arXiv:2402.18510v3 [cs.LG] 10 May 2024

RNNs are not Transformers (Yet): The Key Bottleneck on In-context Retrieval

Kaiyue Wen[1*] **Xingyu Dang**[1*] **Kaifeng Lyu**[2†]
[1]Institute for Interdisciplinary Information Sciences, Tsinghua University
[2]Department of Computer Science & Princeton Language and Intelligence, Princeton University
{wenky20,dangxy20}@mails.tsinghua.edu.cn
klyu@cs.princeton.edu

Abstract

This paper investigates the gap in representation powers of Recurrent Neural Networks (RNNs) and Transformers in the context of solving algorithmic problems. We focus on understanding whether RNNs, known for their memory efficiency in handling long sequences, can match the performance of Transformers, particularly when enhanced with Chain-of-Thought (CoT) prompting. Our theoretical analysis reveals that CoT improves RNNs but is insufficient to close the gap with Transformers. A key bottleneck lies in the inability of RNNs to perfectly retrieve information from the context, even with CoT: for several tasks that explicitly or implicitly require this capability, such as associative recall and determining if a graph is a tree, we prove that RNNs are not expressive enough to solve the tasks while Transformers can solve them with ease. Conversely, we prove that adopting techniques to enhance the in-context retrieval capability of RNNs, including Retrieval-Augmented Generation (RAG) and adding a single Transformer layer, can elevate RNNs to be capable of solving all polynomial-time solvable problems with CoT, hence closing the representation gap with Transformers.[3]

1 Introduction

Transformer models (Vaswani et al., 2017) have become the dominant choice of the backbone for large language models (LLMs). The core component of Transformers is its self-attention module, which allows the model to route information densely across the entire sequence. However, this design leads to high inference costs for modeling long sequences, including a memory cost that is at least linear in the sequence length due to the need for maintaining intermediate attention keys and values for each token, and a time cost quadratic in the sequence length for computing the attention score for each pair of tokens.

Recently, Recurrent Neural Networks (RNNs) have been an increasingly popular choice in sequence modeling tasks due to their ability to maintain a memory size constant in sequence length during inference, thus being more memory efficient than Transformers. Katharopoulos et al. (2020) showed that Transformers with a special type of kernelized linear attention can be expressed as RNNs. Gu et al. (2022) took a different path to design RNNs by structuring latent states as State Space Models (SSMs) from control theory. These ideas have led to a series of development of modern RNNs, including RWKV (Peng et al., 2023), RetNet (Sun et al., 2023), and Mamba (Gu & Dao, 2023). Most notably, Mamba can achieve competitive performance with Transformers on several sequence modeling tasks with linear time and constant memory complexity in sequence length.

Can RNNs replace Transformers yet? The rise of these modern RNNs has led to an interest in understanding their limitations. A recent work by Arora et al. (2023) showed that an important family of RNNs, input-independent gating SSMs, are empirically inferior to Transformers in a task that has a long history in artificial intelligence, *associative recall* (AR) (Willshaw et al., 1969; Hopfield, 1982; Hinton & Anderson, 2014): Given a series of key-value pairs as a string, the model is required to recall the value given a key. On the theory side, Sanford et al. (2023) and Jelassi et al.

[*]Equal contribution
[†]Corresponding author
[3]Code is available at https://github.com/dangxingyu/rnn-icrag

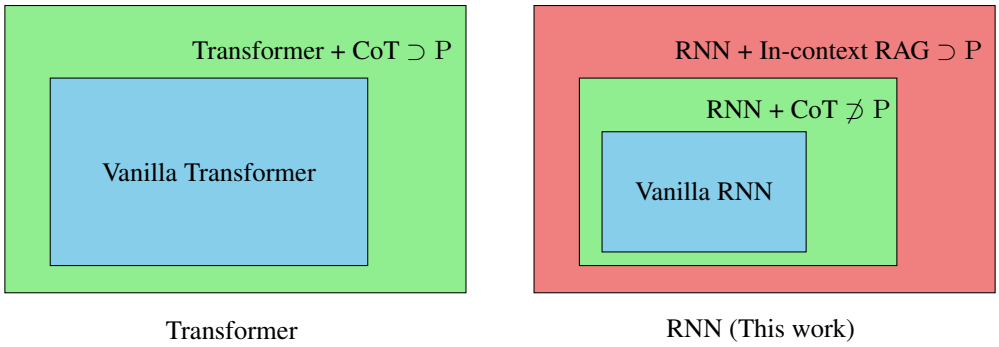


Figure 1: **Hierarchy of Representation Power**. While RNN with chain-of-thought (CoT) with $O(\log n)$ bit memory provably has strictly stronger representation power than RNN without CoT under mild complexity assumptions (Theorem 4.1), it is still exponentially weaker than Transformer with CoT in representing solutions to algorithmic problems (Theorem 4.7). We proceed to show that the incapability of RNNs in In-context Retrieval is the root cause of the gap and propose two forms of In-context Retrieval Augmented Generation (In-context RAG) to close the gap by illustrating their power to simulate any polynomial-time Turing machines (Theorems 5.4 and 5.8).

(2024) demonstrated that constant-memory RNNs do not have sufficient representation power to solve the tasks of averaging a given subset of input vectors (q-sparse averaging) and repeating the input sequence (copying), respectively, while there exist shallow Transformers that can solve these tasks.

However, the above results do not exclude the possibility that enhancing RNNs with additional prompting techniques or minor architectural changes may close the gap with Transformers. In fact, Transformers themselves are not perfect either and may need additional techniques at inference time to perform well on certain tasks. As a notable example, *Chain-of-Thought* (CoT) (Wei et al., 2023), a prompting technique that asks the model to generate a series of intermediate tokens before giving the final answer, has been known to be crucial for Transformers to perform well on tasks that require mathematical or algorithmic reasoning. Feng et al. (2023); Li et al. (2024) explained this from the perspective of representation power: Transformers alone do not have sufficient representation power to solve problems beyond a certain circuit complexity class (TC^0), but with CoT, they can even simulate any polynomial-time Turing machines.

The effectiveness of CoT on Transformers naturally leads to the following question:

Can similar enhancements, such as adopting CoT, improve RNNs so that they can be on par with Transformers?

Our Contributions. This paper answers the above question from theory by examining various ways to close the gap in the representation powers of RNNs and Transformers on algorithmic problems. Through a series of lower and upper bound results, we show that CoT improves the representation power of RNNs, but to close the gap with Transformers, CoT alone is not enough to overcome a key bottleneck of RNNs: their inability to retrieve information from the context, which we call *in-context retrieval* for short. We further illustrate that addressing this in-context retrieval bottleneck is sufficient to close this gap: RNNs can solve all polynomial-time solvable problems if adopting techniques to enhance the in-context retrieval capability, including involving *Retrieval-Augmented Generation* (RAG) and appending a single Transformer layer. Our main contributions are listed as follows:

1. **CoT improves RNNs but cannot close the representation gap with Transformers.** (Section 4)
   - On the positive side, we prove that CoT makes RNNs strictly more expressive under mild assumptions from circuit complexity.
   - On the negative side, we show that adopting CoT is not enough to close the representation gap between RNNs and Transformers: the memory efficiency of RNNs fundamentally limits their ability to perform in-context retrieval, even with CoT. This point is made concrete by proving that RNNs with CoT cannot solve a set of fundamental algorithmic problems that directly ask for in-context retrieval, including associative recall. We further exemplify that in-context retrieval can be implicitly required in tasks that appear unrelated, by proving the inability of RNNs to solve the classic problem of determining if a graph is a tree (IsTree).
   - On the other hand, we prove that Transformers have the representation power to solve many of the above tasks with ease, including IsTree. Moreover, Transformers with CoT can even simulate RNNs with CoT efficiently, with only a small multiplicative factor in the number of parameters.

   Technically, the key insight for the first separation is that RNNs without CoT is a shallow circuit, while RNNs with CoT can be an exponentially deeper circuit. But on the negative side, RNNs are so memory efficient that they can trigger streaming lower bounds (Sanford et al., 2023), especially for problems that require in-context retrieval.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 2

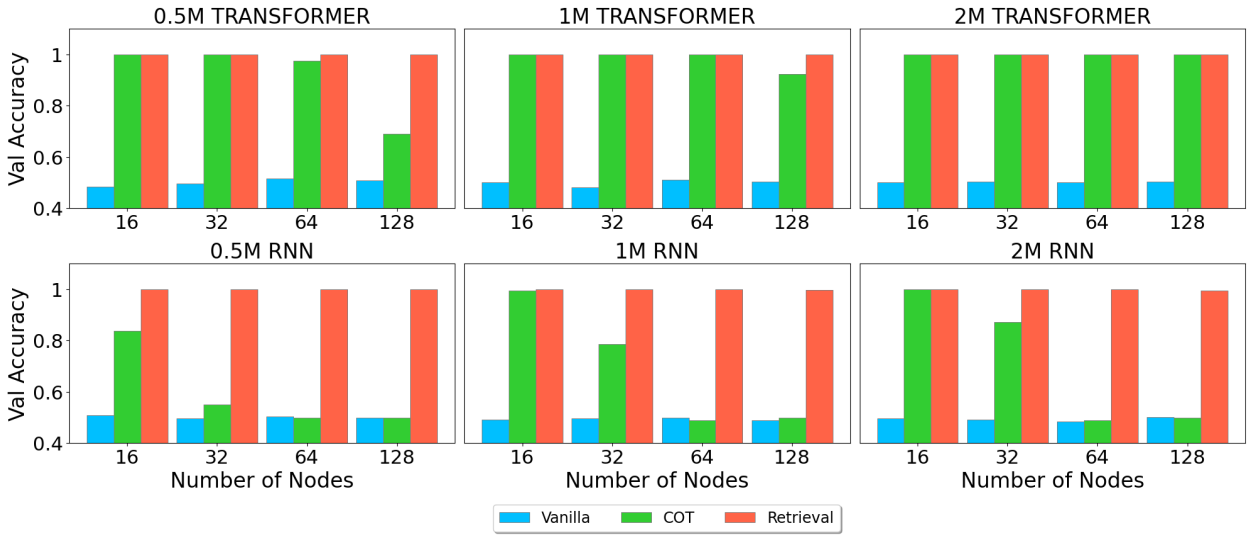


Figure 2: We train RNNs (Mamba) and Transformers (LLaMA 2 Touvron et al. (2023)) with a frozen word embedding and decoding head of three different model sizes (0.5M, 1M, 2M) on IsTree with three different sizes of graph (16, 32, 64) under three different setups. **Vanilla** means the model directly predicts the label. **COT** means the model will generate a chain-of-thought process based on DFS (see Algorithm 1) before prediction. **Retrieval** means the model will generate the chain of search queries and reasoning before prediction (see Algorithm 2). We observe that (1) Both Transformer and RNNs can't solve the IsTree question without a chain of thought; (2) RNNs' performance with chain-of-thought decays quickly when the number of nodes increase, which is consistent with our theory; (3) All models reach almost perfect accuracy when enhanced with retrieval.

2. **Enhancing the in-context retrieval capability of RNNs can close the representation gap.** (Section 5)
   - We prove that allowing RNNs to invoke function calls to perform a certain primitive of in-context retrieval is sufficient to boost their representation power to solve all polynomial-time solvable problems with CoT, hence closing the representation gap between RNNs and Transformers.
   - Alternatively, as one layer of transformer is sufficient to perform many in-context retrieval operations, we prove that implicitly enhancing the in-context retrieval capability of RNNs by adding just one transformer layer at the end of the architecture is also sufficient to close the representation gap.

   Technically, the key insight for the above upper bounds is that RNN can focus on the local reasoning steps and use the in-context retrieval module to adaptively fetch the relevant information from the context.

2 Related Works

State Space Machines and Linear Transformers. There has been a recent surge of interest in state space machines and (kernalized) linear transformers (Gu et al., 2022; Katharopoulos et al., 2020; Peng et al., 2023; Sun et al., 2023; Gu & Dao, 2023; Fu et al., 2023; Poli et al., 2023; Luo et al., 2021; Peng et al., 2021; Wang et al., 2020), which are a class of models that combine the parallelizability of the Transformer with the memory efficiency of the RNN. These models can process both a sequential and a recurrent form, and can use the former for fast parallelizable training and the latter for memory-efficient inference. However, these models are still empirically inferior to the Transformer in terms of performance. Our work investigates the reasons behind this gap and proposes to close it by enhancing the in-context retrieval capability.

Chain of Thought (CoT). Chain of thought (Wei et al., 2023; Nye et al., 2021; Kojima et al., 2023; Wang & Zhou, 2024) is an augmentation to the Transformer, that allows it to solve more complex reasoning tasks by generating a reasoning process before outputting the answer. It has been shown that Transformers with CoT provably have more expressive power than the original Transformer without CoT (Feng et al., 2023; Li et al., 2024). However, the expressive power of RNNs with CoT has not yet been systematically studied. Theorem F.1 in Feng et al. (2023) shows that RNN cannot output a particular format of CoT for evaluating arithmetic expressions and solving linear equations while Transformers with the same amount of parameters can. Concurrent work (Yang et al., 2024) discovers that linear Transformers, a special class of RNNs, are not able to solve some dynamic programming problems with CoT, unless the number of parameters grows with the length of the input. One high-level message our work conveys is similar to

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 3

theirs: RNNs have limited representation power to perform reasoning with CoT. However, we show that such limitation is not specific to the output format or architecture and apply tools from streaming complexity to prove lower bounds on a broader range of tasks and memory-efficient architectures.

Streaming Algorithms. Our lower bound leverages the technique in streaming algorithms. Streaming algorithms are algorithms that take constant (typically just 1) pass over the input and use sublinear space, hence including RNNs with fixed state space as a special case. Works in streaming algorithms date back to the 1980s (Munro & Paterson, 1980) and have been formalized and popularized in the 1990s (Alon et al., 1996) due to the need to process large data streams. The lower bound in our work is a direct application of the technique in streaming algorithms to the study of RNNs and we mainly consider the streaming algorithms for (1) indexing the input (Munro & Paterson, 1980) and (2) determining whether the input is a tree (Henzinger et al., 1998).

Retrieval Augmented Generation. Our work proposes to use retrieval augmentation to close the representation gap between RNNs and Transformers. This is consistent with the recent trend of retrieval augmented generation (Guu et al., 2020; Borgeaud et al., 2022; Rubin & Berant, 2023). Empirically, retrieval augmented generation has been shown to improve the performance of recurrent models in various tasks (Kuratov et al., 2024; Akyürek et al., 2024) and our work provides a theoretical foundation for this phenomenon. Our work also shows that an attention layer can be used to simulate the retrieval process, which is consistent with the finding that attention can improve the performance of RNNs (Vaswani et al., 2017; Arora et al., 2023; Park et al., 2024; Peng et al., 2023; Hao et al., 2019). It has also been shown empirically that attention can be used to simulate complex retrieval process (Jiang et al., 2022).

Comparison Between Transformers and RNNs (Without CoT). A line of works focused on the comparison between RNNs and Transformers in terms of recognizing or generating formal languages (Bhattamishra et al., 2020; Hahn, 2020; Merrill et al., 2022). These works show that the lack of recurrent structure in Transformers makes them fail to recognize some formal languages that RNNs can recognize. However, Liu et al. (2023); Yao et al. (2023); Hao et al. (2022) show that such limitation can be mitigated when we consider bounded length of input or bounded grammar depth. Our work differs from these works in that we consider the expressive power of RNNs and Transformers with CoT and show that in this case, the gap between RNNs and Transformers is one-sided (Theorem 4.8).

Prior work (Arora et al., 2023) has shown that input-independent gating SSMs are inferior to Transformers in the task called *associative recall* (Willshaw et al., 1969; Hopfield, 1982; Hinton & Anderson, 2014). The task requires the model to recall a previously seen pattern given a partial input. They show that input-dependent gating SSMs have better performance in associative recall and also propose a hybrid architecture that combines input-independent state space machines with attention to achieve better performance. Our work differs from this work in the following ways: (1) Our work studies associative recall from a theoretical perspective and proves formal lower bounds on the memory size of RNNs necessary for solving associative recall and other retrieval tasks; (2) We also study hybrid architectures but we provide a proof that appending a single Transformer layer to RNNs can make them expressive enough; (3) Our theory applies to not only input-independent gating SSMs but also all RNNs with $o(n)$-bit memory.

Prior work (Jelassi et al., 2024) proves a representation gap between RNNs and Transformers in repeating a long sequence, which can be seen as a retrieval task. They show that RNNs have difficulty performing the task due to their limited memory. Our work further proves that RNNs are limited in solving many other retrieval tasks, even with CoT. Technically, a key ingredient in their proof is a counting argument on the output sequence to show a limited memory size is not enough to produce too many different output sequences, but our proof can handle retrieval tasks that only require outputting a single token.

Notably, Sanford et al. (2023) apply communication complexity to prove circuit size or memory size lower bounds for RNNs and Transformers on the task of sparse averaging. Sanford et al. (2024) extend this technique to another task called hop_k, a generalization of the associative recall task. Our technique is similar to theirs since our proof is also based on communication complexity. But we consider a broader range of tasks including seemingly irrelevant reasoning tasks such as IsTree, and further explore various ways to close the representation gap.

Representation Theory of RNNs. Another line of works (Li et al., 2021, 2022; Alberti et al., 2023) studies the universal approximation power of RNNs. They show that the upper bound of the approximation power of linear RNNs will be constrained by the dimension of the hidden states. Their works on the high level are consistent with our findings but are not directly comparable because we are considering finite precision compute models with the assistance of CoT or In-context RAG.

3 Preliminaries

We introduce the definitions that are necessary for understanding our results and defer other definitions to Appendix A.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 4

Vocabulary and Embeddings. A vocabulary V is a finite set of tokens. A word embedding for a token $v \in V$ is a vector $W_v^{(E)} \in \mathbb{R}^d$ that represents the token, and a position embedding for the k-th token in a sequence is a vector $W_k^{(P)} \in \mathbb{R}^d$ that represents the position. Given a sequence of tokens $\mathcal{S}$, an embedding function $\text{Emb}(\mathcal{S})$ maps each token to a vector in $\mathbb{R}^d$ by mixing word and position embeddings, resulting in a sequence of vectors. To ease our comparison between RNNs and Transformers, in this paper, we assume fixed word and position embeddings and do not learn them during training. See Appendix A.3 for the formal definitions. This is common practice and has been used in many previous works studying the theory of Transformers (Li et al., 2023; Tian et al., 2023).

Many of the problems we study involve natural numbers up to n, where the input sequence length is linear in n. For simplicity, we assume the vocabulary contains $[n] = \{1, \ldots, n\}$ and the word embedding for i is defined as $W_i^{(E)} = iw_1$, where w_1 is the first coordinate vector. But in practice, the vocabulary size does not increase with n and numbers may be tokenized into a few tokens according to their decimal representations. We note that our results can be easily extended to this more practical case since our lower bounds do not rely on the specific form of the vocabulary and embeddings and for the upper bounds, our embedding can be easily represented by a few RNN or Transformer layers.

Numerical Precision. We will consider computation models with fixed numerical precision in this paper. We will use p to denote the precision of the number of bits to represent real numbers and use $\mathbb{R}_p$ to denote the set of all real numbers that can be represented by p-bit floating point numbers. We defer the details to Appendix A.2. We will assume $p = O(\log n)$ in this paper and state the constant explicitly when necessary. This is a common assumption when studying the finite precision neural networks (Feng et al., 2023; Merrill & Sabharwal, 2023).

Language Modeling. We use V^* and V^+ to denote the set of all finite sequences and all non-empty finite sequences of tokens in V, respectively. We study language models that can predict the next token given a prefix of tokens. For this, we define a language model (LM) as a function $M : V^+ \to \mathbb{P}_V$ that maps a non-empty sequence to a probability distribution over the next token, where $\mathbb{P}_V$ is the probability simplex over V. We specifically study the case where the language model is realized by deep neural networks: first map the input sequence $\mathcal{S}$ into a sequence of embeddings $\text{Emb}(\mathcal{S})$, and then apply a neural network, such as a Transformer or RNN, to process the embeddings and output the probability distribution. We would call a series of parameterized models with increasing input size a *family* of models.

Transformer. We will first define the Transformer architecture used in the theoretical analysis in this paper.

Definition 3.1 (Transformer Block)**.** Let $X \in \mathbb{R}^{d \times l}$ be the input matrix, where l is the sequence length. The output of a Transformer block f is defined as:

$$f(X) = X + \mathcal{A}(X) + g(X + \mathcal{A}(X)),$$
$$\mathcal{A}(X) = \sum_{h=1}^{H} W^{(V,h)} X \text{softmax}\left(\frac{\left(W^{(K,h)}X\right)^\top W^{(Q,h)}X}{\sqrt{d}} + C\right), \tag{1}$$

where g is a column-wise ReGLU feed-forward network [4] with width w and output dimension d, $\mathcal{A}$ is the scaled dot-product attention, softmax is the column-wise softmax function, $W^{(K,h)}$, $W^{(Q,h)}$, $W^{(V,h)}$ are the learnable parameters and H is the number of heads, and $C = \begin{bmatrix} 0 & 0 & \ldots & 0 \\ -\infty & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ -\infty & -\infty & \ldots & 0 \end{bmatrix} \in \mathbb{R}^{l \times l}$ is a mask to prevent the attention from attending to future tokens.

In the context of language modeling, given a sequence of tokens $\mathcal{S}$, a Transformer $T(\mathcal{S})$ is defined as:

Definition 3.2 (Transformer)**.** Let $\mathcal{S} \in |V|^l$ be the tokenized input sequence, the output of a Transformer is defined as:

$$T(\mathcal{S}) = \text{softmax}\left(W^{(E)}\left(f_L\left(\ldots f_1\left(\text{Emb}\left(\mathcal{S}\right)\right)\right)\right)\right)_{:,l}. \tag{2}$$

where softmax is the column-wise softmax function, f_i is the i-th Transformer block. We will call the i-th Transformer block the i-th layer of the Transformer and denote its feed-forward layer and attention layer as g_i and $\mathcal{A}_i$ respectively.

Recurrent Neural Networks Recently there has been a lot of interest in the linear-time Transformer, which replaces the full-attention calculation with linear-time alternatives. These variants are mostly special forms of recurrent neural networks (RNNs) that are parallelizable. Here we define a general form of RNNs containing all of these variants; hence, our analysis will apply to all of them.

[4] ReGLU means $\sigma(x) = \text{ReLU}(W_1 x + b_1) \otimes (W_2 x + b_2)$, this is a surrogate for the commonly used SwiGLU activation and allows the model to perform multiplication of two coordinates.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 5

Definition 3.3 (RNN)**.** An RNN architecture is characterized by two functions: state transition function $\mathbf{t} : \Theta \to \left(\mathbb{R}_p^\Lambda \times \mathbb{R}_p^d \to \mathbb{R}_p^\Lambda\right)$ and output function $\mathbf{o} : \Theta \to \left(\mathbb{R}_p^\Lambda \to \mathbb{R}_p^d\right)$, where Λ is the dimension of the state and Θ is the parameter space. Let $\mathcal{S} \in |V|^l$ be the input sequence, the output of a recurrent neural network with parameter $\theta \in \Theta$ is defined as:

$$R_\theta(\mathcal{S}) = \text{softmax}\left(W^{(E)}\mathbf{o}_\theta(s_l)\right),$$
$$\forall k \in [l], s_k = \mathbf{t}_\theta(s_{k-1}, \text{Emb}(\mathcal{S})_{:,k}),$$

where $s_0 \in \mathbb{R}_p^\Lambda$ is a vector determined by θ and $W^{(E)}$ is the word embedding matrix. We will omit the subscript θ when it is clear from the context.

We can characterize the complexity of an RNN architecture with the following three measures,

1. Parameter size P: the number of bit of parameters determining $\mathbf{t}$ and $\mathbf{o}$.
2. State memory size M: the number of bits to record the state of the RNN, in this case, is $\Lambda \times p$.
3. Circuit size C: the number of bit-wise arithmetic operations needed to calculate $\mathbf{t}$ and $\mathbf{o}$.

We further constrain ourselves to RNNs with the following properties, which still contain all the modern variants of linear-time Transformers to the best of our knowledge.

Definition 3.4. We say that an RNN architecture is regular if $\mathrm{C} = \Theta(\mathrm{P})$, $\mathrm{P} = \Omega(\mathrm{M})$, and $\mathrm{P} = \text{Poly}(\mathrm{M})$.

We note that the above definition of RNNs is general enough to also contain many recent recurrent architecture using streaming context windows or finite KV cache (Xiao et al., 2023; Kuratov et al., 2024; Bulatov et al., 2022; Oren et al., 2024).

For our upper bound on RNNs, we will consider the following linear RNNs, which is a special case of regular RNNs.

Definition 3.5 (RNN block)**.** A Linear RNN block is defined as follows:

$$f(X) = X + \text{LU}(X) + g(X + \text{LU}(X)),$$

where g is a column-wise ReGLU feed-forward network with width w and output dimension d and LU is a linear unit, defined as

$$h_0 = 0, h_{:,t} = Ah_{:,t-1} + BX_{:,t}, \text{LU}(X_{:,1:l}) = h_{:,1:l}.$$

Definition 3.6 (Linear RNN)**.** A Linear RNN is a recurrent neural network

$$R(\mathcal{S}) = \text{softmax}\left(W^{(E)}\left(f_L\left(\ldots f_1\left(\text{Emb}\left(\mathcal{S}\right)\right)\right)\right)\right)_{:,l}. \tag{3}$$

where softmax is the column-wise softmax function, f_i is the i-th Linear RNN block. We will call the i-th Linear RNN block the i-th layer of the Linear RNN and denote its feed-forward layer and linear unit layer as g_i and LU_i respectively.

Language Models for Algorithmic Problems. An algorithmic problem is a problem that may be solved by an algorithm. In this paper, we focus on algorithmic problems $f : V^+ \to V_\text{A}$ that asks for computing $f(\mathcal{S}_\text{in})$ given a sequence of tokens $\mathcal{S}_\text{in}$ as the input, where V_A is the set of possible answers. We say that an LM M can (directly) solve an algorithmic task f if, given the sequence $\mathcal{S}_\text{in}$, the probability distribution $M(\mathcal{S}_\text{in})$ for the next token is peaked at the correct output token $f(\mathcal{S}_\text{in})$, i.e., $\arg\max_{j \in V} M(\mathcal{S}_\text{in})[j] = f(\mathcal{S}_\text{in})$.

Chain-of-Thought (CoT) reasoning is a technique that allows the LM to produce intermediate steps before the final output. We primarily concern with the ability of language models to solve algorithmic problems with CoT, since many algorithmic problems may be too challenging to solve in just one step (Feng et al., 2023; Li et al., 2024) and testing with CoT is now standard in the field (Wei et al., 2023; Nye et al., 2021; Kojima et al., 2023; Wang & Zhou, 2024). In our context, we say that an LM M can solve an algorithmic problem f with CoT if the following process terminates with a sequence ended with $f(\mathcal{S}_\text{in})$. First, let $\mathcal{S}_0 = \mathcal{S}_\text{in}$. For all $i \geq 0$, decode the next token $s_i^\text{next} = \arg\max_{j \in V} M(\mathcal{S}_i)[j]$ from M, and append it to the current sequence $\mathcal{S}_{i+1} = \mathcal{S}_i \oplus s_i^\text{next}$. If $s_i^\text{next} \in V_\text{A}$, then the process terminates with $\mathcal{S}_{i+1}$ with i steps of CoT; otherwise the process continues.

It is evident that if an LM can solve an algorithm problem with 0 steps of CoT, then an LM M can (directly) solve the problem. In this case, we also say that the LM can solve the problem without CoT.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 6

4 Can CoT improve the Representation Power of RNNs?

In this section, we aim to understand the representation power of RNNs with CoT. We first show the positive result that RNNs with CoT can solve tasks that are impossible for RNNs without CoT fixing the state size. We then proceed to understand whether CoT can make RNNs as expressive as Transformers. We show that, even with CoT, RNNs still struggle to solve problems that explicitly require in-context retrieval and this representation gap propagates to seemingly retrieval-irrelevant reasoning tasks such as IsTree. Finally, we show that this gap is indeed one-sided: there only exist tasks where Transformers require exponentially less parameters than RNNs, but not the other way around.

4.1 CoT Strictly Improves RNNs

On the positive side, we show that CoT broadens the representation power of RNNs under mild complexity assumption.

Theorem 4.1. *Assuming* $\mathrm{PSPACE} \not\subset \mathrm{P/Poly}$*, there exists a task with input length* n *that can be solved by a Linear RNN (Definition 3.6) family with* $\Theta(\log n)$ *bit memory with polynomial length CoT but cannot be solved by any regular RNN (Definition 3.4) family with* $\Theta(\log n)$ *bit memory without CoT.*

We note that this result is in the same vein as the recent work on the benefit of CoT for Transformers (Feng et al., 2023; Li et al., 2024), which shows that Transformer with constant size and $O(\log n)$ or constant precision can't simulate any polynomial size circuit family while Transformer with CoT can. Here we utilize a different complexity hypothesis, which intuitively states that not all polynomial space complexity problems (PSPACE) can be solved by a polynomial-size circuit family ($\mathrm{P/Poly}$). We show that this hypothesis implies that RNNs can benefit from CoT in solving reasoning tasks.

Proof Sketch. The intuition behind the proof is that RNNs without CoT are shallow circuits with size $\mathrm{Poly}(\log n)$. However, RNNs with CoT can simulate a Turing machine with $O(\log n)$ space perfectly within $\exp(O(\log n)) = \mathrm{Poly}(n)$ steps. Hence, we only need to show that there exists a problem with linear space complexity that cannot be solved by a polynomial-size circuit family and this can be induced by the (seemingly weaker) assumption that $\mathrm{PSPACE} \not\subset \mathrm{P/Poly}$ (see Lemma B.25).

4.2 CoT Cannot Close the Representation Gap with Transformers

In this section, we aim to understand the representation gap between RNNs and Transformers. We justify that RNNs, even with CoT, struggle to solve algorithmic problems that require the capability of retrieving information from the current context, which we call *In-context Retrieval* for short. This limitation is caused by the memory efficiency of RNNs: if the memory has at most $o(n)$ bits, then we can involve techniques from streaming complexity to prove impossibliity results for in-context retrieval problems.

4.2.1 Simple Problems on In-Context Retrieval

First, we prove that RNNs have a significant representation gap with Transformers in solving several simple algorithmic problems that directly test the in-context retrieval capability.

Definition 4.2 (Index)**.** Index is a problem that given a sequence of tokens with length n and a query token $i \in [n]$, requires the model to output the type of the i-th token.

Definition 4.3 (Associative Recall)**.** Associative Recall (AR) is a problem that given a sequence of tokens with length n consisting of tokens in $[n]$ and a query token $q \in [n]$, requires the model to output the next token of q in the sequence.

Definition 4.4 (c-gram Retrieval)**.** An c-gram is a contiguous subsequence of c tokens in a sequence of tokens. c-gram retrieval is a problem that given a sequence of tokens with length n and a query $(c-1)$-gram that is the prefix of a c-gram in the sequence, requires the model to output the last token of that c-gram.

Definition 4.5 (Counting)**.** Counting is a problem that given a sequence of tokens with length n, a query token $q \in [n]$, and a query number $t \in \mathbb{N}$, requires the model to output 0 or 1 to indicate whether the number of occurrences of q in the sequence is greater than t.

Here, Index and AR are perhaps the most basic problems in retrieval, where Index asks for retrieving a token from the input sequence viewed as a linear array of tokens, and AR asks for retrieving a token from the input sequence viewed as an associative array. These two problems have been studied extensively by different communities. Index is a classic problem in streaming and communication complexity (Munro & Paterson, 1980), known to be impossible to solve with $o(n)$ bits of memory for streaming algorithms. AR has been regarded as a fundamental problem that an artificial intelligence system should be able to solve (Willshaw et al., 1969; Hopfield, 1982; Hinton & Anderson,

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 7

2014; Graves et al., 2014; Ba et al., 2016). In the context of LLMs, AR has been observed to correlate with in-context learning performance (Elhage et al., 2021) and has also been used extensively as synthetic surrogate tasks for pretraining performance (Fu et al., 2023; Poli et al., 2023; Lutati et al., 2023). Besides Index and AR, c-gram retrieval is a natural extension of AR to the case where the query key can contain multiple tokens: instead of retrieving a token given a single-token key, c-gram retrieval asks for retrieving a token when the given key is a $(c-1)$-gram. This task has been studied empirically, but not theoretically in Jelassi et al. (2024). Counting is a problem that asks for the number of occurrences of a token, thereby testing the model's capability to retrieve some statistics of relevant information from the input sequence.

The following theorems show that RNNs with $O(\log n)$ bit memory cannot solve any of the four tasks, while Transformers can solve them perfectly with a constant size and $O(\log n)$ precision.

Theorem 4.6. *For task $T \in \{$Index, AR, c-gram retrieval, Counting$\}$, there exists a Transformer family with constant size and $O(\log n)$ precision that can solve T of size n. On the other hand, for any RNN family R with $o(n)$ bit memory, R cannot solve T of size n with any length of CoT for large enough n.*

Proof Sketch. The key idea of the lower bound of the proof is to put RNNs into the framework of communication complexity and use information-theoretic arguments to prove a lower bound. RNNs have the following property if party A simulates the RNN on the first part of the input and sends the state to party B, then party B can simulate the RNN on the second part of the input with the state received from party A to recover the output of the RNN perfectly. Hence, in the above theorem, if the RNN can solve the problem with $o(n)$ input size, then the information about the input can be compressed to $o(n)$ bit to produce the output, which contradicts the information-theoretic lower bound.

For the upper bound, we show that the Transformer can solve the problem by utilizing an attention mechanism called Match that takes the query token and attends to previous keys that match the query token on certain predefined coordinates. This mechanism allows the Transformer to read its context window like a key-value dictionary and hence can solve the problems perfectly. For the counting problem, we additionally use a COUNT attention mechanism that counts the number of occurrences of the queried token by attending evenly to each appearance of the queried token.

4.2.2 Understanding the Representation Power of RNNs Beyond Simple In-context Retrieval Problems

A natural question would be if an algorithmic problem does not directly test the in-context retrieval capability, can we hope that RNNs would have the representation power to solve it? Do RNNs and Transformers have the same representation power in this case? We show that the limited memory size in RNNs can still be a bottleneck in solving algorithmic problems. Even if the retrieval capability is not explicitly tested in an algorithmic problem, it may still be required implicitly for reasoning about the answer.

We demonstrate this gap on a minimal example of algorithmic problems, called IsTree: given an undirected graph G of n nodes, determine whether G is a tree, i.e., whether every pair of nodes is connected by exactly one simple path. A classical solution to IsTree is running Depth First Search (DFS), which takes $O(n)$ time.

In the context of language modeling, we can write the graph G as a sequence of tokens, and then the task of IsTree is to determine whether G is a tree by predicting a YES/NO token with or without CoT. We use the following tokenization for the graph G:

$$\mathrm{Tokenize}(G) = \{\texttt{<s>}, u_1, \sim, v_1, u_2, \sim v_2, \ldots, u_m, \sim, v_m\}, \tag{4}$$

where <s> and $\sim$ are two special tokens representing the start of the sentence and an edge, and u_i, v_i are numbers denoting the nodes of the graph.

Our result states that RNN with $o(n)$ bit memory cannot solve IsTree, even with an arbitrary choice of chain of thought. On the other hand, there exists a Transformer with constant size and $O(\log n)$ precision that can generate a chain-of-thought of length n following DFS and perfectly solve the same question.

Theorem 4.7. *For any RNN family R with $o(n)$ bit memory, R cannot perfectly solve IsTree of size n for large enough n, with any length of CoT. On the other hand, there exists a Transformer T_1 with constant dimension and depth, and $O(\log n)$ precision that can solve IsTree of size n perfectly with Chain of Thought of length $O(n)$.*

Proof Sketch. The main idea of the proof is that the task of IsTree requires the model to reason about the global structure of the graph, which is beyond the capability of RNNs with limited memory. We prove the lower bound by constructing a graph from a binary sequence and showing that RNNs with $o(n)$ memory cannot solve the problem by a reduction to an information-theoretic lower bound. For the upper bound, we show that the Transformer can simulate the DFS algorithm by outputting the Euler tour of the connected components of vertex 1 and then check the length of the Euler tour with its capability of In-context Retrieval.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 8

The key idea of the lower bound of the proof is to again utilize the information-theoretic lower bound. This idea lies in the core of streaming complexity literature and investigation on the IsTree problem dates back to Henzinger et al. (1998). We hereby restate the proof for completeness. Given any binary sequence x of length $n-2$ and an index $k \in [n-3]$, we can construct a graph as follows: the graph has n nodes, and vertex a is connected to vertex $x_a + n - 1$ for any $a \in [n-2]$. Moreover, vertex k is connected to vertex $k+1$. The graph is a tree if and only if $x_k \neq x_{k+1}$. Now assuming there is an RNN with $o(n)$ memory that can solve IsTree, consider two parties A and B each holding

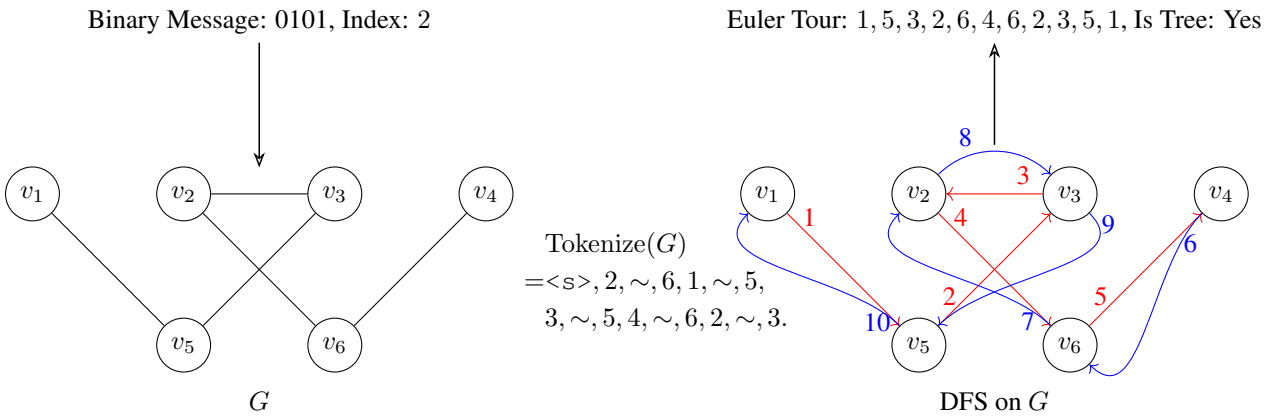


Figure 3: An example of the graph constructed from the binary sequence $x = 0101$ and the index $k = 2$ and the corresponding DFS tour.

the sequence x and the index k, they can construct two parts of the graph using their information and then A can simulate RNN on the first part of the graph and send the state to B, and B can simulate RNN (potentially with CoT) on the second part of the graph to recover the output of the IsTree problem, which is equivalent to whether $x_k \neq x_{k+1}$. However, note that k is never sent to A, and hence actually B can get whether $x_k \neq x_{k+1}$ for any $k \in [n-3]$, which contradicts the information-theoretic lower bound.

Now for the upper bound, we will let the Transformer simulate the DFS algorithm by outputting the Euler tour of the connected components of vertex 1 and then check the length of the Euler tour (see Algorithm 1). To simulate the tour, we will implement two functions through the Transformer Block:

1. Given a prefix of the tour, find the parent of the last vertex in the tour. This can be implemented by copying each token's predecessor's type to that token and then using the Match mechanism to match the first occurrence of the current token in the sequence.
2. Given the tokenized input graph and an edge (u, v), find the next edge after (u, v) containing v in the input graph. We will use another attention mechanism called COUNT to count, for each edge $e = (a, b)$ in tokenized input graph, the number of occurrences of a and b up to that edge and store $1/(n_{e,a} + 1)$ and $1/(n_{e,b} + 1)$ in the token corresponding to the edge, where $n_{e,a}$ and $n_{e,b}$ are the corresponding counts. Then given the edge (u, v), we can use the Match mechanism to find $1/(n_{(u,v),v} + 1)$. Then we will use a feed-forward layer with gated relu activation, constant depth, and constant width to approximate $1/(n_{(u,v),v} + 2)$ and then use the Match mechanism again to find the next edge containing v.

Through the above two functions, the Transformer can simulate the DFS algorithm and hence solve the IsTree problem perfectly.

4.2.3 Transformers are Strictly More Expressive Than RNNs

The above theorems show the existence of tasks where Transformers require exponentially less memory than RNNs. However, they have not rule out the possibility that there exists a corresponding task where the Transformer will be more redundant and require exponentially more parameters than RNNs. However, the following theorem confirms that such a task doesn't exist for regular RNN (Definition 3.4).

The theorem is in the same vein as the recent work on the CoT for Transformer (Li et al., 2024), which shows the constant size and constant precision Transformer with a polynomial-size position embedding can simulate any polynomial size circuit. The major difference of our theorem is that (1) we consider a Transformer with fixed word

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 9

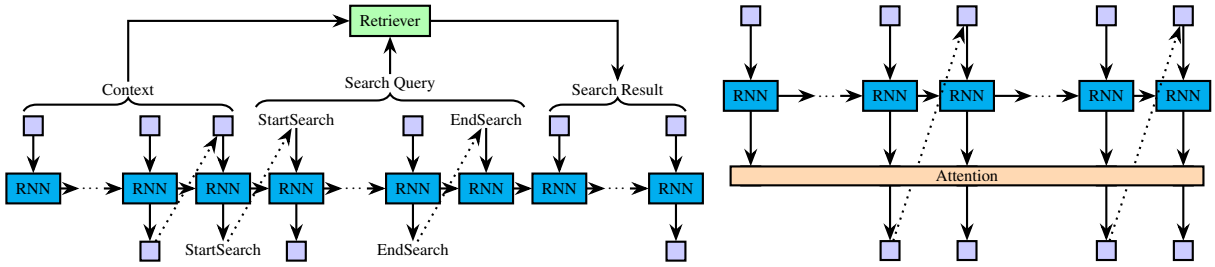
and position embedding, hence allowing the parameter number to be logarithmic in the input size, and (2) we consider simulating RNNs, which is a special kind of circuit family and hence we can use more succinct representation utilizing the structural property attached to the recursive process.

Theorem 4.8. *Given any constant $A > 0$, constant word width and number of special symbols $d, n_S > 0$, for any n, precision $p = \Theta(A \log n)$ and RNN R with word embedding $W^{(E)} \in \mathbb{R}_p^{(n+n_S)\times d}$ such that each recurrent iteration can be calculated with a circuit with size $\mathrm{P}(n) \leq 2^{p/2}$, there exists a Transformer T with $O(\mathrm{P}(n) \log \max\{\mathrm{P}(n), n\})$ bit parameter and word embedding $\begin{bmatrix} W^{(E)} & \mathbf{0}^{(n+n_S)\times d} \end{bmatrix}$ that can simulate the RNN with at most n^A step chain-of-thought precisely, using at most $(\mathrm{P}(n)+1)n^A$ step chain of thought on every input with length n.*

Proof Sketch. The key idea is to encode the RNN circuit into the Transformer's weight and simulate the circuit gate by gate, utilizing the Match mechanism to fetch the input of each gate. In this manner, although the naive circuit to simulate the RNN for n^A steps would require $O(n^A)$ parameter, the Transformer only needs to store one instance of the RNN circuit in its weight and hence we only need $O(P(n) \log \max\{P(n), n\})$ parameter, which is at most polynomial in the parameter size of the RNN.

5 Enhancing the In-context Retrieval Capability Closes the Representation Gap

In Section 4.2, we show that RNNs are deficient at In-context Retrieval, hence leading to a significant representation gap with Transformers. In this section, we aim to understand: if we enhance the In-context Retrieval capability of RNNs, do RNNs remain to have any representation gap with Transformers? We answer this question by considering an explicit and an implicit way to enhance the In-context Retrieval capacity and showing that both ways can close the representation gap between RNNs and Transformers in solving algorithmic problems.



The retrieval augmented RNN

The hybrid architecture

Figure 4: **In-context RAG.** The retrieval augmented RNN (left) and the hybrid architecture (right) close the representation gap between RNNs and Transformers.

5.1 Explicit Retrieval Through Regular Expression

First, we explore the power of RNNs with Retrieval Augmented Generation (RAG), which gives an LM the capability to retrieve relevant information to assist generation. In our context, we are specifically interested in allowing LMs to call functions to retrieve information from their context, which we call *In-context Retrieval Augmented Generation* (In-context RAG).

We will first show that adding function calls to associative recall is not enough to close the representation gap between RNNs and Transformers.

Proposition 5.1. *For any RNN family with $O(\log n)$ bit memory and $O(\log n)$ parameter with an oracle to receive results for the AR problem (Definition 4.3) for any queries, for large enough n, the RNN can't solve the index problem (Definition 4.2) with length n in any CoT steps.*

Proof. Consider a special type of index problem where every token at the even position of the input sequence is a special token κ and the rest of the tokens are uniformly random. Then the oracle for the AR problem can be simulated by the RNN by simply outputting the κ when the query is not κ and outputting the third token when the query is κ. However, following similar proof of Theorem 4.6, we can show that the RNN can't solve this special form of index problem with length n in any CoT steps. □

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 10

In this light, we need to consider a more general form of In-context Retrieval capability. We specifically consider a special form of In-context RAG that enables an LM to perform regular expression matching because the regular expression is a flexible primitive that can be used to describe a wide range of retrieval tasks and can be implemented efficiently on modern hardware.

Given an LM M with vocabulary V (containing two additional special tokens, <StartSearch> and <EndSearch>) and the tokenized input sequence $\mathcal{S}_{\text{in}} \in |V|^{l_0}$, the LM M with In-context RAG generates following sequence of tokenized sequence:

$$\mathcal{S}_0 = \mathcal{S}_{\text{in}}, \qquad s_i^{\text{next}} = \arg\max_{j \in V} M(\mathcal{S}_i)[j],$$

$$\mathcal{S}_{i+1} = \begin{cases} \mathcal{S}_i \oplus s_i^{\text{next}}, & \text{if } s_i^{\text{next}} \neq \text{<EndSearch>} \\ \mathcal{S}_i \oplus s_i^{\text{next}} \oplus \text{RETRIEVE}\left(\mathcal{S}_i\right), & \text{otherwise.} \end{cases}$$

Here RETRIEVE looks for the last occurrence of <StartSearch> at position l_s and <EndSearch> in $\mathcal{S}$ at position l_e and treat $\text{Detokenize}(\mathcal{S}_{l_s:l_e})$ as a regular expression, where Detokenize maps the tokenized sequence back to the string, inserting a space between every pair of adjacent tokens. The algorithm then runs a regular expression matching on $\text{Detokenize}(\mathcal{S}_{1:l_s-1})$, finds the first matching substring, and returns the first capturing group according to the regular expression (i.e., content embraced by a pair bracket in the regular expression). While there are many grammar standards of regular expressions, we adhere to the standard specified in the `re` library of Python. That is, we evaluate the following Python code to get the result of the regular expression matching:

```
re.search(pattern, string).group(1)
```

where $\text{Detokenize}(\mathcal{S}_{l_s:l_e})$ is the pattern and $\text{Detokenize}(\mathcal{S}_{1:l_s-1})$ is the string.

First, we show that In-context RAG with regular expression is powerful enough for RNNs to solve the In-context Retrieval problems in Section 4.2.1 in $O(1)$ CoT steps.

Theorem 5.2. *For task $T \in \{$Index, AR, c-gram retrieval, Counting$\}$, there exists a Linear RNN family with $O(\log n)$ bit memory and $O(\log n)$ parameter, that can solve T with In-context RAG in $O(1)$ CoT steps.*

Proof Sketch. For the Index problem, let the RNN output the regular expression `^(?:\S\s*){`a`}(\S)`, where $a = k - 1$. For AR, let the RNN output `\b`q`\b(\S+)\b`, where q is the number in query. For c-gram retrieval, let the RNN output `\b`$q_1 \ldots q_{c-1}$`\b(\S+)\b`, where q_i is the i-th number in the query. For Counting, let the RNN output `(\b`v`\b){`$k+1$`}`, where v is the query token and k is the query threshold.

Beyond these simple In-context Retrieval problems, in Theorem 4.7, we have shown that RNNs cannot solve IsTree due to its implicit requirement of In-context Retrieval capability. We now show that In-context RAG can help linear RNN solve IsTree in $O(n)$ CoT steps. See Appendix B.9 for the proof.

Theorem 5.3. *There exists a Linear RNN family with $O(\log n)$ bit memory and $O(\log n)$ parameter, that can solve IsTree of size n with In-context RAG in $O(n)$ CoT steps.*

To further show the power of the explicit retrieval, the following theorem further proves a general result, showing that In-context RAG empowers RNNs with $O(\log n)$ bit memory to simulate polynomial-time Turing machines.

Theorem 5.4. *Given any constant A, B, for any polynomial-time Turing machine $T \in \text{TIME}(n^A)$ with B states and vocabulary size B, there exists a retrieval augmented Linear RNN family (see Definitions 3.6 and A.5) with vocabulary of B special symbol, $O(A \log n)$ bit precision and memory, and $O(AB^2 \log n)$ bit parameters, that can simulate the result of T on any input with length n with a chain of thought of length $O(n^A)$.*

Proof Sketch. The retrieval augmented RNN can simulate the Turing machine by maintaining the state of the Turing machine and the position of the tape head in its memory and writing down the tape in the context in the form of $c\ v\ c$ to indicate the value of the tape at position c is updated to v. Given the input tape $\text{TAPE}[1, i]$, the retrieval augmented RNN will first write down the initial tape in the previous form of $i\ \text{TAPE}[1, i]\ i$ using the regular expression used in the Index problem. The RNN can then generate the search queries with forms of p `(.)` p `.*?$` with p being the pointer to retrieve the information from the context.

As a final note, our focus here is to understand the representation power of RNNs given an appropriate RAG, but not to propose a method that immediately leads to practical applications. While the above results show that In-context RAG can close the representation gap between RNNs and Transformers in solving algorithmic problems, a limitation here is that In-context RAG is not an immediate practical solution, as there is no existing training data for this In-context RAG.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 11

5.2 Implicit Retrieval by Appending Just One Transformer Layer

Since Bahdanau et al. (2016), attention mechanisms have been understood as a form of compensation for the fixed memory size of RNNs, allowing the model to attend to the entire context. We show in this section formally that this form of implicit retrieval can close the representation gap between RNNs and Transformers in solving algorithmic problems. We consider the following hybrid architecture, which combines the RNN and the Transformer by appending a single Transformer layer to the RNN output.

Definition 5.5 (Hybrid RNN)**.** A hybrid RNN is a model that consists of an RNN with transition and output function $\mathbf{t}, \mathbf{o}$ and one Transformer layer f, the output of the RNN is used as the input of the Transformer layer and the output of the Transformer layer is used to produce the next token. Concretely, given the input sequence $\mathcal{S}_{\text{in}}$, the output of the hybrid architecture is:

$$\mathcal{H}(\mathcal{S}) = \text{softmax}\left(W^{(E)} f\left([\mathbf{o}(s_k)]_{k\in[l]}\right)\right)_{:,l},$$
$$\forall k \in [l], s_k = \mathbf{t}(s_{k-1}, \text{Emb}(\mathcal{S})_{:,k}),$$

First, we show that hybrid RNNs can solve the In-context Retrieval problems in Section 4.2.1 without CoT.

Theorem 5.6. *For task $T \in \{$Index, AR, c-gram retrieval, Counting$\}$, there exists a hybrid Linear RNN (Definitions 3.6 and 5.5) family with $O(\log n)$ bit memory and $O(\log n)$ parameter, that can solve T without CoT.*

Proof Sketch. The proof is similar to the proof of Theorem 4.6, using the appended Transformer layer to simulate the Match function and COUNT function in the RNN.

Similar to the situation in Section 5.1, the implicit retrieval method can enpower the hybrid linear RNN to solve IsTree with CoT. See Appendix B.12 for the proof.

Theorem 5.7. *There exists a hybrid Linear RNN with $O(\log n)$ bit memory and $O(\log n)$ parameter, that can solve IsTree of size n with a chain of thought of length $O(n \log n)$.*

Further, we show that this hybrid architecture with only one attention block is powerful enough to even simulate any polynomial-time Turing machine with CoT.

Theorem 5.8. *Given any constant A, B, for any polynomial-time Turing machine $T \in \text{TIME}(n^A)$ with B states and vocabulary size B, there exists a hybrid Linear RNN (see Definition 5.5) with vocabulary of B special symbol, $O(A \log n)$ bit precision and memory, and $O(AB^2 \log n)$ bit parameters, that can simulate the result of T on any input with length n in $O(n^A)$ CoT steps.*

Proof Sketch. The proof is similar to the proof of Theorem 5.4. Instead of using regular expressions to retrieve the information from the context, the hybrid architecture can use the attention mechanism in the Transformer layer to implement the Match function to retrieve the information from the context.

6 Experiments

We tested our theory on the IsTree task. To generate the graph, we follow the procedure described in the proof of Theorem 4.7 (see Figure 3). The CoT data is generated using Algorithm 1 and the retrieval data is generated using Algorithm 2. For the CoT model, we decode the reasoning path during inference time until we reach the first YES or NO up to a max token limit greater than the length of all ground truth CoT. For the data points that the model fails to give a prediction, we assume the model gets it correct with 0.5 probability. For the retrieval task, we omit the explicit format of the regular expression and only ask the model to generate the vertices and special tokens in the regular expression to shorten the length of the input sequence. The reported accuracy is calculated over a validation set of 5000 samples using the last iteration of the model.

We train three different architectures: (1) LLaMA architecture (Touvron et al., 2023) representing Transformers, (2) Mamba architecture (Gu & Dao, 2023) representing RNNs, and (3) Mamba with one additional layer of LLaMA block representing hybrid architectures. Following our theory, we freeze and weight-tie the prediction head and word embedding in all the models. For ease of training, we use a different embedding function mapping $i-$th token to $[\sin(\frac{i}{10000^{j/d}}), \cos(\frac{i}{10000^{j/d}})]_{j\in[d/2]}$ with N being the number of different tokens and use standard RoPE (Su et al., 2024) as position embedding. We train every model with at least 1M samples to guarantee convergence using Adam with a cosine learning rate. If the model doesn't converge, we retrain using 5M samples. After a grid search over learning rates, we train all the Transformer models with learning rates 1e-3 and the rest of the models with learning rates 3e-4.

The results are shown in Figures 3 and 5. We observe that:

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 12

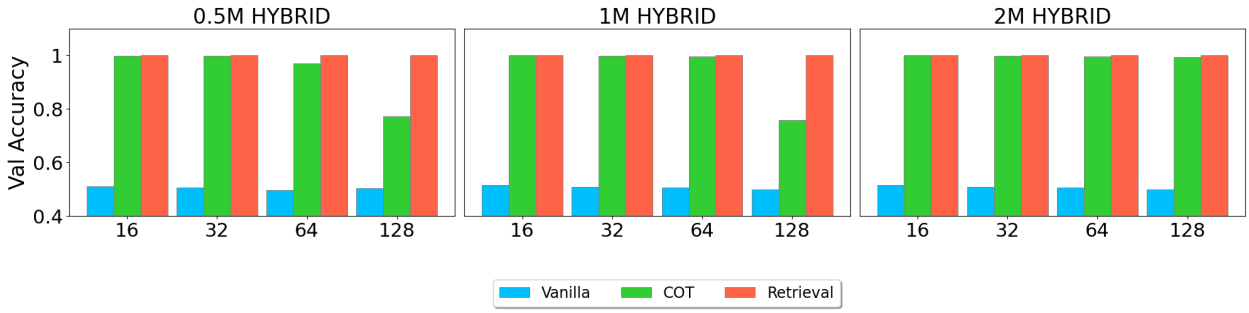


Figure 5: We train Mamba with one additional layer of attention with a frozen word embedding and decoding head of three different model sizes (0.5M, 1M, 2M) on IsTree with three different sizes of graph (16, 32, 64) under three different setups. **Vanilla** means the model directly predicts the label. **COT** means the model will generate a chain-of-thought process based on DFS (see Algorithm 1) before prediction. **Retrieval** means the model will generate the chain of search queries and reasoning before prediction (see Algorithm 2).

1. CoT improves the performance for both Transformers and RNNs. However, the RNNs' performance degrades sharply as the graph size increases and the Transformers consistently outperforms the RNNs. This is consistent with our theory that CoT can improve the expressiveness of the RNN models but the expressiveness is still not enough to solve the IsTree task (see Theorems 4.1 and 4.7).
2. Retrieval Augmentation via regular expression allows all the models to reach near-perfect accuracy. This is consistent with our theory that retrieval augmentation via regular expression can improve the expressiveness of the RNN models to solve algorithmic tasks (see Theorems 5.3 and 5.4).
3. The hybrid model shows performance on par with the Transformer, which is consistent with our theory (see Theorems 5.7 and 5.8).

7 Conclusion and Discussion

This paper studies the representation gap between RNNs and Transformers on algorithmic problems. It is proved that CoT can improve RNNs but is insufficient to close the gap with Transformers: the inability of RNNs to perform in-context retrieval is a key bottleneck. To address this, we show that adopting In-Context RAG or appending a single Transformer layer to RNNs can enhance their in-context retrieval capabilities and close the representation gap with Transformers.

One limitation of this work is that the solution of In-Context RAG through regular expression is for the purpose of understanding the bottleneck of the representation power of RNNs, and may not be a practical method beyond IsTree since there is no existing training data for this type of RAG. Effectively enabling or eliciting LLMs' capability to perform in-context RAG or other types of RAG is an interesting direction for future work. Second, appending a single Transformer layer to RNNs is a minimal example of making architectural changes to RNNs to improve their representation power while marginally increasing the memory cost. It is left unexplored what other architectural changes can pose a similar effect or enjoy a better trade-off between representation power and memory efficiency. Finally, we only study the aspect of representation power, and do not analyze the training dynamics and generalization of RNNs and Transformers. We believe this is the most challenging but important direction for future work.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 13

References

Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. In-context language learning: Architectures and algorithms, 2024.

Silas Alberti, Niclas Dern, Laura Thesing, and Gitta Kutyniok. Sumformer: Universal approximation for efficient transformers, 2023.

Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 20–29, 1996.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. *arXiv preprint arXiv:2312.04927*, 2023.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of transformers to recognize formal languages, 2020.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens, 2022.

Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. Recurrent memory transformer, 2022.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=qHrADgAdYu.

Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models, 2023.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training, 2020.

Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, December 2020. ISSN 2307-387X. doi: 10.1162/tacl_a_00306. URL http://dx.doi.org/10.1162/tacl_a_00306.

Jie Hao, Xing Wang, Baosong Yang, Longyue Wang, Jinfeng Zhang, and Zhaopeng Tu. Modeling recurrence for transformer. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 1198–1207, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1122. URL https://aclanthology.org/N19-1122.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 14

Yiding Hao, Dana Angluin, and Robert Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity, 2022.

Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998.

Geoffrey E Hinton and James A Anderson. *Parallel models of associative memory: updated edition*. Psychology press, 2014.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Samy Jelassi, David Brandfonbrener, Sham M Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. *arXiv preprint arXiv:2402.01032*, 2024.

Zhengbao Jiang, Luyu Gao, Zhiruo Wang, Jun Araki, Haibo Ding, Jamie Callan, and Graham Neubig. Retrieval as attention: End-to-end learning of retrieval and reading within a single transformer. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 2336–2349, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.149. URL https://aclanthology.org/2022.emnlp-main.149.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. In search of needles in a 11m haystack: Recurrent memory finds what llms miss, 2024.

Yuchen Li, Yuanzhi Li, and Andrej Risteski. How do transformers learn topic structure: Towards a mechanistic understanding, 2023.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems, 2024.

Zhong Li, Jiequn Han, Weinan E, and Qianxiao Li. On the curse of memory in recurrent neural networks: Approximation and optimization analysis. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=8Sqhl-nF50.

Zhong Li, Haotian Jiang, and Qianxiao Li. On the approximation properties of recurrent encoder-decoder architectures. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=xDIvIqQ3DXD.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=De4FYqjFueZ.

Shengjie Luo, Shanda Li, Tianle Cai, Di He, Dinglan Peng, Shuxin Zheng, Guolin Ke, Liwei Wang, and Tie-Yan Liu. Stable, fast and accurate: Kernelized attention with relative positional encoding, 2021.

Shahar Lutati, Itamar Zimerman, and Lior Wolf. Focus your attention (with adaptive iir filters), 2023.

William Merrill and Ashish Sabharwal. The Parallelism Tradeoff: Limitations of Log-Precision Transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 06 2023. ISSN 2307-387X. doi: 10.1162/tacl_a_00562. URL https://doi.org/10.1162/tacl_a_00562.

William Merrill, Ashish Sabharwal, and Noah A. Smith. Saturated transformers are constant-depth threshold circuits, 2022.

J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3): 315–323, 1980.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 15

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2021.

Matanel Oren, Michael Hassid, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns, 2024.

Jongho Park, Jaeseung Park, Zheyang Xiong, Nayoung Lee, Jaewoong Cho, Samet Oymak, Kangwook Lee, and Dimitris Papailiopoulos. Can mamba learn how to learn? a comparative study on in-context learning tasks, 2024.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Jiaju Lin, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartlomiej Koptyra, Hayden Lau, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Bolun Wang, Johan S. Wind, Stanislaw Wozniak, Ruichong Zhang, Zhenyuan Zhang, Qihang Zhao, Peng Zhou, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. Rwkv: Reinventing rnns for the transformer era, 2023.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A. Smith, and Lingpeng Kong. Random feature attention, 2021.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models, 2023.

Ohad Rubin and Jonathan Berant. Long-range language modeling with self-retrieval, 2023.

Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Representational strengths and limitations of transformers, 2023.

Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Transformers, parallel computation, and logarithmic depth, 2024.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models, 2023.

Yuandong Tian, Yiping Wang, Beidi Chen, and Simon Du. Scan and snap: Understanding training dynamics and token composition in 1-layer transformer, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020.

Xuezhi Wang and Denny Zhou. Chain-of-thought reasoning without prompting, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

David J Willshaw, O Peter Buneman, and Hugh Christopher Longuet-Higgins. Non-holographic associative memory. *Nature*, 222(5197):960–962, 1969.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 16

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2023.

Kai Yang, Jan Ackermann, Zhenyu He, Guhao Feng, Bohang Zhang, Yunzhen Feng, Qiwei Ye, Di He, and Liwei Wang. Do efficient transformers really save computation?, 2024.

Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages, 2023.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 17

Contents

1 Introduction **1**

2 Related Works **3**

3 Preliminaries **4**

4 Can CoT improve the Representation Power of RNNs? **7**

4.1 CoT Strictly Improves RNNs . . . 7

4.2 CoT Cannot Close the Representation Gap with Transformers . . . 7

4.2.1 Simple Problems on In-Context Retrieval . . . 7

4.2.2 Understanding the Representation Power of RNNs Beyond Simple In-context Retrieval Problems 8

4.2.3 Transformers are Strictly More Expressive Than RNNs . . . 9

5 Enhancing the In-context Retrieval Capability Closes the Representation Gap **10**

5.1 Explicit Retrieval Through Regular Expression . . . 10

5.2 Implicit Retrieval by Appending Just One Transformer Layer . . . 12

6 Experiments **12**

7 Conclusion and Discussion **13**

A Additional Definitions **20**

A.1 Reasoning Tasks on Graphs. . . . 20

A.2 More on Numeric Precisions. . . . 20

A.3 Models . . . 20

A.4 Language Models for Reasoning. . . . 21

B Omitted Proof **22**

B.1 Building Blocks of FFNs Construction . . . 22

B.2 Building Blocks of Transformers Construction . . . 23

B.3 Building Blocks of RNNs Construction . . . 30

B.4 Proof of Theorem 4.6 . . . 30

B.5 Proof of Theorem 4.1 . . . 31

B.6 Proof of Theorem 4.7 . . . 32

B.6.1 Proof of Lemma B.26 . . . 32

B.6.2 Proof of Lemma B.27 . . . 33

B.7 Proof of Theorem 4.8 . . . 35

B.8 Proof of Theorem 5.2 . . . 36

B.9 Proof of Theorem 5.3 . . . 37

B.10 Proof of Theorem 5.4 . . . 37

B.11 Proof of Theorem 5.6 . . . 40

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 18

B.12 Proof of Theorem 5.7 40
B.13 Proof of Theorem 5.8 40

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 19

A Additional Definitions

We will now define some definitions used in the proofs.

A.1 Reasoning Tasks on Graphs.

When resaoning on graphs, without otherwise specified, we will use n as the number of vertices and m as the number of edges. Without loss of generality, we will assume the vertices are labeled by $[n]$.

We will focus on decision problems on graphs, which are defined as follows:

Definition A.1 (Decision Problem on Graphs)**.** A decision problem on graphs is a function $f : \mathcal{G} \to \{\text{YES}, \text{NO}\}$, where $\mathcal{G}$ is the set of all possible graphs.

We will use the following decision problem as our main example:

Definition A.2 (IsTree)**.** $\text{IsTree}(G) = \text{YES}$ if G is a tree, and $\text{IsTree}(G) = \text{NO}$ otherwise.

One can view IsTree as a minimal example of reasoning tasks. One of the classical solutions to IsTree is running Depth First Search and this algorithm takes $O(n)$ time.

A.2 More on Numeric Precisions.

We will use $\text{ROUND}(x, p)$ to denote the rounding function that rounds x to the nearest number in $\mathbb{R}_p$. We will assume p is an odd number without loss of generality.

$$\mathbb{R}_p = \left\{ (2b_p - 1) \left(\sum_{i=1}^{p-1} b_i 2^{(p-1)/2-i} \right) : \forall i \in [p], b_i \in \{0, 1\} \right\}. \tag{5}$$

For calculation over $\mathbb{R}_p$, we will assume the calculation is exact and the result is rounded to $\mathbb{R}_p$ at the end, that is, for operator $\oplus$, we will have

$$\begin{aligned} &\text{ROUND}(x, p) \oplus_p \text{ROUND}(y, p) \\ =&\text{ROUND}\left(\text{ROUND}(x, p) \oplus \text{ROUND}(y, p), p\right). \end{aligned}$$

We will additionally define $\mathbb{Z}_p$ as the set of all integers that can be represented by p-bit floating point numbers. We will define $1/[m]$ as the set of unit fractional $\{\frac{1}{i}\}_{i \in [m]}$. Further, we will define $\text{ROUND}(1/[m], p)$ as the rounding of $1/[m]$ to $\mathbb{R}_p$. We will additionally define for any real number $x \in \{0, 1\}$, $\text{next}(x) = \frac{1}{m+1}$ where $m = \arg\min_{k \in \mathbb{Z}} |x - \frac{1}{k}|$.

A.3 Models

Tokenization. To tokenize a string, we will tokenize all the words separated by the space character into a sequence of tokens. To tokenize a graph G, we will order its edges $\text{E} = \{(u_i, v_i) \mid u_i < v_i\}$ randomly and tokenize it into the following string:

$$\text{Tokenize}(G) = \{\texttt{<s>}, u_1, \sim, v_1, \ldots, u_m, \sim, v_m\}. \tag{6}$$

We hereby assume there are constant more special tokens that are not the same as any number token, which are listed below:

- <s>: the first special token, indicating the start of a sentence.
- $\sim$: the second special token, indicating an edge.
- YES: the third special token, indicating the answer is yes.
- NO: the fourth special token, indicating the answer is no.
- <StartSearch>: the fifth special token, indicating the start of a search query.
- <EndSearch>: the sixth special token, indicating the end of a search query.

We will denote the total number of special tokens as n_S and the total vocabulary size as $|V| = n + n_S$. We will further define the detokenization function Detokenize,

$$\text{Detokenize}(\mathcal{S}) = \text{“}\mathcal{S}_1 \ \mathcal{S}_2 \ \ldots \ \mathcal{S}_l\text{”}.$$

Here each $\mathcal{S}_i$ is either a number or a special token, which we will treat as a word.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 20

Embedding Functions. We will use d to denote the dimension of the embedding and w_i to denote the i-th coordinate vector in $\mathbb{R}^d$.

We will separate the embedding function into two parts: the word embedding and the position embedding. For the word embedding, we will use $iw_1 \in \mathbb{R}^d$ to represent the embedding of the vertice i in the tokenization of the graph. For the k-th special token, we will use w_{2+k} to represent its embedding. For example, the embedding of $\sim$ is w_2. We will denote the word embedding matrix as $W^{(E)} \in \mathbb{R}^{|V| \times d}$.

For the position embedding, we will use lw_d to represent the position embedding of the l-th token in the tokenization of the graph, which is a hyperparameter. The final embedding of any token sequence is the sum of the word embedding and the position embedding. We will use Emb to denote the embedding function.

This embedding function will be fixed and shared across all models we consider in this paper and will not be learned during training, hence we will not consider it as part of the model parameters.

Language Modeling. In this work, we will consider the difference between Transformer and Recurrent Neural Networks on reasoning tasks, which is a special case of language modeling. We will define language modeling as follows:

Definition A.3 (Language Model)**.** A language model is a function $M : \cup_{l=1}^{L} |V|^l \to \mathbb{P}_{|V|}$, where $|V|$ is the vocabulary size, l is the sequence length, and $\mathbb{P}_{|V|}$ is the probability simplex over $|V|$.

A.4 Language Models for Reasoning.

Chain of Thought. We will now define how we use language models to solve reasoning tasks utilizing the following technique called chain of thought.

Definition A.4 (Chain of Thought)**.** Given a language model, M with vocabulary V and the tokenized input sequence $\mathcal{S}_{\text{in}} \in |V|^{l_0}$, chain of thought (CoT) generates the following sequence of tokenized sequence:

$$\mathcal{S}_0 = \mathcal{S}_{\text{in}}, \tag{7}$$

$$s_i^{\text{next}} = \arg\max_{j \in V} M(\mathcal{S}_i)[j], \tag{8}$$

$$\mathcal{S}_{i+1} = \mathcal{S}_i \oplus s_i^{\text{next}}, \forall i \geq 0. \tag{9}$$

The process terminates at $\mathcal{S}_i$ when $\arg\max_{j \in V} M(\mathcal{S}_i)[j]$ is YES or NO. The language model can solve the reasoning task within T steps of CoT if the process terminates at $\mathcal{S}_i$ where $i \leq T$ and the final output is correct. We will call the special case where the language model solves the reasoning task within 0 steps of CoT as solving the reasoning task without CoT.

Retrieval Augmentation. We will show in this paper that retrieval augmentation is a necessary technique to solve reasoning tasks for recurrent neural networks. We will define retrieval augmentation as follows:

Definition A.5 (Retrieval Augmented Generation)**.** Retrieval Augmented Generation means giving the language model the capability to retrieve relevant information to assist generation. We formally described the process here. Given a language model M with vocabulary V (containing two additional special tokens called <StartSearch> and <EndSearch>) and the tokenized input sequence $\mathcal{S}_{\text{in}} \in |V|^{l_0}$, retrieval augmented generation generates following sequence of tokenized sequence:

$$\begin{aligned} \mathcal{S}_0 &= \mathcal{S}_{\text{in}}, \\ s_i^{\text{next}} &= \arg\max_{j \in V} M(\mathcal{S}_i)[j], \\ \mathcal{S}_{i+1} &= \begin{cases} \mathcal{S}_i \oplus s_i^{\text{next}}, s_i^{\text{next}} \neq \text{<EndSearch>} \\ \mathcal{S}_i \oplus s_i^{\text{next}} \oplus \text{RETRIEVE}\left(\mathcal{S}_i\right), \text{otherwise.} \end{cases} \end{aligned}$$

Here RETRIEVE looks for the last occurrence of <StartSearch> at position l_s and <EndSearch> in $\mathcal{S}$ at positon l_e and treat $\text{Detokenize}(\mathcal{S}_{l_s:l_e})$ as a regular expression. The algorithm then uses the regular expression on $\text{Detokenize}(\mathcal{S}_{1:l_s-1})$. If the regular expression ever matches, the RETRIEVE will return the match. If the regular expression never matches, RETRIEVE will return a special token FAILED.

Similar to Definition A.4, we can define the notion of solving the reasoning task within T steps of retrieval augmented generation and solving the reasoning task without retrieval augmented generation.

We will note that assuming $|V| = O(n)$ and every search query and the result is of length $O(1)$, the regular expression evaluation can typically be evaluated in $O(n)$ time.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 21

B Omitted Proof

B.1 Building Blocks of FFNs Construction

We will first show some basic operations that multiple layers of feedforward neural networks with ReGLU activation can perform that will be used in the following proofs.

Lemma B.1 (Multiplication)**.** *Given two dimensions i_1, i_2, there exists a parameter configuration of a 1-layer feedforward neural network with ReGLU activation that for any input $x \in \mathbb{R}^d$ and constant width, computes the product of x_{i_1} and x_{i_2}.*

$$g(x) = [x_{i_1} \times x_{i_2}, 0, \ldots, 0]^\top.$$

Proof. We can construct the following feedforward neural network with ReGLU activation:

$$\begin{aligned}
&W_1 x = [x_{i_1} \quad -x_{i_1} \quad 0 \ldots \quad 0]^\top, \\
&W_2 x = [x_{i_2} \quad x_{i_2} \quad 0 \ldots \quad 0]^\top, \\
&W_3 h = [h_1 + h_2 \quad 0 \quad 0 \quad 0 \quad \ldots \quad 0]^\top, \\
&b_1 = b_2 = b_3 = 0. \\
&g(x) = W_3(\mathrm{ReLU}(W_1 x) \otimes \mathrm{ReLU}(W_2 x)) = [x_{i_1} \times x_{i_2}, 0, \ldots, 0]^\top.
\end{aligned}$$

□

Lemma B.2 (Linear Operations)**.** *Given a linear transformation $W \in \mathbb{R}^{d \times d}$, there exists a parameter configuration of a 1-layer feedforward neural network with ReGLU activation and width $w = d$ that for any input $x \in \mathbb{R}^d$, computes Wx.*

Proof.

$$\begin{aligned}
&b_1 = 1^w, b_2 = 0, b_3 = 0, \\
&W_1 = 0, W_2 = W, W_3 = I_{d \times d}
\end{aligned}$$

□

Lemma B.3 (Indicator)**.** *Given a constant integer $B \leq d$ and a dimension i, there exists a parameter configuration of a 1-layer feedforward neural network with ReGLU activation and width 4 that for any input $x \in \mathbb{R}^d$, computes the indicator function of $x_i = B$ when x_i is an integer.*

Proof.

$$\begin{aligned}
&b_2 = 1^w, b_1 = [-B - 0.5, -B + 0.5, B - 0.6, B + 0.4]^\top, b_3 = 10, \\
&W_2 = 0, W_1 x = [x_i, x_i, -x_i, -x_i]^\top, \\
&\mathrm{ReLU}(W_1 x + b_1) = [\mathrm{ReLU}(x_i - B - 0.5) \quad \mathrm{ReLU}(x_i - B + 0.5) \quad \mathrm{ReLU}(B - x_i - 0.6) \quad \mathrm{ReLU}(B - x_i + 0.4)] \\
&W_3 = 10\,[1 \quad -1 \quad -1 \quad 1]^\top.
\end{aligned}$$

Then,

$$\begin{aligned}
g(x) &= W_3 \mathrm{ReLU}(W_1 x + b_1) + b_3 \\
&= 10\,(\mathrm{ReLU}(x_i - B - 0.5) - \mathrm{ReLU}(x_i - B + 0.5)) + 10\,(\mathrm{ReLU}(B - x_i - 0.6) - \mathrm{ReLU}(B - x_i + 0.4)) + 10 \\
&= \begin{cases} 10 \times 0 + 10 \times -1 + 10 = 0, & \text{if } x_i < B, \\ 10 \times -0.5 + 10 \times -0.4 + 10 = 1, & \text{if } x_i = B, \\ 10 \times -1 + 10 \times 0 + 10 = 0, & \text{if } x_i > B. \end{cases}
\end{aligned}$$

□

Lemma B.4 (Lookup Table)**.** *For constant B and k such that $kB \leq d$, given a lookup table which key is tuple of k integers bounded by B, and value is a scalar in $\mathbb{R}_p$, there exists a parameter configuration of a $1-$ layer feedforward neural network with ReGLU activation with width $O(B^k)$ that for any input $x \in \mathbb{R}^d$, computes the value of the lookup table at the key $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$.*

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 22

Proof. We can calculate $x_{i_1} + B \times x_{i_2} + B^2 \times x_{i_3} + \ldots + B^{k-1} \times x_{i_k}$, and then scale B^k indicator functions to get the value of the lookup table at the key $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$. □

Lemma B.5 (Threshold)**.** *Given any threshold u and constant $\epsilon > 0$, there exists a parameter configuration of a 1-layer feedforward neural network with ReGLU activation and width 2 that for any input $x \in \mathbb{R}^d$, computes the indicator function $x_i > u$ on $x_i \in [-\infty, u - \epsilon] \cup [u, \infty]$.*

Proof.

$$
\begin{aligned}
&b_2 = 1^w, b_1 = [-u + \epsilon, -u + 0.5\epsilon]^\top, b_3 = 0, \\
&W_2 = 0, W_1 x = [x_i, x_i]^\top, \\
&\mathrm{ReLU}(W_1 x + b_1) = [\mathrm{ReLU}(x_i - u + \epsilon) \quad \mathrm{ReLU}(x_i - u + 0.5\epsilon)] \\
&W_3 = \frac{2}{\epsilon} [1 \quad -1 \quad -1 \quad 1]^\top .
\end{aligned}
$$

Then,

$$
\begin{aligned}
g(x) &= W_3 \mathrm{ReLU}(W_1 x + b_1) + b_3 \\
&= 2/\epsilon \left(\mathrm{ReLU}(x_i - u - \epsilon) - \mathrm{ReLU}(x_i - u + 0.5)\right) \\
&= \begin{cases} 0 & \text{if } x_i < u - \epsilon, \\ 1, & \text{if } x_i > u, \end{cases}
\end{aligned}
$$

□

Lemma B.6 (Interval)**.** *Given any constant u, v and $\epsilon > 0$, there exists a parameter configuration of a 1-layer feedforward neural network with ReGLU activation and width 4 that for any input $x \in \mathbb{R}^d$, computes the indicator function $x_i \in [u, v]$ on $x_i \in [-\infty, u - \epsilon] \cup \cup [u, v] \cup [v, \infty]$.*

Proof. The interval function here can be written as the difference of two threshold functions. We can use the same construction as in Lemma B.5 to approximate the indicator function $x_i > u$ and $x_i > v + \epsilon$ and then take the difference. □

B.2 Building Blocks of Transformers Construction

We will show in this section some construction for basic functionality using Transformer Blocks. This construction will be used in the following sections to prove the main results.

We will always use $X \in \mathbb{R}_p^{d \times l}$ as the input to the Transformer Block, where d is the dimension of the input, and l is the length of the sequence. We will first outline all the building functionality and then show how to implement them.

Definition B.7 (Copying Function)**.** For integer s, index set $I_1, I_2 \subset [d - 20]$ satisfying $|I_1| = |I_2|$, a copying function $\mathrm{COPY}[s, I_1, I_2]$ satisfies the following, $\forall X \in \mathbb{R}_p^{d \times l}$, then

$$
\begin{aligned}
&\mathrm{COPY}[s, I_1, I_2](X)_{I_2, k} = x_{I_1, \max\{k-s, 0\}} \quad \forall k \leq [m] \\
&\mathrm{COPY}[s, I_1, I_2](X)_{I_2^c, k} = 0 \quad \forall r \in [m]
\end{aligned}
$$

Definition B.8 (Counting Function)**.** For index set $I_1, I_2 \subset [d - 20]$, $|I_1| = |I_2| \leq 10$ and index i, a counting function $\mathrm{COUNT}[I_1, I_2, i]$ satisfies the following, if $\forall v \in I_1 \cup I_2, k \in [l], X_{v,k} \in \mathbb{Z}_p$ and $X_{v,k} \neq 0$, then

$$
\begin{aligned}
&\mathrm{COUNT}[I_1, I_2, i](X)_{i,k} = \frac{1}{\sum_{h=1}^{k} \mathbf{1}[X_{I_1,h} = X_{I_2,k}] + 1} \quad \forall k \in [l]. \\
&\mathrm{COUNT}[I_1, I_2, i](X)_{i^c,k} = 0 \quad \forall k \in [l].
\end{aligned}
$$

Definition B.9 (Matching Function)**.** For index set $I_1, I_2, I_3, I_4 \subset [d - 20]$, $|I_1| = |I_2| \leq 10$, $|I_3| = |I_4|$, a matching function $\mathrm{Match}[I_1, I_2, I_3, I_4]$ satisfies the following, if $\forall v \in I_1 \cup I_2, k \in [l], X_{v,k} \in \mathbb{Z}_p$, then

$$
\begin{aligned}
&\mathrm{Match}[I_1, I_2, I_3, I_4](x)_{I_3, k} = X_{I_4, k^*} \quad \forall k \in [l] \\
&\text{where } k^* = \begin{cases} \min\{h \mid X_{I_1,h} = X_{I_2,k}\}, \{h \mid X_{I_1,h} = X_{I_2,k}\} \neq \emptyset \\ 1, \text{otherwise} \end{cases} .
\end{aligned}
$$

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 23

Definition B.10 (Matching Closest Function)**.** For index set $I_1, I_2, I_3, I_4 \subset [d-20], |I_1| = |I_2| \leq 10, |I_3| = |I_4|$, a matching closest function $\mathrm{Match}[I_1, I_2, I_3, I_4]$ satisfies the following, if $\forall v \in I_1 \cup I_2, k \in [l], X_{v,k} \in \mathbb{Z}_p$, then

$$\mathrm{MatchClose}[I_1, I_2, I_3, I_4](x)_{I_3,k} = X_{I_4,k^*} \quad \forall k \in [l]$$
$$\text{where } k^* = \begin{cases} \max\{h \mid X_{I_1,h} = X_{I_2,k}\}, \{h \mid X_{I_1,h} = X_{I_2,k}\} \neq \emptyset \\ 1, \text{ otherwise} \end{cases}.$$

Definition B.11 (Matching Nearest Function)**.** For index set $I_1, I_2, I_3, I_4 \subset [d-20], |I_1| = |I_2| \leq 10, |I_3| = |I_4|$ and index i, a matching nearest function $\mathrm{MatchNearest}[I_1, I_2, I_3, I_4, i]$ satisfies the following, if $\forall v \in I_1 \cup I_2, k \in [l], X_{v,k} \in \mathbb{Z}_p$, then

$$\mathrm{MatchNearest}[I_1, I_2, I_3, I_4](x)_{I_3,k} = X_{I_4,k^*} \quad \forall k \in [l]$$
$$\text{where } k^* = \begin{cases} \arg\min_{h \in \{h \mid X_{I_1,h} = X_{I_2,k}\}} |h - X_{i,k}|, \{h \mid X_{I_1,h} = X_{I_2,k}\} \neq \emptyset \\ 1, \text{ otherwise} \end{cases}.$$

Definition B.12 (Matching Next Function)**.** Given any interger constant A, assuming $p > 10A \log n$, for index set $I_1, I_2, I_3, I_4 \subset [d-20], |I_1| = |I_2| \leq 10, |I_3| = |I_4|$, and a special counting index a, a matching next function $\mathrm{MatchNext}[I_1, I_2, I_3, I_4, a]$ satisfies the following, if X satisfies the following condition:

1. $\forall v \in I_1 \cup I_2, k \in [l], X_{v,k} \in \mathbb{Z}_p$,
2. $X_{a,k} \in \mathrm{ROUND}(1/[n^A], p) \cup \{0\}$,
3. For any $k \in [l]$, given any $k \geq k$, the counting index multiset $S_k = \{X_{a,k'} \mid X_{I_1,k'} = X_{I_2,k}\}$ takes consecutive and disjoint values in $\mathrm{ROUND}(1/[n^A], p)$, that is, there exists $u_k, v_k \in \mathrm{ROUND}(1/[n^A], p)$ such that $S_k = [u_k, v_k] \cap \mathrm{ROUND}(1/[n^A], p)$.

then, we have

$$\mathrm{MatchNext}[I_1, I_2, I_3, I_4, a](X)_{I_3,k} = X_{I_4,k^*} \quad \forall k \in [l]$$
$$\text{where } k^* = \arg \min_{h \in \{h \mid X_{I_1,h} = X_{I_2,k}\} \cup \{1\}} |X_{a,h} - \mathrm{next}(X_{a,k})|.$$

Now we will show how to implement these functions using Transformer Blocks. The construction here is motivated by the construction in Feng et al. (2023) with some modifications.

Lemma B.13 (Copying Blocks)**.** *For integer s, index set $I_1, I_2 \subset [d-10]$ satisfying $|I_1| = |I_2|$, a copying function $\mathrm{COPY}[s, I_1, I_2]$ can be implemented with 1 feedforward block g and 1 attention block $\mathcal{A}$ with 1 attention head. Formally, when $X_{d,k} = k$, it holds that*

$$\mathcal{A}(g(X) + X) = \mathrm{COPY}[s, I_1, I_2](X).$$

Proof of Lemma B.13. We will use the feedforward block to calculate $X_{k,d}^2$ and 1 (Lemma B.1) and have

$$(g(X) + X)_{d-1,k} = k^2$$
$$(g(X) + X)_{d-2,k} = 1.$$
$$\forall i \notin \{d-1, d-2\}, (g(X) + X)_{i,k} = X_{i,k}.$$

We will use X' to denote $g(X) + X$. Then we will choose $W^{(K)}, W^{(Q)}$ such that

$$W^{(K)} X'_{:,k'} = n \begin{bmatrix} 1 \\ k' \\ k'^2 \end{bmatrix}$$
$$W^{(Q)} X'_{:,k} = \begin{bmatrix} -(k^2 + s^2 - 2sk) \\ 2k - 2s \\ -1 \end{bmatrix}$$

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 24

Hence

$$\begin{aligned}&\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k}\\=&-n\left(k'^2-k'(2k-2s)+k^2+s^2-2sk\right)\\=&-n(k-s-k')^2\end{aligned}$$

Hence we have

$$\arg\max_{k'<k}\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k}=\max\{k-s,0\}.$$

Also, for any $k'\le k, k'\ne\max\{k-s,0\}$, we have

$$\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k}-\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{\max\{k-s,0\},k}<-n.$$

Hence after the column-wise softmax and rounding to $p=O(\log n)$ bit, we have

$$\left(\text{softmax}\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)\right)_{k',k}=\mathbf{1}[k'=\max\{k-s,0\}].$$

We will then choose $W^{(V)}$ such that

$$\begin{aligned}W^{(V)}X'_{I_2,k'}=X'_{I_1,k'}=X_{I_1,k'}\quad\forall k'\in[l].\\W^{(V)}X'_{I_2^c,k'}=0\quad\forall k'\in[l].\end{aligned}$$

This then concludes that

$$\mathcal{A}\left(g\left(X\right)+X\right)=\text{COPY}[s,I_1,I_2](X).$$

The proof is complete. □

Lemma B.14 (Counting Blocks)**.** *For index set $I\subset[d-20]$ satisfying $|I_1|=|I_2|\le 10$, a counting function* $\text{COUNT}[i,I_1,I_2]$ *can be approximated with* 1 *feedforward block g and* 1 *attention block $\mathcal{A}$ with* 1 *attention head. Formally, when $X_{d,k}=k$ and $X_{3,k}=\mathbf{1}[k=1], X_{I_1,1}=0$, it holds that*

$$\begin{aligned}\mathcal{A}\left(g\left(X\right)+X\right)_{i,k}&=\text{ROUND}\left(\text{COUNT}[s,I_1,I_2](X)_{i,k},p\right).\\\mathcal{A}\left(g\left(X\right)+X\right)_{i^c,k}&=0.\end{aligned}$$

Proof of Lemma B.14. We will use the feedforward block to calculate $X^2_{v,k}, v\in I_1\cup I_2$ (Lemma B.1) and have

$$\begin{aligned}(g(X)+X)_{d-i,k}&=X^2_{I_1[i],k}, i\in[|I|].\\(g(X)+X)_{d-|I|-i,k}&=X^2_{I_2[i],k}, i\in[|I|].\\(g(X)+X)_{d-2|I|-1,k}&=1.\\\forall i\notin\{d-i\mid i\in[2|I|+1]\},(g(X)+X)_{i,k}&=X_{i,k}.\end{aligned}$$

We will use X' to denote $g(X)+X$. Then we will choose $W^{(K)},W^{(Q)}$ such that

$$\begin{aligned}W^{(K)}X'_{:,k'}&=n\begin{bmatrix}1+\mathbf{1}[k'=1]\\X_{I_1[i],k'}\\X^2_{I_1[i],k'}\end{bmatrix}_{i\in[I]}\\W^{(Q)}X'_{:,k}&=\begin{bmatrix}X^2_{I_2[i],k}\\-X_{I_2[i],k}\\1\end{bmatrix}_{i\in[I]}\end{aligned}$$

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 25

Hence,

$$\begin{aligned}
&\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k}\\
=&-n\sum_{i=1}^{|I|}\left(X'^2_{I_2[i],k'}-X_{I_1[i],k'}(2X_{I_2[i],k})+X^2_{I_2[i],k}\right)+n\mathbf{1}[k'=1]\sum_{i=1}^{|I|}X^2_{I[i],k}.\\
=&-n\sum_{i=1}^{|I|}(X_{I_1[i],k'}-X_{I_2[i],k})^2+n\mathbf{1}[k'=1]\sum_{i=1}^{|I|}X^2_{I_2[i],k}.
\end{aligned}$$

Hence we have

$$\max_{k'<k}\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k}=0.$$

Equality holds when $k'=1$ or $X_{I_1[i],k'}=X_{I_2[i],k}$ for all $i\in[|I_1|]$.

Also, for any $k'\le k, k'\ne 1$ or $X_{I_1[i],k'}\ne X_{I_2[i],k}$ for some $i\in[|I_1|]$, we have

$$\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k}<-n.$$

Hence after the column-wise softmax and rounding to $p=O(\log n)$ bit, we have

$$\left(\text{softmax}\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)\right)_{k',k}=\text{ROUND}\left(\frac{1}{\sum_{h=1}^{k}\mathbf{1}[X_{I_1,h}=X_{I_2,k}]+1},p\right)$$

Here the $O\left(\frac{1}{n^A}\right)$ term comes from the fact that the softmax is rounded to $p=O(\log n)$ bit.

We will then choose $W^{(V)}$ such that

$$\begin{aligned}
W^{(V)}X'_{i,k'}=X'_{3,k'}=\mathbf{1}[k'=1]\quad\forall k'\in[l].\\
W^{(V)}X'_{I^c,k'}=0\quad\forall k'\in[l].
\end{aligned}$$

This then concludes that

$$\begin{aligned}
\mathcal{A}\left(g\left(X\right)+X\right)_{i,k}&=\text{ROUND}\left(\text{COUNT}[s,I_1,I_2](X)_{i,k},p\right).\\
\mathcal{A}\left(g\left(X\right)+X\right)_{i^c,k}&=0.
\end{aligned}$$

□

Lemma B.15 (Matching Blocks)**.** *Given any constant c, for index set $I_1,I_2,I_3,I_4\subset[d-20]$, $|I_1|=|I_2|\le 10$, $|I_3|=|I_4|$, a matching function* $\text{Match}[I_1,I_2,I_3,I_4]$ *can be implemented with* 1 *feedforward block g and* 1 *attention block $\mathcal{A}$ with* 1 *attention head. Formally, when $X_{d,k}=k, X_{3,k}=\mathbf{1}[k=1], X_{I_1,1}=0$ and $k\le n^c$, it holds that*

$$\mathcal{A}\left(g\left(X\right)+X\right)=\text{Match}[I_1,I_2,I_3,I_4](X)$$

Proof. We will use the feedforward block to calculate $k^2, X^2_{v,d}, v\in\cup I_1\cup I_2$ as in the proof of Lemmas B.13 and B.14.

We then choose $W^{(K)},W^{(Q)}$ such that

$$\begin{aligned}
&\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k}\\
=&-n^{4c+1}\sum_{i=1}^{|I|}(X_{I_1[i],k'}-X_{I_2[i],k})^2-nk'^2\\
&+\mathbf{1}[k'=1]\left(n^{4c+1}\sum_{i=1}^{|I|}X^2_{I_2[i],k}+n-n^{2c+2}\right).
\end{aligned}$$

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 26

The detailed construction of $W^{(K)}, W^{(Q)}$ is omitted here since it is similar to the proof of Lemmas B.13 and B.14.

We will discuss several cases for the distribution of $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k}$. It always holds that $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{1,k} = -n^{2c+2}$.

1. If there doesn't exists k', such that $X_{k',I_1} = X_{k,I_2}$, then for any $i > 1$, we have $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{i,k} < -n^{4c+1}$.
2. If there exists k', such that $X_{k',I_1} = X_{k,I_2}$, then for such k', we have $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k} = -nk'^2 > -n^{2c+1}$. The rest of the entries are all smaller than $-n^{4c+1}$. Finally, the largest k' satisfying that $X_{k',I_1} = X_{k,I_2}$ will corresponds to a $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k}$ that is at least n larger than the second largest $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k}$, as in the proof of Lemma B.13.

Concluding the above discussion, we have after the column-wise softmax and rounding to $p = O(\log n)$ bit,

$$\left(\text{softmax}\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)\right)_{k',k} = \begin{cases} \mathbf{1}[k' = \min\{h \mid X_{I_1,h} = X_{I_2,k}\}], \{h \mid X_{I_1,h} = X_{I_2,k}\} \neq \emptyset \\ \mathbf{1}[k' = 1], \text{otherwise} \end{cases}$$

Further, we will choose $W^{(V)}$ such that

$$W^{(V)}X'_{I_3,k'} = X'_{I_4,k'} = X_{I_4,k'} \quad \forall k' \in [l].$$
$$W^{(V)}X'_{I_3^c,k'} = 0 \quad \forall k' \in [l].$$

This then concludes that

$$\mathcal{A}\left(g\left(X\right) + X\right) = \text{Match}[I_1, I_2, I_3, I_4](X)$$

This concludes the proof. □

Lemma B.16 (Matching Closest Blocks)**.** *Given any constant c, for index set $I_1, I_2, I_3, I_4 \subset [d-20], |I_1| = |I_2| \leq 10, |I_3| = |I_4|$, a matching closest function* $\text{MatchClose}[I_1, I_2, I_3, I_4]$ *can be implemented with* 1 *feedforward block g and* 1 *attention block $\mathcal{A}$ with* 1 *attention head. Formally, when $X_{d,k} = k, X_{3,k} = \mathbf{1}[k = 1], X_{I_1,1} = 0$ and $k \leq n^c$, it holds that*

$$\mathcal{A}\left(g\left(X\right) + X\right) = \text{MatchClose}[I_1, I_2, I_3, I_4](X)$$

Proof. The proof is similar to the proof of Lemma B.15, and one can design the attention pattern as

$$\begin{aligned} &\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k} \\ =& -n^{4c+1}\sum_{i=1}^{|I|}(X_{I_1[i],k'} - X_{I_2[i],k})^2 - n(k-k')^2 \\ &+ \mathbf{1}[k' = 1]\left(n^{4c+1}\sum_{i=1}^{|I|}X^2_{I_2[i],k} + n(k-1)^2 - n^{2c+2}\right). \end{aligned}$$

The rest of the proof is omitted here. □

Lemma B.17 (Matching Nearest Blocks)**.** *Given any constant c, for index set $I_1, I_2, I_3, I_4 \subset [d-20], |I_1| = |I_2| \leq 10, |I_3| = |I_4|$ and index i , a matching nearest function* $\text{MatchNearest}[I_1, I_2, I_3, I_4, i]$ *can be implemented with* 1 *feedforward block g and* 1 *attention block $\mathcal{A}$ with* 1 *attention head. Formally, when $X_{d,k} = k, X_{3,k} = \mathbf{1}[k = 1], X_{I_1,1} = 0$ and $k \leq n^c$, it holds that*

$$\mathcal{A}\left(g\left(X\right) + X\right) = \text{MatchNearest}[I_1, I_2, I_3, I_4, i](X)$$

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 27

Proof. The proof is similar to the proof of Lemma B.15, and one can design the attention pattern as

$$\begin{aligned}&\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k}\\=&-n^{4c+1}\sum_{u=1}^{|I|}(X_{I_1[u],k'}-X_{I_2[u],k})^2-n(X_{i,k}-k')^2\\&+\mathbf{1}[k'=1]\left(n^{4c+1}\sum_{u=1}^{|I|}X^2_{I_2[u],k}+n(1-X_{i,k})^2-n^{2c+2}\right).\end{aligned}$$

The rest of the proof is omitted here. □

Lemma B.18 (Matching Next Blocks)**.** *Given any constant A, c, for index set $I_1, I_2, I_3, I_4 \subset [d-20], |I_1| = |I_2| \leq 10, |I_3| = |I_4|$ and a special counting index a, a matching next function* $\mathrm{MatchNext}[I_1, I_2, I_3, I_4, a]$ *can implement with 1 feedforward block g and 1 attention block $\mathcal{A}$ with 1 attention head. Formally, when $X_{d,k} = k, X_{3,k} = \mathbf{1}[k=1], X_{I_1,1} = 0$ and $k \leq n^c$, it holds that*

$$\mathcal{A}\left(g\left(X\right)+X\right)=\mathrm{MatchNext}[I_1,I_2,I_3,I_4,a](X)$$

Proof. We will use the feedforward block to calculate the following $\overline{\mathrm{next}}$ function, where

$$\overline{\mathrm{next}}(x)=\begin{cases}\frac{1}{2}, & x\geq\frac{2}{3}.\\ \frac{1}{3}, & \frac{3}{5}>x>\frac{2}{5}.\\ \frac{1}{4}, & \frac{7}{20}>x>\frac{3}{10}\\ x-x^2+x^3, & x\leq\frac{11}{40}.\end{cases}$$

The value can be arbitrary for $x \in [\frac{11}{40}, \frac{3}{10}] \cup [\frac{2}{5}, \frac{7}{20}] \cup [\frac{3}{5}, \frac{2}{3}]$. This function is achievable by a feedforward block through combination of Lemmas B.1 and B.6.

The purpose of this is to approximate the next function for $x \in \mathrm{ROUND}(1/[n^A], p)$, and we have the following lemma.

Lemma B.19. *For large enough n and any $x \in \mathrm{ROUND}(1/[n^A], p)$, we have*

$$|\overline{\mathrm{next}}(x)-\mathrm{next}(x)|\leq\mathrm{next}(x)^3+O\left(\frac{1}{n^{10A}}\right).$$

Proof. We always have $\mathrm{ROUND}(1/[n^A], p) \cap \left([\frac{11}{40}, \frac{3}{10}] \cup [\frac{2}{5}, \frac{7}{20}] \cup [\frac{3}{5}, \frac{2}{3}]\right) = \emptyset$. We will discuss several cases for $x \in \mathrm{ROUND}(1/[n^A], p)$.

1. If $x \geq \frac{3}{10}$, then $\overline{\mathrm{next}}(x) = \mathrm{next}(x)$.
2. If $x \leq \frac{7}{20}$, it holds that $|x - 1/m| \leq 1/n^{10A}, m \geq 3$, then

$$\begin{aligned}\overline{\mathrm{next}}(x)=x-x^2&=\frac{1}{m}-\frac{1}{m^2}+\frac{1}{m^3}+O\left(\frac{1}{n^{10A}}\right)\\&=\frac{1}{m+1}-\frac{1}{m^3(m+1)}+O\left(\frac{1}{n^{10A}}\right)\end{aligned}$$

This then concludes the proof. □

We then choose $W^{(K)}, W^{(Q)}$ such that

$$\begin{aligned}&\left(\left(W^{(K)}X\right)^\top\left(W^{(Q)}X\right)\right)_{k',k}\\=&-n^{4A+3}\sum_{i=1}^{|I|}(X_{I_1[i],k'}-X_{I_2[i],k})^2-n^{4A+1}(\overline{\mathrm{next}}(X_{a,k})-X_{a,k'})^2\\&+\mathbf{1}[k'=1]\left(n^{4A+3}\sum_{i=1}^{|I|}X^2_{I_2[i],k}+n^{4A+1}X^2_{a,k}-n^{4A+2}\right).\end{aligned}$$

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 28

Again, the detailed construction of $W^{(K)}, W^{(Q)}$ is omitted here since it is similar to the proof of Lemmas B.13 and B.14.

We will discuss several cases for the distribution of $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k}$. It always holds that $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{1,k} = -n^{4A+2}$.

1. If there doesn't exists k', such that $X_{k',I_1} = X_{k,I_2}$, then for any $i > 1$, we have $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{i,k} < -n^{4A+3}$.
2. If there exists k', such that $X_{k',I_1} = X_{k,I_2}$, then for such k', we have $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k} = -n^{3A}(\text{next}(X_{a,k}) - X_{a,k'})^2 > -n^{4A+1}$. The rest of the entries are all smaller than $-n^{4A+2}$.

   It remains to discuss the distribution of $\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k}$ for k' satisfying $X_{k',I_1} = X_{k,I_2}$. When X satisfies the condition in Definition B.12, we have that $S_k = \{X_{a,k'} \mid X_{k',I_1} = X_{k,I_2}\}$ takes consecutive and disjoint values in $\text{ROUND}(1/[n^A], p)$. Hence, if $|S_k| > 2$, suppose $y, z \in S_k$ satisfies that

$$|y - \overline{\text{next}}(X_{a,k})| = \min_{x \in S_k} |x - \overline{\text{next}}(X_{a,k})|$$
$$|z - \overline{\text{next}}(X_{a,k})| = \min_{x \in S_k, x \neq y} |x - \overline{\text{next}}(X_{a,k})|.$$

   We will discuss several cases for y, z.

   - If $y - \overline{\text{next}}(X_{a,k})$ and $z - \overline{\text{next}}(X_{a,k})$ are both negative, then $y > z$, we have,

$$\left(y - \overline{\text{next}}(X_{a,k})\right)^2 - \left(z - \overline{\text{next}}(X_{a,k})\right)^2 = (y-z)(y+z-2\overline{\text{next}}(X_{a,k})) \leq -(y-z)^2 \leq -\frac{1}{4n^{4A}}.$$

   .
   - If $y - \overline{\text{next}}(X_{a,k})$ and $z - \overline{\text{next}}(X_{a,k})$ are both positive, then $y < z$, and same as above we have

$$\left(y - \overline{\text{next}}(X_{a,k})\right)^2 - \left(z - \overline{\text{next}}(X_{a,k})\right)^2 = (y-z)(y+z-2\overline{\text{next}}(X_{a,k})) \leq -(y-z)^2 \leq -\frac{1}{4n^{4A}}.$$

   .
   - If $y - \overline{\text{next}}(X_{a,k})$ and $z - \overline{\text{next}}(X_{a,k})$ have different signs, then according to Lemma B.19, we have, $y = \text{ROUND}(\text{next}(X_{a,k}), p)$ because S_k takes consecutive and disjoint values in $\text{ROUND}(1/[n^A], p)$. This then implies that

$$\begin{aligned}&\left(y - \overline{\text{next}}(X_{a,k})\right)^2 - \left(z - \overline{\text{next}}(X_{a,k})\right)^2 \\ \leq & O(\frac{1}{n^{10A}}) + \frac{1}{\text{next}^6(X_{a,k})} - \left(\frac{1}{\text{next}(X_{a,k})(\text{next}(X_{a,k})+1)}\right)^2 \\ \leq & -\frac{1}{4n^{4A}}.\end{aligned}$$

Concluding, we always have for any $k'' \neq k^* = \arg\max_{k',k} \left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k}$

$$\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k',k} - \left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)_{k^*,k} \leq -\frac{n}{4}.$$

Concluding the above discussion, we have after the column-wise softmax and rounding to $p = O(\log n)$ bit,

$$\left(\text{softmax}\left(\left(W^{(K)}X\right)^\top \left(W^{(Q)}X\right)\right)\right)_{k',k} = \mathbf{1}\left[k' = \arg \min_{h \in \{h \mid X_{I_1,h} = X_{I_2,k}\} \cup \{1\}} |X_{a,h} - \text{next}(X_{a,k})|\right].$$

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 29

Further, we will choose $W^{(V)}$ such that

$$W^{(V)}X'_{I_3,k'} = X'_{I_4,k'} = X_{I_4,k'} \quad \forall k' \in [l].$$
$$W^{(V)}X'_{I_3^c,k'} = 0 \quad \forall k' \in [l].$$

This then concludes the proof. □

B.3 Building Blocks of RNNs Construction

We will now describe the building blocks of Linear RNNs construction. We will introduce some basic operations that will be used to build more complex RNNs family.

Lemma B.20 (Recent Input Memorizing)**.** *Given any constant k and C, there exists a parameter configuration of linear unit that maintains C dimensions of last k input vectors in the state.*

Proof. Suppose the input sequence is $x_{1:t} \in \mathbb{R}^d$, and the dimensions that the state should memorize are $d_1, d_2, \ldots, d_C$. We can construct the following linear unit:

$$h_t = \begin{bmatrix} x_{t-1,d_1} & \ldots & x_{t-1,d_C} & h_{t-1,1} & \ldots & h_{t-1,C\times(k-1)} & h_{t-1,C\times k+1} & \ldots & h_{t-1,d} \end{bmatrix}.$$

□

Lemma B.21 (Summation)**.** *Given any constant k and C, there exists a parameter configuration of linear unit that maintains the sum of one dimension of the last k input vectors in the state.*

Proof. Suppose WLOG the input sequence is $x_{1:t} \in \mathbb{R}^d$, and the dimension that the state should memorize is 1. We can construct the following linear unit:

$$h_t = \begin{bmatrix} x_{t-1,1} + h_{t-1,1} & h_{t-1,2} \ldots & h_{t-1,d} \end{bmatrix}.$$

□

Lemma B.22 (Special Position Memorizing)**.** *Given any constant k and C, there exists a parameter configuration of linear unit and a FFN with Ck width that maintains the C dimensions of the input vector at position 1 to k in the state.*

Proof. This is a direct combination of Lemma B.21 and Lemmas B.1 and B.4. The FFN can assign all the input vectors with position greater than k to 0, and permute the corresponding dimensions of first k input vectors to the first Ck dimensions of the state. The linear unit can then maintain the state. □

Lemma B.23 (Recite Fixed Sequence)**.** *Given any constant integer k and C, there exists a FFN with width kC that can output fixed sequence of scalars that takes values in $[C]$ on a fixed set of positions $l_1, \ldots l_k$.*

Proof. This is a direct consequence of Lemma B.4. □

B.4 Proof of Theorem 4.6

We will first restate the theorem for clarity.

Theorem 4.6. *For task $T \in \{$Index, AR, c-gram retrieval, Counting$\}$, there exists a Transformer family with constant size and $O(\log n)$ precision that can solve T of size n. On the other hand, for any RNN family R with $o(n)$ bit memory, R cannot solve T of size n with any length of CoT for large enough n.*

Proof. We will discuss by cases.

When T is Index, we will first show why RNN cannot solve the Index question without $\Omega(n)$ memory. The key observation is that the recurrent form of RNNs allowed the algorithm to be run in a streaming fashion with $o(n)$ bit memory. Here streaming means that the algorithm gets to look at each bit of the memory sequentially and can only update a constant size of memory.

Lemma B.24. *Consider the following two-party game, where Alice receives string $x \in \{0,1\}^n$ and Bob receives an integer k, and Bob wants to know the value of x_k. If only Alice is allowed to send a signal to Bob, then $\Omega(n)$ bit communication complexity is required.*

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 30

Proof of Lemma B.24. Suppose there exists a communication protocol where B only receives $o(n)$ bit and can perfectly decide x_k. Because Alice doesn't know k, the protocol must send the same message to Bob for all k. Hence Bob can reconstruct the whole string x with n bit with $o(n)$ bit communication. This is a contradiction. □

Now if RNN can solve the Index problem with $o(n)$ bit memory, then it can also solve the Index problem with $o(n)$ bit communication complexity. This is because Alice can simply run the RNN on input x and send the hidden state to Bob. Then Bob can run the RNN with the hidden state and k to get the answer. This is a contradiction to Lemma B.24. Hence RNN cannot solve the Index problem with $o(n)$ bit memory.

On the other hand, we will show that Transformers can solve the Index problem with $O(\log n)$ bit parameters. This is because using 2 layers of Transformer, we will implement a Match Block (Lemma B.15) that can match the last query token with the position of the previous token and retrieve the type of the matched token to the query token.

When T is AR, wthout loss of generality, we assume that n is even. The proof is similar to the proof of the proof of theIndex problem. As there are n different types of tokens, we can label them as $[n]$. Now for any boolean sequence $x \in \{0,1\}^{n/2}$, solving AR for the following input is equivalent to solving the Index problem for x:

$$\mathcal{S}_{\text{in}} = \texttt{<s>}, 1, x_1 + n/2, 2, x_2 + n/2, \ldots, n/2, x_{n/2} + n/2, k$$

This then implies that RNN cannot solve AR with $o(n)$ bit memory. Transformers, on the other hand, can still solve AR with $O(\log n)$ bit parameters, we will use one layer of copying function to copy each token's previous token's type to it. Then we can use the Match Block to match the last query token with the position of the previous token and retrieve the type of the matched token to the query token.

When T is c-gram retrieval, without loss of generality, we assume that n is a multiple of c. The proof is similar to the proof of Theorem 4.6. As there are n different types of tokens, we can label them as $[n]$. Now for any boolean sequence $x \in \{0,1\}^{n/2}$, solving AR for the following input is equivalent to solving the Index problem for x:

$$\mathcal{S}_{\text{in}} = \texttt{<s>}, \underbrace{1, \ldots, 1}_{c-1}, x_1 + n/c, \underbrace{2, \ldots, 2}_{c-1}, x_2 + n/c, \ldots, \underbrace{n/c, \ldots, n/c}_{c-1}, x_{n/c} + n/c, \underbrace{k, \ldots, k}_{c-1}$$

This then implies that RNN cannot solve c-gram retrieval with $o(n)$ bit memory. Transformers, on the other hand, can still solve c-gram retrieval with $O(\log n)$ bit parameters, we will use one layer of copying function to copy each token's previous $c-1$ tokens' type to it. Then we can use the Match Block to match the last query token with the position of the previous token and retrieve the type of the matched token to the query token.

When T is Counting, we will first show why RNN cannot solve the Counting question without $\Omega(n)$ memory. Consider the following setup, given any $x \in \{0,1\}^n$, the input string is $j_1 j_2 \ldots j_k$ where $\{j_i \ldots j_k\} = \{j \mid x_j = 1\}$, then solving the Counting question for this input string for queried threshold 1 is equivalent to solving the Index problem for x. This then implies that RNN cannot solve the Counting question with $o(n)$ bit memory.

On the other hand, we will show that Transformers can solve the Counting question with $O(\log n)$ bit parameters. This is because using 2 layers of Transformer, we can first use a COPY block to copy the last query token to the token corresponds to the threshold, and then use a COUNT block (Lemma B.14) that can count the number m of the appearance of the last query token in the input sequence, and then write $1/(m+1)$ to one of the dimension. Finally, we can use the Feed Forward Network on the last layer to multiply threshold $+1$ with this value and compare the result to 1 to get the answer. □

B.5 Proof of Theorem 4.1

We will restate the theorem for clarity.

Theorem 4.1. *Assuming* $\text{PSPACE} \not\subset \text{P/Poly}$*, there exists a task with input length* n *that can be solved by a Linear RNN (Definition 3.6) family with* $\Theta(\log n)$ *bit memory with polynomial length CoT but cannot be solved by any regular RNN (Definition 3.4) family with* $\Theta(\log n)$ *bit memory without CoT.*

Proof. We will first prove a lemma,

Lemma B.25. *If* $\text{PSPACE} \not\subset \text{P/Poly}$*, then there exists a Turing machine* M *with linear space complexity that cannot be simulated by a polynomial-size circuit family.*

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 31

Proof. We will prove this by contradiction. Assuming for every Turing machine M with linear space complexity, there exists a polynomial-size circuit family $\{C_n\}$ that can simulate M. We will construct a polynomial-size circuit family $\{C'_n\}$ that can decide PSPACE, which contradicts the assumption that $\text{PSPACE} \not\subset \text{P/Poly}$.

Given any language $L \in \text{PSPACE}$, we can decide L by a Turing machine M_L with space $O(n^k)$ for some constant k. We can consider another language $L' = \{x1^{|x|^k} \mid x \in L\}$. We can decide L' by a Turing machine $M_{L'}$ with linear space complexity by checking the length of the input and then simulating M_L. By the assumption, there exists a polynomial-size circuit family $\{C_n\}$ that can simulate $M_{L'}$. We can then construct a polynomial-size circuit family $\{C'_n\}$ that can decide L by first counting the length of the input and then simulating C_n on the extended input. This contradicts the assumption that $\text{PSPACE} \not\subset \text{P/Poly}$. □

By Lemma B.25, we know that if $\text{PSPACE} \not\subset \text{P/Poly}$, then there exists a Turing machine M with linear space complexity that cannot be simulated by a polynomial-size circuit family. We will use this result to prove Theorem 4.1.

We will design the task as follows, for any n, let $m = \lfloor \log_2 n \rfloor$, for any boolean input x of length m, we will choose input sequence as $\mathcal{S}_{\text{in}} = \overline{0^{n-m}x}$ and the label as $y = \text{YES}$ if $M(x) = 1$ and $y = \text{NO}$ otherwise.

Because we are considering regular RNN with $O(m)$ memory, we know that we can compute the result of RNN without CoT through a circuit family with size $\text{Poly}(m)$. However, we know that M cannot be simulated by a polynomial-size circuit family. Hence, no RNN family with $O(m)$ memory can solve the task for all n.

On the other hand, we can simulate M by the RNN by maintaining the state, the pointer, and the tape of the M inside the state of the RNN. The RNN can then maintain the transition function of the Turing machine in its output function as a lookup table Lemma B.4 and write down the updated state, the direction of the pointer movement, and the updated cell value at the pointer in its context. Paired with the ability to memorize the recent input Lemma B.20, the RNN can then simulate the running of the Turing machine.

Because the space complexity of M is linear in m, the time complexity of M is $\exp(O(m))$ which is polynomial in n. Hence, we can solve the task by an RNN with CoT and $O(m)$ memory and polynomial-size circuit family. □

B.6 Proof of Theorem 4.7

We will now proceed to prove our main theorem, which states that Transformers with chain-of-thought can solve IsTree perfectly, while RNNs cannot. We will first restate the theorem here.

Theorem 4.7. *For any RNN family R with $o(n)$ bit memory, R cannot perfectly solve IsTree of size n for large enough n, with any length of CoT. On the other hand, there exists a Transformer T_1 with constant dimension and depth, and $O(\log n)$ precision that can solve IsTree of size n perfectly with Chain of Thought of length $O(n)$.*

Proof of Theorem 4.7. We will prove this theorem by proving the following lemmas.

Lemma B.26. *For any n and RNN R with $o(n)$ memory, R can't perfectly solve IsTree of size n.*

Lemma B.27. *There exists a Transformer T with constant depth and width, and $O(\log n)$ precision, that can solve IsTree of size n perfectly with Chain of Thought.*

This proof is a direct combination of Lemmas B.26 and B.27. □

B.6.1 Proof of Lemma B.26

Proof of Lemma B.26. We will first reduce another problem in communication complexity to IsTree.

Lemma B.28. *Consider the following two-party game, where Alice receives string $x \in \{0,1\}^n$ and Bob receives an integer k, and Bob wants to know whether $x_k = x_{k-1}$. If only Alice is allowed to send information, $\Omega(n)$ bit communication complexity is required.*

Proof of Lemma B.28. We can reduce this problem to the problem in Lemma B.24. Considering the game in Lemma B.24, given any $x \in \{0,1\}^n$, we can construct $\tilde{x}_i = \sum_{j=1}^{i} x_i \mod 2$. Then $\tilde{x}$ is a string of length n with only 0 and 1. Moreover, $x_k = x_{k-1}$ if and only if $\tilde{x}_k = \tilde{x}_{k-1}$. Hence, if Bob can solve the problem in Lemma B.28 with $o(n)$ bit, he can solve the problem in Lemma B.24. This is a contradiction. □

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 32

Now suppose that we have a streaming algorithm for IsTree with only $o(n)$ memory. We shall prove Alice and Bob in Lemma B.28 can use it to solve the original question with $o(n)$ memory.

Consider the following graph with $n+2$ nodes. There is a node i corresponding to each x_i for $i \in [n]$ and two special nodes $n+1, n+2$. Node i will be connected to $n+1$ if $x_i = 0$ and to $n+2$ if $x_i = 1$. Moreover, $k-1$ and k will be connected. Now the original answer Bob wants is False if and only if the graph is a Tree. Hence, given access to the streaming algorithm, Alice can run it on the edges that she knows exist and send the memory to Bob. Bob can then run it on the edges that he knows exist. Combining they will be able to solve the original problem. This is a contradiction.

Moreover, as RNN with chain-of-thought is also a streaming algorithm, it also requires $\Omega(n)$ memory. □

B.6.2 Proof of Lemma B.27

Proof of Lemma B.27. The proof is two-folded. We will first define an algorithm that can solve IsTree by generating a sequence of vertices of length $O(n)$, and then we will show that this sequence can be generated by a Transformer with constant depth and width, and $O(\log n)$ precision as a Chain of Thought.

Algorithm 1 Depth-First Search Algorithm

Require: A graph $G = (V, E)$ with n vertices and E has an ordering $e_1, \ldots, e_m$.
Initialize two stacks of vertices S_1, S_2 with $S_1 = [v_1], S_2 = \emptyset$.
while S_1 is not empty **do**
  Let v be the top of S_1. Yield v.
  if there exists a neighbor u of v not in $S_1 \cup S_2$ **then**
    Choose u such that edge (u, v) has the smallest possible order and push u to S_1.
  else
    Pop v from S_1 and push v to S_2.
  end if
end while

Algorithm for IsTree. We define Algorithm 1 as a depth-first search algorithm that can generate a sequence of vertices of length $O(n)$ that can be used to solve IsTree. We will use two stacks S_1, S_2 to store the vertices. S_1 will be used to store the vertices that are not yet visited, and S_2 will be used to store the vertices that are already visited. The algorithm will start with $S_1 = [v_1]$ and $S_2 = \emptyset$. At each step, the algorithm will pop the top of S_1 and push it to S_2. Then it will push all the neighbors of the popped vertex that are not in $S_1 \cup S_2$ to S_1. The algorithm will terminate when S_1 is empty. We will denote the yielded vertice sequence for G as $\mathcal{A}(G)$. The following lemma shows the connection between the result of the algorithm and the IsTree problem.

Lemma B.29. *For any graph G, $\mathcal{A}(G)$ is a tree traversal of a spanning tree of the connected component of G containing v_1. Hence $\mathrm{IsTree}(G)$ is True if and only if G has $n-1$ edges and $\mathcal{A}(G)$ contains $2n-1$ vertices.*

Proof of Lemma B.29. First, every vertex in the connected component of G containing v_1 will be visited. This is because the algorithm will always push all the neighbors of the popped vertex that are not in $S_1 \cup S_2$ to S_1. Hence, the algorithm will terminate when all the vertices in the connected component of G containing v_1 are visited.

Second, every two consecutive vertices in the yielded sequence will be connected by an edge. This is because the algorithm will always push one of the neighbors of the popped vertex that is not in $S_1 \cup S_2$ to S_1. Hence, every two consecutive vertices in the yielded sequence will be connected by an edge. On the other hand, the combination of these edges will form a tree because the algorithm will never push a vertex that is already in $S_1 \cup S_2$ to S_1. Hence, the yielded sequence is a tree traversal of a spanning tree of the connected component of G containing v_1. □

Construction of Transformer. We will now show that the yielded sequence of Algorithm 1 can be generated by a Transformer with constant depth and width, and $O(\log n)$ precision as a Chain of Thought. The Transformer will generate a valid yielded sequence but can terminate early if the graph is not a tree. We will now describe the Transformer in detail. We will assume the input token sequence $\mathcal{S}$ is as follows,

$$\mathcal{S} = \mathrm{Tokenize}(G), v_1, \ldots v_r \tag{10}$$

for some $r \geq 0$ and $v_1 \ldots v_r$ is a valid yielded sequence. The length of $\mathrm{Tokenize}(G)$ is $3n-2$ with 3 tokens for each edges and 1 special token <s>. We will further denote the input to the first layer X as $\mathrm{Emb}(\mathcal{S})$. We will similarly denote the input to layer ℓ as $X^{(\ell)}$. We will also denote the output of the last layer as X^{out}.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 33

1. **Layer 1 and Layer 2 Attention.** The attention at Layer 1 will output zero and the FFN at Layer 1 and Attention at Layer 2 will implement a counting function (Definition B.8) to count the number of vertices n appears in the previous token sequence and write $\mathrm{ROUND}\left(\frac{1}{n}, p\right)$ in a new dimension i_1 as a result.

2. **Layer 2 FFN and Layer 3 Attention.** The FFN at Layer 2 and Attention at Layer 3 will implement a copying function (Definition B.7) copying the first dimension and the counting dimension i_1 of each token to its successor at two new dimensions i_2 and i_3. For each edge, this moves the type of the first vertice and the number of times the first vertice appears to $\sim$. For every vertice in the chain of thought, this moves the type of the previous vertice to them.

3. **Layer 3 FFN and Layer 4 Attention.** The FFN at Layer 3 and Attention at Layer 4 will implement another copying function, copying the dimensions i_2 and i_3 of each token to its successor at two new dimensions i_4 and i_5. Especially, for each edge, this moves the type of the first vertice and the number of times the first vertice appears to the position corresponding to the second vertices.

4. **Layer 4 FFN.** This FFN will process the information gathered from the previous layer and prepare for the next layer. It will make sure the following properties hold for $X^{(5)}$,

   - For every token, the position number, its square, and 1 will be kept in the last three dimensions.
   - For the first vertices in each edges, $\sim$ and <s> The rest dimension will be zero.
   - For the second vertices of each edges (a, b), there will be four dimensions i_6, i_7, i_8, i_9 with value a, b and $n_{a,e}, n_{b,e}$, where $n_{a,e} = \mathrm{ROUND}(\frac{1}{1+\#a\text{appears up to current edge}}, 1)$.
   - For vertice v_l in $v_1, \ldots, v_r$, there will be four dimensions $i_{10}, i_{11}, i_{12}, i_{13}$ with value v_l, v_{l-1} and v_l^2, v_{l-1}^2 ($v_0 = 0$).

5. **Layer 5 Attention.** Combining with the previous Layer 4 FFN layer, we will implement two match functions with two attention heads matching (i_{10}, i_{11}) or (i_{11}, i_{10}) with (i_6, i_7) at Layer 5 Attention, i.e. finding the edge in input for each step in the chain of thought, we will then copy $n_{v_l,(v_l,v_{l-1})}$ to dimensions i_8 and i_9.

6. **Layer 6.** We will use Layer 5 FFN and Layer 6 Attention to implement the match function that matches dimension i_{10} of the current token to i_{10} in the previous token. This will match v_l to the first appearance of v_l in the chain of thought and we will copy i_{11} of the matched token to i_{22}. This dimension will be the first predecessor of v_l in the chain of thought (0 for v_1). We will denote this predecessor of v_l to be $f(v_l)$ as it is the father of v_l in the tree. Now we will need to split into two cases depending on whether v_{l-1} is $f(v_l)$. If $v_{l-1} = f(v_l)$ or $v_{l-1} = 0$ (for v_1), we will set dimension i_8 to be 1 and i_9 to be 0. Otherwise, we will keep dimension i_8 and i_9 as $n_{v_l,(v_l,v_{l-1})}$.

7. **Layer 7.** Now we will use Layer 6 FFN and Layer 7 Attention with two attention heads to implement two $\mathrm{MatchNext}$ functions[5] (Definition B.12) which use i_8 or i_9 as the counting index, and match v_l at i_{10} to i_6 or i_7 respectively. We will then copy dimensions i_6 to i_9 of the matched tokens to i_{14} to i_{21} (because there will be two of them).

   The match next function will be able to retrieve the first edge containing v_1. For any $i \geq 2$, one of the matches next function will be able to retrieve the next edge containing v_i after (v_i, v_{i+1}) if it exists. If it doesn't exist, the corresponding counting dimension will either be zero or no smaller than $n_{v_l,(v_l,v_{l-1})}$. We will use Layer 6 FFN to decide whether the next edge exists and set dimension i_{14} of the output of Layer 6 to be the other edge in the next edge if it exists, or 0 otherwise, and i_{15}, i_{16} of the output of layer 6 to be the counting dimension of the next edge if it exists, or 0 otherwise. For each edge in the original input, we will also set dimension i_{15}, i_{16} to be the counting dimension of the edge.

8. **Layer 8 Attention** We will grab the next edge again, in the same manner as Layer 6, but this time using dimension i_{15} and i_{16}. The necessity of this step is that the next edge containing (v_{i-1}, v_i) in the original graph can be the same as the $(f(v_l), v_i)$ and in such case we need to check whether the next edge after this edge.

9. **Layer 8 FFN.** We now have, at each position corresponding to v_l, the first edge $(f(v_l), v_l)$ in the yielded sequence containing v_l and the other vertex in the edge containing v_l that hasn't been visited if it exists. If they don't exist, the corresponding dimension will be zero. This allows us to use Layer 8s FFN to decide the next vertex in the yielded sequence, which is exactly the first vertex different with $f(v_l)$ in the two edges if they exist, or $f(v_l)$ otherwise. We will use Layer 8 FFN to calculate the potential next vertex and put it in dimension i_{23} and its square in i_{24}.

10. **Layer 9 Attention.** Combining with Layer 8 FFN, we will match i_{23} of the current token to i_{10} of the previous token to find the first appearance of the potential next vertex in the chain of thought. We will then copy dimension d

[5] The constant A here in Definition B.12is 1

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 34

of the matched token to i_{25}. This value being 1 will imply this vertex has never appeared in the chain of thought before and any other value will imply the vertex has appeared before.

11. **Layer 9 FFN.** We can now check several cases,

    - If the potential next vertex v is either $f(v_r) \neq 0$ or never appears in the chain-of-thought sequence, then Layer 9 will output $n[-v, 1, -1, \ldots -1]$, which will decodes to v.
    - If the potential next vertex v is not $f(v_r)$ and appears in the chain-of-thought sequence, then Layer 9 will output nw_6, which will decodes to NO, because the chain of thought has already visited v and hence the graph is not a tree.
    - If $v_r = 1$ and the potential next vertex v is $f(v_r) = 0$, this means the chain of thought has finished. In this case, layer 9 FFN will check whether the position is $3n - 2 + 2n - 1 = 5n - 3$ and output nw_5 if it is, or output nw_6 otherwise, which will decode to YES and NO respectively.

This concludes the construction of the Transformer. □

B.7 Proof of Theorem 4.8

We will now prove Theorem 4.8. We will first restate the theorem for convenience.

Theorem 4.8. *Given any constant $A > 0$, constant word width and number of special symbols $d, n_S > 0$, for any n, precision $p = \Theta(A \log n)$ and RNN R with word embedding $W^{(E)} \in \mathbb{R}_p^{(n+n_S)\times d}$ such that each recurrent iteration can be calculated with a circuit with size $\mathrm{P}(n) \leq 2^{p/2}$, there exists a Transformer T with $O(\mathrm{P}(n) \log \max\{\mathrm{P}(n), n\})$ bit parameter and word embedding $\begin{bmatrix} W^{(E)} & \mathbf{0}^{(n+n_S)\times d} \end{bmatrix}$ that can simulate the RNN with at most n^A step chain-of-thought precisely, using at most $(\mathrm{P}(n) + 1)n^A$ step chain of thought on every input with length n.*

Proof. The proof is motivated by the Theorem B.30 from Li et al. (2024).

Theorem B.30 (Theorem 3.3 of Li et al. (2024))**.** *For any n and any circuit with size $T(n)$ and input size n, there exists a Transformer with constant depth and precision, $O(\log n)$ width, and a position embedding with size $O(T(n) \log n)$, such that for any input $\mathcal{S}$ of length n, the Transformer computes the output of the circuit on $\mathcal{S}$ using $T(n)$ steps.*

However, direct utilization of Theorem B.30 is not feasible because we are interested in (1) $O(\log n)$ precision Transformer, and (2) simulating the RNN for n^A step, which would correspond to a circuit with $n^A \mathrm{P}(n)$ in size. However, as the calculation is recurrent, we can encode the RNN circuit in $O(\mathrm{P}(n))$ parameter instead.

To do so, we will unroll the circuit of each recurrent step of the RNN into $\mathrm{P}(n)$ gates. We will then assign each gate a unique id in $[\mathrm{P}(n)]$ and assume the circuit is calculated in the order of the gate id in the following manner.

1. Each gate has a type $t(i)$, which is either a constant gate outputting 1, an input gate, a hidden state gate, an AND gate, or an XOR gate.
2. Each gate i's output depends on two values $l(i)$ and $r(i)$. If $t(i)$ is a constant gate, then $l(i)$ and $r(i)$ are assigned to be 0. When it is an input gate, $l(i)$ will be assigned to be the coordinate of the input embedding and $r(i)$ will be assigned to be the index of the bit of the value at $l(i)$ coordinate. When it is a hidden state gate, $l(i)$ will be assigned to be the coordinate of the hidden state embedding, and $r(i)$ will be assigned to be the index of the bit of the value at $l(i)$ coordinate. If it is an AND gate or an XOR gate, $l(i)$ and $r(i)$ will be assigned to be the id of the two gates that it depends on.

We will further assume without loss of generality that the hidden state gate is the first $p\Lambda$ gate. The output of the last $p\Lambda$ gate will be the next hidden state. We will also assume that the last $p(\Lambda + d)$ to $p\Lambda - 1$ gates are the output gates. We will now first describe the chain of thought that the Transformer will output and then construct the Transformer.

Chain of thought Taking any input $\mathcal{S}$ with length n, the Transformer will output a sequence of 0 and 1 tokens. The first n tokens will be the same as the input sequence. For each $a \geq 0$ and $b \in [\mathrm{P}(n) + 1]$, the $n + a\,(\mathrm{P}(n) + 1) + b$ token is

1. the output of gate b when RNN circuit is calculating the output at a position plus 1, if $b \leq \mathrm{P}(n)$.
2. the $n + a + 1$ token in the RNN chain of thought, if $b = \mathrm{P}(n) + 1$.

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 35

Construction of the Transformer.

1. **Layer 1.** The first attention layer will output zero and the first FFN layer will be of width $O(\mathrm{P}(n))$, encoding all the gate information. The output of the first layer at position $n + a\,(\mathrm{P}(n) + 1) + b$ will have the following coordinate:
   - The input i will be encoded in the first dimensions.
   - a, a^2, b, b^2 will be encoded in four different dimensions.
   - The gate type $t(s(b))$ will be encoded in the next dimension, where $s(b) = (b + 1) \mod (\mathrm{P}(n) + 1)$ If $b = \mathrm{P}(n) - 1$, then the gate type will be encoded as 0.
   - The necessary dependence $l(s(b)), l^2(s(b))$ and $r(s(b)), r^2(s(b))$ will be encoded in the next two dimensions.
   - A constant 1 will be encoded in the next dimension.
2. **Layer 2 Attention.** Together with the Layer 1 FFN, the Layer 2 Attention will implement two match functions (Definition B.9) to copy the output of gate $l(b + 1)$ and $r(b + 1)$ when RNN circuit is calculating the output at a position. When the type of gate $b + 1$ is not AND or XOR, the result will not be used in the following calculation.
3. **Layer 2 FFN** Layer 2 FFN will be of width $O(1)$. The output of the layer will be
   - When $b < \mathrm{P}(n)$ and $t(s(b))$ is AND or XOR or constant, one dimension of the output will be the output of gate $b + 1$ when RNN circuit is calculating the output at a position.
   - When $b < \mathrm{P}(n)$ and $t(s(b))$ is an input or hidden state gate or $b = \mathrm{P}(n) + 1$, one dimension of the output will be the position in the current chain of thought where the input bit or hidden state bit is copied from and the other dimension will be the square of that position
   - When $b = \mathrm{P}(n)$, the output remains the same as the input to Layer 2 FFN.
4. **Layer 3 Attention.** Layer 3 Attention will be of width $O(1)$. Together with Layer 2 FFN, Layer 3 Attention will implement Match heads (Definition B.9) to copy the output at the position where the input bit or hidden state bit is copied from. When the type of gate $b + 1$ is not input or hidden state gate, the result will not be used in the following calculation.
5. **Layer 3 FFN** Layer 3 FFN will be of width $O(1)$. The output of the layer will be
   - When $b \neq \mathrm{P}(n)$, one dimension of the output will be output of gate $s(b)$ when RNN circuit is calculating the output at $a + \mathbf{1}[b = \mathrm{P}(n) + 1]$ position.
   - When $b = \mathrm{P}(n)$, the output remains the same as the input to Layer 3 FFN.
6. **Layer 4** Layer 4 Attention will have $p - 1$ heads and each head will be of width $O(1)$. Head $h \in [p - 1]$ will copy the first dimension of the output of Layer 3 FFN at position $n + a(P(n) + 1) + b - (p - h)$ and weight each of them by $2^{-h+(p-1)/2}$ and add them in one dimension. The Layer 4 FFN will calculate r when the first dimension of the input is 1 and $-r$ otherwise. Hence, for each $a \geq 0$, the $n + a\,(\mathrm{P}(n) + 1) - hp, h \in [\Lambda : \Lambda + d]$ token contains a dimension i_1 which is the $k - \Lambda$ dimension of the output of the RNN at position a.
7. **Layer 5** Layer 5 Attention will have $d + 1$ heads and each head will be of width $O(1)$. Head $h \in [d + 1]$ will copy the dimension i_1 of the output of Layer 4 FFN at position $n + a(P(n) + 1) + b - (h + \Lambda)p$ to a disjoint dimension i_{h+1}. The Layer 5 FFN will then make sure the output of Layer 5 satisfies the following:
   - When $b \neq \mathrm{P}(n)$, one dimension of the output will be n times the output of gate $s(b)$ when RNN circuit is calculating the output at $a + \mathbf{1}[b = \mathrm{P}(n) + 1]$ position plus 1, which will decode to the corresponding value.
   - When $b = \mathrm{P}(n)$, the first d dimension of the output will be the same as the output of the RNN at position a, and the rest dimension will be 0, which will decode to the same token as the chain of thought of the RNN at position $a + 1$.

This concludes the construction of the Transformer. □

B.8 Proof of Theorem 5.2

We will first restate the theorem for clarity.

Theorem 5.2. *For task $T \in \{$Index, AR, c-gram retrieval, Counting$\}$, there exists a Linear RNN family with $O(\log n)$ bit memory and $O(\log n)$ parameter, that can solve T with In-context RAG in $O(1)$ CoT steps.*

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 36

Proof. When task T is Index, given an input sequence that ends with a query token k, RNN will generate the following search query sequence:<StartSearch>, `^(?:\S\s*){`$k-1$`}(\S)`, <EndSearch>.

Then the regular expression will match the k-th token in the input sequence. The RNN needs to recite the format of the query, remember the index k and calculate $k-1$ to generate the regular expression. As we have shown in Lemmas B.20 and B.23, RNN can recite a fixed sequence at fixed position and memorize the recent input, the above sequence can be generated by an RNN. The explicit construction of the RNN is omitted here.

When task T is AR, given an input sequence that ends with a query token k, RNN will generate the following search query sequence: <StartSearch>, `\b` k `\b(\S+)\b`, <EndSearch>.

Then the regular expression will match the next token after the first occurrence of k in the input sequence. The RNN needs to recite the format of the query and remember the query token k to generate the regular expression. The explicit construction of the RNN is omitted here.

When task T is c-gram retrieval, given an input sequence that ends with query tokens, RNN will generate the following search query sequence: <StartSearch>, `\b` $k_1 \dots k_{c-1}$ `\b(\S+)\b`, <EndSearch>.

Then the regular expression will match the next token after the first occurrence of $k_1, \dots k_{c-1}$ in the input sequence. The RNN needs to recite the format of the query and remember the query tokens $k_1, \dots k_{c-1}$ to generate the regular expression. The explicit construction of the RNN is omitted here.

When task T is Counting, given an input sequence that ends with a query token v and a query threshold k, RNN will generate the following search query sequence <StartSearch>, `(\b` v `\b){`$k+1$`}`, <EndSearch>.

Then the regular expression will match the k-th occurrence of v in the input sequence. The RNN needs to recite the format of the query and remember the query token v and the threshold k to generate the regular expression. The RNN can then check whether the retrieval result is FAILED to determine if the count is less than k. The explicit construction of the RNN is omitted here.

□

B.9 Proof of Theorem 5.3

In this section, we will prove Theorem 5.3. We will first restate the theorem for convenience.

Theorem 5.3. *There exists a Linear RNN family with $O(\log n)$ bit memory and $O(\log n)$ parameter, that can solve IsTree of size n with In-context RAG in $O(n)$ CoT steps.*

Proof of Theorem 5.3. We will first define the sequence that the retrieval-augmented RNN will generate and then construct an RNN that can generate such a sequence.

Sequence Generation. We will use a variant of Algorithm 1 to generate the sequence and we will use the concatenation of the tokenization of the sequence returned by Algorithm 2 as the sequence that the retrieval augmented RNN will generate.

RNN Construction. We can use similar proof in Theorem 5.4 by having the RNN memorize local sequences and determine the phase of Algorithm 2 it is in. The RNN will maintain the length of S_2 (Lemma B.21) and the top of S_1 in the state (Lemma B.20) and it is easy to check that the retrieval function will retrieve the correct result for each search query. The way to determine the next vertex in the stack is the same as in the proof of Lemma B.27. We will omit the simple but tedious detailed construction here. □

B.10 Proof of Theorem 5.4

In this section, we will prove Theorem 5.4. We will first restate the theorem for convenience.

Theorem 5.4. *Given any constant A, B, for any polynomial-time Turing machine $T \in \mathrm{TIME}(n^A)$ with B states and vocabulary size B, there exists a retrieval augmented Linear RNN family (see Definitions 3.6 and A.5) with vocabulary of B special symbol, $O(A \log n)$ bit precision and memory, and $O(AB^2 \log n)$ bit parameters, that can simulate the result of T on any input with length n with a chain of thought of length $O(n^A)$.*

Proof of Theorem 5.4. We will denote the state of T as $1, \dots, B$ (we will use 1 as the initial state) and the vocabulary of T as $1, \dots, B$. We will assume T operates on an infinite tape TAPE, which is a sequence of cells indexed by $\mathbb{Z}$. We will also assume that the tape is initialized with all cells being 0 except for the n cell starting at 1. The Turing machine

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 37

Algorithm 2 Depth-First Search Algorithm with Retrieving

Require: A graph $G = (V, E)$ with n vertices and E has an ordering $e_1, \ldots, e_m$.
Initialize two stacks of vertices S_1, S_2 with $S_1 = [v_1]$, $S_2 = \emptyset$, a list L with $L = \emptyset$, and a vertex $v' =$ FAILED.
while S_1 is not empty **do**
  Let v be the top of S_1. Push v to L.
  Generate the regular expression $r_2 =$ `\b(\S+)\b-\b`v`\b.`
  Let $f(v)$ be the predecessor of v in S_1 for the first time and FAILED when $v = v_1$.
  Push <StartSearch>, r_2, <EndSearch>, $f(v)$ to L.
  if $v' \neq f(v)$ **then**
    Generate the regular expression
    $r_1 =$ `(\b`v'`\b~\b`v`\b|\b`v`\b~\b`v'`\b).*?(\b(\S+)\b~\b`v`\b|\b`v`\b~\b(\S+)\b)`
  else
    Generate the regular expression $r_1 =$`\b(\S+)\b~\b`v`\b|\b`v`\b~\b(\S+)\b`
  end if
  Push <StartSearch>, r_1, <EndSearch> to L.
  if there exists a neighbor u of v such that (u, v) has larger order than (v, v') when $v' \neq f(v)$ or there exists a neighbor u of v such that $u \neq f(v)$ when $v' = f(v)$ **then**
    Choose u such that edge (u, v) has the smallest possible order and push u to L. Let $v'' = u$.
  else
    Push FAILED to L. Let $v'' =$ FAILED.
  end if
  if $v'' = f(v) \neq$ FAILED **then**
    Generate the regular expression
    $r_3 =$ `(\b`v''`\b~\b`v`\b|\b`v`\b~\b`v''`\b).*?(\b(\S+)\b~\b`v`\b|\b`v`\b~\b(\S+)\b)`
    Push <StartSearch>, r_3, <EndSearch> to L.
    if there exists a neighbor u of v such that (u, v) has larger order than (v, v'') **then**
      Choose u such that edge (u, v) has the smallest possible order and push u to L. Let $v'' = u$.
    else
      Push FAILED to L. Let $v'' =$ FAILED.
    end if
  end if
  if $v'' =$ FAILED **then**
    Pop v from S_1. Push v to S_2. Let $v' = v$.
  else
    Generate the regular expression $r_4 =$`\b(\S+)\b-\b`v''`\b`
    Push <StartSearch>, r_4, <EndSearch>, 0 to L.
    if v'' is not in S_1 **then**
      Push FAILED, v, $-$, v'' to L.
      Push v'' to S_1. Let $v' = v$.
    else
      Let $f(v'')$ be the predecessor of v in S_1 for the first time and FAILED when $v'' = v_1$.
      Push $f(v'')$, NO to L.
      return L.
    end if
  end if
end while
if S_2 has n vertices **then**
  Push YES to L.
  return L.
else
  Push NO to L.
  return L.
end if

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 38

also has a pointer p that points to a cell in the tape. The pointer is initialized to 1. At each time step, the Turing machine reads the cell pointed by POINTER and updates the cell pointed by POINTER and the pointer p according to the transition function $\delta : [B+1] \times [B] \to [B] \times [B] \times \{-1, 1\}$, which takes the current state and the current cell value (could be empty, which corresponds to $B+1$) as input and outputs the new state, the new cell value and the direction to move the pointer. The Turing machine halts when the state is B. Because $T \in \mathrm{TIME}(n^A)$, the Turing machine will always halt in n^A steps. We will use $\mathrm{TAPE}[t, i]$ as the value on the i-th cell on TAPE before the t timestep. We will use $\mathrm{POINTER}[t]$ as the value of the pointer before the t timestep and $\mathrm{State}[t]$ as the state of the Turing machine before the t timestep. We will further use $\mathrm{Direction}[t]$ as the direction of the pointer movement before the t timestep.

We will first define the sequence that the retrieval-augmented RNN will generate and then construct an RNN that can generate such a sequence.

Sequence generation. The input token sequence $\mathcal{S}_{\text{in}}$ will be as followed,

$$\mathcal{S}_{\text{in}} = \texttt{<s>}, \mathrm{TAPE}[1,1], \mathrm{TAPE}[1,2], \ldots, \mathrm{TAPE}[1,n]$$

Here all the symbols on the tape are represented by one special symbol in the vocabulary. Given this input token sequence, the retrieval augmented RNN will generate the following output token sequence,

$$\begin{aligned}
\mathcal{S} = \mathcal{S}_{\text{in}}, &\texttt{<StartSearch>}, \texttt{^(?:\S\s*).\{1\}(\S)}, \texttt{<EndSearch>}, \\
&\mathrm{TAPE}[1,1] \\
&1, \mathrm{TAPE}[1,1], 1, \\
&\ldots \\
&\texttt{<StartSearch>}, \texttt{^(?:\S\s*).\{n\}(\S)}, \texttt{<EndSearch>}, \\
&\mathrm{TAPE}[1,n] \\
&n, \mathrm{TAPE}[1,n], n, \\
&\texttt{<StartSearch>}, ((\mathrm{POINTER}[1]\ \ (.)\ \ \mathrm{POINTER}[1]\ \texttt{.*?\$})), \texttt{<EndSearch>}, \\
&\mathrm{SearchResult}(1), \\
&\mathrm{POINTER}[1], \mathrm{TAPE}[2, \mathrm{POINTER}[1]], \mathrm{POINTER}[1] \\
&\mathrm{State}[2], \mathrm{Direction}[2], \\
&\ldots \\
&\texttt{<StartSearch>}, (\mathrm{POINTER}[t]\ \ (.)\ \ \mathrm{POINTER}[t]\ \texttt{.*?\$}), \texttt{<EndSearch>}, \\
&\mathrm{SearchResult}(t) \\
&\mathrm{POINTER}[t], \mathrm{TAPE}[t+1, \mathrm{POINTER}[t]], \mathrm{POINTER}[t], \\
&\mathrm{State}[t+1], \mathrm{Direction}[t+1],
\end{aligned}$$

Here $\mathrm{SearchResult}(t)$ is defined as

$$\mathrm{SearchResult}(t) = \begin{cases} \text{FAILED; if } \mathrm{POINTER}[t] \text{ is empty cell before } t \\ \mathrm{TAPE}[t, \mathrm{POINTER}[t]]; \text{otherwise} \end{cases}$$

The output token sequence simulates the Turing machine T on the tape TAPE due to the following lemma.

Lemma B.31. *Given any $t \in [n^A]$ and $i \in [n^A]$, the last string in $\mathcal{S}$ that contains i, i as a substring is $i, \mathrm{TAPE}[t,i], i$ if $\mathrm{TAPE}[t,i]$ is not empty and is the empty string otherwise.*

Proof. The proof is by induction, for $t = 1$, the result holds. For any $t \geq 2$, we only need to notice that $\mathrm{POINTER}[t-1]$ is the only cell that can be updated at time $t-1$. □

Construction of RNN Given the input, the RNN can first iterate over 1 to n and generate the first n search queries and results by maintaining a counter in its state and memorizing the most recent search result (Lemma B.20). Then it is easy to see that the retrieval oracle will generate the correct $\mathrm{SearchResult}(t)$ given the input $\mathcal{S}$. Therefore, we will only need to construct an RNN that can generate the rest part of $\mathcal{S}$.

We will assume the RNN maintains the state and pointer of the Turing machine in its state and show that they can be updated.

Based on Lemma B.20, the RNN can maintain constant recent token types in its state, we will assume the RNN memorize the last tokens up to the most recent <StartSearch> and also calculate the position relative to the most recent

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 39

<StartSearch>. By a lookup table in the FFN Lemma B.4, the RNN can output the fixed format of the search query. Similarly, RNN can output the $\mathrm{POINTER}[t]$. To generate the update $\mathrm{TAPE}[t+1, \mathrm{POINTER}[t]], \mathrm{State}[t], \mathrm{Direction}[t]$, the RNN can use a FFN with $O(B^2)$ width to memorize the transition function of the Turing machine and output the update. Then, the RNN can use the memorized recent input to update the state and the pointer of the Turing machine at the next <StartSearch>. The proof is then complete.

□

B.11 Proof of Theorem 5.6

Theorem 5.6. *For task $T \in \{$Index, AR, c-gram retrieval, Counting$\}$, there exists a hybrid Linear RNN (Definitions 3.6 and 5.5) family with $O(\log n)$ bit memory and $O(\log n)$ parameter, that can solve T without CoT.*

Proof. The proof here is essentially the same as the construction of the Transformer in Theorem 4.6. We would use the same Transformer layer to solve T. The only difference is that we would use the output of the RNN, instead of FFN, as the input of the Transformer layer.Also for Counting, instead of using a COPY function, we write the query token in the state of the RNN (Lemma B.20). □

B.12 Proof of Theorem 5.7

Theorem 5.7. *There exists a hybrid Linear RNN with $O(\log n)$ bit memory and $O(\log n)$ parameter, that can solve IsTree of size n with a chain of thought of length $O(n \log n)$.*

Proof. The proof is similar to the proof of Theorem 5.3. However, instead of using regular expressions to retrieve the next neighbor and parent, we will need to use the Transformer layer. The Transformer layer can retrieve the parent through an attention head implementing the match closest head (Lemma B.16) if the RNN part maintains the predecessor of each node in the chain of thought.

Retrieving the next neighbor is more complicated and we will use $O(\log n)$ steps of the chain of thought to do that. Given an edge (v, v'), we will first use one match head to retrieve the position p of (v, v') in the input sequence and write it to the chain of thought. Then we will use two $\mathrm{MatchClose}$ heads to retrieve the edge that contains v and is closest to $p + 2^i$ for $i = 0, 1, \ldots, \log_2 n$ iteratively until the heads return an edge that is not (v, v') or i reaches $\log_2 n$. Here 2^i can be computed through doubling one of the dimensions in the state of the RNN and reset that dimension to 1 after termination. We will then compare the retrieved edge with the father of v to check if it is the same. If it is the same, we will search the next neighbor of v after the parent of v in the same way. The other part of the proof is similar to the proof of Theorem 5.3. □

B.13 Proof of Theorem 5.8

Theorem 5.8. *Given any constant A, B, for any polynomial-time Turing machine $T \in \mathrm{TIME}(n^A)$ with B states and vocabulary size B, there exists a hybrid Linear RNN (see Definition 5.5) with vocabulary of B special symbol, $O(A \log n)$ bit precision and memory, and $O(AB^2 \log n)$ bit parameters, that can simulate the result of T on any input with length n in $O(n^A)$ CoT steps.*

Proof. **Sequence Generation.** Under the same formulation of proof of the Theorem 5.4. The hybrid RNN will output the following sequence.

$$\begin{aligned}\mathcal{S} = \mathcal{S}_{\text{in}}, &\mathrm{POINTER}[1], \mathrm{TAPE}[2, \mathrm{POINTER}[1]], \mathrm{POINTER}[1] \\ &\mathrm{State}[2], \mathrm{Direction}[2], \\ &\ldots \\ &\mathrm{POINTER}[t], \mathrm{TAPE}[t+1, \mathrm{POINTER}[t]], \mathrm{POINTER}[t], \\ &\mathrm{State}[t+1], \mathrm{Direction}[t+1],\end{aligned}$$

Note that Lemma B.31 still holds. We only need to prove that the hybrid architecture can generate the above sequence.

Hybrid Construction. The way RNN maintains the pointers and the states is the same as the proof of Theorem 5.4. Given each pointer value, we can retrieve the last value of the cell at the pointer through the one layer of attention by implementing a match closest head (Lemma B.16). □

FriendliAI Inc. / Hugging Face, Inc.
Exhibit 2019
Page 40