Neural Networks and the Chomsky Hierarchy

Gregoire Deletang et al.

Presented by: Michael Van Huffel
05.12.2023

Outline

Outline

Generalization of Neural Networks

- What can Neural Networks learn? When do they generalize well and when do they not?
- The central assumption of statistical learning theory is i.i.d. data, which is fairly restrictive. Most interesting problems are not i.i.d.
- Can neural networks reverse an arbitrary string? Can they sum/multiply two arbitrary numbers?

In this work

What computational power have different types of neural network architectures in practice?

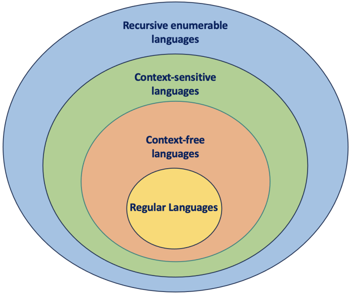
Outline

1. Introduction
2. Background
3. Contributions and Related Work
4. Methods
5. Results
6. Summary and Discussion
7. Appendix

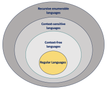
Exploring the Chomsky Hierarchy and Its Relation to Formal Languages

Grammar type (low $\rightarrow$ high)	Automaton	Memory
Regular (R)	Finite-state automaton (FSA)	Automaton state
Context-free (CF)	Push-down automaton (PDA)	+ infinite stack (only top entry accessible)
Context-sensitive (CS)	Linear bounded automaton (LBA)	+ bounded tape (all entries accessible)
Recursively enumerable (RE)	Turing machine (TM)	+ infinite tape (all entries accessible)

Automata serve as finite representations of formal languages and are typically categorized based on the class of formal languages they can recognize, known as the Chomsky hierarchy.



Finite-state automata (FSA)



Definition: A finite-state automaton (FSA) $\mathcal{A}$ is a 5 -tuple $(\Sigma, Q, I, F, \delta)$.

Example

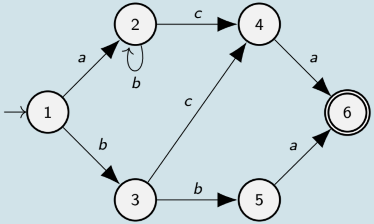


Figure 2: A simple finite-state automaton.

Deterministic vs nondeterministic FSA? I

Deterministic finite automaton:

- One state has only one transition for given input symbol
- Accept input iff last state is a terminal (or accept) state

Nondeterministic finite automaton:

- Any number of choices per step, which branch into "parallel universes"
- Accept input iff any branch ends in a terminal state

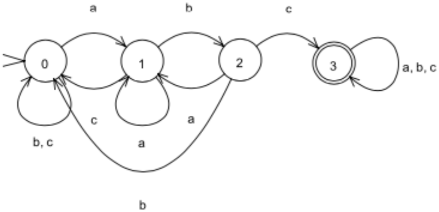


Figure: DFA accepting string abc

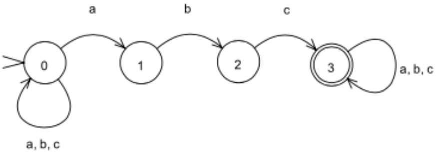


Figure: NFA accepting string abc

Deterministic vs nondeterministic FSA? II

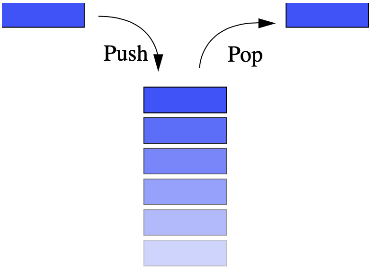
- Every language that can be described by an NFA (i.e. the set of all strings accepted by the automaton) can also be described by some DFA.

Finite-state automata recognize regular languages

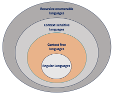
- FSA are capable of recognizing elements of a language such as $L = \{a^n \mid n \text{ even }\}$
- Proven to be too simple to recognize the elements of a language such as $L = \{a^n b^n \mid n > 0\}$.

What is a stack?

- A data structure that acts like a stack of dinner plates
- You can push new things on top
- You can pop things off the top
- Can't access things beneath the top without popping it



Pushdown Automata (PDA)



Pushdown automata $=$ FSA + stack.

NPDAs recognize the class of context-free languages (CFLs).
DPDAs can only recognize deterministic context-free languages (DCFLs).

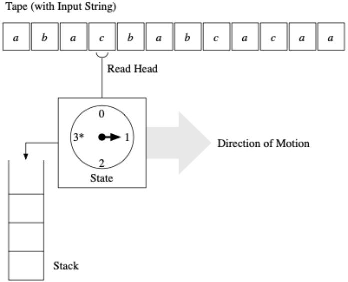


Figure: Illustration of a push-down automaton.

PDA Example

Example of a PDA $\mathfrak{P}$ accepting the language $L(\mathfrak{P}) = \{a^n b^n \mid n \in \mathbb{N}\}$

- $(1 \xrightarrow{a,\varepsilon \to \mathrm{X}} 1, 1 \xrightarrow{\varepsilon,\varepsilon \to \varepsilon} 2, 2 \xrightarrow{b,\mathrm{X} \to \varepsilon} 2)$ is an accepting run of $\mathfrak{P}$.

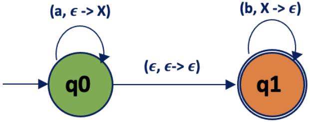
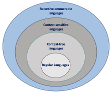


Figure: A PDA $\mathfrak{P}$ that recognizes the language $\{a^n b^n \mid n \in \mathbb{N}\}$.

Turing Machines (TM)



- PDAs have limits, cannot recognize context sensitive language $L = \{a^n b^n c^n \mid n > 0\}$
- **Turing Machine:** automation with tape, read-write head, finite controller, halting state

TM recognize the class of ecursively enumerable languages.

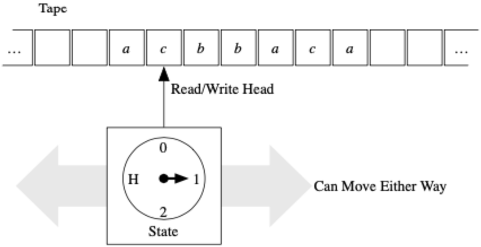


Figure: Illustration of a Turing machine.

TM Example

Example of a TM $\mathfrak{T}$ duplicating a string with only one character.

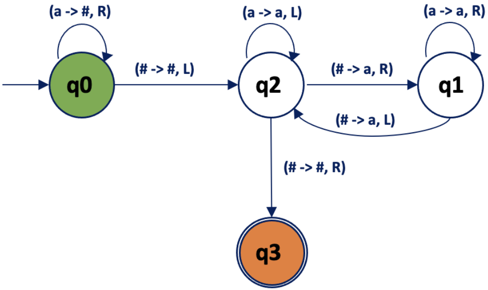
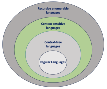


Figure: A TM $\mathfrak{T}$ that duplicates the input made of only one character a.

What about context-sensitive grammars?

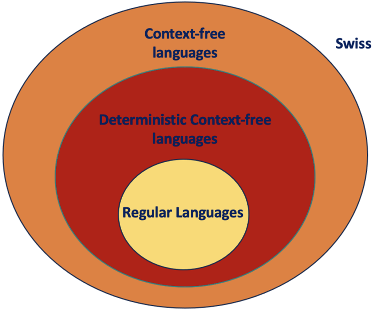


- Automaton for context-sensitive grammars: Linear Bounded Turing Machines (LBTM).
- TM with tape length limited to $k \cdot n$ cells (n = input length, k = constant)

. . . why do we want memory augmented neural networks?

Better expressivity and learning:

- RNN with finite precision = really big finite automaton
- RNN + deterministic PDA = Neural DPDA (cannot model all CFLs)
- RNN + nondeterministic PDA = Neural PDA (can model all CFLs)
- Syntactic ambiguity in natural language



Natural Language?
Swiss German, Dutch above context-free!

Ambiguity

CORONAVIRUS

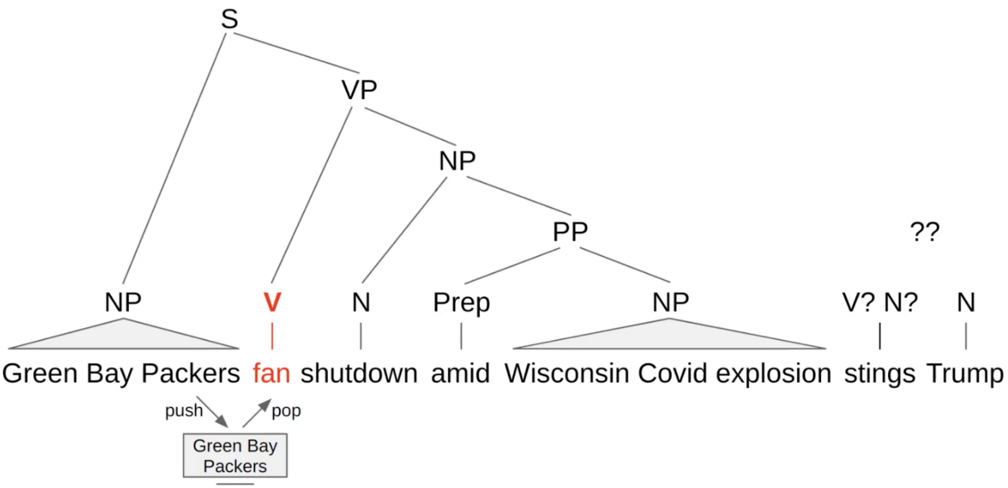
Green Bay Packers fan shutdown amid Wisconsin Covid explosion stings Trump

The NFL team won't allow fans at its home games.

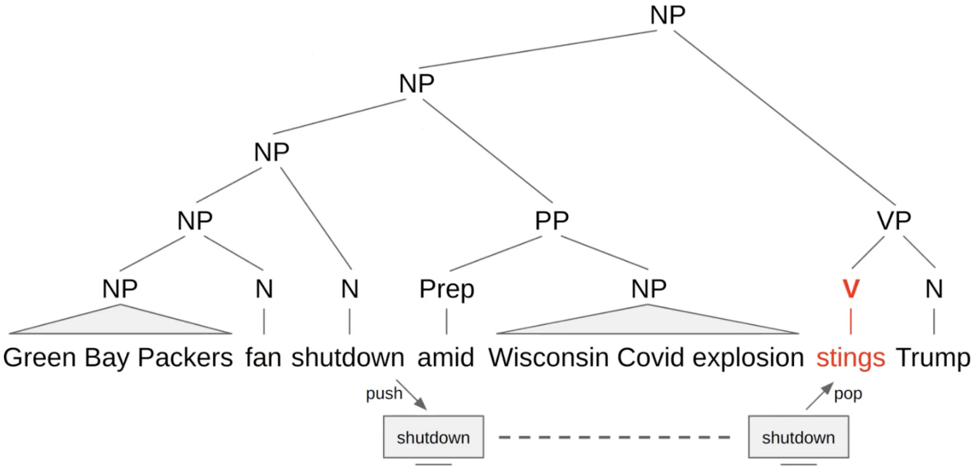
Fans waves towels as the Green Bay Packers take on the Seattle Seahawks in a playoff game at Lambeau Field on Jan. 12, 2020, in Green Bay, Wis. | Dylan Buell/Getty Images

By NATASHA KORECKI
10/07/2020 07:37 PM EDT

Ambiguity

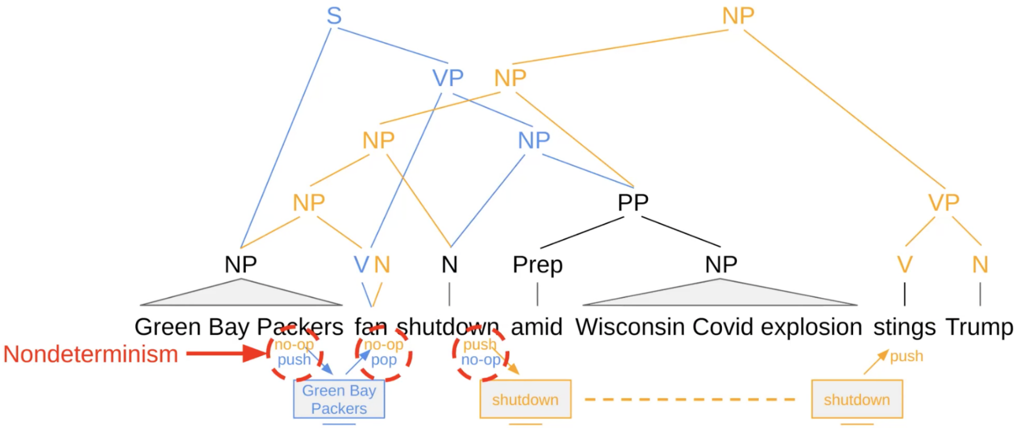

S
VP
NP
PP
??
NP
V
N
Prep
NP
V? N?
N
Green Bay Packers fan shutdown amid Wisconsin Covid explosion stings Trump
push
pop
Green Bay Packers

Ambiguity



NP
NP
NP
NP
PP
VP
NP
N
N
Prep
NP
V
N
Green Bay Packers fan shutdown amid Wisconsin Covid explosion stings Trump
push
pop
shutdown
shutdown

Ambiguity

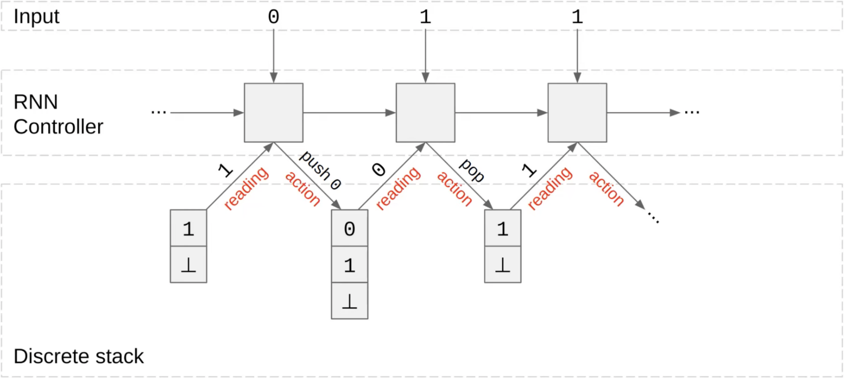


S
NP
VP
NP
NP
NP
NP
PP
VP
NP
V N
N
Prep
NP
V
N
Green Bay Packers fan shutdown amid Wisconsin Covid explosion stings Trump
Nondeterminism
no-op
push
no-op
pop
push
no-op
push
Green Bay Packers
shutdown
shutdown

RNN equipped with differentiable stacks

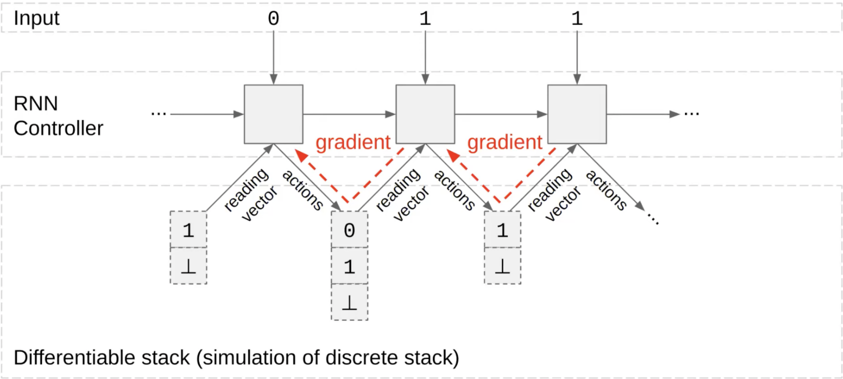
- *Differentiable* $\rightarrow$ can be trained with backprop
- Only simulates a discrete stack (not a discrete data structure itself)
- Three main styles
  - Stratification (Das et al. 1992, Sun et al. 1995, Grefenstette et al. 2015)
  - **Superposition (Joulin & Mikolov 2015)**
  - Nondeterminism (DuSell & Chiang 2020)

Stack RNNs



1. At each time step, RNN send "action" to stack
2. Stack action performed, return "top element" to RNN

Stack RNNs

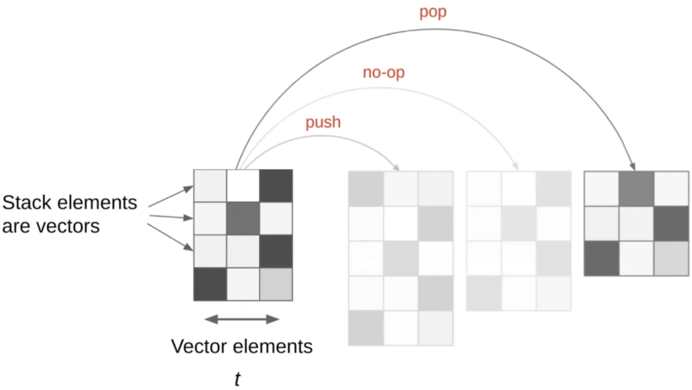


In reality: differentiable stacks, discrete stack is just simulation of differentiable stack

1. Instead of performing actions, controller outputs distributions over actions
2. Combine fractional actions to approximation of updated stack ("reading vector")
3. Stack actions must be differentiable wrt reading vector!

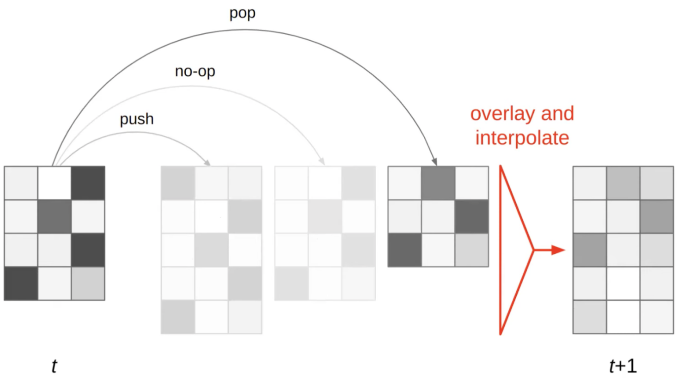
Superposition stack (Joulin & Mikolov 2015)

Controller actions include probability distributions over the 3 stack operations (**push**, **no-op**, **pop**)!



Superposition stack (Joulin & Mikolov 2015)

Controller actions include probability distributions over the 3 stack operations (**push**, **no-op**, **pop**)!



Tape RNNs (or Baby NTM, Suzgun et al. 2019)

- Extension of Stack-RNN
- Stack-RNN: Unbounded stack with only PUSH and POP operations (beside NO-OP)
- Baby-NTM: Fixed memory size, but more freedom to the model $\rightarrow$ 5 operations: ROTATE-RIGHT, ROTATE-LEFT, NO-OP, POP-LEFT, POP-RIGHT

Example: suppose current memory configuration is $[a, b, c, d, e]$

$$\begin{aligned} \text{ROTATE-RIGHT} &: [e, a, b, c, d]. \\ \text{ROTATE-LEFT} &: [b, c, d, e, a]. \\ \text{NO-OP} &: [a, b, c, d, e]. \\ \text{POP-RIGHT} &: [0, a, b, c, d]. \\ \text{POP-LEFT} &: [b, c, d, e, 0]. \end{aligned}$$

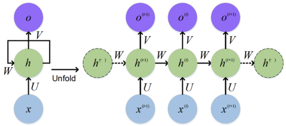
Outline

1. Introduction
2. Background
3. Contributions and Related Work
4. Methods
5. Results
6. Summary and Discussion
7. Appendix

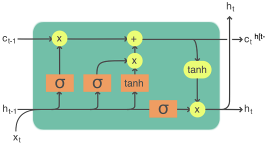
Related Work - Learning fromal languages I

Learning formal languages has been extensively explored in prior research using various machine learning architectures.

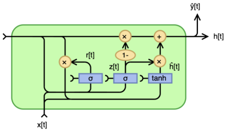
RNN (Elman, 1990)



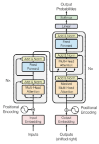
LSTMs (Hochreiter et al, 1997)



GRUs (Cho et al, 2014)



Transformers (Vaswani et al, 2017)



Related Work - Learning fromal languages II

Key insights from prior work include:

- RNNs and LSTMs require exponential memory for advanced languages
- Limited generalization for context-sensitive languages
- Transformers struggle with Dyck-n languages for n>1 and long sequences

Authors' contribution: Unified experimental protocol across Chomsky hierarchy levels for diverse models.

Related Work - Neural Networks and the Chomsky hierarchy

Key insights from prior work include:

- RNNs and Transformers are Turing complete (Chen et al, 2018; Pérez et al, 2019; 2021)
- RNNs, GRUs, and LSTMs recognize regular languages (Ackerman & Cybenko, 2020; Bhattamishra et al., 2020; Hao et al., 2022).
- Transformers don't align well with Chomsky hierarchy, unable to recognize certain regular languages (Weiss et al., 2021).

Authors' contribution: This research explores gradient-based learning's effectiveness in using inductive bias to recognize languages at different Chomsky hierarchy levels.

Related Work - Memory augmented Networks

Key insights from prior work include:

- Past approaches added external memory to neural networks (e.g., stacks, memory matrices) (Das et al., 1992a; Grefenstette et al., 2015; Graves et al., 2014).
- Memory-augmented networks can handle complex languages but require linear steps per token for superlinear tasks (Freivalds & Liepins, 2017; Kaiser & Sutskever, 2016).

Authors' contribution: This paper also delves into the exploration of the role of memory-augmented networks within the Chomsky hierarchy.

Outline

Data Generation

- The authors train on sequences of length ℓ, where ℓ is sampled from $\mathcal{U}(1, N)$ and $N = 40$.
- For testing, the authors consider sequences of length sampled from $\mathcal{U}(N + 1, M)$, where $M = 500$.

Length Generalization Benchmark

Deletang et al. introduce 15 novel length generalization tasks, spanning the entire Chomsky hierarchy:

- Regular: Solvable by finite-state automata,
- Deterministic Context-Free: Solvable by deterministic push-down automata,
- Context-Sensitive: Solvable by linear bounded automata

Regular tasks

Level	Name	Example Input	Example Output
R	`Even Pairs`	$aabba$	True
	`Modular Arithmetic (Simple)`	$1+2-4$	4
	`Parity Check`†	$aaabba$	True
	`Cycle Navigation`†	011210	2
DCF	`Stack Manipulation`	$abbaa$ POP PUSH a POP	$abba$
	`Reverse String`	$aabba$	$abbaa$
	`Modular Arithmetic`	$-(1-2)\cdot(4-3\cdot(-2))$	0
	`Solve Equation`$^\circ$	$-(x-2)\cdot(4-3\cdot(-2))$	1
CS	`Duplicate String`	$abaab$	$abaababaab$
	`Missing Duplicate`	10011021	0
	`Odds First`	$aaabaa$	$aaaaba$
	`Binary Addition`	$10010+101$	10111
	`Binary Multiplication`$^\times$	$10010 * 101$	1001000
	`Compute Sqrt`	100010	110
	`Bucket Sort`†*	421302214	011222344

Parity Check: Given a binary string, e.g., $aaabba$, compute if the number of b's is even.

Deterministic Context-Free tasks

Level	Name	Example Input	Example Output
R	`Even Pairs`	$aabba$	True
	`Modular Arithmetic (Simple)`	$1+2-4$	4
	`Parity Check`†	$aaabba$	True
	`Cycle Navigation`†	011210	2
DCF	`Stack Manipulation`	$abbaa$ POP PUSH a POP	$abba$
	`Reverse String`	$aabba$	$abbaa$
	`Modular Arithmetic`	$-(1-2)\cdot(4-3\cdot(-2))$	0
	`Solve Equation`$^{\circ}$	$-(x-2)\cdot(4-3\cdot(-2))$	1
CS	`Duplicate String`	$abaab$	$abaababaab$
	`Missing Duplicate`	10011021	0
	`Odds First`	$aaabaa$	$aaaaba$
	`Binary Addition`	$10010+101$	10111
	`Binary Multiplication`$^{\times}$	$10010 * 101$	1001000
	`Compute Sqrt`	100010	110
	`Bucket Sort`†*	421302214	011222344

Modular Arithmetic: Given a sequence of numbers in $\{0, 1, 2, 3, 4\}$, brackets, and operations in $\{+, -, \cdot\}$, compute the result modulo 5. For example, the sequence $\boldsymbol{x} = -(1-2)\cdot(4-3\cdot(-2))$ evaluates to $y = 0$.

Context-Sensitive tasks

Level	Name	Example Input	Example Output
R	Even Pairs	$aabba$	True
	Modular Arithmetic (Simple)	$1+2-4$	4
	Parity Check†	$aaabba$	True
	Cycle Navigation†	011210	2
DCF	Stack Manipulation	$abbaa$ POP PUSH a POP	$abba$
	Reverse String	$aabba$	$abbaa$
	Modular Arithmetic	$-(1-2)\cdot(4-3\cdot(-2))$	0
	Solve Equation°	$-(x-2)\cdot(4-3\cdot(-2))$	1
CS	Duplicate String	$abaab$	$abaababaab$
	Missing Duplicate	10011021	0
	Odds First	$aaabaa$	$aaaaba$
	Binary Addition	$10010+101$	10111
	Binary Multiplication×	$10010*101$	1001000
	Compute Sqrt	100010	110
	Bucket Sort†★	421302214	011222344

Duplicate string: Given a binary string, e.g., $x = abaab$, output the string twice, i.e., $y = abaababaab$.

Model Suite

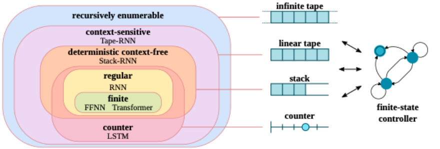
- RNN: Vanilla single-layer RNN (Elman, 1990).
- Stack-RNN, Tape-RNN: Single-layer RNN controller with access to a differentiable stack/tape (SRNN: Joulin & Mikolov, 2015, TRNN: Suzgun et al., 2019).
- Transformer: Vanilla Transformer encoder (Vaswani et al., 2017) with 5 positional encodings.
- LSTM: Vanilla single-layer LSTM (Hochreiter & Schmidhuber, 1997).

Outline

The Chomsky Hierarchy Forecasts Generalization

Level	Task	RNN	Stack-RNN	Tape-RNN	Transformer	LSTM
R	Even Pairs	**100.0**	**100.0**	**100.0**	**96.4**	**100.0**
	Modular Arithmetic (Simple)	**100.0**	**100.0**	**100.0**	24.2	**100.0**
	Parity Check†	**100.0**	**100.0**	**100.0**	52.0	**100.0**
	Cycle Navigation†	**100.0**	**100.0**	**100.0**	61.9	**100.0**
DCF	Stack Manipulation	56.0	**100.0**	**100.0**	57.5	59.1
	Reverse String	62.0	**100.0**	**100.0**	62.3	60.9
	Modular Arithmetic	41.3	**96.1**	**95.4**	32.5	59.2
	Solve Equation°	51.0	56.2	64.4	25.7	67.8
CS	Duplicate String	50.3	52.8	**100.0**	52.8	57.6
	Missing Duplicate	52.3	55.2	**100.0**	56.4	54.3
	Odds First	51.0	51.9	**100.0**	52.8	55.6
	Binary Addition	50.3	52.7	**100.0**	54.3	55.5
	Binary Multiplication	50.0	52.7	58.5	52.2	53.1
	Compute Sqrt	54.3	56.5	57.8	52.4	57.5
	Bucket Sort†★	27.9	78.1	70.7	**91.9**	**99.3**

(a) Average accuracy on lengths $[41, 500]$, bold number when $> 90\%$ (task is solved).



(b) Formal language classes with their minimal automaton needed to recognize or generate the language

Learned Program Structure I: Regular Tasks

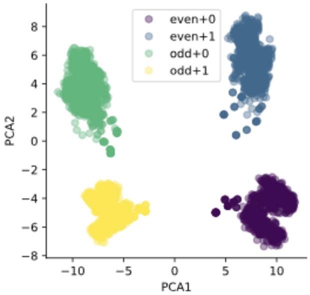


Figure: Simple RNN on Parity Check

- **Automaton States:** Hidden states correspond to automaton states.
- **Transitions:** Movement between states in FSA signifies transition in hidden states of RNN.
- **Four Clusters:** Points form four distinct clusters.
- **FSA Representation:** Clusters imply an FSA with four states.

Learned Program Structure II: Deterministic Context-Free Tasks

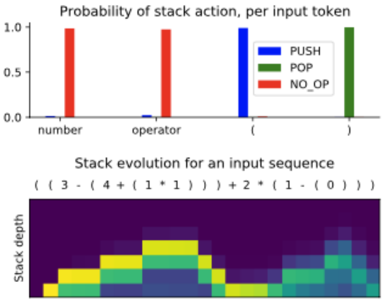


Figure: Stack-RNN on Modular Arithmetic

- **Upper part:** probabilities of actions in Stack RNN averaged over each time step
- **Lower part:** stack content over time, cells color according to 1^{st} principal component after PCA (most significant variation in stack states).

Learned Program Structure III: Context Sensitive Tasks

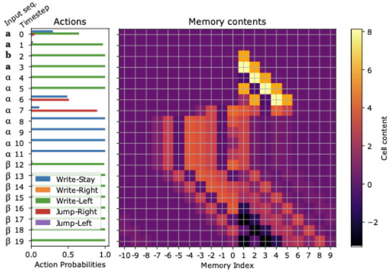


Figure: Tape-RNN on Duplicate String

- **Left part:** action probabilities after reading each input symbol
- **Right part:** tape evolution, each row represents one time step.

Why Do Transformers Perform So Poorly?

Positional encodings do not generalize to longer sequences. The follow-up work (Ruoss et al., 2023) addresses this by randomizing the positional encodings.

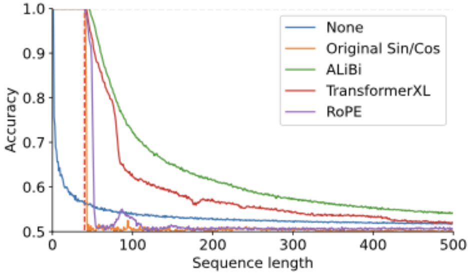


Figure: Accuracy per length by positional encoding.

Outline

Summary

- This paper uses formal language theory to analyze the generalization capabilities and computational complexity in sequence prediction tasks.
- It demonstrates that networks can generalize to longer out-of-distribution sequences, indicating correct algorithm learning, despite finite input and output languages.
- RNNs, LSTMs, and Transformers exhibit limitations in generalizing to longer test sequences due to numerical inaccuracies, suggesting potential hard limits in these architectures.
- The study highlights a model hierarchy where architectures integrated with external memory structures show potential in overcoming these generalization limitations.

Discussion - Paper Evaluation I

Weakness of the paper:

- Probing study conclusions are somewhat provisional
- It is an empirical study and so there is never enough experiments to 100% confirm the points. The authors have clearly articulated that in the limitations section.
- Poor explaination of the figures (Tape-RNN figure not explained at all)
- If the empirical nature is the main focus, some statistical analysis would be useful.

Discussion - Paper Evaluation II

Strengths of the paper:

- The paper presents a large-scale, thorough study of transformer-like architectures.
- Valuable insights, results are useful, pointing to directions of memory, which has been somewhat neglected in the past years.

Thank You for Your Attention!

Questions?

Outline

Dyck-n languages

A Dyck language is a set of well-formed strings made of brackets. A Dyck-n string can contain n types of brackets (e.g., for $n = 3$ you can have $"(", "[", "\{"$).

Example:

$$S \to \varepsilon \mid "["S"]"S$$

Contributions by the authors

- The authors introduce an open-source length generalization benchmark (15 tasks) that is employed for training and evaluating the models.
- An extensive generalization study (20’910 models) is conducted on state-of-the-art neural network architectures (RNN, LSTM, Transformer) and memory-augmented variants.
- The team demonstrates the effectiveness of augmenting architectures with differentiable structured memory (e.g., a stack or a tape) in enabling them to tackle more complex tasks.
- It is shown that increasing the volume of training data does not necessarily lead to improved generalization on more complex tasks, possibly suggesting inherent limitations in scaling laws (Kaplan et al., 2020).

Regular grammars

A grammar $\mathrm{G} = (\Sigma, N, S, R)$ is called regular iff

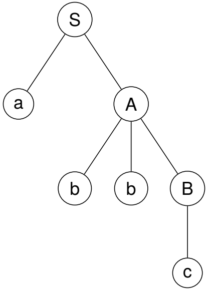
- all rules R are of the form

$$A \rightarrow w \text{ or } A \rightarrow wB(\text{ or } A \rightarrow Bw), \text{ where } A, B \in N \text{ and } w \in \Sigma^*$$

- i.e., LHS: a single NT symbol; RHS: a (possibly empty) sequence of T symbols, optionally followed (preceded) by a NT symbol

Example: $\Sigma = \{a, b, c\}, N = \{S, A, B\}, R = \{S \rightarrow aA, A \rightarrow bbB, B \rightarrow c\}$

$$\mathrm{S} \Rightarrow \mathrm{a}A \Rightarrow abbB \Rightarrow abbc$$



Insight: Context free grammars

A grammar $\mathrm{G} = (\Sigma, N, S, R)$ is called context free iff all rules R are of the form

$$A \to \alpha, \text{ where } A, \in N \text{ and } \alpha \in (\Sigma \cup N)^*$$

Example: $\Sigma = \{a, b\}, N = \{S, A\}, R = \{S \to ASA, S \to b, A \to a\}$

$$L = \{a^n b a^n \mid n \geq 1\}$$

Context-sensitive grammars

A grammar $\mathrm{G} = (\Sigma, N, S, R) >$ is called context sensitive iff

- all rules R are of the form $\alpha A \gamma \rightarrow \alpha\beta\gamma$, or $S \rightarrow \epsilon$ (with no S symbol on RHS) where $A \in N$, and $\alpha\beta\gamma \in (\Sigma \cup N)^{\star}, \beta \neq \epsilon$
- I.e., LHS: non-empty sequence of NT or T symbols with at least one NT symbol and RHS a nonempty sequence of NT or T symbols (exception: $S \rightarrow \epsilon$)
- For all rules $LHS \rightarrow RHS$: $|LHS| \leq |RHS|$

Example: $\Sigma = \{a, b, c\}, N = \{S, A, B, C\}, R = \{S \rightarrow aSBC, S \rightarrow aBC, CB \rightarrow BC, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\}$

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

Recursively enumerable grammars

A grammar $\mathrm{G} = (\Sigma, N, S, R) >$ is called recursively enumerable iff

- all rules R are of the form $\alpha \rightarrow \beta$, where $\alpha \in (N \cup \Sigma)^+, \beta \in (N \cup \Sigma)^*$
- I.e., LHS a nonempty sequence of NT or T symbols with at least one NT symbol and RHS a possibly empty sequence of NT or T symbols
- no restrictions on the form of production rules: arbitrary strings on LHS and RHS of rules

Example: $\Sigma = \{a\}, N = \{S, A, B, C, D, E\}, R = \{S \rightarrow ACaB, Ca \rightarrow aaC, CB \rightarrow DB, CB \rightarrow E, aD \rightarrow Da, AD \rightarrow AC, aE \rightarrow Ea, AE \rightarrow \epsilon\}$

$$L = \left\{a^{2^n} \mid n \geq 1\right\}$$