100
	EMW'22	ReLU-RNN	$M_{cat'}$	10,000	50	1,000
	LGCK'22	MDLRNN	$M_{cat}$	500	16	$\infty$

Table 1: ANN performance in selected probes of formal language learning. **Metrics** (see Section 3.5):  $M_{det}$  = deterministic accuracy;  $M_{cat}$  = categorical accuracy;  $M_{cat'}$  = a non-probabilistic version of  $M_{cat}$ ;  $Bin$  = binary classification from hidden state to accept/reject labels, based on positive and negative samples. **Training size**: † = the paper did not explicitly specify the training set size, the value here is derived by assuming the training set was an exhaustive list of all strings up to ‘max train  $n$ ’. **‘Max test  $n$ ’**: the largest  $n$  for which the criterion was reached. For Dyck-1,  $n$  represents overall sequence length. ‘ $\infty$ ’ = the paper provides evidence that the network is correct for any  $n$ . When a paper reports several experiments as in GS'01, we take the best result based on the smallest training set. **Papers**: GS'01 = Gers and Schmidhuber (2001); JM'15 = Joulin and Mikolov (2015); WGY'18 = Weiss et al. (2018); SGBS'19a = Suzgun et al. (2019a); SGBS'19b = Suzgun et al. (2019b); EMW'22 = El-Naggar et al. (2022); LGCK'22 = Lan et al. (2022).

(ii) ANNs’ expressive power; i.e., whether they can represent the relevant target grammars (often probed with reference to the Chomsky hierarchy of formal languages, as in Delétang et al., 2022). Here we will focus on the generalization question. We will show how it might be related to another under-exploited line of inquiry regarding the training objective of ANNs.

A long line of theoretical work has probed the computational power of ANNs. Siegelmann and Sontag (1992) originally showed that RNNs with a sigmoid activation can emulate multiple-stack Turing machines under certain permissive conditions (infinite activation precision and unbounded running time). Since these conditions cannot be met in practice, another line of work probed the computational power of RNNs under practical conditions (finite precision and input-bound running time). Weiss et al. (2018) have shown that under these conditions LSTMs are able to hold weight configurations that perform unbounded counting, and so they should be able to recognize counter languages (CL), a family of formal languages that can be recognized using one or more counting devices (following some formal restrictions, Merrill, 2021). Recently, El-Naggar et al. (2023a) and El-Naggar et al. (2023b) have shown that two simpler RNN ar-

chitectures, with linear- and ReLU-based cells, are also able to hold counting weight configurations, with similar consequences for recognizing CL.

Empirically, another line of work provided promising results regarding the capability of ANNs to learn formal languages. This was most often done by training networks on strings up to a certain length and then showing good performance on longer ones (Bodén and Wiles, 2000, Gers and Schmidhuber, 2001; see Table 1). Gers and Schmidhuber (2001) have shown that LSTMs trained on languages such as  $a^n b^n$  and  $a^n b^n c^n$  with  $n$  values in the low dozens perform well on  $n$ ’s in the high hundreds. Suzgun et al. (2019a) found that LSTMs trained on Dyck-1 sequences (strings of well-balanced pairs of brackets) up to length 50 performed well on lengths up to 100. Suzgun et al. (2019b) proposed RNN variants that are equipped with external differentiable memory devices and showed that they yield improved performance on non-regular languages.

However, other empirical results show that in practice ANNs generalize only to very restricted ranges. Weiss et al. (2018) found that while LSTMs are theoretically able to hold counting solutions, these are not found through training: LSTMs trained on  $a^n b^n$  and  $a^n b^n c^n$  with max  $n$  100 and

50, respectively, start accepting illicit strings with  $n$  values as low as 256 and 100. As mentioned above, [Suzgun et al. \(2019a\)](#) tested LSTMs on Dyck-1 sequences but only up to length 100, and concluded that this language was learned by LSTMs. [El-Naggar et al. \(2022\)](#) extended this work to longer sequences, and found that LSTMs fail to generalize in practice, outputting incorrect predictions at lengths under 1,000. This, despite Dyck-1 being a CL and so theoretically learnable by LSTMs ([Weiss et al., 2018](#)).

Apart from LSTMs, recent probes by [El-Naggar et al. \(2023a\)](#) and [El-Naggar et al. \(2023b\)](#) have shown that linear and ReLU RNNs, theoretically capable of counting, fail to find the counting weight configurations in practice when trained using back-propagation and standard loss functions; [El-Naggar et al. \(2023b\)](#) went further with determining the source of this discrepancy, showing that the counting weight configuration is not an optimum of these loss functions.

Moreover, even in works that report successful generalization to some degree beyond the training set, the fact that networks stop generalizing at an arbitrary point is often left unexplained ([Gers and Schmidhuber, 2001](#), [Suzgun et al., 2019a, 2019b](#), [Delétang et al., 2022](#), a.o.).<sup>1</sup>

The literature on the generalization abilities of ANNs has made use of a range of measures of success, making results difficult to compare. Different probes of the same model often use different success criteria, and generate training and test sets using different sampling methods and of different orders of magnitude. [Table 1](#) summarizes selected probes of ANN generalization and highlights the fragmented nature of this literature. In the following sections we propose a unified method to consolidate these efforts and better understand the generalization capabilities of ANNs.

### 3 The BLISS index

We present the Benchmark for Language Induction from Small Sets (BLISS). We provide a formal description of the method, followed by a concrete application to specific tasks.

<sup>1</sup>Technical limitations such as finite activation precision can be ruled out as explanations for generalization failures, at least for counter languages and models where network states serve as memory: as shown in works mentioned above, ANNs often start outputting wrong predictions for  $n$  values in the low hundreds. Even restricted representations such as 16-bit floats can hold much larger values, and modern implementations such as PyTorch use 32-bit floats by default.

The current release consists of three parts: (i) A specification for the generalization index  $\mathbb{B}$ , calculated for a given pair of formal language and ANN; (ii) A dataset containing a set of formal languages for benchmarking; (iii) An evaluation of different ANN architectures using this dataset.

#### 3.1 General setting: models and tasks

For a given model  $A$ , e.g., an LSTM, a task is composed of the following components:

- $G$  – a grammar, e.g., a probabilistic context-free grammar (PCFG).
- $S$  – a sampling method from  $\mathcal{L}(G)$ , the language generated by  $G$ .
- $\mathcal{C} = S(G)$  – a training corpus, may contain repetitions.
- $\mathcal{T} \subseteq \mathcal{L}(G) \setminus \mathcal{C}$  – a test corpus.
- $M$  – a task-specific accuracy metric with adjustable error margin  $\varepsilon \in [0, 1]$ . It uses predictions  $A(s)$  on strings  $s \in \mathcal{T}$  to calculate an accuracy score  $M(A, \mathcal{T}, \varepsilon) \in [0, 1]$ .
- $N$  – a task-specific constant for setting the order of magnitude of dataset sizes. For example,  $N = 3$  sets the order of magnitude at  $10^3$ . Training and test sizes are then derived as described below. Selecting  $N$  is done empirically based on properties of the task, e.g., languages with large vocabularies require larger amounts of training data, hence a larger  $N$ .

#### 3.2 From task to generalization index

For a given task, the generalization index of order  $N$  for a model  $A$  is then defined as:

$$\mathbb{B}_N^{\mathcal{L}}(A) = \max \left\{ b \mid \begin{array}{l} |\mathcal{T}| = 10^N \times b, \\ |\mathcal{C}| = 10^N / b, \\ M(A, \mathcal{T}, \varepsilon) = 1.0 \end{array} \right\} \quad (1)$$

Intuitively, the index compares a model’s performance on a test size  $|\mathcal{T}|$  to the inverse of its training data size  $|\mathcal{C}|$ .

The index is expressed as the maximal  $b$  factor which scales the training set and the corresponding test set in opposite directions: The accuracy condition at the bottom of (1) means that the model should be  $\varepsilon$ -close to perfect generalization on the test set. A model’s generalization index  $\mathbb{B}$  thus represents the performance that can be maximally ‘squeezed out’ of an inversely small amount of data.

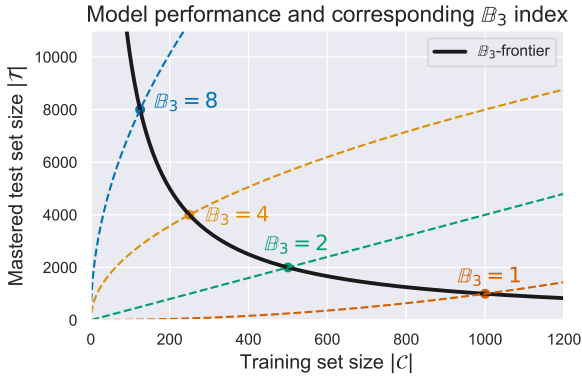


Figure 1: Example generalization index scores  $\mathbb{B}_3$ , i.e., for a baseline training size of  $10^3$ . Each dashed line represents the performance profile of some hypothetical model, as a function of the size of the training set. The intersection with the  $\mathbb{B}_3$ -frontier indicates its  $\mathbb{B}_3$  index.

Figure 1 exemplifies selected  $\mathbb{B}$  values calculated based on (1). For illustration, for  $a^n b^n$ , using the order of magnitude  $N = 3$ , a model that was trained on  $|C| = 10^3/2 = 500$  samples and was 100% accurate on a test set of size  $10^3 \times 2 = 2,000$  will have an index score  $\mathbb{B}_3^{a^n b^n} \geq 2$ . A model for the same language that was trained on 250 samples only and generalized to a subsequent set of 4,000 samples will reach  $\mathbb{B}_3^{a^n b^n} \geq 4$ .

For practical reasons, one cannot exhaust all values of  $b$  to find  $\mathbb{B}$ . However, training and evaluating a model using a few  $b$  values is enough to reveal its generalization dynamics, as shown in experiments in Section 5. The following sections describe the specific choices made for the different benchmark components in these experiments.

### 3.3 Learning setup

Previous work surveyed here differed in their learning setup. Gers and Schmidhuber (2001) and Suzgun et al. (2019a, 2019b) trained networks in a non-probabilistic, supervised setup by exposing the model to all possible next symbols and minimizing the mean-squared error (MSE) – i.e., the model is given explicit information about the distribution of possible symbols. Joulin and Mikolov (2015) and Lan et al. (2022) used a setup that we adopt below, in which model outputs are probabilistic, and training is self-supervised language modeling (i.e., the model is exposed to the next symbol only) with a cross-entropy loss. Weiss et al. (2018) trained a binary classifier with accept/reject labels based on positive and negative examples.

Since our focus is grammar induction, here we

adopt the more demanding setup of learning from positive examples alone. All tasks are thus designed as self-supervised language modeling. At each time step, a model assigns a probability distribution to the next symbols in the string.

The benchmark is agnostic as to the internals of the model and its training, as long as its outputs represent a probability distribution over symbols. In practice, then, the method can be applied to any language model, not necessarily an ANN.

### 3.4 Sampling

To construct the training and test sets  $\mathcal{C}$  and  $\mathcal{T}$  we use the following as method  $S$ :

- To construct  $\mathcal{C}$ , we sample strings according to the distribution defined by  $G$ , with repetitions. For example, if  $G$  is a PCFG, it can be sampled by applying derivation rules chosen proportionally to their expansion probabilities. Repetitions are allowed so that  $\mathcal{C}$  follows a similar surface distribution to  $\mathcal{L}(G)$  and so that the model can pick up on the underlying probabilities in  $G$ .
- To construct  $\mathcal{T}$ , we take the  $|\mathcal{T}|$  subsequent strings starting right after the longest string in  $\mathcal{T}$ , sorted by length.<sup>2</sup> For example, for the language  $a^n b^n$ , if the longest string in the training set  $\mathcal{C}$  was  $a^{17} b^{17}$ , and the model needs to be tested on a set of 2000 strings,  $\mathcal{T}$  will be composed of the strings  $a^{18} b^{18}, \dots, a^{2017} b^{2017}$ .

The sampling method  $S$  can be either probabilistic as described here, or exhaustive, training on all strings in  $\mathcal{L}$  up to a certain length. We opt for probabilistic sampling because of the nature of the task at hand: the models under discussion here are trained to assign probabilities to the next symbol in a string, most often minimizing a cross-entropy loss. In practice, then, they always learn distributions over strings. Thus if  $\mathcal{C}$  follows a similar surface distribution to  $\mathcal{L}$  (given a large enough sample size), the model should eventually learn this distribution in order to minimize its loss.

Probabilistic sampling thus makes it possible to probe both a model’s knowledge about the surface forms of  $\mathcal{L}$  (by treating model outputs as categorical classes), and about their distribution. The modularity of the index makes it possible to choose

<sup>2</sup>Test strings may need to be sorted further according to specific properties of a language, see Section 4.1.

either option by varying the accuracy metric  $M$ , as we show in the next section.

### 3.5 Accuracy metrics

Ultimately we are interested in knowing whether a model accepts all strings in  $\mathcal{L}$  and rejects all others. In classical formal language theory, where discrete automata are used, acceptance is clear cut and taken as going into an accepting state. ANNs on the other hand use continuous representations with no standard acceptance criterion.

Different acceptance criteria have been used in previous works to measure success for ANNs: Gers and Schmidhuber (2001) and Suzgun et al. (2019b) defined acceptance of a string as a model assigning output values above a certain threshold to valid symbols only; Joulin and Mikolov (2015) measure accuracy at parts of strings that are completely predictable; and Weiss et al. (2018) turn a network into a recognizer by training a binary classifier from network states to accept/reject labels. Below we provide general versions of these accuracy metrics (omitting Weiss et al., 2018 who rely on negative examples).

Choosing which metric to use is based on the properties of the language at hand. Well-performing models might still deviate slightly from perfect accuracy due to practical limitations, such as a softmax function preventing a model from expressing categorical decisions. Thus for each

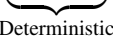
Input:	#	(	(	)	(	)	)
	↓	↓	↓	↓	↓	↓	↓
Target:	#/(	(/)	(/)	(/)	(/)	(/)	#/(
Input:	#	a	a	a	b	b	b
	↓	↓	↓	↓	↓	↓	↓
Target:	#/a	a/b	a/b	a/b	<b>b</b>	<b>b</b>	#
							

Figure 2: Inputs and valid next symbols at each step of a Dyck-1 string (top) and  $a^n b^n$  (bottom), including the start/end-of-sequence symbol ‘#’. For  $a^n b^n$ , accuracy is measured at deterministic steps, after the first ‘b’. For Dyck-1, accuracy is the fraction of time steps where a model predicts only valid next symbols: ‘#’ should be predicted only when brackets are well balanced.

accuracy metric we add an adjustable error margin  $\varepsilon$ . Acceptance of a string is defined as reaching 100% accuracy (minus  $\varepsilon$ ) on the string. Success on the test set is then defined as accepting all strings in the set (third condition in (1)).

1. *Deterministic accuracy* ( $M_{det}$ ). Some languages contain strings with deterministic phases, where the next symbol is fully predictable. For example, strings in the language  $a^n b^n$  have two phases, the  $a$  phase and the  $b$  phase. As long as only  $a$ ’s are seen, the next symbol remains unpredictable as the sequence can continue with another  $a$  or switch to the  $b$  phase. The string becomes deterministic once the first  $b$  appears.  $M_{det}$  is defined as the fraction of deterministic time steps in which the model assigns the majority probability to the correct next symbol. This metric is used in Joulin and Mikolov (2015).

A string is considered accepted if the model is  $1 - \varepsilon$  accurate over all deterministic time steps. Note however that even a very small  $\varepsilon$  might benefit models that do not recognize strings well. For example, for the language  $a^n b^n$ , the deterministic steps in a string are the  $b$ ’s and the final end-of-sequence symbol. A degenerate model that predicts only  $b$ ’s will get only the end-of-sequence symbol wrong out of all deterministic steps, and will reach a very high accuracy score. For any large enough test set these errors will be hidden within the  $\varepsilon$  margin and the model will be deemed successful.  $\varepsilon$  should therefore be chosen with care per task.

$M_{det}$  is used below for the following languages that have deterministic phases:  $a^n b^n$ ,  $a^n b^n c^n$ ,  $a^n b^n c^n d^n$ , and  $a^n b^m c^{n+m}$ .

2. *Categorical accuracy* ( $M_{cat}$ ). Some language strings do not have any predictable phases. This is the case in the Dyck family of languages. At each time step in a Dyck string, one may open a new bracket (see Figure 2).  $M_{cat}$  is therefore defined as the fraction of steps in which a network assigns probability  $p > \varepsilon$  to each possible next symbol, and  $p \leq \varepsilon$  to irrelevant symbols. Non-probabilistic versions of  $M_{cat}$  are used in Gers and Schmidhuber (2001) and Suzgun et al. (2019a, 2019b) who do not treat network outputs as probability distributions.  $M_{cat}$  is used below for Dyck languages.

As specified in Section 3.1, the index  $\mathbb{B}$  is calculated based on the largest test set for which a model reaches an  $\varepsilon$ -perfect accuracy score.

Beyond accuracy, one might be interested in inspecting a model’s knowledge of the distribution of strings in  $\mathcal{L}$  induced by a probabilistic  $G$ . This can be done by using the probabilistic sampling method described in Section 3.4 and accompanying it with a probabilistic accuracy measure – for example, one based on an optimal cross-entropy score, which is known from  $G$ ’s expansion probabilities (as done in Lan et al., 2022). Feeding loss values into an accuracy metric will require normalizing them across tasks. We leave this extension for future work.

### 3.6 String structure

Following Gers and Schmidhuber (2001), each sequence starts and ends with a start/end-of-sequence symbol ‘#’. This turns the task into a strict acceptance/rejection task – predicting the end-of-sequence symbol is taken as going into an accept state. The start- and end-of-sequence symbols are added to the task-specific vocabulary and are assigned probabilities by the model at each step. Figure 2 illustrates input and target sequences for  $a^n b^n$  and Dyck-1.

### 3.7 Limitations

One shortcoming of the proposed index score is that it does not reflect perfect generalization, i.e., it is an empirical index that cannot point out a model that outputs correct predictions for *any* string in  $\mathcal{L}(G)$ . For most models, this will not be a problem, and  $\mathbb{B}$  will simply represent the model’s best training vs. test size ratio. In the case of a model that reaches perfect generalization on any input, the index score will represent the critical training size that brings the model to this performance.

Assigning a generalization score to infinitely correct models will remain a problem for any empirical metric that assigns scores to models based on finite test values. An alternative to such empirical probes would be to analytically show that a model is correct (as done in Lan et al., 2022).

## 4 Datasets

We provide training and test datasets for a preliminary set of formal languages for evaluation using the  $\mathbb{B}$  index. The dataset includes the languages  $a^n b^n$ ,  $a^n b^n c^n$ ,  $a^n b^n c^n d^n$ ,  $a^n b^m c^{n+m}$ ,

Dyck-1, and Dyck-2. The source code, datasets, and specifications for the benchmark are available at <https://github.com/taucompling/bliss>.

### 4.1 Training and test sets

Training sets for context-free languages are sampled from PCFGs as described in Section 3.4. The PCFGs are given in Appendix B. Training sets for context-sensitive languages are generated by sampling values for  $n$  from a geometric distribution.

Test sets are generated using the method described in Section 3.4: All test sets consist of an exhaustive list of strings ordered by length starting right after the longest string seen during training. Test sets for  $a^n b^m c^{n+m}$  consist of the list of strings starting after the last seen pair of  $n, m$ , sorted by  $n + m$  values to test all possible combinations.

## 5 Experiments

### 5.1 Models

We test the following models: LSTM RNNs (Hochreiter and Schmidhuber, 1997); Memory-augmented RNNs (MARNN; Suzgun et al., 2019a); and Minimum Description Length RNNs (MDL-RNN; Lan et al., 2022).

LSTM architectures were developed with the task of keeping items in memory over long distances in mind. As mentioned above, Weiss et al. (2018) have shown that LSTMs are theoretically capable of recognizing CL.

MARNNs (Suzgun et al., 2019b) are RNNs equipped with external memory devices, and were shown to yield better performance when learning languages that require stack-like devices and beyond. Here we use Stack-LSTM, an LSTM augmented with a pushdown automaton; and Baby Neural Turing Machines (Baby-NTM; itself a variant of NTMs, Graves et al., 2014), an RNN with a more freely manipulable memory.<sup>3</sup>

MDLRNNs are RNNs trained to optimize the Minimum Description Length objective (MDL; Rissanen, 1978), a computable approximation of Kolmogorov complexity, the algorithmic complexity of a model. The intuition behind the objective is equating compression with finding regularities in the data: a model that compresses the data well will generalize better and avoid overfitting. In practice, optimization is done by minimizing the sum of

<sup>3</sup>We modify Suzgun et al. (2019a)’s models to output probability distributions, replacing the final sigmoids with a softmax layer and the MSE loss with cross-entropy. See Section 3.3.

the architecture encoding length and the standard cross-entropy loss, both measured in bits based on a specific encoding scheme.

MDL is a stricter regularizer than standard regularization techniques such as L1/L2: the latter penalize large weight values but cannot prevent models from overfitting using a solution that uses many small, but informative, weights. MDL penalizes the actual information content of the network, forcing it to be general and avoid overfitting. MDLRNNs were shown to learn some of the languages discussed here in full generality using small architectures of only 1 or 2 hidden units and to outperform L1/L2 (Lan et al., 2022).

MDL is a non-differentiable objective, which requires that MDLRNN be optimized using a non-gradient based search method, such as an evolutionary algorithm that searches the network architecture space. Since this method is not based on gradient descent, Lan et al. (2022) were able to use non-standard, non-differentiable activations such as step functions. Here we restrict the architecture space to only standard activations: the linear function, ReLU, and tanh. This serves both to compare MDLRNN with standard networks and to limit the architecture search space. We publish the resulting nets as part of the MDLRNN-Torch release at <https://github.com/0xnurl/mdlrnn-torch>.

Appendix A lists the hyper-params for all runs.

## 5.2 Training sets

We used training sizes  $|\mathcal{C}| = 100, 250, 500, 1000$ . We stopped at the smallest size 100 because in our setup this size results in test strings of lengths  $> 10,000$ , leading to very long running times.

## 5.3 Index parameters

We calculate the  $\mathbb{B}$  index for all trained networks using the following index parameters:

Magnitude parameter  $N = 3$ , i.e., training and test sizes are derived from a baseline size  $10^3$ . This order of magnitude was selected based on the training set sizes used in previous works for the languages inspected here (Table 1).

$M_{det} \varepsilon = 0.005$ , i.e., a model needs to correctly predict the next symbol for at least 99.5% of all deterministic steps. Since even this high threshold allows a degenerate model to reach good scores as described in Section 3.5, we also calculate the index score using  $\varepsilon = 0$ , i.e. a model must predict *all* deterministic symbols correctly.

Language	Model	$\mathbb{B}$ -index	
		$\varepsilon = 0.005$	$\varepsilon = 0$
$a^n b^n$	LSTM	<b>10</b>	<1
	Stack-LSTM	<b>10</b>	<1
	Baby-NTM	<b>10</b>	1
	MDLRNN	<b>10</b>	<b>10</b>
$a^n b^n c^n$	LSTM	<1	<1
	Stack-LSTM	2	<1
	Baby-NTM	<b>10</b>	<1
	MDLRNN	<1	<1
$a^n b^n c^n d^n$	LSTM	<1	<1
	Stack-LSTM	1	<1
	Baby-NTM	<b>4</b>	<1
	MDLRNN	<1	<1
$a^n b^m c^{n+m}$	LSTM	<1	<1
	Stack-LSTM	<b>10</b>	<1
	Baby-NTM	4	<1
	MDLRNN	4	<b>4</b>
Dyck-1	LSTM	<1	<1
	Stack-LSTM	<1	<1
	Baby-NTM	<1	<1
	MDLRNN	<b>2</b>	<b>2</b>
Dyck-2	LSTM	<1	<1
	Stack-LSTM	<1	<1
	Baby-NTM	<1	<1
	MDLRNN	<1	<1

Table 2: Generalization scores  $\mathbb{B}$ . The index represents how well a model generalizes in relation to its training size. A score  $\mathbb{B} = 4$  indicates that a model trained on 250 samples reached the accuracy criterion on the consecutive 4,000 unseen test samples.  $\mathbb{B} < 1$  indicates that the model did not reach the accuracy criterion when the test size was greater than the training size, but might reach it for larger training and smaller test sets.

$M_{cat} \varepsilon = 0.005$ , i.e., for Dyck, a model needs to assign  $p \leq 0.005$  to each irrelevant symbol and  $p > 0.005$  to possible ones. Here as well we report results for  $\varepsilon = 0$ , i.e., a model must assign non-zero probabilities to valid symbols only.

## 6 Results

### 6.1 Non-perfect accuracy

The generalization index obtained by each model for each language is presented in Table 2.

We start by inspecting the indexes calculated using the more lenient accuracy margin  $\varepsilon = 0.005$ .

For  $a^n b^n$ , under this accuracy margin, all models are assigned index  $\mathbb{B} = 10$ , i.e., reaching the success criterion for the next unseen 10,000 samples

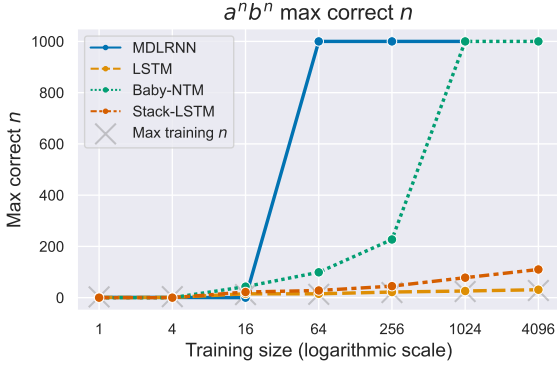


Figure 3: Generalization performance of the models tested here. Models were trained on strings drawn from  $a^n b^n$  and tested on acceptance of strings up to  $n = 1,000$ . X's mark the maximum  $n$  seen during training.

after being trained on 100 samples. For the specific combination of random seed and sampling prior in these experiments, this means that the models were trained on strings up to  $a^{20}b^{20}$  and generalized to all strings up to at least  $a^{10020}b^{10020}$  with deterministic accuracy  $M_{det} \geq 99.5\%$ .

For  $a^n b^n c^n$ , MARNNs reach  $\mathbb{B} = 10$  and 2, while LSTM and MDLRNNs do not reach the success criterion, resulting in  $\mathbb{B} < 1$ . For  $a^n b^n c^n d^n$  only MARNNs reach a specified index, with a Baby-NTM reaching  $\mathbb{B} = 4$ , indicating that it generalized to strings as long as  $a^{4020}b^{4020}c^{4020}d^{4020}$  with  $M_{det} \geq 99.5\%$ .

For the addition language  $a^n b^m c^{n+m}$ , Stack-LSTM and MDLRNN reached index scores  $\mathbb{B} = 10$  and 4 respectively. For the specific combination of random seed and the sampling prior used here, this means that the winning Stack-LSTM saw maximum values of  $n = 18, m = 20$  during training, and generalized to all strings up to  $a^{120}b^{120}c^{240}$  with  $M_{det} \geq 99.5\%$ .

## 6.2 Perfect accuracy

We report the generalization scores using a strict  $\varepsilon = 0$  as well, i.e., when a model is required to predict *all* deterministic steps correctly or assign non-zero probability to valid symbols only. For languages with deterministic steps such as  $a^n b^n$ , this means that the model needs to always predict the end-of-sequence symbol correctly, thus making a distinction between accepting a string and approximating its surface structure.

Here, only MDLRNNs remain at the same scores, indicating that they predicted all time steps correctly. Baby-NTM reaches  $\mathbb{B} = 1$  for  $a^n b^n$ , a

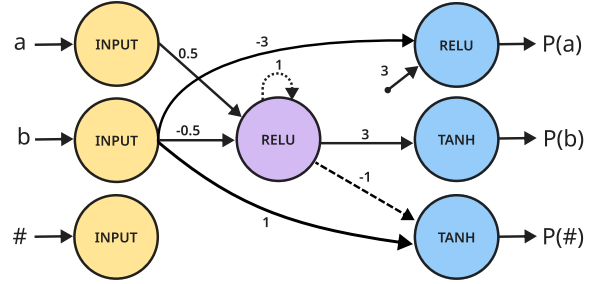


Figure 4: RNN cell architecture of the best-performing MDLRNN for  $a^n b^n$ , which trained on 100 samples and reached  $\mathbb{B}_3 = 10$ . The network uses only one hidden unit and standard activation functions, and generalizes up to at least  $a^{35000}b^{35000}$ . Dashed arrows are recurrent connections across time steps. The loop from the hidden ReLU unit to itself is a counter mechanism evolved by the evolutionary algorithm to count and compare the number of  $a$ 's and  $b$ 's.

drop from 10. The rest of the networks drop to  $\mathbb{B} < 1$ , revealing that their good scores in the previous comparison calculated with a non-zero  $\varepsilon$  was due to them approximating the target languages, even at low  $n$  values.

MDLRNN performance here is in line with results from Lan et al. (2022), who provided evidence that MDLRNNs for these languages do not only perform empirically well on large test values, but are also provably correct for any input. However, here we limited activations to standard, non-discrete functions (Section 5.1), potentially limiting the network's ability to generalize well in the limit. While we do not provide correctness proofs for the networks found here, the index scores indicate that MDLRNNs generalize well to large values using only standard activations. Figure 4 presents the MDLRNN found for  $a^n b^n$ . We checked whether this network also accepts  $n$  values beyond those needed to reach the score  $\mathbb{B} = 10$  ( $n = 10,020$ ). The network reached 100%  $M_{det}$  for all values up to  $n = 35,000$ , at which point we stopped the test due to long feeding times.

Beyond the benchmark scores, Figure 3 plots the largest  $n$  value for  $a^n b^n$  strings predicted by the models tested here with 100%  $M_{det}$  accuracy ( $\varepsilon = 0$ ), as a function of training set size. Both MDLRNNs and Baby-NTMs reach perfect accuracy up to the tested maximum of  $n = 1,000$ . MDLRNNs however require two orders of magnitude less data to reach this performance (and the benchmark scores in Table 2 show that in fact MDLRNNs generalized up to at least  $n = 10,000$ , while Baby-NTMs remained at 1,000). LSTMs



and Stack-RNNs did not generalize well beyond the training samples. This is in line with previous works showing that these models may need substantially more training data in order to learn these languages (Table 1).

## 7 Discussion

We provided a simple index for how well a model generalizes: how much it can learn from how little data. We illustrated the usefulness of this index in a comparison of several current models over several formal languages. Beyond showing which current models generalize better than others, the benchmark also highlights which aspects of artificial neural networks work well for grammar induction, and what is still missing.

Among languages that were learned with perfect accuracy ( $a^n b^n$ ,  $a^n b^m c^{n+m}$ , Dyck-1), MDLRNNs generalized best, but still failed on others ( $a^n b^n c^n$ ,  $a^n b^n c^n d^n$ , and Dyck-2). Previous work has shown that this model’s search procedure, an evolutionary algorithm, fails to find networks that are manually shown to have better MDL scores (Lan et al., 2022). We take this to show that the optimization procedure limits the model and prevents it from taking full advantage of the MDL objective. The benefit of the MDL objective is nevertheless evident in the generalization performance for several languages.

MARNNs clearly benefit from their memory devices and reach good generalization scores, but testing for perfect accuracy ( $\varepsilon = 0$ ) reveals that their learning outcome is mostly approximate, and that they fail to maintain perfect accuracy for long stretches beyond their training data. This could be the result of an inadequate objective function (cross-entropy), limitations of the search (backpropagation/gradient descent), or both. We do not currently have results that help decide this matter, but recent results for other architectures (El-Naggar et al., 2023b) hint that the problem lies at least in part in the objective function.

## 8 Acknowledgements

This work was granted access to the HPC resources of IDRIS under the allocation 2023-AD011013783 made by GENCI.

## References

Mikael Bodén and Janet Wiles. 2000. Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science*, 12(3-4):197–210.

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. 2022. [Neural Networks and the Chomsky Hierarchy](#).

Nadine El-Naggar, Pranava Madhyastha, and Tillman Weyde. 2022. [Exploring the Long-Term Generalization of Counting Behavior in RNNs](#).

Nadine El-Naggar, Pranava Madhyastha, and Tillman Weyde. 2023a. [Theoretical Conditions and Empirical Failure of Bracket Counting on Long Sequences with Linear Recurrent Networks](#).

Nadine El-Naggar, Andrew Ryzhikov, Laure Daviaud, Pranava Madhyastha, and Tillman Weyde. 2023b. Formal and empirical studies of counting behaviour in ReLU RNNs. In *Proceedings of 16th Edition of the International Conference on Grammatical Inference*, volume 217 of *Proceedings of Machine Learning Research*, pages 199–222. PMLR.

W. Tecumseh Fitch and Marc D. Hauser. 2004. Computational constraints on syntactic processing in a nonhuman primate. *Science*, 303(5656):377–380.

Felix Gers and Jürgen Schmidhuber. 2001. [LSTM recurrent networks learn simple context-free and context-sensitive languages](#). *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. [Neural Turing Machines](#). *arXiv:1410.5401 [cs]*.

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

Alon Jacovi, Avi Caciularu, Omer Goldman, and Yoav Goldberg. 2023. [Stop Uploading Test Data in Plain Text: Practical Strategies for Mitigating Data Contamination by Evaluation Benchmarks](#).

Armand Joulin and Tomas Mikolov. 2015. Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Diederik P. Kingma and Jimmy Ba. 2017. [Adam: A Method for Stochastic Optimization](#).

Nur Lan, Michal Geyer, Emmanuel Chemla, and Roni Katzir. 2022. [Minimum Description Length Recurrent Neural Networks](#). *Transactions of the Association for Computational Linguistics*, 10:785–799.

Raphaëlle Malassis, Stanislas Dehaene, and Joël Fagot. 2020. [Baboons \(Papio papio\) Process a Context-Free but Not a Context-Sensitive Grammar](#). *Scientific Reports*, 10(1):7381.

William Merrill. 2021. [On the Linguistic Capacity of Real-Time Counter Automata](#).

J. Rissanen. 1978. [Modeling by shortest data description](#). *Automatica*, 14(5):465–471.

Hava T. Siegelmann and Eduardo D. Sontag. 1992. [On the computational power of neural nets](#). In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92*, pages 440–449, New York, NY, USA. Association for Computing Machinery.

Mirac Suzgun, Yonatan Belinkov, Stuart Shieber, and Sebastian Gehrmann. 2019a. [LSTM Networks Can Perform Dynamic Counting](#). In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 44–54, Florence. Association for Computational Linguistics.

Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M. Shieber. 2019b. [Memory-Augmented Recurrent Neural Networks Can Learn Generalized Dyck Languages](#). *arXiv:1911.03329 [cs]*.

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: Hyper-parameters

### A.1 Training corpora

All training sets were generated using the same random seed 100 and prior probability  $p = 0.3$ . The datasets are available at <https://github.com/taucoplmling/bliss>. Following [Jacovi et al. \(2023\)](#), the datasets are zipped and password-protected to prevent test data contamination of large language models through crawling.

Each of the LSTM and MARNN hyper-param configurations below was run 3 times using different random seeds (100, 101, 102). MDLRRNs were run once per configuration because of their longer running time.

### A.2 LSTM

LSTMs were trained based on the following hyper-params grid: hidden state size (2/32/128), regularization technique (L1/L2/none), and the regularization constant in case regularization was applied ( $\lambda = 1.0/0.1/0.01$ ). Networks were trained using the Adam optimizer ([Kingma and Ba, 2017](#)) with learning rate 0.001,  $\beta_1 = 0.9$ , and  $\beta_2 = 0.999$ . The networks were trained by feeding the full batch of training data for 1,000 epochs.

### A.3 MARNN

MARNNs were trained by varying the architecture type (Softmax Stack-LSTM/Softmax Baby-NTM) and stack/memory size (50/100 for Stack-LSTM, 2050 for Baby-NTM). For Stack-LSTM,

stack sizes were selected so they were always larger than the largest values seen during training:  $n + m = 22 + 24$  for  $a^n b^m c^{n+m}$  and  $n = 24$  for all other languages. During testing the stack size was enlarged to 2050, beyond the maximum needed to reach scores  $\mathbb{B} = 1$  and 2. Baby-NTM memory was set to 2050 already during training because this model’s memory size affects the weight dimensions and cannot be changed after training.

The rest of the hyper-parameters were set to the default values from [Suzgun et al. \(2019b\)](#). Stack-LSTM: hidden size 8; 1 layer; memory dimension 5; epochs 3/50; learning rate 0.01; Baby-NTM: hidden size 8; 1 layer; memory dimension 5; epochs 3/50; learning rate 0.01.

The original MARNN setup was modified here so that the network outputs represent probability distributions and not multi-label outputs. This was done by replacing the sigmoid outputs with a softmax layer and the MSE loss with cross-entropy.

### A.4 MDLRRN

MDLRRNs were trained using the evolutionary algorithm and the same hyper-params reported in [Lan et al. \(2022\)](#): population size 500; islands size 250; 25,000 generations; tournament size 2; early stop after 2 hours of no improvement; elite ratio 0.001; migration interval 1,000 generations/30 minutes.

## B Appendix: PCFGs

### B.1 $a^n b^n$

$$S \rightarrow \begin{cases} aSb & 1 - p \\ \varepsilon & p \end{cases}$$

### B.2 $a^n b^m c^{n+m}$

$$S \rightarrow \begin{cases} aSc & 1 - p \\ X & p \end{cases}$$

$$X \rightarrow \begin{cases} bXc & 1 - p \\ \varepsilon & p \end{cases}$$

### B.3 Dyck-1

$$S \rightarrow \begin{cases} (S)S & p \\ \varepsilon & 1 - p \end{cases}$$

### B.4 Dyck-2

$$S \rightarrow \begin{cases} (S)S & p/2 \\ [S]S & p/2 \\ \varepsilon & 1 - p \end{cases}$$