Understanding Machine Learning with Language and Tensors

Jon Rawski
Linguistics Department
Institute for Advanced Computational Science
Stony Brook University

Thinking Like A Linguist

1. Language, like physics, is not just data you throw at a machine

2. Language is a fundamentally computational process, uniquely learned by humans from **small, sparse, impoverished** data.

3. We can use core properties of language to understand how other systems generalize, learn, and perform inference.

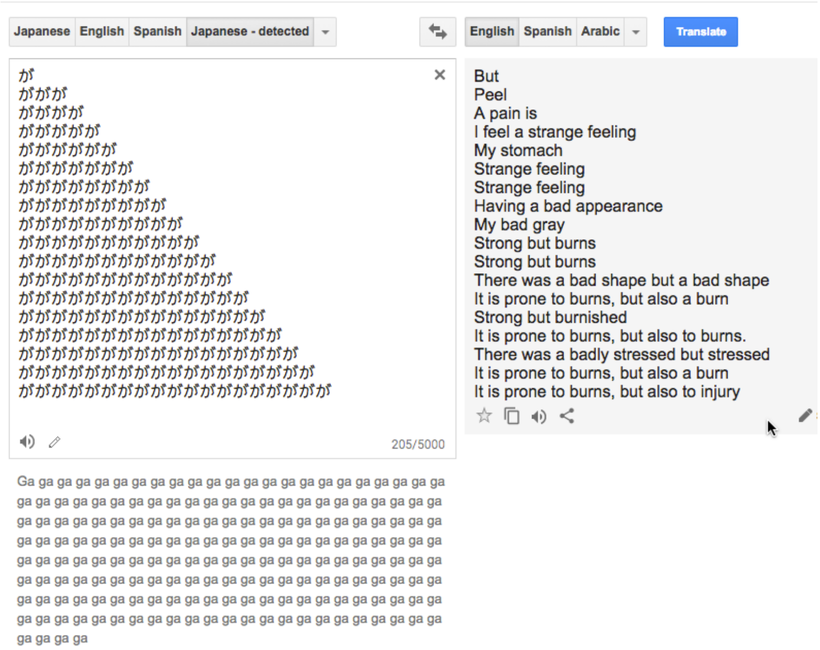
Gaps Between wet and dry brains

Data gap

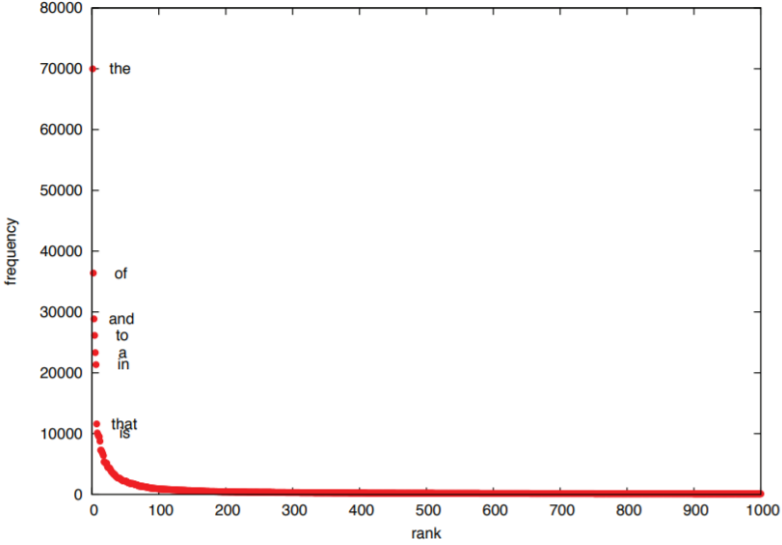
- Modern ML is training-data hungry, requires orders of magnitude more training data than biological brains
- Biological brains have species-specific, adaptively-evolved prior structure, encoded in the species genome and reflected in mesoscale brain connectivity

Energy Gap

- Modern computational infrastructure is energy-hungry, consuming orders of magnitude more power than biological brains. IT sector is a growing contributor to climate destruction


Japanese English Spanish Japanese - detected
English Spanish Arabic
Translate
が
ががが
がががが
ががががが
がががががが
ががががががが
がががががががが
ががががががががが
がががががががががが
ががががががががががが
がががががががががががが
ががががががががががががが
がががががががががががががが
ががががががががががががががが
がががががががががががががががが
ががががががががががががががががが
がががががががががががががががががが
ががががががががががががががががががが
205/5000
But
Peel
A pain is
I feel a strange feeling
My stomach
Strange feeling
Strange feeling
Having a bad appearance
My bad gray
Strong but burns
Strong but burns
There was a bad shape but a bad shape
It is prone to burns, but also a burn
Strong but burnished
It is prone to burns, but also to burns.
There was a badly stressed but stressed
It is prone to burns, but also a burn
It is prone to burns, but also to injury
Ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga
ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga
ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga
ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga
ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga
ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga
ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga
ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga ga
ga ga ga ga

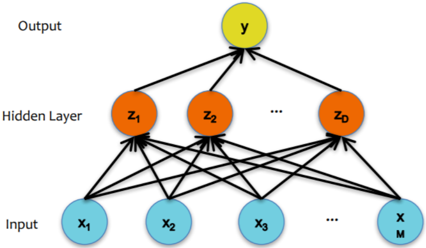
The Zipf Problem (Yang 2013)



A Recipe for Machine Learning

1. Given training data: $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$
2. Choose each of these:
   - Decision Function: $\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$
   - Loss Function: $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$
3. Define Goal:
   $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$
4. Train (take small steps opposite the gradient):
   $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$

"Neural" Networks & Automatic Differentiation



(F) Loss
$J = \frac{1}{2}(y - y^*)^2$

(E) Output (sigmoid)
$y = \frac{1}{1+\exp(-b)}$

(D) Output (linear)
$b = \sum_{j=0}^{D} \beta_j z_j$

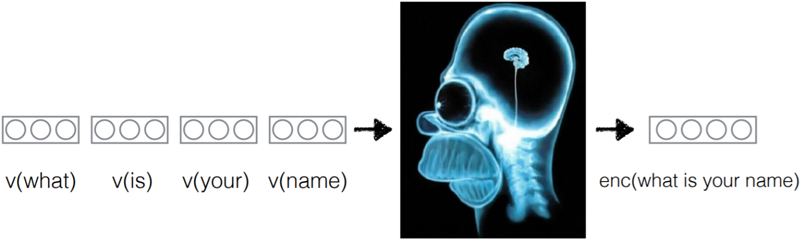
(C) Hidden (sigmoid)
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) Input
Given $x_i, \ \forall i$

p.c. Matt Gormley

Recurrent Neural Networks (RNN)

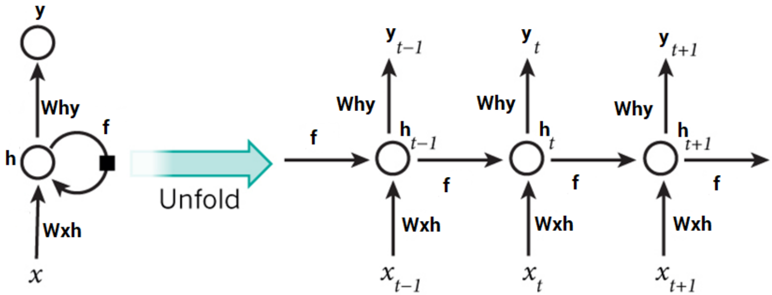


Acceptor: Read in a sequence. Predict from the end state.
Backprop the error all the way back.

---

p.c. Yoav Goldberg

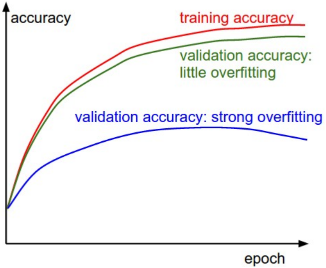
Recurrent Neural Networks (RNN)



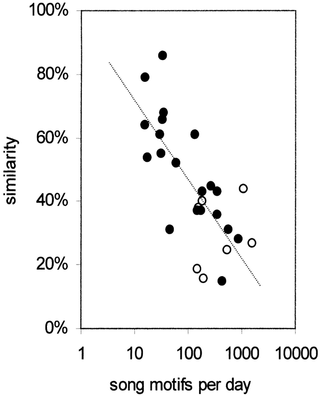
Acceptor: Read in a sequence. Predict from the end state. Backprop the error all the way back.

p.c. Yoav Goldberg

Expected behavior in Machine Learning: Multiple presentations should yield a better fit to training samples



Zebrafinches exhibit the opposite behavior: when presented the same song multiple times, imitation accuracy decreases



Tchernichovsky et al, PNAS 1999

> *When we consider it carefully, it is clear that no system — computer program or human — has any basis to reliably classify new examples that go beyond those it has already seen during training, unless that system has some additional prior knowledge or assumptions that go beyond the training examples. In short, there is no free lunch — no way to generalize beyond the specific training examples, unless the learner commits to some additional assumptions.*

Tom Mitchell, *Machine Learning, 2nd ed*

Don't "confuse ignorance of biases with abscence of biases" (Rawski and Heinz 2019)

What is a function for language?

Alphabet: $\Sigma = \{a, b, c, ...\}$

- Examples: letters, DNA peptides, words, map directions, etc.

Σ^*: all possible sequences (strings) using alphabet

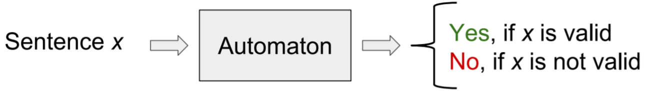
- Examples: aaaaaaaa, baba, bcabaca,...

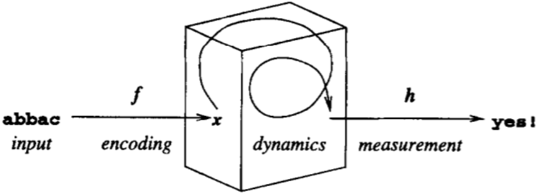
Languages: Subsets of Σ^* following some pattern

- Examples:
  - {ba, baba, bababa, babababa, ...}: 1 or more ba
  - {ab, aabb, aaabbb, aaaaabbbbb,...}: $a^n b^n$
  - {aa, aab, aba, aabbaabbaa,...}: Even # of a's

What is a function for language?

- **Grammar/Automaton:** Computational device that decides whether a string is in a language (says yes/no)
- Functional perspective: $f : \Sigma^* \to \{0, 1\}$



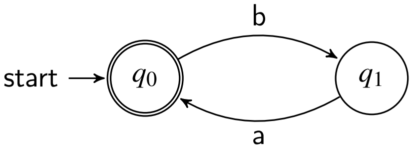


p.c. Casey 1996

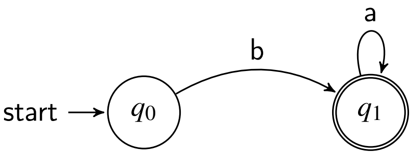
Regular Languages & Finite-State Automata

Regular Language: Memory required is finite w.r.t. input

(ba)*: {ba, baba, bababa,...}

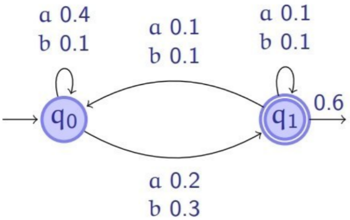


b(a*): {b, ba, baaaaaa,....}



Regular Languages & Finite-State Automata

$f : \Sigma^* \to \mathbb{R}$



Operator Representation

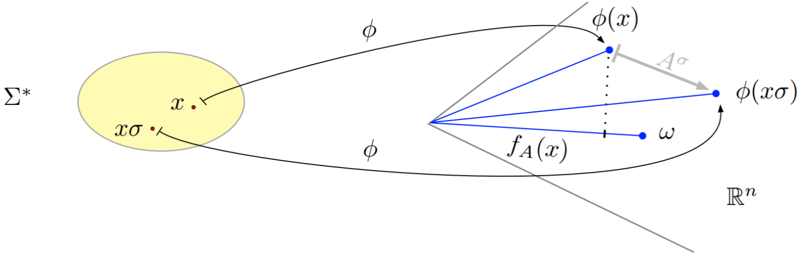
$$\boldsymbol{\alpha} = \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix} \quad \mathbf{A}^{a} = \begin{bmatrix} 0.4 & 0.2 \\ 0.1 & 0.1 \end{bmatrix}$$

$$\boldsymbol{\omega} = \begin{bmatrix} 0.0 \\ 0.6 \end{bmatrix} \quad \mathbf{A}^{b} = \begin{bmatrix} 0.1 & 0.3 \\ 0.1 & 0.1 \end{bmatrix}$$

$$\begin{aligned} f(ab) &= 0.4 \times 0.3 \times 0.6 + 0.2 \times 0.1 \times 0.6 = 0.084 \\ &= \boldsymbol{\alpha}^{\top} \mathbf{A}^{a} \mathbf{A}^{b} \boldsymbol{\omega} \end{aligned}$$

p.c. B. Balle, X. Carreras, A. Quattoni - ENMLP'14 tutorial

Finite-State Automata & Representation Learning



- An FSA induces a mapping $\phi : \Sigma^* \to \mathbb{R}$
- The mapping ϕ is compositional
- The output $f_A(x) = \langle \phi(x), \omega \rangle$ is linear in $\phi(x)$

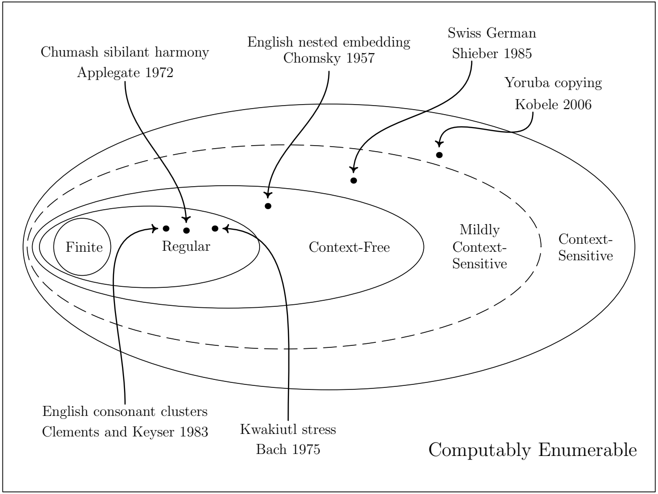
p.c. Guillaume Rabusseau

Supra-Regularity in Natural Language

The dog the cat the mouse feared loathed barked.

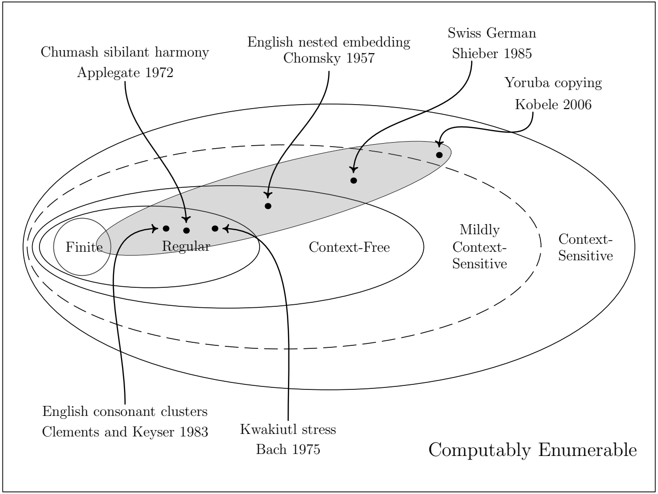
dat ik Anna Maria Hans zag laten helpen zwemmen.
THAT I ANNA MARIA HANS SAW LET HELP SWIM
that I saw Anna let Mary help Hans swim

Chomsky Hierarchy



p.c. Rawski & Heinz 2019

Chomsky Hierarchy

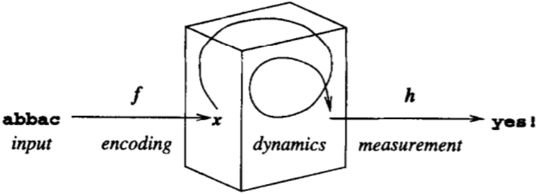


p.c. Rawski & Heinz 2019

RNN and regular languages

Language: *Does string w belong to stringset (language) L*

- Computed by different classes of grammars (**acceptors**)



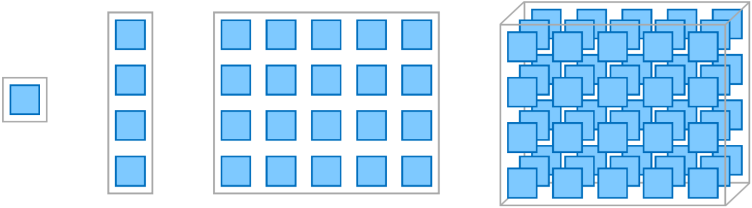
How expressive are RNNs?

Turing complete	infinite precision+time	(Siegelmann 2012)
$\subseteq$ counter languages	LSTM/ReLU	(Weiss et al. 2018)
Regular	SRNN/GRU	(Weiss et al. 2018)
	asymptotic acceptance	(Merrill 2019)
Weighted FSA	Linear 2nd Order RNN	(Rabusseau et al. 2019)
Subregular	LSTM problems	(Avcu et al. 2017)

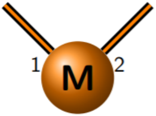
pic credit: Casey 1996

Tensors: Quick and Dirty Overview

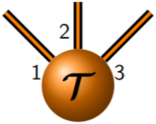
- Order 1 — vector:
$$\vec{v} \in A = \sum_i C_i^v \vec{a_i}$$
- Order 2 — matrix:
$$M \in A \otimes B = \sum_{ij} C_{ij}^M \vec{a_i} \otimes \vec{b_j}$$
- Order 3 — Cuboid:
$$R \in A \otimes B \otimes C = \sum_{ijk} C_{ijk}^R \vec{a_i} \otimes \vec{b_j} \otimes \vec{c_k}$$



Tensor Networks (Penrose Notation?)

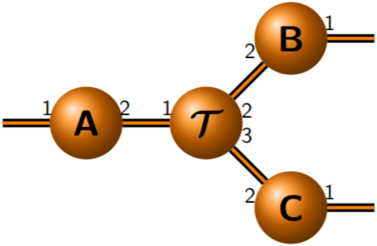


Matrix: $\mathbf{M}_{i_1 i_2}$



3rd order tensor: $\mathcal{T}_{i_1 i_2 i_3}$

Tensor times matrices:

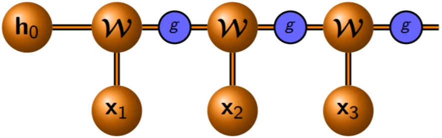


$$(\mathcal{T} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C})_{i_1,i_2,i_3} = \sum_{k_1 k_2 k_3} \mathcal{T}_{k_1 k_2 k_3} \mathbf{A}_{i_1 k_1} \mathbf{B}_{i_2 k_2} \mathbf{C}_{i_3 k_3}$$

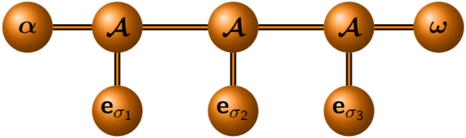
p.c. Guillaume Rabusseau

Second-Order RNN

Hidden state is computed by $\mathbf{h}_t = g\left(\mathcal{W} \times_2 \mathbf{x}_t \times_3 \mathbf{h}_{t-1}\right)$



The computation of a finite-state machine is very similar!



where $\mathcal{A} \in \mathbb{R}^{n \times \Sigma \times n}$ defined by $\mathcal{A}_{:,\sigma,:} = \mathbf{A}^{\sigma}$

p.c. Guillaume Rabusseau

Theorem (Rabusseau et al 2019)

Weighted FSA are expressively equivalent to second-order linear RNNs (linear 2-RNNs) for computing functions over sequences of discrete symbols.

Theorem (Merrill 2019)

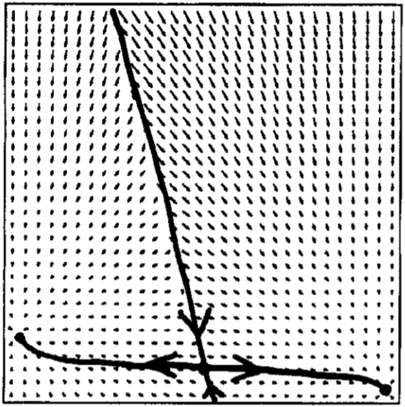
RNNs asymptotically accept exactly the regular languages

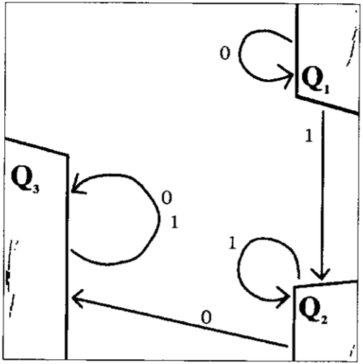
Theorem (Casey 1996)

A finite-dimensional RNN can robustly perform only finite-state computations.

Theorem (Casey 1996)

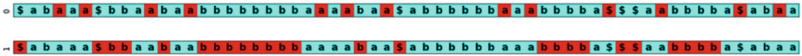
An RNN with finite-state behavior necessarily partitions its state space into disjoint regions that correspond to the states of the minimal FSA





Analyzing Specific Neuron Dynamics

- RNN with only 2 neurons in its hidden state trained on “Even-A" language.
- Input: stream of strings separated by $ symbol
- Neuron 1: all even as, and $ symbol after a rejected string
- Neuron B: all b’s following even number of a’s, and $ after an accepted string.

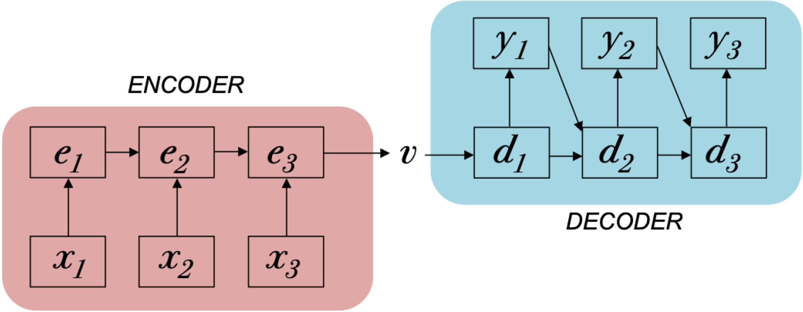


---

p.c. Oliva & Lago-Fernàndez 2019

RNN Encoder-Decoder and Transducers

- **Function**: *Given string* w, *generate* $f(w) = v$
  - $=$ accepted pairs of input & output strings
  - Computed by different classes of grammars (**transducers**)
- Recurrent encoder maps a sequence to $v \in \mathbb{R}^n$, recurrent decoder language model conditioned on v (Sutskever et al. 2014)
- How expressive are they?



Our idea: Use functions that copy!

(1) Total reduplication = unbounded copy (∼83%)

a. wanita→wanita∼wanita
'woman'→'women' (Indonesian)

(2) Partial reduplication = bounded copy (∼75%)

a. C: gen→g∼gen
'to sleep'→'to be sleeping' (Shilh)

b. CV: guyon→gu∼guyon
'to jest'→'to jest repeatedly' (Sundanese)

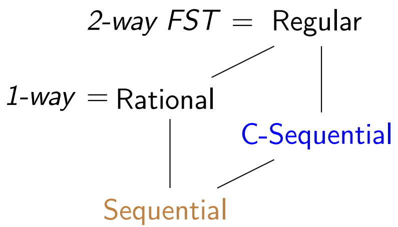
c. CVC: takki→ tak∼takki
'leg'→'legs' (Agta)

d. CVCV: banagaɲu→bana∼banagaɲu
'return' (Dyirbal)

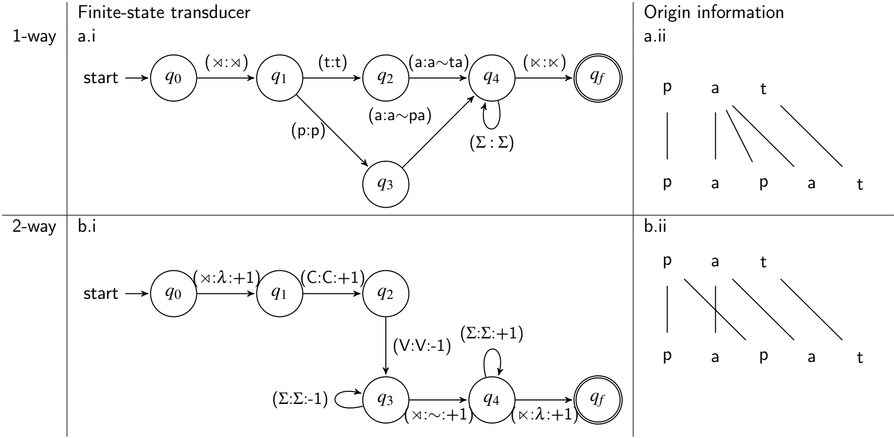
Subregular computing of reduplication

- Why reduplication (Red)?
  - inhabits **sub**classes of **regular** string-to-string functions
  - computed by restricted types of **Finite-State Transducers**

1. **1-way FST**: reads input once in one direction
   ∼ computes Rational functions
   e.g., Sequential functions like partial Red
2. **2-way FST**: reads multiple times, moves back and forth
   ∼ computes Regular functions
   e.g., Concatenated-Sequential functions like partial & total Red



1-way and 2-way Finite-State Transducers



Learning Reduplication

Reduplication is *provably* learnable in polynomial time and data (Chandlee et al. 2015; Dolatian and Heinz 2018)

RNNs with segmental inputs cannot be trained as reduplication acceptors (Gasser 1993; Marcus et al. 1999)

- Recognizing reduplication requires the comparison of static subsequences - difficult for an RNN to store

Encoder-Decoders learn reduplication with a fixed-size reduplicant in a small toy language (Prickett et al. 2018)

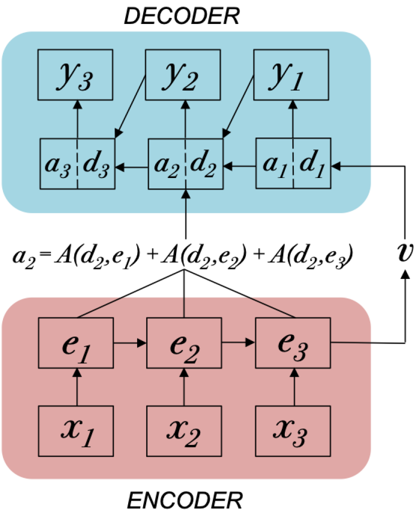
- Generalizable to novel segments and sequences
- Generalization to novel lengths not tested, computable by 1-way FST that uses featural representations

Recurrence

- **Recurrence relation:** The function relating hidden states in the encoder and decoder RNNs - affects practical expressivity of network
- Two types of recurrence tested:
  - **sRNN** - t^{th} state is a nonlinear function of the t^{th} input and state $t-1$ (Elman 1990)
  - **GRU** - t^{th} state is a linear function of three functions (gates) of the t^{th} input and state $t-1$ (Cho et al. 2014)
- Saturating nonlinearities ($tanh$) - sRNNs and GRUs cannot count with finite precision (Weiss et al. 2018)
- LSTM is supra-regular, we are testing necessary properties of RNN and GRU, which are finite-state (Merrill 2019)

Attention

- In standard ED, the encoded representation is the only link between the encoder and decoder
- **Global attention** allows the decoder to selectively pull information from hidden states of the encoder (Bahdanau et al. 2014)
- **FLT Analog**: 2-way FST has full access to the input by moving back and forth



Test data

- Input-output mappings generated with 2-way FSTs from RedTyp database[1]
  1. Initial-CV tasgati→ta∼tasgati
     Fixed-size reduplicant
  2. Initial two-syllable (C*VC*V)
     tasgati→tasga∼tasgati
     Onset maximizing, fixed over vowels
  3. Total
     tasgati→tasgati∼tasgati
     Variably sized reduplicant
- 10,000 generated for each language, 70/30 train/test split
- Minimum string length 3 - maximum string length varied
- Alphabet of 10, 16, or 26 characters
- Boundary symbols (∼) are not present

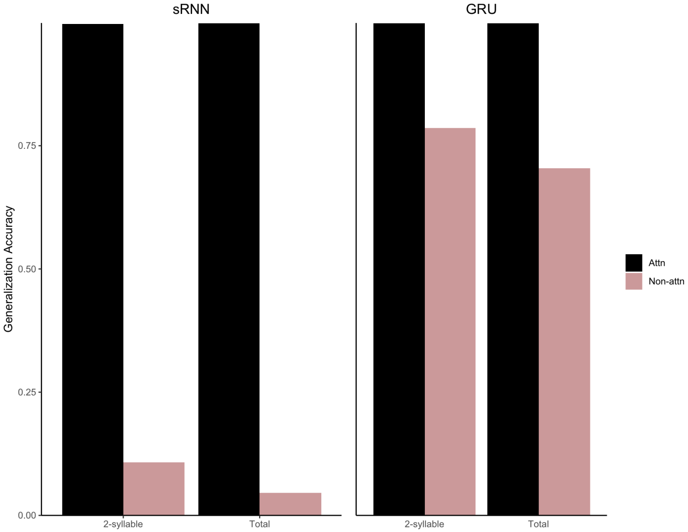
[1] Dolatian and Heinz (2019); also available on GitHub

Experiment 1

- Interaction between reduplication type, recurrence, and attention
  - Total and partial (two-syllable) reduplication
  - sRNN and GRU with and without attention
- Max string length: 9
- 10 symbols alphabet

Attention should improve function generalization across reduplication types and recurrence relations

Experiment 1

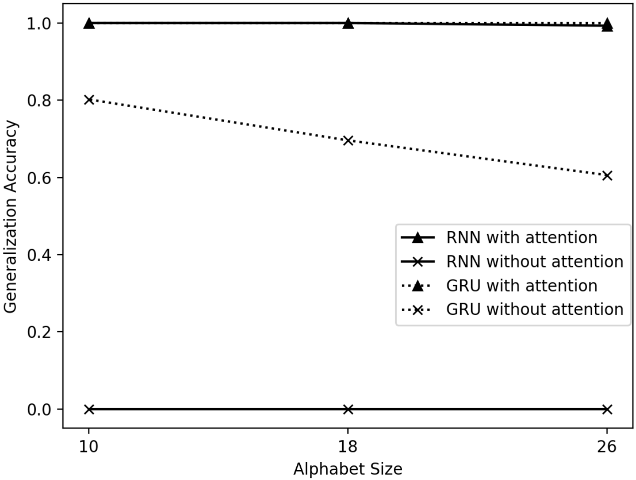


Experiment 2

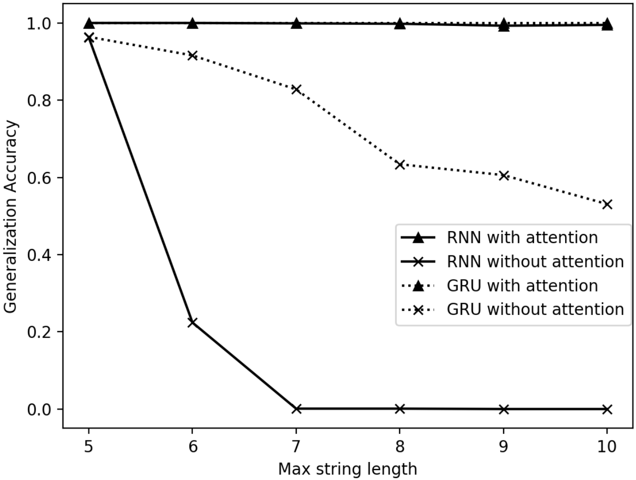
- Effects of alphabet size and range of permitted string lengths
- CV reduplication only
- sRNN/GRU $\times$ attention/non-attention $\times$ 3 alphabet sizes $\times$ 7 length ranges

Network generalization while learning a general reduplication function should be invariant to language composition

Experiment 2



Experiment 2

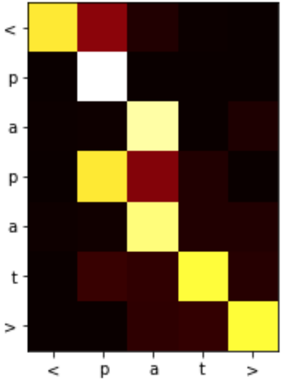


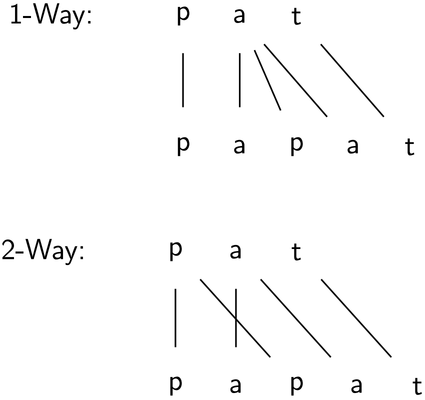
Discussion

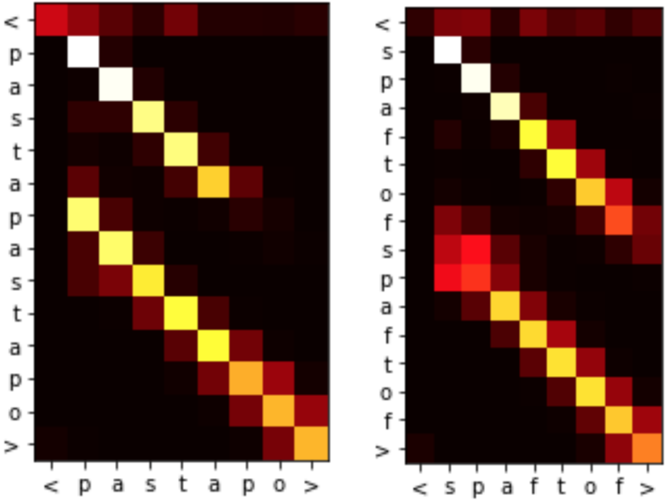
- Networks with global attention learn and generalize all types of reduplication and seem robust to string length and alphabet size
- sRNNs without attention show slightly better generalization of partial reduplication than total reduplication
  - Confound with less attested reduplicant lengths or a bias preferring the regular pattern?
- GRUs perform better than sRNNs across all conditions
  - Without attention not robust to length/alphabet - likely learning heuristics that capture most data rather than a general function

Networks that cannot see material in the input multiple times cannot learn generalizable reduplication

Attention and Origin Semantics






< p a s t a p a s t a p o >
< p a s t a p o >
< s p a f t o f s p a f t o f >
< s p a f t o f >

Summary

1. **Why use reduplication functions?**
   - properties define fine-grained subregular function classes
   - Allows us to test the generalization capacity of neural nets
2. **Expressivity of attention**
   - Attention is necessary and sufficient for robustly learning and generalizing reduplication functions using Encoder-Decoders
3. **FST approximations**
   - Non-attention networks are limited to a single input pass, approximating 1-way FST
   - Attention networks can read the input again during decoding, approximating 2-way FST,
4. **Attention weights and origin information**
   - Evidence for approximation comes from attention weights
   - IO correspondence relations mirror origin semantics of 2-way FST
5. **Next step**: trying more copying and non-copying functions

Main Points

1. Language is not just data you throw at a machine
2. Language is a fundamentally computational process uniquely learned by humans.
3. We can use core properties of language to understand how other systems learn.

References I

Avcu, Enes, Chihiro Shibata, and Jeffrey Heinz. 2017. Subregular complexity and deep learning. In *CLASP Papers in Computational Linguistics: Proceedings of the Conference on Logic and Machine Learning in Natural Language (LaML 2017), Gothenburg, 12 –13 June*, ed. Simon Dobnik and Shalom Lappin, 20–33.

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

Chandlee, Jane, Rémi Eyraud, and Jeffrey Heinz. 2015. Output strictly local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, 112–125. Chicago, USA.

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

Dolatian, Hossep, and Jeffrey Heinz. 2018. Learning reduplication with 2-way finite-state transducers. In *Proceedings of Machine Learning Research: International Conference on Grammatical Inference*, ed. Olgierd Unold, Witold Dyrka, , and Wojciech Wieczorek, volume 93 of *Proceedings of Machine Learning Research*, 67–80. Wroclaw, Poland.

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

References II

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

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

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

Merrill, William. 2019. Sequential neural networks as automata. In *Proceedings of the Deep Learning and Formal Languages workshop at ACL 2019*.

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

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

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

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

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. *CoRR* abs/1409.3215. URL http://arxiv.org/abs/1409.3215.

Weiss, Gail, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision rnns for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 740–745.