Investigating what Transformers do with formal languages

Stepan Shabalin

Hopf Algebra Seminar

Introduction

- Neural networks have mostly overtaken the field of NLP
- Why do they work so well?
- Opening the black box
  - Converting the model into a human-interpretable form
  - Approach: see if a network can learn to recognize formal languages by training it on them
  - Approach: see if a network has learned a formal language from real-world data by looking at its circuits
  - Instead of passively observing the model to see if it works, intervene on it
  - Ultimate goal: don't just observe the performance (including the components). Replace the model with a human-interpretable one

Part 1: Expressivity of RNNs

- RNNs are the simplest neural networks for sequence processing
- Only variable is the hidden state
- Universal approximation theorems (RNNs can represent Turing machines), RNNs can efficiently represent stacks
- If the RNN learned a DFA, we can sometimes extract the language from it

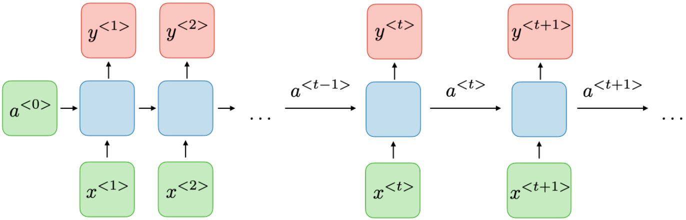


Figure 1: RNN architecture

Rule extraction for RNNs

- Early work
  - Clustering for finding DFAs: Omlin and Giles (1996)
    - Quantize neuron activations
    - Start from a state, run through all possible inputs and record transitions between quantized states
- Modern approaches use L* (Weiss et al. 2018)
  - Also uses quantization, but instead of BFS uses RNN for membership and equivalence queries
  - RNN:
    - Answers if a word is in the language
    - Checks if the DFA matches the RNN and provides counterexamples

Low-dimensional linear dynamics

- Susillo and Barak (2013)
  - Linearize RNN
    - $\dot{x} = F(x)$
    - $F(x^* + \delta x) = F(x^*) + F'(x^*)\delta x + \frac{1}{2}\delta x F''(x^*)\delta x + ..$
  - Linearization is interesting when
    - $F'(x^*)\delta x > |F(x^*)|$
    - $F'(x^*)\delta x > |F(x^*)| > |\frac{1}{2}\delta x F''(x^*)\delta x|$
  - Fixed points tick the first condition
  - "Kinetic energy" at a point. If we minimize this, we can find fixed points and slow points.
    - $q(x) = \frac{1}{2}|F(x)|^2$
- DFA parallel: fixed points are states

Low-dimensional linear dynamics

- Susillo and Barak (2013)
  - Linearize RNN
  - Find fixed points and slow points

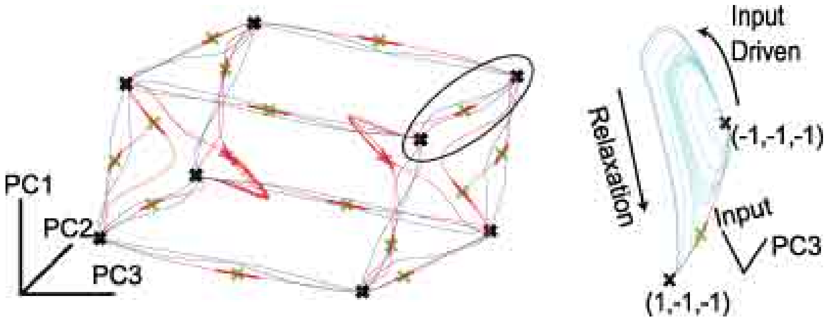
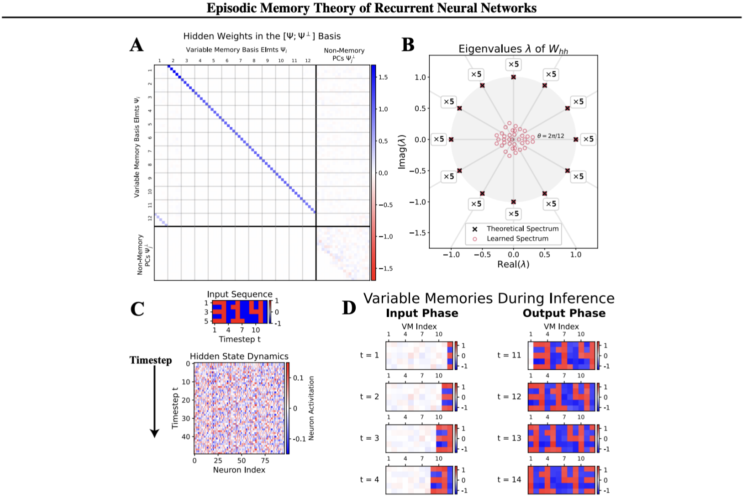


Figure 2: Dynamics in a PCA of a 3-bit RNN

Linear RNNs

- Elman RNN
  - $h(t+1) = tanh(W_{hh}h(t) + W_{uh}u(t))$
  - $y(t) = W_r h(t+1)$
- Reading and writing memories
  - Start with a linearized RNN
  - $|h(t)\rangle = (\xi^i_\mu \Phi^\mu_\nu (\xi^\dagger)^\nu_j |e_i\rangle\langle e^j|)|h(t-1)\rangle$
  - $|h(t+1)\rangle = (\Phi^\mu_\nu |\phi_\mu\rangle\langle\phi^\nu|)|h(t)\rangle$
  - $\Phi = \sum_{\mu=1}^{(N-1)\kappa} |\psi_\mu\rangle\langle\psi^{\mu+\kappa}| + \sum_{\mu=(N-1)\kappa}^{N\kappa} \Phi^\mu_\nu |\psi_\mu\rangle\langle\psi^\nu|$
  - $W_r = \sum_{\mu=(N-1)\kappa+1}^{N_\kappa} |e_{\mu-(N-1)\kappa}\rangle\langle\psi^\mu|$
  - $W_{uh} = |\psi_{(N-1)\kappa+j}\rangle\langle e^j|$
- Repeat copy task, which is just *ww*
  - $\Phi = \sum_{\mu=1}^{(s-1)\kappa} |\psi_\mu\rangle\langle\psi^{\mu+k}| + \sum_{\mu=(s-1)\kappa+1}^{s\kappa} |\psi_\mu\rangle\langle\psi^{\mu-(s-1)k}|$
  - $|h(s+t)\rangle = \sum_{\mu=1}^{s} u^i(\mu)|\psi_{[((\mu-t-1 \mod s)+1)\kappa+i]}\rangle$

Linear RNNs



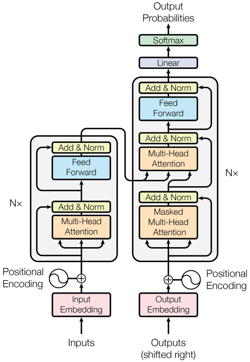
Episodic Memory Theory of Recurrent Neural Networks
A
Hidden Weights in the [Ψ; Ψ⊥] Basis
Variable Memory Basis Elmts Ψi
Non-Memory PCs Ψj⊥
B
Eigenvalues λ of Whh
Theoretical Spectrum
Learned Spectrum
θ = 2π/12
Imag(λ)
Real(λ)
C
Input Sequence
Timestep t
Timestep
Hidden State Dynamics
Neuron Index
Neuron Activation
D
Variable Memories During Inference
Input Phase
Output Phase
VM Index

Linear RNNs

- LRU: nonlinear RNNs can be replaced with linear RNNs with efficiency gains
  - See also: SRU, QRNN
  - See also: state-space models (which are convolutions)
- Justification in Koopman theory
  - "In essence, Koopman operator theory, provides the following guarantee: *any regular nonlinear dynamical system is representable by a linear RNN after proper nonlinear reparameterization of the inputs* — which can be performed by an MLP."
- Looks suspiciously Hopf Algebra-like

Part 2: Transformers

- Transformers are the new hotness
  - They are inherently more parallelizable
  - They are capable of using much more information from the input



Expressivity of transformers

- Dyck-k
- First-order logic with majority or counting
  - Doesn't seem particularly useful

RASP and Tracr

- RASP (Weiss et al. 2021)
    - Language for writing transformers by hand
    - Operations:
        - Elementwise operations - can use index
        - Select: create an attention matrix between queries and keys based on a boolean predicate
        - Aggregate: averages over values matching the predicate
- Tracr (Lindner et al. 2023) is an implementation of a compiler from a RASP-like DSL to Transformer weights

Learning Transformer Programs

- An algorithm for learning programs using Transformers
  - (It's in the title)
- A more discrete transformer
  - Use "variables"
  - Each attention head reads three variables and outputs one
  - Binary predicate matrix
  - Hard Attention
  - Gumbel Softmax for optimizing this mess
- Learns Dyck-1, Dyck2 and sorting

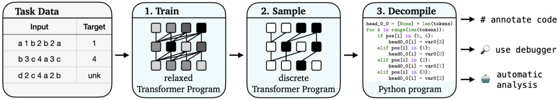


Figure 1: We design a modified Transformer that can be trained on data and then automatically discretized and converted into a human-readable program. The program is functionally identical to the Transformer, but easier to understand—for example, using an off-the-shelf Python debugger.

ACDC

- One way to interpret a model is to ask it to predict some feature at different layers and see its accuracy
- But this can only measure correlations, it doesn't tell us if the model is actually using the feature
- Instead, we can ablate circuits in the model and see how the accuracy changes
- Conmy, Arthur et al. (2023) - ACDC: recursive pruning of a model to keep only components causally important for a task
- This lets us isolate components in real-world transformers that perform a task
    - but the user has to interpret the results

Part 3: Case studies

- There are some toy problems for which if we understand how transformers solve them, we can understand how transformers solve other linguistic tasks
- We can also use these problems to test the expressivity of transformers (~~not recommended~~)

Case study: Sorting

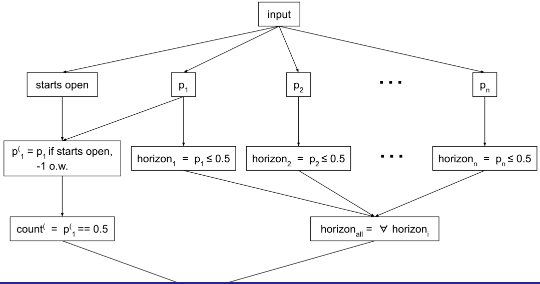
- Sequence-to-sequence task: encode a list of integers and output a sorted list
- Transformers can sort integer tokens with just one layer and one head!
- Looking inside, we find that each generated token attends to the input tokens that are greater than it

Case study: Modular addition

- Neural networks for group composition: addition modulo 113
  - An MLP mapping from a pair of tokens to a token
  - Grokking: validation accuracy rapidly increases long after the network finished overfitting
  - Embedding matrices are suspiciously low-rank and sparse under FFT
  - MLP activations are second-order polynomials of cosines of the input
- The networks use group theory. Input elements are mapped to matrices and multipled inside MLPs
  - $a, b \mapsto \rho(a), \rho(b) \mapsto \rho(a)\rho(b) = \rho(ab)$
  - It's possible to handcode the weights of the network with a formula
- Similar principels apply to permutations

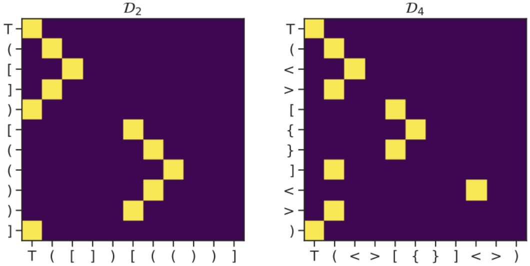
Case study: Dyck-k

- Chan et al. (2022):
  - Attribute components of a simple circuit. It didn't actually work.
- Ebrahimi et al. (2020):
  - Interesting attention map, but not mechanistic
- Bhattamishra et al. (2020):
  - Single-layer transformers can't learn a "reset" token for Dyck-1
- RASP/Tracr have their Dyck-k implementation, it's completely



Case study: Dyck-k

- Chan et al. (2022):
  - Attribute components of a simple circuit. It didn't actually work.
- Ebrahimi et al. (2020):
  - Interesting attention map, but not mechanistic
- Bhattamishra et al. (2020):
  - Single-layer transformers can't learn a "reset" token for Dyck-1
- RASP/Tracr have their Dyck-k implementation, it's completely



Remaining questions

- RNNs can represent Dyck-(k, m) in $O(m \log k)$.
  - Interesting fact: the syntactic monoid of Dyck-k is the bicyclic monoid. It has no faithful finite-dimensional representations.
  - Is there perhaps some connection between the representations of the syntactic monoid and the solution the network learns?
- How is the Dyck-k network represented at neuron level?
- LRUs and state-space models are basically convolutions. Do they have a Hopf algebraic interpretation?
- Does the shift result from EMT correspond to some operation? Is there something deeper here than just reading off memories and overwriting them?
- What if the modular addition transformer used bilinear activations?
- Is there a way to improve on the Tracr solution for Dyck-k?

References

- Siegelmann, Hava T., and Eduardo D. Sontag. "On the computational power of neural nets." Proceedings of the fifth annual workshop on Computational learning theory. 1992.
- Hewitt, John, et al. "RNNs can generate bounded hierarchical languages with optimal memory." arXiv preprint arXiv:2010.07515 (2020).
- Omlin, Christian W., and C. Lee Giles. "Extraction of rules from discrete-time recurrent neural networks." Neural networks 9.1 (1996): 41-52.
- Weiss, Gail, Yoav Goldberg, and Eran Yahav. "Extracting automata from recurrent neural networks using queries and counterexamples." International Conference on Machine Learning. PMLR, 2018.

References

- Karuvally, Arjun, Peter DelMastro, and Hava T. Siegelmann. "Episodic Memory Theory of Recurrent Neural Networks: Insights into Long-Term Information Storage and Manipulation." Topological, Algebraic and Geometric Learning Workshops 2023. PMLR, 2023.
- Sussillo, David, and Omri Barak. "Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks." Neural computation 25.3 (2013): 626-649.
- Orvieto, Antonio, et al. "Resurrecting recurrent neural networks for long sequences." arXiv preprint arXiv:2303.06349 (2023).
- Lei, Tao, et al. "Simple recurrent units for highly parallelizable recurrence." arXiv preprint arXiv:1709.02755 (2017).

References

- Strobl, Lena, et al. "Transformers as Recognizers of Formal Languages: A Survey on Expressivity." arXiv preprint arXiv:2311.00208 (2023).
- Bhattamishra, Satwik, Kabir Ahuja, and Navin Goyal. "On the ability and limitations of transformers to recognize formal languages." arXiv preprint arXiv:2009.11264 (2020).
- Weiss, Gail, Yoav Goldberg, and Eran Yahav. "Thinking like transformers." International Conference on Machine Learning. PMLR, 2021.
- Lindner, David, et al. "Tracr: Compiled transformers as a laboratory for interpretability." arXiv preprint arXiv:2301.05062 (2023).

References

- Friedman, Dan, Alexander Wettig, and Danqi Chen. "Learning Transformer Programs." arXiv preprint arXiv:2306.01128 (2023).
- Conmy, Arthur, et al. "Towards automated circuit discovery for mechanistic interpretability." arXiv preprint arXiv:2304.14997 (2023).

References

- Ebrahimi, Javid, Dhruv Gelda, and Wei Zhang. "How Can Self-Attention Networks Recognize Dyck-n Languages?." arXiv preprint arXiv:2010.04303 (2020).
- Chan et al. "Causal scrubbing: results on a paren balance checker." LessWrong (2022).
- Mateusz Bagiński and Gabin Kolly. "One Attention Head Is All You Need for Sorting Fixed-Length Lists." Apart Research Alignment Jam #4 (Mechanistic Interpretability) (2023).
- Chughtai, Bilal, Lawrence Chan, and Neel Nanda. "A toy model of universality: Reverse engineering how networks learn group operations." arXiv preprint arXiv:2302.03025 (2023).

Empty slide