MLRegTest: A Benchmark for the Machine Learning of Regular Languages

Sam van der Poel   SAMVANDERPOEL@GATECH.EDU
School of Mathematics
Georgia Institute of Technology

Dakotah Lambert   DAKOTAHLAMBERT@ACM.ORG
Laboratoire Hubert Curien
Université Jean Monnet Saint-Etienne, CNRS

Kalina Kostyszyn   KALINA.KOSTYSZYN@STONYBROOK.EDU
Department of Linguistics &
Institute for Advanced Computational Science
Stony Brook University

Tiantian Gao   TIAGAO@CS.STONYBROOK.EDU
Rahul Verma   RXVERMA1@GMAIL.COM
Department of Computer Science
Stony Brook University

Derek Andersen   DEREK.ANDERSEN@ALUMNI.STONYBROOK.EDU
Joanne Chau   CHORYAN.CHAU@ALUMNI.STONYBROOK.EDU
Emily Peterson   EMILY.PETERSON@ALUMNI.STONYBROOK.EDU
Cody St. Clair   CODY.STCLAIR@ALUMNI.STONYBROOK.EDU
Department of Linguistics
Stony Brook University

Paul Fodor   PFODOR@CS.STONYBROOK.EDU
Department of Computer Science
Stony Brook University

Chihiro Shibata   CHIHIRO@HOSEI.AC.JP
Department of Advanced Sciences
Graduate School of Science and Engineering
Hosei University

Jeffrey Heinz   JEFFREY.HEINZ@STONYBROOK.EDU
Department of Linguistics &
Institute for Advanced Computational Science
Stony Brook University

Abstract

Evaluating machine learning (ML) systems on their ability to learn known classifiers allows fine-grained examination of the patterns they can learn, which builds confidence when they are applied to the learning of unknown classifiers. This article presents a new benchmark for ML systems on sequence classification called MLRegTest, which contains training, development, and test sets from 1,800 regular languages.

Different kinds of formal languages represent different kinds of long-distance dependencies, and correctly identifying long-distance dependencies in sequences is a known challenge for ML systems to generalize successfully. MLRegTest organizes its languages according to their logical complexity (monadic second order, first order, propositional, or monomial expressions) and the kind of logical literals (string, tier-string, subsequence, or combinations thereof). The logical complexity and choice of literal provides a systematic way to understand different kinds of long-distance dependencies in regular languages, and therefore to understand the capacities of different ML systems to learn such long-distance dependencies.

Finally, the performance of different neural networks (simple RNN, LSTM, GRU, transformer) on MLRegTest is examined. The main conclusion is that their performance depends significantly on the kind of test set, the class of language, and the neural network architecture.

Keywords: formal languages, regular languages, subregular languages, sequence classification, neural networks, long-distance dependencies

1. Introduction

This article presents a new benchmark for the machine learning (ML) of regular languages called MLRegTest.[1] Regular languages are formal languages, which are sets of sequences definable with certain kinds of formal grammars, including regular expressions, finite-state acceptors, and monadic second order logic with either the successor or precedence relation in the model signature for words (Kleene, 1956; Scott and Rabin, 1959; Büchi, 1960).

This benchmark allows one to better understand the learning capabilities and limitations of practical ML systems on learning patterns over sequences. It was specifically designed to help identify those factors, specifically the kinds of long-distance dependencies, that can make it difficult for ML systems to successfully learn to classify sequences. MLRegTest contains 1,800 languages from 16 distinct subregular classes whose formal properties are well-understood. It is the most comprehensive suite of regular languages we are aware of. Finally, experimental results on the benchmark can be aggregated to form a complete block design, which facilitates statistical analysis of the results.

For each language, the benchmark includes three nested training sizes with equal numbers of positive and negative examples, three nested development sizes with equal numbers of positive and negative examples, and three nested sizes of four distinct test sets with equal numbers of positive and negative examples. The four test sets manipulate two ways in which testing can be difficult: (1) the test strings can either be at most as long as the longest training strings or they can be longer, and (2) the test strings can either be randomly

---

generated or designed to occur in pairs of strings x and y such that $x \in L$, $y \notin L$ and the string edit distance of x and y equals 1. We refer to such pairs of strings as forming the 'border' of the language.

Another aspect of MLRegTest's design was its attention to the role of long-distance dependencies in sequence classification. Long-distance dependencies are widely recognized as a key challenge to generalizing successfully. Bengio et al. (1994) define long-term dependencies this way: "A task displays long-term dependencies if prediction of the desired output at time t depends on input presented at an earlier time $\tau \ll t$." Many examples of such long-term dependencies abound in nature and engineering. For example, generative linguists, beginning with Chomsky (1956, 1957), have studied the grammatical basis of long-term dependencies in natural languages and have raised the question of how such dependencies are learned (Chomsky, 1965). However, there are many different ways in which a long-term dependency can manifest itself, and we should be interested in classifying long-term dependencies to the same degree as we are interested in classifying types of non-linear, numerical functions.

Formal languages provide a way to achieve such a classification, and the 16 classes used in this article are characterized by the kinds of long-term dependencies required to successfully distinguish strings. MLRegTest organizes its languages along two dimensions. One is according to their logical complexity. Can the formal language be expressed with a sentence of monadic second order logic, first order logic, propositional logic, or a particular kind of monomial? The other is according to the kind of logical literal. Are the primitives in the logical language based on the notion of a string (successive symbols), a tier-string (successive salient symbols), a subsequence (not-necessarily adjacent symbols in order), or combinations thereof? The logical complexity and choice of literal provides a systematic way to understand different kinds of long-distance dependencies in regular languages. In this way, we can study precisely the challenges certain kinds of long-distance dependencies, in terms of their logical complexity, bring to the learning of sequential classifiers.

We examine one such experimental design to broadly consider the question of where the difficulties lie for neural networks (NNs) learning to classify sequences drawn from regular languages from positive and negative examples. While we acknowledge that there may exist some ML system we did not consider whose performance erases the distinctions we find, our main objective was the development of the benchmark. Our investigation suggests that it will be an important milestone if other researchers are able to find a ML system that succeeds across the board on MLRegTest.

From our experiments, we were able to draw two main conclusions. First, neural networks generally perform worse on the test sets that examine the border of the language. Consequently, performance on randomly generated test data can mislead researchers into believing correct generalization has been obtained, and stricter testing can reveal it has not. This is not the first time such an observation has been made (Weiss et al., 2018, and others), but the degree to which it is observed here is striking.

Second, the formal properties of the languages are more important in determining its learning difficulty than the size of its grammatical representation. This conclusion is established in two parts. First we find that neither the size of the minimal deterministic finite-state acceptor nor the size of its syntactic monoid correlate well with NN performance. We also find that, across the board, neural networks have difficulty learning periodic regular languages; i.e those that require monadic second order logic. Also, the neural networks

generally performed worse on classifying strings on languages that are defined in terms of propositional logic with precedence as opposed to propositional logic with successor. While there could be other measures of grammar size that do correlate with learning difficulty, it remains an open question what those grammatical representations would be.

2. Background and Related Work

There is much precedent in exploring the use of formal languages to investigate the learning capabilities of machine learning systems, and neural networks in particular. Indeed this history goes right back to the foundational chapters in computer science. For example, the introduction of regular expressions into computer science (Kleene, 1956) was primarily motivated to understand the nerve nets of (McCulloch and Pitts, 1943). This kind of theoretical work which establishes equivalencies and relationships between neural network architectures and formal grammars continues to the present day (Li et al., 2020).

The reasons for making formal languages the targets of learning are as valid today as they were decades ago. First, the grammars generating the formal languages are known and understood. Therefore training and test data can be generated as desired to run controlled experiments to see whether particular generalizations are reliably acquired under particular training regimens.

Regular languages have often been used to benchmark ML systems. Casey (1996); Smith and Zipser (1989) studied how well first-order RNNs can learn to predict the next symbol of a string using regular languages based on the Reber grammar (Reber, 1967). Pollack (1991); Watrous and Kuhn (1992); Giles et al. (1992) studied how well second-order RNNs could learn to discriminate strings on the Tomita regular languages (Tomita, 1982). Over time, the Tomita languages have become a de facto benchmark for learning regular languages (Zeng et al., 1994; Weiss et al., 2018).

Later research also targeted nonregular languages (Schmidhuber et al., 2002; Chalup and Blair, 2003; Pérez-Ortiz et al., 2003). Readers are encouraged to read Schmidhuber (2015, sec. 5.13), which provides an extensive review of this literature up to 2015, with extensive focus on neural network ML architectures. More recent contributions in this area include Sennhauser and Berwick (2018); Skachkova et al. (2018); Ebrahimi et al. (2020); Delétang et al. (2022).

There are some key differences between the present paper and past research. First, the regular languages chosen here are known to have certain properties. The Reber grammars and Tomita languages were not understood in terms of their abstract properties or pattern complexity. While it was recognized some encoded a long-distance dependency and some did not, there was little recognition of the computational nature of these formal languages beyond that. In contrast, the formal languages in this paper are much better understood. While *subregular* distinctions had already been studied by the time of that research (Mc-Naughton and Papert, 1971), it went unrecognized how that branch of computer science could inform machine learning. Since then, there has been much work on clarifying particular subregular classes of languages in terms of their logical complexity as well as their significance for cognition (Rogers and Pullum, 2011; Rogers et al., 2013; Heinz and Idsardi, 2013; Rogers and Lambert, 2019; Lambert, 2023).

Second, MLRegTest is much more comprehensive and makes more fine-grained distinctions than previous work. For example, consider Tomita (1982). There were seven Tomita languages altogether, the alphabet size was restricted to two symbols, and the largest DFA has four states. MLRegTest improves each of these metrics and so it is much more comprehensive. There are 1,800 languages; 3 alphabet sizes are used (4, 16, and 64); and the minimum, maximum, median, mean and standard deviations of the sizes of the minimal DFA and their syntactic monoids are shown in Table 1.

Type of Machine	min	max	median	mean	s.d.
Minimal DFA	2	613	11	23.35	53.19
Monoid of Minimal DFA	2	2701	45	142	304.76

Table 1: Summary statistics of the numbers of states in the minimal deterministic acceptors and their syntactic monoids of the languages in MLRegTest.

Another example comes from recent work which studied transformer and LSTM performance on regular languages organized by their dot-depth (Bhattamishra et al., 2020). They consider 30 regular languages whose complexity varies according to where they fall on the dot-depth hierarchy. Like the classes presented here, the dot-depth classes are mathematically well-understood. However, the simplest class they consider, the dot-depth one class, is defined nearly identically to what we call the Piecewise Local Testable (PLT) class (Lambert, 2022), and MLRegTest considers hundreds of languages from 11 subclasses of PLT. In other words, there are many interesting, linguistically relevant, classes of languages within the simplest class considered by Bhattamishra et al. which are not distinguished in their study, but which MLRegTest does distinguish. To our knowledge, MLRegTest is the most comprehensive, fine-grained suite of artificial regular languages ever constructed.

3. Languages

This section describes the 16 classes of formal languages from which the 1,800 languages were drawn. The first part discusses the classes themselves, and the second part discusses how we designed the 1,800 languages in the dataset.

3.1 Subregular Formal Languages

An underlying theme to the 16 classes we consider is the notion of string containment. This notion can ultimately be dissected along two dimensions, logical power and representation, using the tools of mathematical logic and model theory (Enderton, 2001; Libkin, 2004). Figure 1 shows the 16 classes considered in this paper with arrows indicating proper subset relationships among them. It is worth mentioning that for each class C in Figure 1, there is an algorithm which can take any finite-state acceptor and decides whether the language recognized by that acceptor belongs to C or not. Many of these algorithms are based on the
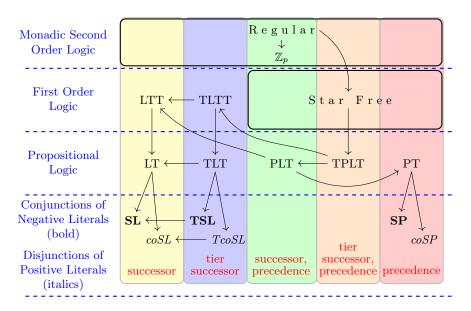
Figure 1: Regular and Subregular Classes of Formal Languages

algebraic properties of these classes (Pin, 2021). These algorithms have been implemented in the software package The Language Toolkit and Amalgam (Lambert, 2022).[2]

We explain these classes of languages by first considering languages defined via the containment of substrings (the "successor" column), and then by exploring different logics based on this notion. We then expand the notion of substring containment to subsequences (the "precedence" column) and then to substrings on projected tiers ("tier-successor") and then to their combinations. In addition to their natural mathematical character (as evidenced by their many characterizations), these classes are also motivated by linguistic and cognitive considerations (Rogers et al., 2010; Heinz et al., 2011; Rogers et al., 2013; Heinz, 2018).

3.1.1 The Local Family

For $w \in \Sigma^*$, the regular expression $\Sigma^* w \Sigma^*$ represents the set of all and only those strings which contain w as a substring. Let $C(w) = \Sigma^* w \Sigma^*$. As an example, Figure 2 shows a finite-state acceptor which recognizes $C(aa)$. One class of languages we consider can be defined
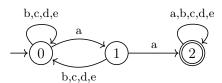

Figure 2: A finite-state acceptor recognizing the language of all and only those strings which contain the aa substring.

---

by taking finitely many strings $w_1, w_2, \ldots w_n$ and constructing the union of the languages which contain them.

$$\bigcup_{1 \leq i \leq n} C(w_i) \tag{1}$$

To decide whether a string x belongs to such a language requires identifying whether x contains any one of the substrings $w_1, w_2, \ldots w_n$. If it does then x belongs to the language, and otherwise it does not. The w_i are "licensing" substrings, and strings must possess at least one licensing substring.

The complements of the aforementioned languages present another class of languages. In this case, by DeMorgan's law, any complement language would have the form shown in Equation 2, where $\overline{C(w_i)}$ indicates the complement of $C(w_i)$; that is, the set of all strings which do not contain w_i as a substring.

$$\bigcap_{1 \leq i \leq n} \overline{C(w_i)} \tag{2}$$

To decide whether a string x belongs to such a language also requires identifying whether x contains any one of the substrings $w_1, w_2, \ldots w_n$. If it does then x does not belong to the language, and otherwise it does. Here, the w_i are "forbidden" substrings, and strings must not possess any forbidden substring.

Historically, the latter class of languages was studied first, and it is called the Strictly Local (SL) class, or sometimes the Locally Testable in the Strict Sense (McNaughton and Papert, 1971). Following Rogers and Lambert (2019), we call the former class Complements of Strictly Local (coSL). It can be argued that neither of these classes makes use of long-term dependencies. This is because there are only finitely many w_i and so there is a longest one of length k. Therefore, deciding whether a string x contains any w_i comes down to scanning x with windows of size k. All the information needed to decide string membership is local within bounded windows of size k.

From a logical perspective, the SL class can be understood as the conjunctions of negative literals and the coSL class can be understood as the disjunctions of positive literals. Here positive literals are strings w and they are interpreted as $C(w)$. A negative literal is $\neg w$, which is interpreted as $\overline{C(w_i)}$. In this way, we obtain a direct translation of these language classes into particular Boolean expressions (Rogers and Lambert, 2019). Specifically, in terms of these logical expressions SL languages will have the logical form shown in Equation 3, and and coSL languages will have the logical form shown in Equation 4.

$$\bigwedge_{1 \leq i \leq n} \neg w_i \tag{3}$$

$$\bigvee_{1 \leq i \leq n} w_i \tag{4}$$

Long-distance dependencies appear when the logical formalism introduced above is generalized to any Boolean expression over strings as literals. For example, the Boolean expression in Equation 5 would be interpreted as the set of strings x such that if x contains the substring aa then it also contains the substring ab.

$$aa \rightarrow ab \tag{5}$$

7

Note in this language, the substrings aa and ab do not need to be adjacent, or even in any particular order. For example, strings aab and $c^{10}abc^{20}aac^{20}$ belong to this language and strings baa and $c^{10}aac^{20}ac^{20}$ do not. This class of languages is called the Locally Testable (LT) class (McNaughton and Papert, 1971). Deciding whether a string x belongs to a LT language requires keeping track of the substrings that occur in k-sized windows in x (Rogers et al., 2013). Again because the Boolean expression is of finite length, there is a longest literal w_i of length k.

Boolean expressions are a propositional logical language, and a more powerful logic is First Order (FO) logic. FO logic includes universal and existential quantification over elements in a structure. To define a FO logical language, then, requires us to be clear about the structures we are describing. Model theory provides a way to talk about mathematical structures and the relations that make up such structures (Enderton, 2001; Hedman, 2004). Strings are a mathematical structure that are well studied in this way. Thomas (1982) showed that the class of formal languages definable with First Order logic with the successor relation is exactly the Locally Threshold Testable (LTT) class. In model-theoretic representations of strings, the successor relation is one way in which the order of the elements can be encoded, and its usage yields the notion of substring that was used in defining LT, SL, and coSL. It follows from Thomas' characterization that deciding whether a string x belongs to a LTT language requires keeping track of how many substrings there are, counting them up to some threshold t, that occur in k-sized windows in x (Rogers et al., 2013). LTT is a superclass of LT. In fact, LT is the subclass of LTT where the threshold t equals 1.

An example of a LTT language is one that requires there to be at least two aa substrings in a word. In this language, for all n, $b^n aab^n aab^n$ belongs to this language but $b^n aab^n$ does not. One can prove that there is no Boolean expression which represents this language and so it is not LT. But it is LTT, where the threshold equals 2, and so it is possible to distinguish between 0, 1 and 2 or more occurrences of substrings of length k. Note this is a kind of long-term dependency distinct in kind from the ones presented in the LT class.

The last move in logical power is to move from FO logic to Monadic Second Order (MSO) logic. MSO logic extends FO logic by additionally allowing quantification over sets of elements in structures. Büchi (1960) established that the languages definable with finite-state acceptors are exactly the ones definable with MSO logic over the successor relation. These are thus a proper superset of LTT.

Parity languages are examples of regular languages that are not LTT. A parity language is a language which counts modulo n. For example, a language that requires there to be an even number of as in string is an example of a parity language. Pure modulo-counting with a prime modulus forms a class we call $\mathbb{Z}_p$, named for the algebraic groups $(\mathbb{Z}/p\mathbb{Z})$ that their automata invoke.

These classes, SL, coSL, LT, LTT and Regular are shown in the leftmost column labeled "successor" in Figure 1. The class $\mathbb{Z}_p$ is a proper subset of the Regular languages and disjoint from these others.

3.1.2 The Piecewise Family

We next modify the notion of containment from substring to subsequence. For $w = a_1 a_2 \ldots a_n \in \Sigma^*$, the regular expression $\Sigma^* a_1 \Sigma^* a_2 \Sigma^* \ldots \Sigma^* a_n \Sigma^*$ represents the set of all and

only those strings which contain w as a subsequence. Let $C_<(w) = \Sigma^* a_1 \Sigma^* a_2 \Sigma^* \ldots \Sigma^* a_n \Sigma^*$. The choice of subscript for $C_<$ is motivated by the fact that the precedence relation ($<$) is used to represent the order of elements in a string in place of the successor relation in model-theoretic treatments (McNaughton and Papert, 1971; Rogers et al., 2013).

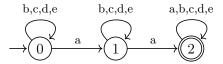As an example, Figure 3 shows a finite-state acceptor which recognizes $C_<(aa)$. Words

Figure 3: A finite-state acceptor recognizing the language of all and only those strings which contain the aa subsequence.

like $c^{20} a c^{20} a c^{20}$ belong to this language but words like $c^{20} a c^{20} b c^{20}$ do not.

The "Piecewise" families of languages can then be constructed exactly as before. The Strictly Piecewise (SP) class of languages is defined by taking finitely many strings $w_1, w_2, \ldots w_n$ and constructing the intersection of the languages which do not contain these strings as subsequences (Equation 6).

$$\bigcap_{1 \leq i \leq n} \overline{C_<(w_i)} \tag{6}$$

To decide whether a string x belongs to such a language requires identifying whether x contains any one of the substrings $w_1, w_2, \ldots w_n$. If it does not x belongs to the language, and otherwise it does. The w_i are "forbidden" subsequences, and strings must not possess any forbidden subsequences (Rogers et al., 2010). Using the Boolean expressions mentioned previously, each SP language can be expressed as shown in Equation 7 with where each w_i is interpreted as the language containing w_i as a subsequence (Rogers et al., 2013).

$$\bigwedge_{1 \leq i \leq n} \neg w_i \tag{7}$$

Similarly, the Complement of Strictly Piecewise (coSP) class of languages is defined by taking finitely many strings $w_1, w_2, \ldots w_n$ and constructing the union of the languages which contain these strings as subsequences.

$$\bigcup_{1 \leq i \leq n} C_<(w_i) \tag{8}$$

Here, the w_i are "licensing" subsequences, and to belong to the language, a string must possess at least one licensing subsequence (Rogers and Lambert, 2019). It follows that the coSP languages can be expressed with the Boolean expression shown in Equation 9

$$\bigvee_{1 \leq i \leq n} w_i \tag{9}$$

The SP and coSP language classes are incomparable with the LTT class. In other words, they generally encode different kinds of long-term dependencies than those in the LTT and LT languages.

Like the LT class, the Piecewise Testable (PT) class of languages (Simon, 1975) is characterized with any Boolean expression over literals. The literals are now interpreted as containment by subsequence. For example, the Boolean expression in Equation 10 would be interpreted as the set of strings x such that if x contains the subsequence aa then it also contains the subsequence ab.

$$aa \to ab \tag{10}$$

For example, strings aba and $c^{10}ac^{20}ac^{20}bc^{20}$ belong to this language and strings baa and $c^{10}ac^{20}ac^{20}$ do not. It follows that deciding whether a string x belongs to a PT language whose longest literal is of length k requires keeping track of the subsequences of size k that occur in x (Rogers et al., 2013).

The logical language obtained by combining the precedence relation with FO logic yields formulas whose corresponding languages form exactly the Star-Free (SF) class of languages (McNaughton and Papert, 1971). The name Star-Free comes from one of the first definitions of this class in terms of star-free regular expressions; that is languages describable with the base cases $a \in \Sigma$, $\emptyset$, and ϵ (the empty string), and operations union, intersection, concatenation, and complement with respect to Σ^*, but crucially the Kleene star operation is omitted. This celebrated result, along with other characterizations, is due to McNaughton and Papert (McNaughton and Papert, 1971).

That the SF class properly contains the LTT class follows from the fact that the successor relation is FO definable from precedence, but not vice versa.[3] A concrete example of a SF language that is neither LTT nor PT is the language obtained by concatenating all words which end with the symbol a with all words that do not contain a bc substring. Formally, with an alphabet $\Sigma = \{a, b, c, d\}$, this language can be expressed as $\Sigma^* a\overline{C(bc)}$.

When the precedence relation is combined with MSO logic, exactly the class of regular languages is obtained again. This is because precedence is MSO definable with successor and vice versa. The classes, SP, coSP, Star-Free, and Regular are shown in the rightmost column labeled "precedence" in Figure 1.

3.1.3 The Tier-Local Family

The tier-local family of classes introduces yet another kind of long-distance dependency that similarly interacts with the logical languages already introduced. In this family of language classes, the notion of containment is sequence of relevant symbols where irrelevant symbols are ignored. We also use the terms 'salient' and 'non-salient' for the relevant and ignored symbols, respectively. The set of relevant symbols is called the "tier" and is some subset $T \subseteq \Sigma$. For example, if $\Sigma = \{a, b, c, d, e\}$ and $T = \{a, e\}$ and $w = daceba$ then the string on tier T is aea. The notion of "contains the substring on the tier" can be generally expressed as follows. For $w = a_1 a_2 \ldots a_n \in T^*$, the regular expression $\Sigma^* a_1 \overline{T}^* a_2 \overline{T}^* \ldots \overline{T}^* a_n \Sigma^*$ where $\overline{T}^* = (\Sigma/T)^*$ represents the set of all and only those strings which contain w as a substring when all the non-tier symbols are removed.

For example, Figure 4 shows a finite-state acceptor which recognizes $C_T(aa)$ where $T = \{a, e\}$. This is the language which must contain the substring aa on the $\{a, e\}$ tier. So words like $c^{20}ac^{20}ac^{20}ec^{20}$ belong to this language but words like $c^{20}ac^{20}ec^{20}ac^{20}$ do not. The former has the string aae on this tier whereas the latter has aea.

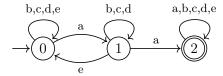3. $s(x, y) := x < y \wedge \neg \exists z[x < z < y]$.

Figure 4: A finite-state acceptor recognizing the language of all and only those strings which contain aa as a substring on the $\{a, e\}$ tier.

Letting $C_T(w) = \Sigma^* a_1 \overline{T}^* a_2 \overline{T}^* \ldots \overline{T}^* a_n \Sigma^*$, we can define the Tier Strictly Local (TSL) and Complement of Tier Strictly Local (coTSL) classes using conjunctive and disjunctive fragments of Boolean logic analogously to the SL, SP, coSL, and coSP classes. Similarly the use of propositional logic will characterize the Tier Locally Testable (TLT) class. Note for all languages within all of these classes, the tier T remains invariant. For example the formula shown in Equation 11 will be interpreted as $\overline{C_T(ae)} \cap \overline{C_T(ae)}$.

$$\neg ae \wedge \neg ea \tag{11}$$

So each term in the formula is interpreted with respect to the same tier T. The tier T does vary across these formal languages within the class, but not within individual languages. (See Aksënova and Deshmukh (2018) and Lambert (2022) for research on languages incorporating multiple tiers.)

When we move to FO logic, instead of the successor or precedence relations, the order relation is the tier-successor relation (specific again to some T). This representation of strings combined with FO logic yields the TLTT class. If MSO logic is used with the tier-successor relation, the class of regular languages is again obtained.

When $T = \Sigma$, every symbol is relevant and on the tier and this special case reduces to Local family of languages. It follows that TSL is a proper superset of SL, coTSL is a proper superset of coSL, and so on as shown in Figure 1.

Interestingly, projecting relevant symbols on tiers and checking for subsequence containment (precedence) does not lead to more expressive classes. In other words, everything tier-precedence can do, precedence can already do.

3.1.4 More Than One Order Relation

Including the precedence relation with either the successor relation or the tier-successor relation in the model-theoretic representation yields more expressive power than any of these relations on its own with propositional logics. For instance when both the precedence and successor relations are included, the literals in the logical language refer to sequences which are contiguous at some points and discontiguous at others. For example, let $\lhd$ denote the successor relation and $<$ the precedence relation. The substring aaa would now be written $a \lhd a \lhd a$ and the subsequence aaa would now be written $a < a < a$. A literal such as $a \lhd a < a \lhd b$ now denotes the set of all strings which contain a substring aa which precedes a substring ab.

In this way, combining adjacency and general precedence at the propositional level in order to allow local triggers for long-distance constraints or vice-versa. This is the Piecewise

Locally Testable (PLT) class. Similarly, The Tier Piecewise Locally Testable and (TPLT) combines tier-adjacency and general precedence for a similar purpose. TPLT properly includes PLT. Interestingly LTT $\subsetneq$ PLT and TLTT $\subsetneq$ TPLT (Rogers and Lambert, 2019; Lambert, 2022).

If FO logic is used, then the addition of successor or tier-successor to precedence does not increase the expressive power of the logical language, which yields the Star-Free languages.

The "strict" counterparts of PLT and TPLT also exist. They are omitted from this study because we are not aware of any procedure which can decide membership in them, contra the situation for PLT and TPLT (Lambert, 2022).

3.2 Summary

The classes presented here identify several types of formal languages. Among the simplest are the SL languages which forbid specific substrings from occurring. This kind of constraint, based on a conjunction of negative literals, specifies a local dependency. Its complement (coSL), a disjunction of positive literals, would be a different sort of local dependency, where substrings license, rather than forbid, strings in the language.

The other classes enable different sorts of long-term dependencies. For example, the Piecewise classes encode long-distance dependencies based on subsequences, and the Tier-Local classes encode long-distance dependencies based on strings of salient symbols. Consequently, the SP languages forbid subsequences from occurring, and the Tier Strictly Local languages forbid substrings from occurring on tiers of salient symbols.

We call these different kinds of strings—substrings, subsequences, projected substrings on tiers, and combinations thereof—*factors*. Adding arbitrary Boolean combinations results in a full propositional logic, which allows conditional constraints so that the presence or absence of a particular set of factors can trigger the enforcement of another local dependency. These are the Testable languages (LT, PT, TLT, PLT, and TPLT). FO logic lets one count instances of factors up to some threshold (LTT, TLTT) and MSO logic lets one count them relative to some modulus (Regular).

Finally it is worth mentioning that given a selection of model signature, any class at or below the propositional level has an associated parameterized learner that converges with complete accuracy without any negative data at all and whose sample complexity is relatively small (Lambert et al., 2021).

3.3 The Languages in MLRegTest

MLRegTest contains representations of 1,800 languages drawn from the 16 classes described above. This section explains how those 1,800 languages were constructed, and the design choices that went into their construction.

An important design goal was to ensure that each language in MLRegTest counts as a representative of a single class. Since classes may fully or partially include other classes, a typical formal language actually belongs to more than one class. For example, every SL language is LT but not vice versa. Consequently, we designed MLRegTest such that a language L counts as a representative of a class X provided that L belongs to class X, and L does not belong to any class Y which is a subset of, or incomparable with, X. Following this principle, a SL language could count as a representative of the SL class, but not the LT

class. Furthermore, the languages representative of the LT class will not belong to SL, coSL, TSL, coTSL, or PT. Henceforth, when referring to languages and classes in MLRegTest and we write "L belongs to class X", or "the languages in class X", "language L from class X", or anything similar, we mean the language L counts as representative of the class X in the manner described here.

After presenting some additional parameters of our design, we explain how we algorithmically verified that we achieved the aforementioned design goal. However, the specific choices of parameter values was influenced by our ability to conduct verification. In particular, we often adopted numbers that made verification possible. Now we turn to those parameterizations.

For each class, we developed ten base patterns. For example, in the SL class, one base pattern only forbids strings containing a^k, for the symbol a in the alphabet and for some k. Another pattern forbids $(ab)^{k/2}$ when k is even. These base patterns are then actualized by specifying both the alphabet and the value k, dimensions of variation to which we now turn.

For all language classes, the base patterns were embedded in three alphabet sizes {4, 16, 64}. The alphabets are nested. The sizes were chosen to grow exponentially. Specifically, the alphabets were the first 4, 16, or 64 letters of the sequence ⟨`abcdefghijklmnopqrstuvwxyz` `ABCDEFGHIJKLMNOPQRSTUVWXYZáàăéèěóòǒúùǔ`⟩.

One of the key properties of the languages in all the classes except SF, $\mathbb{Z}_p$, and Reg is the window size k, which corresponds to the length of the longest literal (string) in the logical expression describing the pattern. We considered three k values: {2,4,6}.

The language classes SL, coSL, SP, coSP, LT, PLT, and PT only vary across the dimensions of base, alphabet, and k value. Therefore, we constructed 90 languages in these classes (10 bases × 3 alphabets × 3 k values).

The SF, $\mathbb{Z}_p$, and Reg classes are not specified in terms of k value, tiers, or thresholds. Therefore, there are only 30 languages in these classes (10 bases × 3 alphabets).

An additional parameter of variation for the classes TSL, TcoSL, TLT, and TPLT is the number of salient symbols (those that project onto the tier). Because having more or fewer symbols be salient might affect learning difficulty, we provided two tier sizes for alphabets 16 and 64. For the alphabet of size 16, the tier sizes were {4, 7}; and for the alphabet of size 64, the tier sizes were {6, 11}. When the alphabet was of size 4, we only included one tier of size {3} because we could not otherwise easily construct languages that we could verify as representatives of TLT and TPLT.

The LTT class does not have a tier, but it does have an additional parameter, which is the counting threshold. We considered three thresholds {2,3,5} but they are not equally represented in MLRegTest. Instead, they occur in a 3:2:1 ratio so that we have 90 languages with threshold 2, 60 with threshold 3, and 30 with threshold 5. Consequently, there were a total of 180 languages in LTT.

Finally the TLTT class has both a tier and a threshold. The thresholds were chosen the same way as the LTT class. Also, the tiers were chosen the same way as the other tier classes. Therefore, there was a total of 300 TLTT languages (60 languages for each alphabet size/tier size combination, of which there are 5: 4/3, 16/7, 16/4, 64/11, 64/6). Within each group of 60, there are 30 languages with threshold 2, 20 with threshold 3, and 10 with threshold 5.

Tables 2 and 3 summarize the design parameters that led to the construction of 1,800 languages in MLRegTest.

class	bases	alphabets	windows	thresholds	total
SL	10	3	3		90
coSL	10	3	3		90
SP	10	3	3		90
coSP	10	3	3		90
LT	10	3	3		90
PLT	10	3	3		90
PT	10	3	3		90
LTT	10	3	3	2(3)*	180
SF	10	3			30
$\mathbb{Z}_p$	10	3			30
Reg	10	3			30
total					**900**

Table 2: A summary of the number of languages without tiers in each class and the dimensions along which they vary. The asterisk indicates that while there were actually 3 thresholds, since they occur in 3:2:1 ratio, they only doubled the number of languages.

class	bases	alphabets	windows	tiers	thresholds	total
TSL	10	1	3	1		30
	10	2	3	2		120
TcoSL	10	1	3	1		30
	10	2	3	2		120
TLT	10	1	3	1		30
	10	2	3	2		120
TPLT	10	1	3	1		30
	10	2	3	2		120
TLTT	10	1	3	1	2(3)*	60
	10	2	3	2	2(3)*	240
total						**900**

Table 3: A summary of the number of languages with tiers in each class and the dimensions along which they vary. The asterisk indicates that while there were actually 3 thresholds, since they occur in 3:2:1 ratio, they doubled the number of languages.

The automata representing the languages were generated by the Language Toolkit (Lambert, 2022) from source code which was itself generated by a Python program.[4] Tables 2 and 3 show the number of languages in each class. The Language Toolkit extends traditional regular expressions with basic terms that are interpreted as languages which contain substrings and/or subsequences. This does not increase the expressivity of traditional regular expressions, but it does facilitate the construction of languages belonging to the aforementioned classes. These expressions are then compiled into finite-state automata.

The languages in the coSL, TcoSL, and coSP classes were chosen to be the complements of the languages in the SL, TSL, and SP classes, respectively.

For each class C above, the programs The Language Toolkit and Amalgam include algorithms which decide whether a given finite-state automaton belongs to C. Therefore, to verify that a language L counts as a representative of class C_0 and not to classes C_1, C_2 and so on, we ran the decision algorithms for classes C_0, C_1, C_2 on the finite-state automaton for L and ensured that L belonged to C but not to C_1, C_2 and so on. This was done for each of the 1,800 languages in MLRegTest.

The decision procedures for many of these classes can be found in the algebraic literature on automata theory (Pin, 2021). The decision procedures for the tier-based classes are presented in (Lambert, 2023). Generally speaking, these algorithms run in time polynomial in the size of the syntactic monoid or in the size of the minimal DFA. While computing the syntactic monoid from the minimal DFA is in the worst case exponential, we were able to construct the syntactic monoids for all the languages in MLRegTest using The Language Toolkit.

Amalgam typically consumes considerably less time and memory in practice than The Language Toolkit when deciding class membership. Using Amalgam, we verified that every language L labeled as belonging to class C in MLRegTest (except those belonging to the SP and coSP classes) counts as a representative of class C. The only exceptions to this are the Strictly Piecewise class and its complement, for which Amalgam currently has no test. For these classes, using The Language Toolkit, we were able to verify that all the languages in SP and coSP count as a representative of class SP and coSP, respectively. In this way, all languages in MLRegTest were verified as being representative of their designated class.

3.4 Randomly Constructing Finite-State Automata

One motivation for the careful curation and construction of the languages in MLRegTest was that languages in most of these classes are unlikely to be generated randomly. As evidence for this claim, we randomly constructed finite-state automata and then used some of the decision procedures mentioned above to classify them.

The procedure for randomly generating machines was as follows. We fixed a number of states n, a number of symbols s, a start state, an edge-probability $0 \leq p_e \leq 1$, and an acceptance-probability $0 \leq p_f \leq 1$. For each state, it was accepting with probability p_f. For each $\sigma \in \Sigma$, and for each pair of states (q, r), we included an edge from q to r labeled σ with probability p_e. We ran several experiments, varying in each of these parameters. Our

---

4. In the software, languages were named according to the scheme SIGMA.TAU.CLASS.K.T.I, where SIGMA is a two-digit alphabet size, TAU a two-digit number of salient symbols (the 'tier'), CLASS the named subregular class, K the width of factors used (if applicable), T the threshold counted to (if applicable), and I a unique identifier.
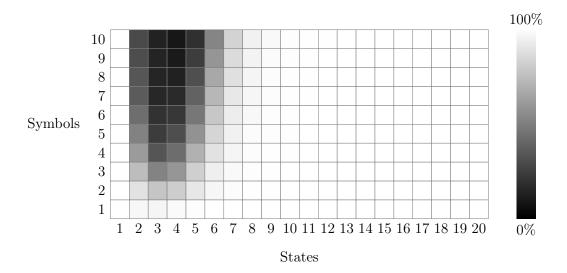
Figure 5: The proportion of Strictly Local languages upon fair generation, $p_e = p_f = 0.5$.

primary result is that as n increases, it was more likely that the automaton generated was Strictly Local.

We began with a fair construction, with $p_e = p_f = 0.5$, varying q and s over a range of values. For each combination of $q \in [1, 20]$ and $s \in [1, 10]$, we generated ten thousand automata by this method and determined how many of those ten thousand were Strictly Local. A heat map of the results is shown in Figure 5. As one can see, unless the alphabet is sufficiently large compared to the number of states, a vast majority of languages generated by this method are Strictly Local. The mean average had 87.57% in this class, with $n = 7$ and $s = 8$ as the parameters yielding the result closest to this value.

From there, we fixed $n = 7$ and $s = 8$ and varied p_e and p_f from 0 to 1 in intervals of 0.1. For each parameterization here, we generated one thousand machines and cataloged which were Strictly Local. Of course, when the p_f is exactly 0 or exactly 1, the resulting language is the empty set or its complement, and thus strictly local, so those cases are not exactly interesting. And if p_e is exactly 0 only the empty set is generated. But outside of these special cases, the effect of p_f is dwarfed by that of p_e, where a sparser graph is significantly less likely to be Strictly Local. The heat map is shown in Figure 6.

In sum, we cannot in good faith recommend random generation as a mechanism for producing test languages, as, without careful consideration of parameterization, the resulting languages are overwhelmingly Strictly Local. As this is among the simplest possible subregular classes, such generation could easily lead one to believe that a machine-learning algorithm performs significantly better than it might on a more diverse test set.

4. Data Sets

For each language in MLRegTest, we separately generated training, development, and test data sets. To generate data sets, we used the software library Pynini (Gorman, 2016; Gorman
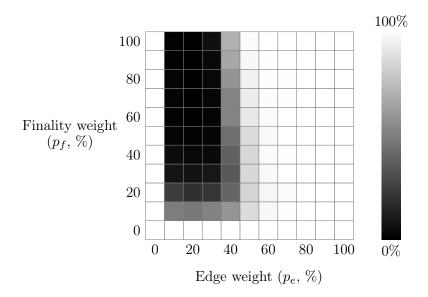
Figure 6: The proportion of Strictly Local languages for 7 states, 8 symbols.

and Sproat, 2021), which is a Python front-end to OpenFst (Allauzen et al., 2007).[5] The automaton constructed with The Language Toolkit was exported to the `att` format and then converted to a binary format by OpenFst, which is a format Pynini reads.

For each language L we generated a training set which included 100,000 strings, half of which belonged to the language and half of which did not. We call the strings belonging to L positive, and the strings not belonging to L negative. Additionally, there were equally many strings of length ℓ where ℓ ranged between 20 and 29. The positive strings were generated in a few steps. First, the automaton for L was intersected with the automaton for Σ^ℓ. Second, probabilities were assigned to the edges of this acyclic automaton to ensure a uniform distribution over its paths. This was accomplished with a reverse topological sort. Finally, paths were selected randomly from this weighted automaton.[6] The negative strings were similarly generated using the complement of L. Note that it was possible for the same string to be generated more than once (duplicates were allowed).

For each language L we similarly generated a development set which included 100,000 strings, half of which were positive and half of which were negative. As with the training

---

5. OpenFst is available at `https://www.openfst.org/twiki/bin/view/FST/WebHome` and Pynini at `https://www.openfst.org/twiki/bin/view/GRM/Pynini`.

6. We observed that randomly selecting paths by assuming a uniform distribution on outbound edges was not effective. For example, consider the coSL language where words must contain an aa substring (language 64.64.coSL.2.1.0). If there is a uniform distribution over outbound edges in the acyclic automaton generating words of this language of length 20, then in the first state, the probability of selecting the edge labeled a is $1/64$. Similarly, for the second state. In general, the probability of producing an aa substring at any given point is $(1/64)^2 = 1/4096$. It is very unlikely for aa to occur by chance under these conditions, for all but the latest states. If no aa substring has occured by the antepenultimate state, then the probability of selecting an edge with a becomes 1. And similarly for the penultimate state for the simple reason that the this acyclic machine only generates words with the substring aa. Assuming uniform distribution over the outbound edges of each state for this language has the consequence that aa substrings overwhelmingly occur at the right edge of the word.

17

set, there were equally many strings of length ℓ where ℓ ranged between 20 and 29. The positive strings were generated by intersecting the automaton for L with the automaton for Σ^ℓ, removing the positive strings from the training set, and then weighting the edges of this acyclic automaton as before to ensure a uniform distribution over the paths. The negative strings were similarly generated using the complement of L and removing the negative strings in the training set. In this way, we ensured the training and development sets for every L were disjoint.

For each language L we generated four test sets, each with 100,000 strings, half of which were positive and half of which were negative. We call these four test sets "Short Random" (SR), "Short Adversarial" (SA), "Long Random" (LR), "Long Adversarial" (LA). The Short Test sets included equally many strings of length ℓ where ℓ ranged between 20 and 29. The Long Test sets included equally many strings of length ℓ where ℓ ranged between 31 and 50. The Random Test sets sampled positive and negative strings without replacement. In the Adversarial Test sets, each positive string x was paired with a negative string y such that the string edit distance $d(x, y) = 1$. No positive or negative string in any Test set occurred in the Training or Development sets. Below we describe how we generated the data to meet these specifications.

The Short Random Test sets generated positive strings as follows. For each length ℓ, the automaton A was constructed by intersecting the automaton for L with the automaton for Σ^ℓ, and removing the positive strings from both the training and development sets. The acyclic automaton A was weighted to ensure a uniform probability distribution over its paths. Then the following procedure was repeated. Let n be the number of strings remaining to be generated (initially 5,000) and P the list of strings currently obtained (initially empty). Then n many positive strings were generated by selecting n paths from A. Strings were added to a list only if they did not already occur in this list. Then n was updated to $5,000$ minus the length of this accumulating list. This process repeated until all desired unique strings were obtained. A similar procedure was followed for generating the negative strings. In this way, we ensured the SR Test set was disjoint from both the training and development sets, and that each string in the SR Test sets was unique. The Long Random Test sets were generated similarly by randomly sampling strings of each length without replacement.

The Short Adversarial Test sets for each L were constructed according to the following procedure. We constructed the transducer $C \circ T \circ A$ where A is the original automaton used to construct SR Test, T is the transducer recognizing the relation $\{(x, y) \mid x, y \in \Sigma^*, d(x, y) = 1\}$, and C is the automaton recognizing the complement of L, and where $\circ$ indicates composition. Consequently $C \circ T \circ A$ is the transducer whose paths correspond to positive strings x of length ℓ and negative strings y such that $d(x, y) = 1$. This machine was weighted to ensure a uniform distribution over its paths. For each ℓ, 5,000 unique paths were randomly selected to ultimately obtain 50,000 unique pairs of positive and negative strings. The Long Adversarial Test sets were generated similarly to the SA Test sets.

The above procedures produced 6 data sets (Train, Dev, SR, SA, LR, LA), each with 50,000 positive and 50,000 negative strings. We then made additional Train, Dev, SR, SA, LR, LA sets of 1/10th and 1/100th the size by downsampling. Consequently, for every language we prepared 3 training sets, 3 development sets and 12 test sets. The sets with 100,000 words we call "Large", those with 10k words we call "Mid", and those with 1,000

words we call "Small." These sets are nested so that every string in the Small set is included in the Mid set, which is included in Large set.

The above procedures were followed for all languages except the languages in the coSL, TcoSL, and coSP classes. The datasets for coSL, TcoSL, and coSP languages were generated simply by switching the positive and negative strings in the corresponding datasets for the corresponding SL, TSL and SP languages.

5. Experiments

This section reports a set of experiments and analyses that were conducted to assess the capabilities of generic neural networks to model the languages in MLRegTest. Our goal is to understand the strengths and limitations of neural networks in modeling regular languages and to identify the features that make languages easier or harder to learn. We achieve this by analyzing how neural network performance changes as several linguistic and model parameters (Table 5) are varied. By independently training and evaluating neural networks on nearly all combinations of the factors in Table 5, a large sample size ($n = 108,000$) of accuracy scores was collected, making possible powerful tests of statistical association. The experiments follow a factorial design outlined in §5.1, with factors listed in Table 5.

In our experimental results, measures of neural network performance including F-score, precision, and Brier score are each highly correlated with accuracy (Table 4). Further, since positive and negative data are balanced in MLRegTest, we find accuracy to be an effective measure of neural network performance and thus use it throughout.

	Accuracy	AUC	Brier
AUC	0.950	–	–
Brier	−0.940	−0.897	–
F-score	0.873	0.834	−0.811

Table 4: Correlation matrix of performance metrics.

5.1 Experimental Design

We examine the effects of the design factors in Table 5 on model accuracy: six factors describe languages, two describe datasets, and one describes neural network architecture. A realization of the factors `Alph`, `Tier`, `Class`, `k`, `j`, and `i` fully specifies a regular language in MLRegTest. A *model configuration* in the present design refers to a choice of regular language, training set size, and neural network architecture, which is to say a realization of all factors in Table 5 except `TestType`. Our experimental design therefore consists of

$$1,800 \text{ (languages)} \times 3 \text{ (training set sizes)} \times 5 \text{ (NN architectures)} = 27,000$$

model configurations, each corresponding to a unique trained model. For all model configurations in the design, the associated trained model was evaluated on the four distinct test sets SR, SA, LR, and LA (described in §4) corresponding with the model configuration's regular language. In total, 108,000 observations of model accuracy scores were collected.

Factor Name	Description (Levels)
`Alph`	Alphabet size (4, 16, 64)
`Tier`	(2, 3, 4, 6, 7, 11, 16, 64)
`Class`	Subregular language class (SL, coSL, TSL, TcoSL, SP, coSP, LT, TLT, PT, LTT, TLTT, PLT, TPLT, SF, Zp, Reg)
`k`	Factor width (0, 2, 3, 4, 5, 6)
`j`	Threshold (0, 1, 2, 3, 5)
`i`	Language identification number (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
`TrainSize`	Size of training set (Small: 1k, Mid: 10k, Large: 100k)
`TestType`	Type of test set: whether strings are short or long, random or adversarial (SR, SA, LR, LA)
`NNType`	Neural network architecture (Simple RNN, GRU, LSTM, Stacked LSTM, Transformer)

Table 5: Factors comprising the experimental design.

All 2,880 combinations of `Alph`, `Class`, `TrainSize`, `TestType`, and `NNType` were tested in our experiments. Importantly, not all combinations of `Tier`, `k`, `j`, and `i` were tested because the values of these variables depend on those of `Alph` and `Class`. For example, not all language classes have different tier alphabets or thresholds (see §3).

Our experimental design and statistical analysis follows the approaches advocated by Demšar (2006) and Stąpor (2018). We form a full factorial design by mean-aggregating the factors `Tier`, `k`, `j`, and `i`. Specifically, the cells of the design are indexed by the factors `Alph`, `Class`, `TrainSize`, `TestType`, and `NNType`. The value of each cell is the average of accuracy scores of observations that have the same values of `Alph`, `Class`, `TrainSize`, `TestType`, and `NNType` but different values of `Tier`, `k`, `j`, and `i`. The full factorial design thus has 2,880 aggregated observations and makes possible repeated-measures non-parametric statistical tests, in particular the Friedman test (Demšar, 2006; Stąpor, 2018). Indeed, any of the five factors `Alph`, `Class`, `TrainSize`, `TestType`, or `NNType` can be considered a treatment variable since we hypothesize them to have an effect on accuracy, while the remaining four form blocking variables. To apply the Friedman test, the full factorial design is cast to a matrix whose rows represent distinct combinations of blocking variable levels and whose columns represent treatment levels. The Friedman test is then used to ask whether any of the treatment levels differ (described in §6). In those cases that we reject the Friedman test null hypothesis of equal treatment effects, we conduct post hoc analyses using the Nemenyi-Wilcoxon-Wilcox all-pairs test to determine precisely which treatment levels are pairwise significantly different from one another (Pereira et al., 2015; Singh et al., 2016).

The Friedman test and Nemenyi-Wilcoxon-Wilcox all-pairs test both report p-values, which represent "the probability, calculated under the null hypothesis, that a test statistic is as extreme or more extreme than its observed value" (Benjamin et al., 2018). If this probability is deemed low enough to be significant, then the null hypothesis is rejected and it is concluded that an effect is present. It is therefore important to determine the cutoff α, known as the significance level, at which p-values are deemed significant. Many communities set the significance level to $\alpha = 0.05$ though Benjamin et al. (2018) argue it should be set

	Network Type				
Alphabet Size	Simple RNN	Transformer	GRU	LSTM	2-Layer LSTM
4	41,102	114,030	122,102	161,702	322,502
16	42,302	115,230	123,302	162,902	323,702
64	47,102	120,030	128,102	167,702	328,502

Table 6: Number of trainable parameters by alphabet size and network type.

lower to $\alpha = 0.005$. Some communities, such as researchers in high-energy physics and genetics, have stricter levels. We follow the recommendation of Benjamin et al. (2018), though we note that nearly all of our results of statistical significance are established by p-values on the order of 10^{-5} or smaller.

5.2 Neural Network Details

We implemented five distinct neural network architectures using the Tensorflow (Abadi et al., 2015) and Keras (Chollet et al., 2015) machine learning APIs: simple recurrent neural network (RNN), gated recurrent unit (GRU), long short-term memory (LSTM), two-layer LSTM, and transformer. The five architectures form the `NNType` experimental design factor. Each neural network model consists of the following ordered modules: token embedding (with embedding dimension 100), recurrent or attention module, linear layer, and softmax activation. The number of trainable parameters in each neural network depends on the alphabet size of the language being trained on and the network type, as detailed in Table 6.

The following hyperparameters were the same across all neural network architectures in our experimental pipeline: 30 training epochs, batch size of 64, binary cross-entropy loss, Adam optimizer, and learning rate of 2×10^{-5}.

6. Results

This section presents the results obtained from the experimental design described in Section 5.1 as well as their interpretation with regards to salient research questions.

6.1 Sanity Checks

We first report some results that give us confidence in the validity of our experimental setup and factorial design.

6.1.1 Training Size

Because the Small, Mid, and Large data sets are nested, we fully expect additional data will improve accuracy. Our first question was whether our results bore out this expectation.

Setting the treatment variable to `TrainSize` and the other variables as blocking variables, Table 7 shows the average accuracy scores of each treatment level which increase as expected. Furthermore, the Friedman test shows that the type of training set leads to sta-

Small	Mid	Large
0.668	0.772	0.863

Table 7: Average accuracy by `TrainSize`.

tistically significant differences in accuracy (Friedman chi-squared = 1864.6, df = 2, p-value < 2.2e−16). Post-hoc pairwise comparisons using Nemenyi-Wilcoxon-Wilcox all-pairs test for a two-way balanced complete block design showed that each treatment level differed significantly from the others (Table 8). Visual inspection of Figure 7 indicates that these results appear to hold across individual language classes, also as expected.

	Large	Mid
Mid	<2e−16	–
Small	<2e−16	<2e−16

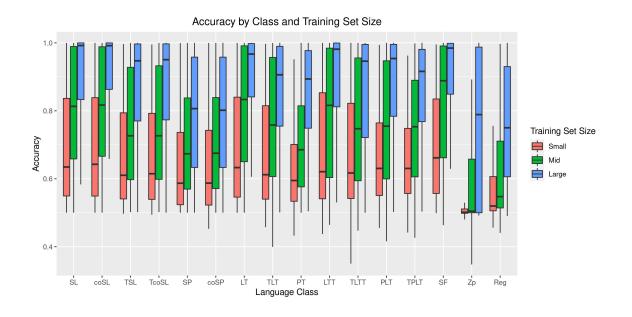Table 8: p-values from the Nemenyi-Wilcoxon-Wilcox all-pairs test with treatment variable `TrainSize`.

Figure 7: Accuracy by `TrainSize` and `Class`.

6.1.2 Complement Classes

We were also interested in comparing the performance on the pairs of language classes SL/coSL, TSL/TcoSL, and SP/coSP. Recall that the languages in these classes are paired,

in the sense that for every language L in class X $\in$ {SL, SP, TSL}, the complement $\bar{L}$ of L is in coX. Furthermore, the training, development, and test sets for every complement language $\bar{L}$ in coX was made simply by switching the labels in the training, development, and test sets for the corresponding language L in class X. Therefore we expected there would be no difference in performance.

Setting the treatment variable to `Class` and the other variables as blocking variables, the Friedman rank sum test shows that the type of class leads to a statistically significant difference in accuracy (Friedman chi-squared = 831.03, df = 15, p-value $< 2.2e{-}16$). However, our question at this point is whether these differences are found between the specific pairs of classes highlighted above.

While the Friedman test answers the question whether any treatment levels differ, it does not tell us where or how the treatment levels do so. We perform post-hoc analysis using the Nemenyi-Wilcoxon-Wilcox all-pairs test for a two-way balanced complete block design to answer these questions. Table 25 in the appendix shows the p-values reported by the post-hoc Nemenyi-Wilcoxon-Wilcox all-pairs test for a two-way balanced complete block design for all language classes. Table 9 presents the mean-aggregated accuracies for the classes of interest here. This shows that for these pairs of classes, there is no significant difference in accuracy as expected. Mean accuracies for all languages are listed in Table 26 in the appendix.

Accuracy		All-pairs p-value
SL	coSL	1.000
0.806	0.806	
SP	coSP	1.000
0.731	0.731	
TSL	TcoSL	1.000
0.769	0.769	

Table 9: Average accuracy for SL, coSL, SP, coSP, TSL and TcoSL classes.

6.2 Research Questions

This section examines the questions posed earlier in this article that influenced the design of MLRegTest and the neural networks used in our experiments. These questions are summarized below.

1. What is the effect of the type of test set (SR, SA, LR, LA)?
2. What is the effect of the language class?

   (a) What is the effect of logical level?
   (b) What is the effect of order relation (successor, tier-successor, precedence)?

3. What is the effect of alphabet size?
4. What is the effect of neural network architecture?
5. What is the effect of the size of the automata?

The remainder of this section analyzes these questions one by one.

6.2.1 THE TEST SET

Setting the treatment variable to `TestType` and the other variables as blocking variables, Table 10 shows the average accuracy scores of each treatment level. The Friedman rank sum

SR	LR	SA	LA
0.884	0.847	0.686	0.653

Table 10: Average accuracy by `TestType`.

test shows that the null hypothesis that all test set types have the same accuracies should be rejected (Friedman chi-squared = 1965.5, df = 3, p-value < 2.2e−16).

Post-hoc pairwise comparisons using Nemenyi-Wilcoxon-Wilcox all-pairs test for a two-way balanced complete block design showed that each pair of treatment levels differed significantly with a p-value < 2e−16.

While each pair of test types exhibits a statistically significant difference in accuracy, we examined Cohen's d effect size between each pair, which quantifies the "degree of departure from the null hypothesis" (Cohen, 1988). It is advised by Cohen (1988) to interpret $|d| \approx 0.2$ as a small effect, $|d| \approx 0.5$ as a medium effect, and $|d| \approx 0.8$ or greater as a large effect. Table 11 shows Cohen's d values for each pair of tests, and indicates that the adversariality

Pair	Effect Size	Interpretation
SR, LR	0.286	Small
LR, SA	1.274	Large
SA, LA	0.188	Small

Table 11: Effect sizes (Cohen's d) regarding `TestType`.

of the data set has a more significant effect on accuracy than the lengths of the strings within the data set. These differences are generally visible across each class as shown in Figure 8.

6.2.2 THE LANGUAGE CLASS

It was already mentioned in §6.1 that the Friedman rank sum test shows that generally the type of language class does lead to a statistically significant difference in accuracy (Friedman chi-squared = 831.03, df = 15, p-value < 2.2e−16). There we also discussed that the Nemenyi-Wilcoxon-Wilcox all-pairs test revealed no significant differences between the pairs SL/coSL, SP/coSP, and TSL/TcoSL.

In this section, we focus on whether the parameters by which we classified our language classes (cf. Figure 1) can help explain why the Friedman test rejects the null hypothesis that accuracies for `Class` would be the same. Specifically, we investigate the impacts of the kind of logic needed (CNL, DPL, Propositional, FO, MSO) and the kind of representational primitive (successor, precedence, tier-successor).

Figure 8: Accuracy by `Class` and `TestType`.

First, we investigate the logical level. Table 12 shows how the language classes are grouped into logical levels. Does accuracy generally decrease as expressivity increases logi-

Group	Classes
CNL	SL, SP, TSL
DPL	coSL, coSP, TcoSL
PROP	LT, PLT, PT, TLT, TPLT
FO	LTT, TLTT, SF
REG	Zp, Reg

Table 12: Language Classes Grouped by Logical Level

cally? If so, we would expect the accuracies to follow the ordering CNL ~ DPL > Prop > FO > MSO.

The Friedman rank sum test shows that the logical level leads to a statistically significant difference in accuracy (Friedman chi-squared = 145.85, df = 4, p-value < 2.2e−16).

FO	PROP	CNL	DPL	REG
0.781	0.776	0.768	0.768	0.697

Table 13: Average accuracy by logical level in decreasing order.

Table 14 shows the p-values from the Nemenyi-Wilcoxon-Wilcox all-pairs test for the groups organized by logical level. It is not surprising that DPL and CNL show no statistically significant difference. Contrary to expectations, however, PROP and FO also show no

25

	CNL	DPL	FO	PROP
DPL	0.99	–	–	–
FO	3.4e−6	3.5e−5	–	–
PROP	5.7e−6	5.7e−5	1.00	–
MSO	1.7e−6	1.2e−7	4.8e−14	4.2e−14

Table 14: p-values from the Nemenyi-Wilcoxon-Wilcox all-pairs test for classes grouped by logical level.

statistically significant difference. There are statistically significant differences in the other comparisons. However, it is not clear how meaningful these difference are. Table 15 shows the the pairwise effect sizes between logical levels. The effect sizes between REG and each

	CNL	DPL	FO	PROP
DPL	nss	–	–	–
FO	−0.079	−0.077	–	–
PROP	−0.059	−0.057	nss	–
REG	0.394	0.397	0.463	0.452

Table 15: Effect sizes as measured by Cohen's d for classes grouped by logical level. Differences that are not statistically significantly different (Table 14) are indicated by 'nss'.

other class shows a medium effect, while all other effect sizes indicate a small effect. In other words, at best there is a clear divide between MSO and all other logical levels. At the FO level and below, the differences, while statistically significant, are so small that they may not be meaningful.

Next we fix the logical level and examine the effect of the successor, precedence, and tier-successor. Table 16 shows how the language classes are grouped by the order relations.

Group	Classes
SUCC	SL, coSL, LT, LTT
PREC	coSP, PT, SF, SP
TSUCC	TcoSL, TLT, TLTT, TSL
OTHER	PLT, TPLT, Reg, Zp

Table 16: Language Classes Grouped by Order Relation

The Friedman rank sum test shows that the order relation leads to a statistically significant difference in accuracy (Friedman chi-squared = 135.97, df = 3, p-value < 2.2e−16).

SUCC	TSUCC	OTHER	PREC
0.800	0.777	0.759	0.747

Table 17: Average accuracy for classes grouped by order relation in descending order.

Table 18 shows the p-values from the Nemenyi-Wilcoxon-Wilcox all-pairs test for the groups defined with order relations. The SUCC group shows statistically significant dif-

	OTHER	PREC	SUCC
PREC	0.007	–	–
SUCC	1.6e−13	4.0e−14	–
TSUCC	0.559	0.220	2.8e−14

Table 18: p-values from the Nemenyi-Wilcoxon-Wilcox all-pairs test for classes grouped by order relation.

ferences in accuracy as compared to the other groups. The PREC group has the lowest. Analysis of the effect sizes using Cohen's d indicates that these effects are modest. There is a small effect ($d = 0.292$) when comparing the SUCC group to the PREC group. The SUCC/TSUCC comparison has a very small, perhaps negligible, effect ($d = 0.155$). Similarly, the effect size for the PREC/OTHER comparison is very small ($d = -0.101$).

6.2.3 ALPHABET SIZE

We also study the effect of the alphabet size. Average accuracy by alphabet size is depicted in Table 19. The Friedman rank sum test shows that the alphabet size leads to a statistically significant difference in accuracy (Friedman chi-squared = 267.82, df = 2, p-value < 2.2e−16). Table 20 shows the p-values from the Nemenyi-Wilcoxon-Wilcox all-pairs test.

64	16	4
0.798	0.764	0.740

Table 19: Average accuracy by alphabet size in descending order.

There are statistically significant differences between the accuracies on languages with the largest alphabet size as compared to the smaller ones. The effect size between sizes 64 and 16 is negligible ($d = 0.115$). The effect size between alphabet sizes 16 and 4 is also negligible ($d = 0.089$). The effect size between alphabet sizes 64 and 4 is small ($d = 0.200$). In summary, alphabet size is a statistically significant factor, though its effect is small to negligible as measured by Cohen's d.

6.2.4 WHAT IS THE EFFECT OF THE NEURAL NETWORK?

The Friedman rank sum test shows that the neural network type also leads to a statistically significant difference in accuracy (Friedman chi-squared = 460.12, df = 4, p-

	4	16
16	1.2e−11	–
64	<2e−16	<2e−16

Table 20: *p*-values from the Nemenyi-Wilcoxon-Wilcox all-pairs test for alphabet size.

value $< 2.2\mathrm{e}{-}16$). Those accuracies are shown in Table 21. According to Cohen's d, the

Simple RNN	2-layer LSTM	LSTM	GRU	Transformer
0.784	0.776	0.773	0.770	0.734

Table 21: Accuracies by neural network type in descending order.

GRU/Transformer comparison shows a small effect size ($d = 0.208$), but the other successive comparisons (SRNN/2LSTM, 2LSTM/LSTM and LSTM/GRU) have a negligible effect size with $|d| < 0.041$. Table 22 shows the p-values from the Nemenyi-Wilcoxon-Wilcox all-pairs test.

	Simple RNN	GRU	LSTM	2-layer LSTM
GRU	7.8e−6	–	–	–
LSTM	4.7e−11	0.278	–	–
2-layer LSTM	0.471	0.008	1.3e−6	–
Transformer	<2e−16	<2e−16	<2e−16	<2e−16

Table 22: *p*-values from the Nemenyi-Wilcoxon-Wilcox all-pairs test for neural network type.

Since the above results aggregate over all training set sizes, we repeated the above analysis by partitioning the data according to the training set size. After restricting to different training set sizes, the Friedman rank sum test continued to show that the network type significantly impacted accuracy with $p <2.2\mathrm{e}{-}16$. The Nemenyi-Wilcoxon-Wilcox all-pairs test revealed which differences were statistically significant. Summarized in Table 23, these results show each network type's overall accuracy on each training set. This analysis reveals some important differences. On small training sets, simple RNNs perform best, but on large training sets, 2-layer LSTMs and GRUs perform best. On Mid training sets, the simple RNNs, GRUs, and 2-layer LSTMs showed comparable performance. Somewhat surprisingly, transformers were consistently the worst performing architecture on MLRegTest.

These statistical differences have clear, though mostly small, effect sizes. For the small training set, the effect size on using a Simple RNN instead of the next-best neural network, the LSTM, has a Cohen's $d = 0.295$. For the Mid-sized training regimes, the effect sizes between the highest-performing network, the 2-layer LSTM, and the LSTM and transformer were negligible ($d = 0.069$) and medium ($d = 0.210$), respectively. For the Large-sized training regimes, the effect sizes between the highest-performing network, the 2-layer LSTM,

	Small	Mid	Large
Simple RNN	**0.719**	**0.781**	0.853
GRU	0.645	**0.776**	**0.889**
LSTM	0.671	0.770	0.879
2-layer LSTM	0.649	**0.783**	**0.897**
Transformer	0.656	0.750	0.795

Table 23: Average accuracy by neural network type and training size. In each column, bold-faced scores are statistically significantly different than non-bold scores, and not significantly different from each other according to the Nemenyi-Wilcoxon-Wilcox all-pairs test.

and the LSTM, the Simple RNN, and the transformer were small ($d = 0.105$), small ($d = 0.267$), and medium ($d = 0.646$), respectively.

Figures 9 and 10 provide a visualization of the performance of the neural networks on each language class aggregating across training regimes and for the Large training set, respectively. Visual inspection reveals that there is significant variation both across language
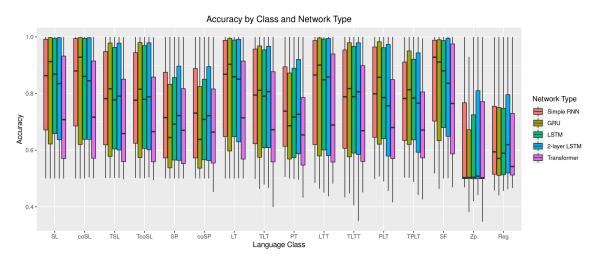

Figure 9: Accuracy by Class and Neural Network

classes, and across network types within language classes. For a given language class, it is natural to ask which network type performs best on that language class. For example, it appears that the Transformer architecture achieves the highest mean accuracy on learning class $\mathbb{Z}_p$ on the Large data set.

6.2.5 Grammar size

While the size of the grammar of the formal language was not a treatment variable, we chose to examine it anyway. The number of states of the minimal DFA is a standard measure for

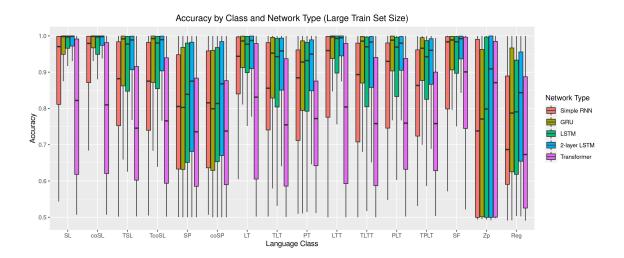Accuracy by Class and Network Type (Large Train Set Size)

Figure 10: Accuracy by Class and Neural Network on the Large Training Set.

the size of the grammar for a regular language. We also considered the number of states of the syntactic monoid of the minimal DFA. Table 1 provides summary statistics on these size measures of the representations of the MLRegTest languages.

We were interested in how well accuracy scores were inversely correlated with grammar size. Table 24 shows these overall correlations calculated for all the training sets, as well as for the large training set. These results show that while there is an inverse correlation

	All train	Large train
DFA size $\sim$ accuracy	-0.050	-0.129
monoid size $\sim$ accuracy	-0.045	-0.121

Table 24: Correlations between accuracies and the size of the target pattern measured by the size of the minimal DFA and its syntactic monoid on all training sets and the large training set.

(i.e. generally accuracy decreases the larger the automata), it is not strong since these correlations are much closer to 0 than to 1.

We calculated other correlations making finer distinctions by network type, test type, and training size. The strongest correlation we found was for the 2-layer LSTM trained on the Large training set and evaluated on the Short Adversarial test set. The correlation for the minimal DFA was -0.460 and for its syntactic monoid -0.470. While these correlations are better, they are still closer to 0 than to 1.

7. Discussion

The results of §6 support the following conclusions.

- Regardless of language class and neural network type, high performance on Random test sets does not imply correct generalization as measured by performance on the Adversarial test sets.

- Learning classifiers which depend on counting modulo n is more difficult for neural ML systems.

- Learning classifiers for languages definable with propositional logic and precedence is more difficult for neural ML systems than learning classifiers for languages definable with propositional logic and successor.

- Classification ability of neural ML systems does not correlate well with the size of the minimal DFA or monoid size.

- On small training sets, simple RNNs generally perform best.

- On large training sets, 2-layer LSTMs and GRUs generally perform best.

One potential objection to our conclusions above is that the neural networks utilized have insufficiently many parameters or inadequately set hyperparameters, and that networks with larger layers, more layers, or better tuned hyperparameters would be more successful. To address this issue, we selected six languages from an earlier version of MLRegTest and conducted a grid search to find the best hyperparameters on the large training set and small development set.

Two of the chosen languages had mean accuracy scores close to 1, two had mean accuracies close to 0.5, and the other two had mean accuracy scores close to 0.75. Additionally, we wanted the the standard deviations of the accuracies to be close to zero. We could find such languages for the four languages with mean accuracies that were close 1 or 0.5, but not for the ones with mean accuracies close to 0.75. We refer to these languages as Easy, Medium, and Hard.

For these languages, a grid search was performed with the following hyperparameter ranges.[7]

- learning rate: [0.01,0.0001]
- size of the hidden layer: [32,512]
- RNN cell type: [lstm, gru, ellman]
- number of layers: [1,4]
- bidirectionality: [True,False]
- losses: [binary cross entropy, mean squared error]
- optimizers: [RMSprop, Adam, Gradient Descent]
- padding: [end, begin]
- trainable initial hidden vectors: [True, False]

As expected, the grid search completed with the Easy languages with near ceiling performance in less than one hour. The grid search completed in about five hours on the Medium languages and showed improvements in mean accuracy (one increased to about 82% and

---

7. We thank Rémi Eyraud and Badr Tahri Joutei for running this on the high-performance computing clusters at Jean Monnet University.

the other to 95%). For the Hard languages, the models never acheived an accuracy beyond 51%, even after several hours of grid search.

We acknowledge that this experiment with 6 languages is a small sample, but computing resources prevented us from conducting grid searches on a larger sample of languages. That said, the results of the hyperparameter tuning that we did conduct indicate that it would help for some languages, but not for all of them, and also that it helps to different degrees.

8. Conclusion

This article presented a new benchmark for machine learning systems on sequence classification. This benchmark, called MLRegTest, contains training, development, and test sets from 1,800 regular languages spread across 16 subregular classes. These languages are organized according to their logical complexity (monadic second order, first order, propositional, or monomial expressions) and the kind of logical literals (string, tier-string, subsequence, or combinations thereof). The logical complexity and choice of literal provides a systematic way to understand different kinds of long-distance dependencies in regular languages, and therefore to understand the capacities of different ML systems to learn such long-distance dependencies.

In addition to providing three nested training sets for each language, MLRegTest provides four test sets according to two binary parameters: string length (short/long) and data generation (random/adversarial).

Finally, we examined the performance of different neural networks (simple RNN, LSTMs, GRU, Transformer) on MLRegTest. While there is much variation in the performance, some statistical trends were clear. First, the neural networks generally performed worse on the adversarial test sets; these contained pairs of strings of string edit distance 1 with the property that one belonged to the target language and the other did not.

Second, the formal properties of the languages themselves are more important in determining their learning difficulty than the size of the minimal DFA or the size of its syntactic monoid, neither of which correlated well with NN performance. It was also shown that neural networks have difficulty learning periodic regular languages; i.e those that require monadic second order logic. Another conclusion was that the neural networks generally performed worse on classifying strings on languages which are defined in terms of propositional logic with precedence as opposed to propositional logic with successor.

Third, simple RNNs performed best when training data was small, but 2-layer LSTMs and GRUs performed best when training data was large. These lattter two architectures achieved aggregate accuracies close to 90%.

Altogether, we hope that MLRegTest provides a useful tool for researchers in machine learning interested in sequence classification. We believe that systems that are able, for all languages, to achieve near perfect accuracy on all the test sets from small amounts of data will be revolutionary.

Acknowledgments

Appendix

Statistical Tables

Tables 25 and 26 list statistical results corresponding with the analyses of Section 6.

	coSL	coSP	LT	LTT	PLT	PT	Reg	SF
coSP	<2e−16	–	–	–	–	–	–	–
LT	1.000	<2e−16	–	–	–	–	–	–
LTT	0.848	1.6e−13	0.791	–	–	–	–	–
PLT	0.134	1.1e−13	0.102	0.999	–	–	–	–
PT	1.2e−13	0.504	1.8e−13	2.0e−12	6.7e−9	–	–	–
Reg	<2e−16	0.007	<2e−16	<2e−16	<2e−16	1.4e−8	–	–
SF	1.000	<2e−16	1.000	0.741	0.082	1.6e−13	<2e−16	–
SL	1.000	<2e−16	1.000	0.770	0.093	1.7e−13	<2e−16	1.000
SP	<2e−16	1.000	<2e−16	1.7e−13	1.1e−13	0.537	0.006	<2e−16
TcoSL	1.9e−13	0.209	1.7e−13	5.9e−11	1.3e−7	1.000	6.4e−10	1.5e−13
TLT	2.9e−13	0.006	2.3e−13	1.2e−7	7.3e−5	0.973	3.4e−13	2.0e−13
TLTT	1.2e−13	0.082	1.1e−13	7.5e−10	1.1e−6	1.000	5.0e−11	1.1e−13
TPLT	5.8e−13	0.003	3.7e−13	3.5e−7	0.000	0.934	2.2e−13	2.9e−13
TSL	1.7e−13	0.250	1.5e−13	3.3e−11	7.8e−8	1.000	1.1e−9	1.4e−13
Zp	5.0e−14	1.000	2.8e−14	1.8e−13	2.9e−13	0.981	9.7e−5	<2e−16

	SL	SP	TcoSL	TLT	TLTT	TPLT	TSL
coSP	–	–	–	–	–	–	–
LT	–	–	–	–	–	–	–
LTT	–	–	–	–	–	–	–
PLT	–	–	–	–	–	–	–
PT	–	–	–	–	–	–	–
Reg	–	–	–	–	–	–	–
SF	–	–	–	–	–	–	–
SL	–	–	–	–	–	–	–
SP	<2e−16	–	–	–	–	–	–
TcoSL	1.6e−13	0.232	–	–	–	–	–
TLT	2.1e−13	0.007	0.999	–	–	–	–
TLTT	1.1e−13	0.093	1.000	1.000	–	–	–
TPLT	3.3e−13	0.003	0.996	1.000	1.000	–	–
TSL	1.5e−13	0.275	1.000	0.998	1.000	0.993	–
Zp	2.1e−14	1.000	0.842	0.134	0.605	0.082	0.880

Table 25: P-values from the Nemenyi-Wilcoxon-Wilcox all-pairs test with treatment variable `Class`.

Accuracy	Class
0.820	SF
0.806	coSL
0.806	SL
0.802	LT
0.793	PLT
0.793	LTT
0.773	TLT
0.770	TLTT
0.769	TcoSL
0.769	TSL
0.769	TPLT
0.754	PT
0.731	coSP
0.731	SP
0.697	Zp
0.697	Reg

Table 26: Mean accuracy in decreasing order by language `Class`.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Alëna Aksënova and Sanket Deshmukh. Formal restrictions on multiple tiers. In *Proceedings of the Society for Computation in Linguistics*, volume 1, 2018. doi: https://doi.org/10. 7275/R5K64G8S. Article 8.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. Open-Fst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer, 2007. URL `http://www.openfst.org`.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, Mar 1994. doi: 10.1109/72.279181.

Daniel J. Benjamin, James O. Berger, Magnus Johannesson, Brian A. Nosek, E.-J. Wagenmakers, Richard Berk, Kenneth A. Bollen, Björn Brembs, Lawrence Brown, Colin Camerer, David Cesarini, Christopher D. Chambers, Merlise Clyde, Thomas D. Cook, Paul De Boeck, Zoltan Dienes, Anna Dreber, Kenny Easwaran, Charles Efferson, Ernst Fehr, Fiona Fidler, Andy P. Field, Malcolm Forster, Edward I. George, Richard Gonzalez, Steven Goodman, Edwin Green, Donald P. Green, Anthony G. Greenwald, Jarrod D. Hadfield, Larry V. Hedges, Leonhard Held, Teck Hua Ho, Herbert Hoijtink, Daniel J. Hruschka, Kosuke Imai, Guido Imbens, John P. A. Ioannidis, Minjeong Jeon, James Holland Jones, Michael Kirchler, David Laibson, John List, Roderick Little, Arthur Lupia, Edouard Machery, Scott E. Maxwell, Michael McCarthy, Don A. Moore, Stephen L. Morgan, Marcus Munafó, Shinichi Nakagawa, Brendan Nyhan, Timothy H. Parker, Luis Pericchi, Marco Perugini, Jeff Rouder, Judith Rousseau, Victoria Savalei, Felix D. Schönbrodt, Thomas Sellke, Betsy Sinclair, Dustin Tingley, Trisha Van Zandt, Simine Vazire, Duncan J. Watts, Christopher Winship, Robert L. Wolpert, Yu Xie, Cristobal Young, Jonathan Zinman, and Valen E. Johnson. Redefine statistical significance. *Nature Human Behaviour*, 2:6–10, 2018. doi: https://doi.org/10.1038/s41562-017-0189-z.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020. emnlp-main.576. URL `https://aclanthology.org/2020.emnlp-main.576`.

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

Mike Casey. The dynamics of discrete-time computation with application to recurrent neural networks and finite state machine extraction. *Neural computation*, 8(6):1135–1178, 1996.

Stephan K. Chalup and Alan D. Blair. Incremental training of first order recurrent neural networks to predict a context-sensitive language. *Neural Networks*, 16(7):955–972, 2003.

François Chollet et al. Keras. `https://keras.io`, 2015.

Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, page 113–124, 1956. IT-2.

Noam Chomsky. *Syntactic Structures*. Mouton & Co., Printers, The Hague, 1957.

Noam Chomsky. *Aspects of the theory of syntax*. Cambridge, MA: MIT Press, 1965.

Jacob Cohen. *Statistical power analysis for the behavioral sciences*. Lawrence Erlbaum Associates, 1988.

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. Neural networks and the chomsky hierarchy, 2022. URL `https://arxiv.org/abs/2207.02098`.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, dec 2006. ISSN 1532-4435.

Javid Ebrahimi, Dhruv Gelda, and Wei Zhang. How can self-attention networks recognize Dyck-n languages? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4301–4306, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.384. URL `https://aclanthology.org/2020.findings-emnlp.384`.

Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 2nd edition, 2001.

C L Giles, C B Miller, D Chen, H H Chen, G Z Sun, and Y C Lee. Learning and Extracting Finite State Automata with 2nd-Order Recurrent Neural Networks. *Neural Computation*, 4(3):393–405, 1992.

Kyle Gorman. Pynini: a Python library for weighted finite-state grammar compilation. In *Proceedings of the ACL Workshop on Statistical NLP and Weighted Automata*, pages 75–80, 2016.

Kyle Gorman and Richard Sproat. *Finite-State Text Processing*. Morgan & Claypool Publishers, 2021.

Shawn Hedman. *A First Course in Logic*. Oxford University Press, 2004.

Jeffrey Heinz. The computational nature of phonological generalizations. In Larry Hyman and Frans Plank, editors, *Phonological Typology*, Phonetics and Phonology, chapter 5, pages 126–195. De Gruyter Mouton, 2018.

Jeffrey Heinz and William Idsardi. What complexity differences reveal about domains in language. *Topics in Cognitive Science*, 5(1):111–131, 2013.

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 58–64, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

S.C. Kleene. Representation of events in nerve nets. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–40. Princeton University. Press, 1956.

Dakotah Lambert. *Unifying Classification Schemes for Languages and Processes With Attention to Locality and Relativizations Thereof*. PhD thesis, Stony Brook University, 2022. URL `https://vvulpes0.github.io/`.

Dakotah Lambert. Relativized adjacency. *Journal of Logic Language and Information*, 32 (4):707–731, 2023. doi: https://doi.org/10.1007/s10849-023-09398-x.

Dakotah Lambert, Jonathan Rawski, and Jeffrey Heinz. Typology emerges from simplicity in representations and learning. *Journal of Language Modelling*, 9(1):151–194, 2021.

Tianyu Li, Doina Precup, and Guillaume Rabusseau. Connecting weighted automata, tensor networks and recurrent neural networks through spectral learning, 2020. URL `https://arxiv.org/abs/2010.10029`.

Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Robert McNaughton and Seymour Papert. *Counter-Free Automata*. MIT Press, 1971.

Dulce G. Pereira, Anabela Afonso, and Fátima Melo Medeiros. Overview of friedman's test and post-hoc analysis. *Communications in Statistics - Simulation and Computation*, 44 (10):2636–2653, 2015. doi: 10.1080/03610918.2014.931971.

Jean-Éric Pin. *Handbook of Automata Theory*. European Mathematical Society Publishing House, September 2021. doi: 10.4171/Automata. URL `https://hal.archives-ouvertes.fr/hal-03579131`.

Jordan B. Pollack. The Induction of Dynamical Recognizers. *Machine Learning*, 7(2):227–252, 1991.

Juan Antonio Pérez-Ortiz, Felix A. Gers, Douglas Eck, and Jürgen Schmidhuber. Kalman filters improve lstm network performance in problems unsolvable by traditional recurrent nets. *Neural Networks*, 16(2):241–250, 2003. ISSN 0893-6080. doi: https://doi.

org/10.1016/S0893-6080(02)00219-8. URL `https://www.sciencedirect.com/science/article/pii/S0893608002002198`.

A. S. Reber. Implicit learning of artificial grammars. *Journal of Verbal Learning and Verbal Behavior*, 6:855–863, 1967.

James Rogers and Dakotah Lambert. Some classes of sets of structures definable without quantifiers. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 63–77, Toronto, Canada, July 2019. Association for Computational Linguistics. doi: 10. 18653/v1/W19-5706. URL `https://www.aclweb.org/anthology/W19-5706`.

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

James Rogers, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. On languages piecewise testable in the strict sense. In Christian Ebert, Gerhard Jäger, and Jens Michaelis, editors, *The Mathematics of Language*, volume 6149 of *Lecture Notes in Artifical Intelligence*, pages 255–265. Springer, 2010.

James Rogers, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. Cognitive and sub-regular complexity. In Glyn Morrill and Mark-Jan Nederhof, editors, *Formal Grammar*, volume 8036 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61: 85–117, 2015.

J. Schmidhuber, F. Gers, and D. Eck. Learning nonregular languages: A comparison of simple recurrent networks and lstm. *Neural Computation*, 14:2039–2041, 2002.

Dana Scott and Michael Rabin. Finite automata and their decision problems. *IBM Journal of Research and Development*, 5(2):114–125, 1959.

Luzi Sennhauser and Robert Berwick. Evaluating the ability of LSTMs to learn context-free grammars. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 115–124, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5414. URL `https://aclanthology.org/W18-5414`.

Imre Simon. Piecewise testable events. In *Automata Theory and Formal Languages*, pages 214–222. 1975.

Pawan Kumar Singh, Ram Sarkar, and Mita Nasipuri. Significance of non-parametric statistical tests for comparison of classifiers over multiple datasets. *Int. J. Comput. Sci. Math.*, 7(5):410–442, jan 2016. ISSN 1752-5055. doi: 10.1504/IJCSM.2016.080073. URL `https://doi.org/10.1504/IJCSM.2016.080073`.

Natalia Skachkova, Thomas Trost, and Dietrich Klakow. Closing brackets with recurrent neural networks. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing*

and Interpreting Neural Networks for NLP, pages 232–239, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5425. URL https://aclanthology.org/W18-5425.

Andrew W. Smith and David Zipser. Encoding sequential structure: experience with the real-time recurrent learning algorithm. In *International 1989 Joint Conference on Neural Networks*, pages 645–648, 1989. doi: 10.1109/IJCNN.1989.118646.

Katarzyna Stąpor. Evaluating and comparing classifiers: Review, some recommendations and limitations. In Marek Kurzynski, Michal Wozniak, and Robert Burduk, editors, *Proceedings of the 10th International Conference on Computer Recognition Systems CORES 2017*, pages 12–21, Cham, 2018. Springer International Publishing.

Wolfgang Thomas. Classifying regular events in symbolic logic. *Journal of Computer and Systems Sciences*, 25:370–376, 1982.

Masaru Tomita. Learning of construction of finite automata from examples using hillclimbing. *Proc. Fourth Int. Cog. Sci. Conf.*, pages 105 – 108, 1982.

Raymond L Watrous and G M Kuhn. Induction of Finite-State Automata Using Second-Order Recurrent Networks. *Advances in Neural Information Processing Systems*, (10): 309–316, 1992.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5247–5256, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/weiss18a.html.

Zheng Zeng, R.M. Goodman, and P. Smyth. Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks*, 5(2):320–330, 1994. doi: 10.1109/72.279194.