Extracting Finite State Machines from Transformers

Rik Adriaensen [1] **Jaron Maene** [1]

Abstract

Fueled by the popularity of the transformer architecture in deep learning, several works have investigated what formal languages a transformer can learn. Nonetheless, existing results remain hard to compare and a fine-grained understanding of the trainability of transformers on regular languages is still lacking. We investigate transformers trained on regular languages from a mechanistic interpretability perspective. Using an extension of the L^* algorithm, we extract Moore machines from transformers. We empirically find tighter lower bounds on the trainability of transformers, when a finite number of symbols determine the state. Additionally, our mechanistic insight allows us to characterise the regular languages a one-layer transformer can learn with good length generalisation. However, we also identify failure cases where the determining symbols get misrecognised due to saturation of the attention mechanism.

1. Introduction

Transformers (Vaswani et al., 2017) have become a mainstay architecture in natural language processing and deep learning more generally. There has hence been increasing interest in characterising the expressive power of this architecture. Significant effort has been devoted to proving what formal languages can or cannot be *expressed* by the transformer architecture (Ackerman & Cybenko, 2020; Merrill, 2021; Strobl et al., 2023). The *trainability* remains less well understood, however. Indeed, the fact that a transformer can express a formal language does not necessarily mean it can also learn it from data.

As an example, consider the parity language, which consists of all binary sequences with an odd number of ones. Chiang & Cholak (2022) proved that soft attention transformers can express this language. However, empirical studies found that transformers trained on the parity language fail to generalise to longer sequences (Anil et al., 2022; Delétang et al., 2022; Liu et al., 2023). Worryingly, Bhattamishra et al. (2020) even found that transformers fail to learn the parity language for sequences with in-distribution lengths.

How can we draw consistent conclusions from these seemingly contradictory results? Strobl et al. (2023) show that different underlying assumptions cause these inconsistencies. Additionally, the specific setup of the training task and evaluation metric are important. We argue that using *mechanistic interpretability*, by unveiling exactly what the transformers learn, allows us to draw more robust conclusions on trainability, thus preventing future contradictions.

To achieve a mechanistic understanding of transformers trained on regular languages we automatically reverse-engineer the finite state machines they learned. This approach, originally proposed for recurrent neural networks (RNN) (Weiss et al., 2018b), relies on the classical L^* algorithm (Angluin, 1987) to extract state machines using queries and counterexamples. We further localise the states of finite state machines in the transformer and investigate how faithfully an extracted state machine models the transformer. We observe that when a transformer has effectively generalised on a language, the states of the target finite state machine are represented as directions in the transformer's output layer. Upon identifying these directions, we can precisely determine what occurs in the instances where the transformer fails.

2. Preliminaries

We will assume basic familiarity with transformers (Vaswani et al., 2017). In this section, we briefly introduce two formalisms commonly used to characterise transformers: finite state machines and Boolean circuits.

We write Σ or Γ for a finite set of symbols, also called an *alphabet*. The *Kleene closure* Σ^* of an alphabet Σ is the set of all finite sequences using the symbols of Σ. For example, the binary alphabet $\Sigma = \{0, 1\}$ has as closure $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$. We denote the empty sequence with ϵ. A *language* L over the alphabet Σ is a subset of Σ^*.

[1]KU Leuven, Department of Computer Science, Leuven, Belgium. Correspondence to: Rik Adriaensen <rik.adriaensen@student.kuleuven.be>.

Mechanistic Interpretability Workshop, Vienna, Austria. 2024.

2.1. Finite State Machines

Finite state machines are among the most well-known models to describe formal languages. Although several variants exist, most can be reduced to deterministic finite automatons. For a more thorough introduction, we refer to Hopcroft et al. (2006).

Definition 2.1. A *deterministic finite automaton* (DFA) is a tuple $(Q, \Sigma, \Gamma, q_0, \delta)$, where

- Q is a finite set of states.
- Σ is the input alphabet.
- $q_0 \in Q$ is the starting state.
- $\delta : Q \times \Sigma \to Q$ is the transition function, mapping the current state and an input symbol to a new state.
- $F \subset Q$ is the set of final states.

Given a sequence $s \in \Sigma^*$, a symbol $\sigma \in \Sigma$ and a state $q \in Q$, we define the repeated application of the transition function $\hat{\delta} : Q \times \Sigma^* \to Q$ as $\hat{\delta}(q, \sigma.s) = \hat{\delta}(\delta(q, \sigma), s)$, with $\hat{\delta}(q, \epsilon) = q$ in the base case. We say that a DFA *accepts* a sequence s if and only if $\hat{\delta}(q_0, s) \in F$. Each DFA $\mathcal{D}$ hence represents a language $L(\mathcal{D}) = \{s \mid \hat{\delta}(q_0, s) \in F\}$. A language is *regular* if it can be represented by a DFA.

A *Moore machine* is a DFA without final states and with an output function $\gamma : Q \to \Gamma$, where Γ is the output alphabet. On each transition, a Moore machine not only takes an input symbol but also produces an output symbol using γ. So while DFAs are sequence classification models, Moore machines are sequence-to-sequence models, aligning more closely with transformers. Note that DFAs can be seen as a special case of Moore machines, where $\Gamma = \{\text{Reject}, \text{Accept}\}$.

2.2. Circuit Complexity

A Boolean circuit C_n is a Boolean formula represented as a directed acyclic graph. For a full introduction, we refer to Arora & Barak (2006). A circuit C_n has n input nodes and a single output node with no outgoing edges. Inner nodes represent either the AND, OR, or NOT operations. The NOT nodes have a single incoming edge, while AND and OR nodes can have an arbitrary number of incoming edges. A circuit computes a function $C_n : \{0,1\}^n \mapsto \{0,1\}$, accepting a sequence $s \in \{0,1\}^n$ if $C_n(s) = 1$. The depth of a circuit $D(C_n)$ is the length of the longest path between an input node and the output node. The size of a circuit $|C_n|$ is the number of nodes in C_n.

A *circuit family* $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$ is a set containing a Boolean circuit for each sequence length. The family $\mathcal{C}$ induces a function $\mathcal{C}(s) = C_{|s|}(s)$ and accepts the language $L(\mathcal{C}) = \{w \mid \mathcal{C}(w) = 1\}$. The depth and size of $\mathcal{C}$ are now functions $n \mapsto D(C_n)$ and $n \mapsto |C_n|$.

We next present two relevant classes of languages in terms of Boolean circuits.

Definition 2.2. A language is in AC^0 if it can be recognised by a family of circuits with constant depth and size polynomial in n.

Definition 2.3. The class TC^0 is defined the same as AC^0, but also allows MAJORITY nodes. These nodes output 1 if and only if at least half of their inputs are 1.

It is well-known that $AC^0 \subsetneq TC^0$ (Vollmer, 1999). Notably, the hierarchy of Boolean complexity classes does not coincide with the Chomsky hierarchy. Indeed, the regular languages span different circuit classes, that are known or conjectured to be different. It has been conjectured that the expressivity of transformers aligns with Boolean complexity, and not the Chomsky hierarchy (Strobl et al., 2023).

The class of regular languages in AC^0 can be further divided in the star-free and non-star-free languages. The star-free can in turn be subdivided according to their dot-depth.

Definition 2.4. (Barrington et al., 1992) The *star-free* regular languages are those subsets obtained by closing over concatenation, complement and union beginning with the letters of the alphabet and the empty sequence.

For example, the language containing no consecutive zeros can be constructed as $\overline{\Sigma^* 00 \Sigma^*}$ and is therefore star-free.

Definition 2.5. The *dot-depth* of a star-free language is defined inductively. Closing the letters of the alphabet and the empty sequence over complement and union yields the languages of dot-depth zero. The dot-depth of a star-free language is the smallest n, such that it can be constructed from the closure of star-free languages of dot-depth $n-1$ under concatenation, using complement and union.

2.3. Example Languages

We consider the following regular languages defined over the binary alphabet $\Sigma = \{0, 1\}$. These languages are chosen to cover the star-free languages, AC^0 and TC^0, all known to be expressable by transformers (Merrill & Sabharwal, 2023).

- **Ones**: 1^*. Ones includes all sequences containing only ones. Ones is star-free with dot-depth one.
- **First**: $1(0 \mid 1)^*$. First contains all sequences that start with a one. First is star-free with dot-depth one.
- **Depth-bounded Dyck**: $\mathcal{D}_i$. The depth-bounded Dyck languages contain all sequences where the two symbols are correctly balanced and the maximal nesting depth is bounded. We denote the Dyck language with maximum depth i as $\mathcal{D}_i$. For example, $\mathcal{D}_1 = (01)^*$ and $\mathcal{D}_2 = (0(01)^*1)^*$. The language $\mathcal{D}_i$ is star-free with dot-depth i.

- **Modulo length**: $\mathcal{C}_q$. The languages $\mathcal{C}_q$ have the unary alphabet $\Sigma = \{0\}$ and contain all sequences s with length $|s|$ modulo $q = 0$. All languages $\mathcal{C}_q$ are in AC^0, but not star-free.
- **Parity**. Parity contains all sequences with an odd number of ones. Parity is famously within TC^0, but outside of AC^0.

3. Trainability of Transformers

In line with previous work, we distinguish between *expressivity* and *trainability* (Strobl et al., 2023). Expressivity asks whether a transformer can *represent* a language, while trainability asks whether a transformer can *learn* a language from data. Although we investigate trainability, expressivity is still relevant as it is a prerequisite for trainability.

As opposed to expressivity, the study of trainability on transformers has been mostly empirical. Unfortunately, these studies can be hard to compare due to subtle variations in the task. We argue that all tasks can be unified as learning to simulate a Moore machine.

Our work focuses on the trainability of regular languages and finite state machines. Although it is generally accepted that transformers are not well aligned with the Chomsky hierarchy (Delétang et al., 2022), the existing work on expressivity is often still structured along this hierarchy (Bhattamishra et al., 2020; Merrill, 2021). Moreover, this allows us to draw upon the methods and results for RNNs, which are more closely related to regular languages (Giles et al., 1991). Finally, Bricken et al. (2023) recently observed transition functions resembling state machines in transformers trained on natural language, demonstrating the real-world relevance of this formalism.

Next character prediction Bhattamishra et al. (2020) conducted a large experimental study on the trainability of transformers on different languages in the Chomsky hierarchy. They trained on a multilabel classification task where, given a sequence, the transformer predicts which next symbols are a valid continuation. A symbol σ is a valid continuation of a sequence w if there exists a $w' \in \Sigma^*$ such that $\hat{\delta}(w.\sigma.w') \in F$. The transformer accepts a sequence when it considers each symbol a valid continuation and predicts the end-of-sequence symbol valid on the final position. We call this setting next character prediction. We can frame this as a Moore machine with the output alphabet $\Gamma = 2^{\Sigma \cup \{\text{Accept}\}}$ where Σ is the language alphabet.

Bhattamishra et al. (2020) presented the transformer only with positive examples, resembling a language modelling task. In their analysis, they hypothesised that transformers only generalise well on star-free languages with dot-depth 1, and not for higher depths.

State prediction Liu et al. (2023) investigated transformers learning semiautomata. Semiautomata are a DFA variant without a notion of final or non-final states. Given an input sequence s, the transformer is trained to predict the sequence of states visited by the target semiautomaton. So when δ is the transition function, the transformer must predict the generalised transition function $\hat{\delta}(q_0, s_{1:i})$ for each position $1 \leq i \leq |s|$. This is a multiclass classification task and can be seen as a Moore machine with $\Gamma = Q$, meaning γ is the identity function.

Liu et al. (2023) argue that transformers must use *shortcut solutions* to compute the recurrent dynamics of these automata in their limited number of layers. They proved that any semiautomaton can be simulated up to length T by a transformer with $O(\log(T))$ layers and a sufficiently large embedding, attention and dense layer width. Furthermore, a constant depth solution exists for solvable semiautomata, with sufficiently large embedding and attention width, and a dense layer width linear in T. Experimentally, they found transformers can learn any semiautomaton with near-perfect accuracy, including Parity (a non-star-free language), which is seemingly at odds with the results of Bhattamishra et al. (2020). However, the found solutions did not generalise to longer sequence lengths. The authors did not conduct a full mechanistic analysis, nor claim transformers find the shortcuts they proposed.

Membership prediction Predicting membership of a language can also be modelled as a Moore machine. The output function is then binary, $\Gamma = \{\text{Accept}, \text{Reject}\}$, and the problem reduces to simulating a DFA as γ encodes the final states (Weiss et al., 2018a;b).

We refer to the minimal DFA accepting a language as the target DFA of that language. The target Moore machine of a language is then equivalent to the target DFA with the appropriate output function and alphabet.

The relationship between different training tasks is now more explicit. Membership prediction is similar to state prediction but with partial observability, as all accepting states (and all non-accepting states) get the same label. For the next character prediction task, the observability depends on the specific machine, in particular, whether the sets of next valid symbols are unique in each of the states.

4. Extracting State Machines

Assuming that a transformer faithfully implements a state machine, it must be possible to *extract* said state machine. Weiss et al. (2018b) proposed a method for extracting DFAs from RNNs using the exact learning algorithm L^* (Angluin, 1987). We briefly summarise their approach, before specifying our changes.

The L^* algorithm learns a DFA by asking queries to a teacher oracle. These queries are either 1) *membership* queries, asking whether the target language accepts a given sequence, or 2) *equivalence* queries, asking whether a given DFA implements the target language and returning a counterexample should they differ. For a full explanation of the L^* algorithm, we refer to Appendix A.

The method of Weiss et al. (2018b) boils down to implementing these two queries using an RNN. A membership query is straightforward as the RNNs are trained on membership prediction. Answering equivalence queries is generally intractable, however. For this reason, the hidden activation space is partitioned, such that each partition represents a state and state transitions are defined by a single sample in that partition (Giles et al., 1991). In other words, Weiss et al. (2018b) keep track of two DFAs: the DFA $\mathcal{D}$ being extracted by L^*, and the DFA $\mathcal{D}'$ specified by the partitioning of the activation space.

To answer equivalence queries, the states of $\mathcal{D}$ and $\mathcal{D}'$ are traversed in parallel. Two types of conflicts may occur. The first possibility is that $\mathcal{D}$ is different from the target language of the RNN, in which case we can return a counterexample. The second possibility is that the partitioning $\mathcal{D}'$ differs from $\mathcal{D}$, in which case the partitioning is refined.

Weiss et al. (2018b) employ three additional tactics to improve practical performance. 1) They apply a time limit after which the algorithm is stopped and the final proposed DFA is returned. 2) They use an aggressive initial splitting strategy on the first set of conflicting vectors, by splitting along the dimensions where these vectors differ the most. The number of dimensions d to initially split on is a hyperparameter called the *initial split depth*. 3) They sometimes supply starting examples of at least one positive and one negative example for the L^* learner to get started. These last two additions prevent the algorithm from terminating prematurely on an automaton with a single state.

4.1. Extracting Moore Machines

With only minor changes, the above method can be adapted to extract Moore machines from transformers instead of DFAs from RNNs.

The method of Weiss et al. (2018b) is not specific to RNNs but can be used on any *neural acceptor* as long as the chosen state vectors are consistent, i.e., they can be partitioned such that all vectors belonging to the same cluster are either rejecting or accepting. In a one-layer transformer, we propose using the final activations of the residual stream after layer normalisation. Layer normalisation projects the activations on a $d-1$ dimensional hypersphere of radius $\sqrt{d}$, with d the dimension of the residual stream (Brody et al., 2023). So at that point, all activation vectors are normalised*. From these activation vectors, the output is computed with a final affine transformation. As such, they are analogous to the state vectors of the RNNs in the original study.

To find Moore machines instead of DFAs we use a generalisation of L^*. More specifically, we learn a single Moore machine, which is a special case of the Moore machine product considered by Moerman (2019). As opposed to learning DFAs, learning Moore machines allows us to consider transformers with an arbitrary output alphabet.

5. Experiments

In this section, we analyse the performance of transformers trained on regular languages. Furthermore, we apply our extraction method to analyse the Moore machines learned by transformers on the different settings discussed in Section 3.

5.1. Experiment Setup

We use the same training method and transformer architecture throughout all experiments to ensure reproducibility and comparability.

Datasets The training datasets contain 10,000 sequences of length 32 labelled using the task-specific target Moore machine. The validation sets contain 2,000 sequences of length 100. To evaluate the transformers after training we use one test set of length 100 and another of length 1,000, both containing 1,000 sequences. The data is either sampled uniformly at random or contains only positive examples, i.e. examples in the target language. We call the latter positive-only learning and employ it when reproducing the results of Bhattamishra et al. (2020). The positive examples are generated by traversing the target DFA at random after removing the garbage state for efficiency.

We often use length ranges as for some languages the sequence classification is identical for sequences of certain fixed lengths. For example, exactly the sequences of even length are part of the language $\mathcal{C}_2$. Each example is prepended with a unique beginning-of-sequence symbol, allowing the transformer to learn an output for the starting state.

Architecture We use one-layer transformers with soft attention (Vaswani et al., 2017), pre-norm layer normalisation (Nguyen & Salazar, 2019), and rotary positional encodings (Su et al., 2024). The latter two techniques have been

*After centering and normalisation the resulting vectors are typically *recentered* and *rescaled* by the scaling vector γ and bias β. However, at inference time this is simply one more affine transformation that can be folded into the final output transformation (Elhage et al., 2021).

demonstrated to enhance the original transformer architecture in terms of training efficiency and length generalisation, respectively. We use a residual stream width of 16, a single-layer MLP layer of width 64 and 4 attention heads. This is a subset of the transformer configurations studied in Bhattamishra et al. (2020). We do not compare different transformer configurations as this has been extensively done before and we instead aim to show transformers can be mechanistically interpreted.

Training We use the Adam optimizer (Kingma & Ba, 2015) with a learning rate of $3 \cdot 10^{-4}$, no learning rate schedule, and a batch size of 32. We employ early stopping on the validation loss, with a patience of 100 epochs, and select the best-performing weights on the validation set. We run each experiment three times with identical hyperparameters and a different seed.

5.2. Positive-Only Character Prediction

First, we consider the next character prediction setting from Bhattamishra et al. (2020). Remember that in this setting, a sequence is accepted when the transformer considers each symbol valid and predicts the end-of-sequence symbol as valid on the final position. We refer to this evaluation measure as the *sequence accuracy*.

Table 1 shows the F1 scores and sequence accuracies on the training and validation sets and the sequence accuracy on two datasets containing *positive-only* data with lengths ranging from 100 to 104 and 1000 to 1004. In Table 2, we repeat the same evaluation on *randomly sampled* data. We also include the F1 score of a trivial baseline that predicts all symbols as valid. In both tables, results are averaged over three runs. It is enough for a single experiment to learn a language perfectly to demonstrate trainability. Therefore, if at least one run achieved a perfect score, we indicate this with an asterisk.

Only considering the evaluation on positive-only data, the results in Table 1 indicate transformers generalise well on the star-free languages (Ones, First, $\mathcal{D}_1$ and $\mathcal{D}_2$). On the non-star-free languages in AC^0 ($\mathcal{C}_2$ and $\mathcal{C}_3$), the transformers perfectly fit the training set but fail to generalise. On Parity, which lies outside AC^0, they do not perfectly learn the training set. However, the evaluation on random data in Table 2 reveals that the transformers do worse or barely outperform the baseline, although the sequence accuracy stays high.

How can we explain this? The gradient update of a transformer is calculated based on the predictions made at all positions of the input sequence. Therefore, it is effectively trained on *all prefixes* of the sequences in the training set. When training on positive examples only, if the target DFA contains a garbage state, the transformer is never trained on any sequences that lead to that garbage state. Conversely, when the target DFA does not contain a garbage state, the transformer is trained on *all sequences* up to a length slightly smaller than the training length. That is, also the sequences ending in a rejecting states, as they are a prefix of an accepting sequence. If we then validate the performance of a transformer only on positive data – in line with the original study – the behaviour of the transformer on sequences for which the target DFA ends in its garbage state is *not* taken into account.

Furthermore, a transformer performs perfectly on the sequence accuracy measure, even when evaluated on randomly sampled data, provided that it correctly identifies when the target DFA transitions to the garbage state, i.e. when it identifies a symbol to be invalid. The transformer's predictions after an invalid symbol do not matter. When the sequence accuracy is high, it is therefore entirely possible for a transformer to have learned a variation on the intended target language with arbitrary behaviour on sequences ending in the garbage state.

Table 1. Support weighted F1 score on the train and validation set and sequence accuracy on test sets ranging from 100 to 104 and 1000 to 1004. All datasets contain only positive examples. Three transformers with identical hyperparameters were trained each time and the average is reported. An asterisk is added in the table if at least one of the transformers scores perfectly.

	F1 Score		Sequence accuracy	
	Train	Validation	Test	
Language	l=32	l=100	l=100[†]	l=1000[†]
Ones	1.000*	1.000*	1.000*	1.000*
First	1.000*	1.000*	1.000*	1.000*
$\mathcal{D}_1$	1.000*	1.000*	1.000*	0.589
$\mathcal{D}_2$	1.000*	0.994*	0.856*	0.590
$\mathcal{C}_2$	1.000*	0.883	0.592	0.461
$\mathcal{C}_3$	1.000*	0.874	0.595	0.528
Parity	0.975	0.928	0.543	0.467

Extraction We now extract a Moore machine from each of the transformers using the method described in Section 4. In accordance with Weiss et al. (2018b), we make use of an initial split depth of 10 and supply a short positive and negative example whenever the algorithm terminates on the trivial one-state machine. Furthermore, the algorithm times out after 30 seconds. The results are shown in Table 3. Besides the number of states of the extracted machine and target machine, we report the *agreement* metric.

[†] This dataset includes examples of up to four characters more than specified to achieve more accurate results on languages where classification depends on the sequence length.

Table 2. Support weighted F1 score and sequence accuracy on randomly sampled test sets ranging from from 100 to 104 and 1000 to 1004. All datasets contain random data. Three transformers with identical hyperparameters were trained each time and the average is reported. An asterisk is added if at least one of the transformers scored perfectly. The baseline predicts all tokens as valid.

	l=100†			l=1000†		
Language	Baseline F1	Transformer F1	Sequence accuracy	Baseline F1	Transformer F1	Sequence accuracy
Ones	0.040	**0.042**	1.000*	**0.004**	**0.004**	1.000*
First	0.673	**0.676**	1.000*	**0.684**	**0.684**	1.000*
$\mathcal{D}_1$	0.023	**0.045**	1.000*	0.002	**0.005**	1.000*
$\mathcal{D}_2$	0.036	**0.060**	1.000*	0.004	**0.007**	1.000*
$\mathcal{C}_2$	**0.890**	0.882	0.592	**0.889**	0.785	0.461
$\mathcal{C}_3$	**0.875**	**0.875**	0.667	**0.875**	0.809	0.603
Parity	**0.933**	0.928	0.492	**0.933**	0.892	0.498

Table 3. Results of extracting Moore machines from the transformers trained under the positive only next character prediction task. The results are averaged over the three identically trained transformers unless this obscures important details. T/O indicates a timeout.

	Size		Agreement F1		
Language	Extracted	Target	l=10†	l=100†	l=1000†
Ones	1	2	0.983	0.988	0.999
First	2	3	1.000	1.000	1.000
$\mathcal{D}_1$	2	3	0.999	1.000	0.927
$\mathcal{D}_2$	3,4,3	4	0.823	0.712	0.762
$\mathcal{C}_2$	2	2	1.000	0.882	0.785
$\mathcal{C}_3$	3	3	1.000	0.875	0.809
Parity	2,T/O,2	2	0.999	0.928	0.891

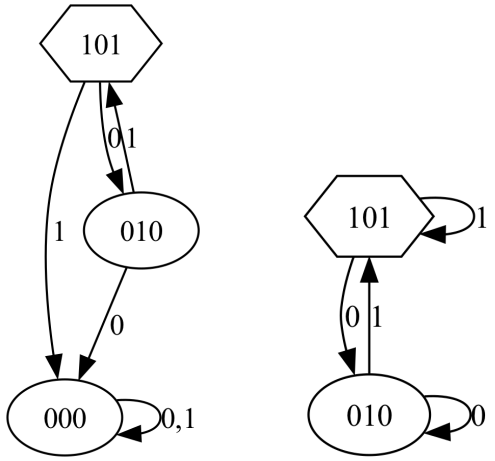


Figure 1. Target Moore Machine (left) and extracted Moore machine (right) from a transformer trained on $\mathcal{D}_1$. The labels show whether the 0, 1 and end-of-sequence symbols are considered valid in that state.

The agreement is the F1 score on random data using the extracted Moore machine as ground truth, indicating how well the extracted machine describes the transformer.

The extraction only timed out on one of the transformers trained on Parity. For the machines without a garbage state ($\mathcal{C}_2$,$\mathcal{C}_3$ and Parity), the extraction found the exact target Moore machine. In that case, the F1 score is equivalent to the agreement metric. For all languages with a garbage state (Ones, First, $\mathcal{D}_1$ and $\mathcal{D}_2$), the algorithm extracted a Moore machine with one state less than the target machine, except for two transformers trained on $\mathcal{D}_2$. For all these languages the agreement is much higher than the F1 score on random data, which indicates that the extracted machine describes the behaviour of the transformer much better than the target machine.

Figure 1 compares the extracted Moore machines for $\mathcal{D}_1$ with the target machines. The same machine was extracted from all three transformers. The labels of the states show the output function of the Moore machine. Under this task, the output function maps a state to three binary values indicating whether the 0, 1, and end-of-sequence symbols are considered valid in that state. The extraction results for the other languages are similarly in Appendix B.

As hypothesised, the transformer learns a variation of the target machine without the garbage state. Note that the transformer could have rerouted the transitions ending in the garbage state arbitrarily. Interestingly, the configuration we have extracted was studied by Liu et al. (2023). They call these languages gridworld$_i$, so we will refer to them as $\mathcal{G}_i$. They are equivalent to $\mathcal{D}_i$ with the garbage state removed and the transitions rerouted to be self-loops. Liu et al. (2023) showed that for these languages even shallower shortcuts exist than the general logarithmic depth and constant depth shortcuts they derived. The authors found that a two-layer transformer with large enough dimensions can learn this family of languages. Remarkably, our transformers can learn this language with only a single layer, empirically

setting an even tighter lower bound.

5.3. Positive-Negative State Prediction

From here on, we analyse transformers trained on randomly sampled data under the state prediction task to avoid learning unintended target languages. We additionally study the languages $\mathcal{G}_i$ as we have shown the former are the actual languages transformers manage to generalise well on under the positive-only character prediction setting.

We train three transformers on the studied languages, now including $\mathcal{G}_1$ and $\mathcal{G}_2$, and extract Moore machines from them. Both the training and extraction are performed as before. The full results of this training and extraction are in Appendix C.

The transformers learned perfectly on the training set for all languages, except Parity. On $\mathcal{C}_2$ and $\mathcal{C}_3$ they did not generalise well outside of the training set. At least one transformer had a perfect score even for sequences of length 1000 for Ones, $\mathcal{G}_1$ and $\mathcal{D}_1$. For First, $\mathcal{G}_2$ and $\mathcal{D}_2$, there was generalisation but not up to such a long length. The extraction algorithm terminated and extracted the exact target Moore machine for all transformers except for one trained on Parity and two trained on $\mathcal{D}_2$, where it timed-out, returning a large machine.

6. State Localisation

We next attempt to localise how the states of the extracted finite state machines are represented in the final layer of our transformers. For each language, we focus on the model that generalised best. We summarise the main results here. Appendix D contains the full analysis for each language.

As discussed before, the activations of the final layer all have norm $\sqrt{d}$ due to the layer normalisation, with d the dimension of the residual stream. Therefore, only the direction of the activation vector matters. Within the activation space, we can find inherently meaningful directions by looking at those directions along which the probability of one of the output labels is maximised. We will refer to them as *maximal probability directions*. We observe that these directions are arranged opposingly when there are two outputs and in a plane when there are three outputs. This makes sense as, due to the softmax activation, making one output large causes the others to be small. These arrangements maximise the angles between the directions of the different outputs.

The computation of the transformer can thus be decomposed in two steps. Firstly, the transformer computes, for each symbol, a fixed-norm activation vector in its final layer. Next, this activation vector is transformed into the output. The directions that maximise the output probabilities span a subspace of the activation space of the final layer. Therefore, the only part of the activation vector that influences the output is the component in that subspace. If an activation vector has a large perpendicular component, the component in the subspace must be small and therefore the confidence of the predictions drops.

When a transformer has learned to generalise well, we assume the transformer has similar activations on input sequences for which the extracted machine is in the same state. We therefore introduce the notion of returning suffixes.

Definition 6.1. A *returning suffix* associated with a state $q \in Q$ of a Moore machine $M = (Q, q_0, \delta, \gamma, \Sigma, \Gamma)$ is a sequence of symbols $s_q \in \Sigma^*$, such that $\hat{\delta}(q, s_q) = q$.

Notice that the empty sequence ϵ is a returning sequence for all states. All returning suffixes for a state q up to length L can be found by performing a breadth-first traversal of depth L of the extracted Moore machine and adding all paths returning to q to the set of returning suffixes. For some states, the number of returning suffixes is exponential in L. Therefore, we used a beam search to generate a dataset of returning suffixes. We then look at the transformers activation on $p_q.s_q$, with $p_q \in \Sigma^*$ being a prefix, such that $\delta(q_0, p_q) = q$. We always choose the shortest such p_q. If the transformer has generalised well, we expect to find the activations on the returning suffixes to lie in the same direction.

Of the languages we study, our transformers managed to generalise well on the languages Ones, First, $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{D}_1$ and $\mathcal{D}_2$. Indeed, we observe that they have, for each state, a designated direction in their final layer along which they orient the activation vector of a sequence ending in that state. Moreover, these designated directions are well aligned with the direction maximising the correct output probability. The languages $\mathcal{D}_1$ and $\mathcal{D}_2$ are an exception to the rule, as for these languages, transformers only seem to represent the garbage state well.

For the languages $\mathcal{C}_2$ and $\mathcal{C}_3$, we observe that transformers also can align the activation vectors along such a designated direction, but only for lengths seen in training. For the Parity language, our transformers never seem to align the activation vectors at all.

Further analysis is necessary to explain exactly how this alignment comes about. However, observing when a transformer is successful in representing a state and when it is not conveys a great deal of information and allows for a hypothesis on the trainability of one-layer transformers.

7. Characterising Trainability

Finally, we can propose a coherent characterisation of the trainability of one-layer transformers on regular languages

using our mechanistic insights. Concretely, we hypothesise that regular languages are trainable when the state of their minimal DFA is determined by a finite number of symbols. To better understand this, we consider the concept of reset sequences (Maler & Pnueli, 1994). A sequence of symbols is a reset sequence for a DFA if it maps every state to a single state under the generalised transition function.

Definition 7.1. We call $s \in \Sigma^*$ a *reset sequence* for state q_r of DFA $\mathcal{D} = (Q, F, \Sigma, q_0, \delta)$ if

$$\forall q \in Q : \hat{\delta}(q, s) = q_r \tag{1}$$

For $\mathcal{G}_1$, all sequences of length one are reset sequences. Therefore, the correct state can be determined by looking only at the final symbol. For $\mathcal{G}_2$, the sequences 00 and 11 are reset sequences, but 01 and 10 are not. For Ones, a single zero always brings the target DFA to the garbage state and is therefore a reset sequence. Moreover, when the sequence contains a zero anywhere the target DFA is in the garbage state. The occurrence of a zero thus determines the state. Finally, for First, the correct state depends solely on the first symbol.

For $\mathcal{D}_1$ and $\mathcal{D}_2$, only the garbage states have reset sequences. Therefore, the condition in our hypothesis is only met in these garbage states. This is in line with our observation in the previous section that our transformers only represent the garbage state well for these languages.

These are thus three slightly different cases where a finite number of symbols determines the automaton's state. Either there are *reset sequences*, the *occurence* of a particular symbol anywhere in the sequence determines the state, or the symbol at a *certain position* determines the state.

Our results in Appendix D.8 are in line with our hypothesis that one-layer transformers use this property to generalise well on these languages. Through inspection of the attention patterns, we indeed observe that the transformer trained on Ones pays strong attention to zero symbols. For the language First, we observe that the transformer pays strong attention to the symbol at the first position. The transformer trained on $\mathcal{G}_1$ pays stronger attention to the current symbol. The transformer trained on $\mathcal{G}_2$ pays stronger attention to the current symbol and the two previous symbols, having one head for each of these three positions.

7.1. Attention Saturation

Although we seem to have a clear understanding of how the transformers represent states, it is still unclear why they do not generalise perfectly. Hahn's lemma provides an explanation.

Theorem 7.2. *(Hahn, 2020) Given a soft attention transformer on an input of n symbols. If we exchange one single input symbol, then the change in the resulting activation at the decoder layer is bounded as $O(\frac{1}{n})$ with constants depending on the parameter matrices.*

As the one-layer transformers always use a finite number of symbols to determine the correct output, they eventually fail when these determining symbols are unable to sufficiently influence the output due to the effect of the unimportant symbols. In other words, the attention mechanism becomes saturated.

We verified that this indeed happens in practice. A transformer trained on the language Ones fails on a long sequence $011\ldots11$ when the effect of the zero which determines the state becomes too small. A transformer trained on the language First also fails on a sequence $011\ldots11$ as the influence of the first symbol becomes insufficient. A transformer trained on $\mathcal{G}_1$, fails on a sequence $00\ldots001$ because the effect of the one is not sufficient.

Somewhat differently, a transformer trained on $\mathcal{G}_2$ fails on a sequence $0101\ldots0101$ as there are no reset sequences for the transformer to use. Transformers trained on $\mathcal{D}_1$ and $\mathcal{D}_2$ also quickly fail on a sequence $0101\ldots0101$. This again shows that the transformer has difficulty representing the non-garbage states for these two languages.

This further validates our hypothesis that one-layer transformers depend on a finite number of symbols to recognise these languages, as they fail when this property no longer holds. Finally, we look at how the activation vector computed by the transformer varies in these failure cases. We find that the activation vectors start to deviate from the direction the correctly classified activation vectors were aligned with, confirming that the wrong output is indeed caused by the inability to detect which state the sequence should be classified as.

8. Conclusion

We identified a general task setting in which the seemingly contradictory results of previous studies regarding the trainability of transformers can be compared. As illustrated on the positive-only next character prediction task, having a deeper understanding of what is learned during training leads to more robust conclusions on trainability. We showed that the extraction technique proposed by Weiss et al. (2018b) can be used to extract Moore machines from transformers with minimal adaptations. We showed that mechanistic interpretability is a productive approach to empirically investigate and verify the trainability of transformers on formal languages. We successfully localised meaningful directions in the output layer of the transformers along which the transformers orient the activations on sequences that end in the same state. We empirically found that one-layer transformers can find generalisable solutions

for the languages Ones, First, $\mathcal{G}_1$ and $\mathcal{G}_2$ which implies that for languages where a finite number of symbols determine the automaton's state, a stronger lower bound exists than proposed by Liu et al. (2023). Finally, we showed failure cases can be found by reasoning over the structure of the target finite state machine.

9. Future Work

The languages studied in this paper are anecdotal evidence of tighter lower bounds on the trainability of transformers on some classes of regular languages. A more thorough theoretical characterisation of this lower bound is necessary, as well as further experimental validation of our trainability hypothesis, for example, through ablations and causal interventions.

Several other training tasks could be investigated. Most of these can be framed as simulating a Moore machine with a task-specific output function, such that our method works without adaptations. We are particularly interested in direct *membership prediction*. This might be more difficult as the output function does not explicitly distinguish between two accepting (or two non-accepting) states, as was the case under the state prediction task. Additionally, other transformer variants could be studied, including transformers with multiple layers.

In general, a transformer maps a sequence of input symbols to a probability distribution over sequences of output symbols. If the confidence of the predictions is high, this can be seen as a sequence-to-sequence transformation from the input to the output alphabet, as we have done. This does not hold, however, when the model is less confident. By instead looking at *weighted finite automata*, it might be possible to model transformers more accurately. This might allow extraction of small machines with high agreement even when the transformers do not perform the original task perfectly.

The extraction of state machines from transformers could also be applied as structure learning for neurosymbolic frameworks which rely on weighted grammars (Winters et al., 2022).

Our work is restricted to regular languages, although the trainability of transformers is not well aligned with the Chomsky hierarchy. Therefore, a mechanistic interpretability approach looking at non-regular formal languages would likely improve our current understanding.

Acknowledgements

Jaron Maene received funding from the Flemish Government (AI Research Program) and the Flanders Research Foundation (FWO) under project G097720N. We are grateful to Luc De Raedt for his feedback.

References

Ackerman, J. and Cybenko, G. A survey of neural networks and formal languages. *arXiv preprint arXiv:2006.01338*, 2020.

Angluin, D. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987. ISSN 0890-5401. doi: https://doi.org/10.1016/0890-5401(87)90052-6.

Anil, C., Wu, Y., Andreassen, A., Lewkowycz, A., Misra, V., Ramasesh, V., Slone, A., Gur-Ari, G., Dyer, E., and Neyshabur, B. Exploring length generalization in large language models. *Advances in Neural Information Processing Systems*, 35:38546–38556, 2022.

Arora, S. and Barak, B. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2006. ISBN 978-0-521-42426-4.

Barrington, D. A. M., Compton, K., Straubing, H., and Thérien, D. Regular languages in nc1. *Journal of Computer and System Sciences*, 44(3):478–499, 1992. ISSN 0022-0000. doi: https://doi.org/10.1016/0022-0000(92)90014-A.

Bhattamishra, S., Ahuja, K., and Goyal, N. On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7096–7116. Association for Computational Linguistics, 2020.

Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N., Anil, C., Denison, C., Askell, A., Lasenby, R., Wu, Y., Kravec, S., Schiefer, N., Maxwell, T., Joseph, N., Hatfield-Dodds, Z., Tamkin, A., Nguyen, K., McLean, B., Burke, J. E., Hume, T., Carter, S., Henighan, T., and Olah, C. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023.

Brody, S., Alon, U., and Yahav, E. On the expressivity role of LayerNorm in transformers' attention. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 14211–14221, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.895.

Chiang, D. and Cholak, P. Overcoming a theoretical limitation of self-attention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, 2022.

Delétang, G., Ruoss, A., Grau-Moya, J., Genewein, T., Wenliang, L. K., Catt, E., Cundy, C., Hutter, M., Legg, S.,

Veness, J., and Ortega, P. A. Neural networks and the chomsky hierarchy. *The Eleventh International Conference on Learning Representations*, 2022.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021.

Giles, C. L., Miller, C. B., Chen, D., Sun, G. Z., Chen, H. H., and Lee, Y. C. Extracting and learning an unknown grammar with recurrent neural networks. In Moody, J., Hanson, S., and Lippmann, R. (eds.), *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1991.

Hahn, M. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020. doi: 10.1162/tacl_a_00306.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2006. ISBN 0321455363.

Kingma, D. P. and Ba, J. L. Adam: A method for stochastic gradient descent. In *International Conference on Learning Representations*, pp. 1–15, 2015.

Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., and Zhang, C. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*, 2023.

Maler, O. and Pnueli, A. On the cascaded decomposition of automata, its complexity and its application to logic. *ACTS Mobile Communication*, 48, 1994.

Merrill, W. Formal language theory meets modern nlp. *arXiv preprint arXiv:2102.10094*, 2021.

Merrill, W. and Sabharwal, A. 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.

Moerman, J. Learning product automata. In Unold, O., Dyrka, W., and Wieczorek, W. (eds.), *Proceedings of Machine Learning Research*, volume 93, pp. 54–66. PMLR, 2019.

Nguyen, T. Q. and Salazar, J. Transformers without tears: Improving the normalization of self-attention. In Niehues, J., Cattoni, R., Stüker, S., Negri, M., Turchi, M., Ha, T.-L., Salesky, E., Sanabria, R., Barrault, L., Specia, L., and Federico, M. (eds.), *Proceedings of the 16th International Conference on Spoken Language Translation*, Hong Kong, November 2-3 2019. Association for Computational Linguistics.

Strobl, L., Merrill, W., Weiss, G., Chiang, D., and Angluin, D. Transformers as recognizers of formal languages: A survey on expressivity, 2023.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2023.127063.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Vollmer, H. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag, Berlin, Heidelberg, 1999. ISBN 3540643109.

Weiss, G., Goldberg, Y., and Yahav, E. On the practical computational power of finite precision RNNs for language recognition. In Gurevych, I. and Miyao, Y. (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 740–745, Melbourne, Australia, July 2018a. Association for Computational Linguistics. doi: 10.18653/v1/P18-2117.

Weiss, G., Goldberg, Y., and Yahav, E. Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples. *Proceedings of the 35th International Conference on Machine Learning*, 2018b. doi: doi:10.48550/arXiv.1711.09576.

Winters, T., Marra, G., Manhaeve, R., and De Raedt, L. Deepstochlog: Neural stochastic logic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 10090–10100, 2022.

A. Angluin's L^* algorithm

The L* algorithm (Angluin, 1987) is an exact learning algorithm to learn a minimal DFA accepting the language L from a minimally adequate teacher.

Definition A.1. An oracle is a *minimally adequate* teacher for language L if it answers two types of queries correctly:

- A membership query, consisting of a sequence s. The reply is *yes* if the sequence belongs to L and *no* otherwise.
- An equivalence query, consisting of a proposed regular language L'. The answer is *yes* if the proposal equals L and a counterexample from the symmetric difference of L and L' otherwise.

Internally, the learner keeps track of an observation table $\mathcal{O}$, where answers to membership queries are stored and from which proposals can be constructed.

Definition A.2. (Observation Table) An *observation table* over alphabet Σ is a triple $\mathcal{O} = (S, E, T)$, with:

- S a non-empty finite set of sequences, where $\forall w \in S, \text{prefix}(w) \in S$. S is said to be *prefix-closed*.
- E a nonempty finite set of sequences, where $\forall w \in E, \text{suffix}(w) \in E$. E is said to be *suffix-closed*.
- T a function mapping $((S \cup S.\Sigma).E)$ to $\{0, 1\}$.

It can be thought of as a table with the rows labelled with elements from $(S \cup S.\Sigma)$ and the columns with elements from E. The entry for row s and column e is then $T(s.e)$.

An observation table is called *closed* if for each $t \in S.\Sigma$, there is an $s \in S$ such that $row(t) = row(s)$. An observation table is called *consistent* if for all $s_1, s_2 \in S$ and $\sigma \in \Sigma$, $row(s_1) = row(s_2) \Rightarrow row(s_1.\sigma) = row(s_2.\sigma)$.

A DFA $M(S, E, T)$ can be defined from an observation table as follows.

- $Q = \{row(s) \mid s \in S\}$
- $q_0 = row(\epsilon)$
- $F = \{row(s) \mid s \in S \wedge T(s) = 1\}$
- $\delta(row(s), \sigma) = row(s.\sigma)$

In the original paper, the following property is proven.

Theorem A.3. *(Angluin, 1987) If (S, E, T) is a closed, consistent observation table, the DFA $M(S, E, T)$ is consistent with the finite function T.*

It is then possible to describe Angluin's L* algorithm, as shown in algorithm 1.

Algorithm 1 Angluin's L* algorithm

$S \leftarrow \{\epsilon\}$
$E \leftarrow \{\epsilon\}$
repeat
  while $\neg isClosed((S, E, T)) \vee \neg isConsistent((S, E, T))$ **do**
    if $\neg isClosed(O)$ **then**
      1. $\exists s' \in S, \sigma \in \Sigma : \forall s \in S : row(s'.\sigma) \neq row(s)$
      2. Add $s'.\sigma$ to S
      3. Extend T to cover $((S \cup S.\Sigma).E)$ using membership queries
    if $\neg isConsistent(O)$ **then**
      1. $\exists s_1, s_2 \in S, \sigma \in \Sigma, e \in E : row(s_1) = row(s_2) \wedge T(s_1.\sigma.e) \neq T(s_2.\sigma.e)$
      2. Add $\sigma.e$ to E
      3. Extend T to cover $((S \cup S.\Sigma).E)$ using membership queries
  if $EquivalenceQuery(M(S, E, T))$ *gives a counterexample c* **then**
    1. Add c and all its prefixes to S
    2. Extend T to cover $((S \cup S.\Sigma).E)$ using membership queries
  else
    return $M(S, E, T)$

The L^* algorithm can learn Moore machines with three simple changes. 1) The observation table $\mathcal{O} = (S, E, T)$ remains the same, but T now maps $((S \cup S.\Sigma).E)$ to an output alphabet Γ instead of $\{0, 1\}$. 2) On a membership query, the teacher returns a symbol from Γ instead of 0 or 1. 3) The proposal in an equivalence query takes the form of a Moore machine $M(S, E, T)$.

A Moore machine $M(S, E, T)$ is constructed from the observation table as follows.

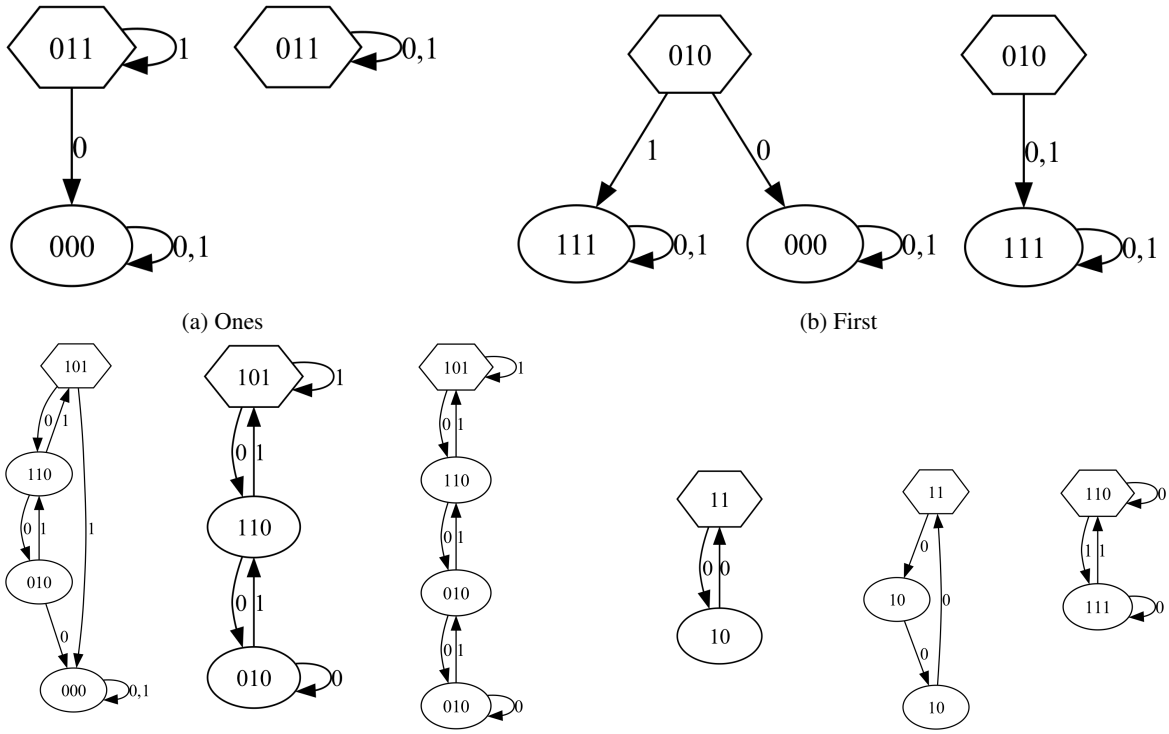
- $Q = \{row(s) \mid s \in S\}$,
- $q_0 = row(\epsilon)$,
- $\delta(row(s), \sigma) = row(s.\sigma)$,
- $\gamma(s) = T(s)$

The correctness of this extension follows directly from the variant of L^* that learns product Moore machines (Moerman, 2019).

B. Extraction Results for Ones, First and $\mathcal{D}_2$ on Character Prediction

The extracted machines from the transformers on positive-only data under the next character prediction task trained for languages Ones, First and $\mathcal{D}_2$ are included below. We also include the target machine for comparison. From the transformers trained on Ones and First, the same machine was extracted every time. From the transformers trained on $\mathcal{D}_2$, the same machine with three states was extracted from two of the transformers and the other machine from the third. For $\mathcal{C}_2$,$\mathcal{C}_3$ and Parity the extracted machines were equal to the targets.

As mentioned in the main text, the transformers have learned a variant of the intended target language for all languages with a garbage state. The behaviour of the transformer in the garbage state can be arbitrary but we often see that the garbage state is not modelled and the transitions leading to the garbage state are rerouted to be self-loops. The only exception is $\mathcal{D}_2$, where the transformer duplicates the last state.



(a) Ones

(b) First

(c) $\mathcal{D}_2$

(d) $\mathcal{C}_2$,$\mathcal{C}_3$ and Parity

Figure 2. The extracted and target Moore machines for languages Ones, First and $\mathcal{D}_2$, $\mathcal{C}_2$,$\mathcal{C}_3$ and Parity. The target machine is always shown on the left. For $\mathcal{C}_2$,$\mathcal{C}_3$ and Parity the target and extracted machines are the same. The labels show whether the zero, one and end-of-sequence symbols are considered valid in that state.

C. Training and Extraction Results on State Prediction

Table 4. Support weighted F1 score of transformers trained on all studied languages under the state prediction task. Three transformers with identical hyperparameters were trained each time and the average is reported. An asterisk is added in the table if at least one of the transformers scored perfectly. The baseline predicts the majority class.

	F1 Score						
	Train	l=10		l=100		l=1000	
Language	l=32	baseline	model	baseline	model	baseline	model
Ones	1.000	0.744	**1.000***	0.970	**1.000***	0.997	**1.000***
First	1.000	0.287	**1.000***	0.330	**1.000***	0.338	**0.525**
$\mathcal{G}_1$	1.000	0.390	**1.000***	0.341	**1.000***	0.334	**0.956***
$\mathcal{G}_2$	1.000	0.254	**1.000***	0.174	**1.000***	0.169	**0.503**
$\mathcal{D}_1$	1.000	0.721	**1.000***	0.970	**1.000***	0.997	**1.000***
$\mathcal{D}_2$	0.999	0.625	**1.000***	0.956	**1.000**	0.996	**1.000**
$\mathcal{C}_2$	1.000	0.385	**1.000***	0.339	**0.638**	0.334	**0.436**
$\mathcal{C}_3$	1.000	0.194	**1.000***	0.170	**0.537**	0.167	**0.264**
Parity	0.802	0.390	**0.982**	0.340	**0.591**	0.334	**0.455**

Table 5. Results of extracting Moore machines from the transformers trained under the state prediction task. The results are averaged over the three identically trained transformers. T/O indicates a timeout.

	Size		Agreement		
Language	Extracted	Target	l=10	l=100	l=1000
Ones	2	2	1.000	1.000	1.000
First	3	3	1.000	1.000	0.525
$\mathcal{G}_1$	2	2	1.000	1.000	0.956
$\mathcal{G}_2$	3	3	1.000	1.000	0.503
$\mathcal{D}_1$	3	3	1.000	1.000	1.000
$\mathcal{D}_2$	T/O,T/O,4	4	1.000	1.000	1.000
$\mathcal{C}_2$	2	2	1.000	0.638	0.436
$\mathcal{C}_3$	3	3	1.000	0.537	0.264
Parity	2,T/O,2	2	0.982	0.610	0.551

D. Mechanistic Interpretability Analysis

In this appendix, we provide a detailed overview of the mechanistic analysis we have conducted on our transformers and how we interpret the results.

For the extraction method to work, we assumed the transformer managed to learn some representation of the finite state machine in its final layer. As the extractions were successful, this must hold true to a certain extent. We will now attempt to interpret the final layer of the transformers trained on the state prediction task. For each language, we focus on the transformer that generalises the best.

As discussed in Section 4.1, due to layer normalisation, the activation of the final layer all have a Euclidian norm of $\sqrt{d}$ with d the dimension of the residual stream. Therefore, only the direction of the activation vector matters. We first introduce the concepts of maximal probability directions and returning suffixes.

D.1. Maximal Probability Direction

Within the activation space, we can find inherently meaningful directions by looking at those directions along which the probability of one of the output labels is maximised. We call these directions the *maximal probability directions* or

m-directions and denote them as $\hat{\mathbf{m}}_i$, where $0 \leq i \leq |\Gamma| - 1$. For the state prediction task, we have $Q = \Gamma$ and we therefore say $\hat{\mathbf{m}}_i$ is the direction along which the label for state q_i is maximised.

The function mapping the activation vector, $\mathbf{t}$, to the output probability for label i, $p(\mathbf{t})_i$, is an affine transformation $f_A : \mathbb{R}^d \mapsto \mathbb{R}^{|\Gamma|}$ followed by the softmax function.

$$p(\mathbf{t})_i = \text{softmax}(f_A(\mathbf{t}))_i \tag{2}$$

We thus have a multivariate optimisation problem towards $p(\mathbf{t})_i$ under the restriction $\|\mathbf{t}\|_2^2 - 1 = 0$. By introducing a Lagrange multiplier we get the unconstraint objective function $W(\mathbf{t}, \lambda) = p(\mathbf{t})_i + \lambda(\|\mathbf{t}\|_2^2 - 1)$. Solving the resulting set of differential equations yields a solution for $\hat{\mathbf{m}}_i = max_{\mathbf{t}}(W(\mathbf{t}, \lambda))$.

$$\hat{\mathbf{m}}_i = \frac{sgn \nabla_t p(\mathbf{t})_i}{\|\nabla_t p(\mathbf{t})_i\|_2} \tag{3}$$

The sign sgn must be chosen such that the probability is maximal and not minimal, or $p(\hat{\mathbf{m}}_i)_i \geq p(-\hat{\mathbf{m}}_i)_i$.

As both the affine transformation and the softmax function are monotone, for any activation vector with a fixed norm, the probability of output i is maximised if it is aligned with direction $\hat{\mathbf{m}}_i$.

For any activation vector with a fixed norm, the probability of output i is maximised if it is aligned with direction $\hat{\mathbf{m}}_i$.

D.2. Returning Suffixes

When a transformer has learned to generalise well, we assume the transformer will have similar activations on input sequences for which the extracted machine is in the same state. We therefore introduce the notion of returning suffixes:

Definition D.1. A *returning suffix* associated with a state $q \in Q$ of a Moore machine $M = (Q, q_0, \delta_1, \delta_2, \Sigma, \Gamma)$ is a sequence of symbols $s_q \in \Sigma^*$, such that $\hat{\delta}(q, s_q) = q$.

Notice that the empty sequence ϵ is a returning sequence for all states. All returning suffixes for a state q up to length L can be found by performing a breadth-first traversal of depth L of the extracted Moore machine and adding all paths returning to q to the set of returning suffixes. For some states, the number of returning suffixes is exponential in L. Therefore, we used a beam search to generate a dataset of returning suffixes. We then look at the transformer's activation on $p_q.s_q$, with $p_q \in \Sigma^*$ being a prefix, such that $\delta(q_0, p_q) = q$. We always choose the shortest such p_q. If the transformer has generalised well, we expect to find the activation vectors on the returning suffixes of a state to lie in the same direction. It can therefore be useful to look at the average activation direction or a-direction on the returning suffixes of a state. We denote the average direction for state q_i as $\hat{\mathbf{a}}_i$.

D.3. Ones

We start by investigating the language Ones. Initially, we investigate whether the activations on all returning suffixes for a given state lie in the same direction. To accomplish this, we generated a dataset of returning suffixes with a beam traversal of the machine extracted from one of the transformers trained on Ones with a width of 10, and a maximum suffix length of 100. The average cosine similarity between the activations on these suffixes and their mean is 0.960 for q_0 and 0.990 for q_1. The variation of this similarity with increasing length is depicted by the solid lines in Figure 3. We observe that the cosine similarity stays constant as the length increases.

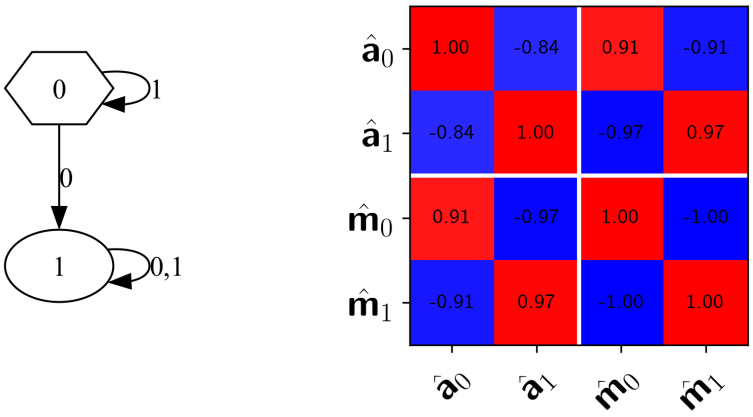


Figure 4. The extracted Moore machine and the similarity matrix for a transformer trained on Ones.

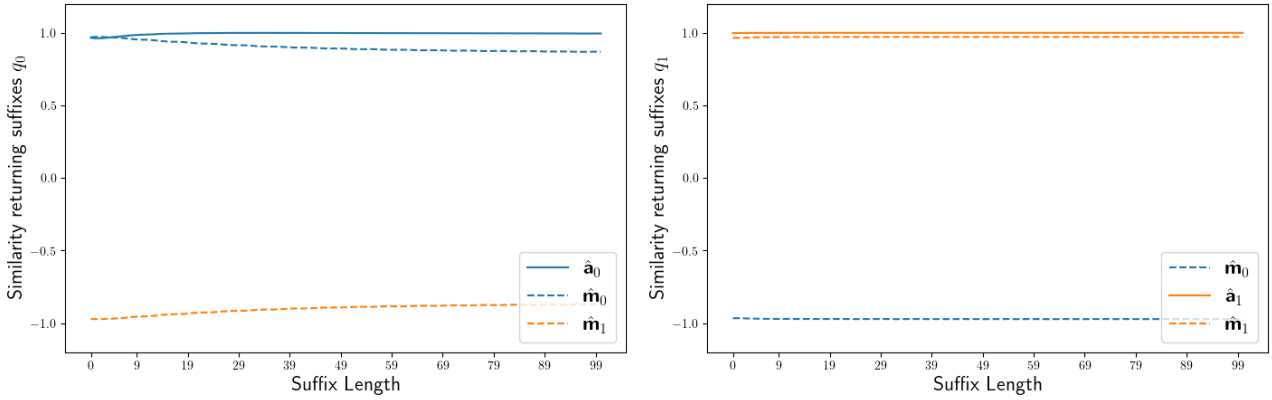


Figure 3. The cosine similarities between the activations on the returning suffixes and their average as well as the m-directions for the language Ones.

Next, we examine the similarity matrix between the a-direction and the m-direction, shown in Figure 4. The m-directions are exactly opposite, which is expected since maximising one output probability minimises the other. In this light, we consider the dashed lines in 3, which are the similarities between the activation vectors on the returning suffixes and the m-directions. We observe that for q_1, the activations are consistently closely aligned with $\hat{\mathbf{m}}_1$, whereas for q_0 there is a slight decrease in alignment as the sequence length increases.

To understand what is happening, we introduce the concept of reset sequences, as described in Maler & Pnueli (1994).

Definition D.2. We call $w \in \Sigma^*$ a *reset sequence* for state q_r of DFA $S = (Q, F, \Sigma, q_0, \delta)$ if

$$\forall q \in Q : \hat{\delta}(q, w) = q_r \tag{4}$$

For Ones, the occurrence of a zero symbol acts as a reset sequence. Whenever it is seen, the machine always transitions to state q_1. Moreover, in this language, if a zero occurs *anywhere* in the sequence up to the current position, the machine must be in state q_1; otherwise it is in state q_0. We indeed observed that the transformer pays strong attention to zero symbols.

We interpret the slight decrease in similarity for state q_0 as follows: for long sequences of consecutive ones, the component of the activation vector *perpendicular* to the line on which both m-directions lie increases. Intuitively, we can understand why for a one-layer transformer this perpendicular component must be there. A sequence of consecutive ones ends in q_1, while the same sequence prepended by a zero ends in q_0. But, due to the same phenomenon underlying Hahn's lemma, see 7.2, the transformer must represent them similarly. When using soft-attention, a single symbol can only influence the output with a factor bounded by $O(\frac{1}{T})$, with T the sequence length.

Based on this reasoning, we can find a case in which this transformer will fail. Surprisingly, it will fail to represent q_1 before q_0. The returning suffixes for q_1 are randomly sampled and therefore often contain additional zeros, which the transformer can use to align the activation vector with $\hat{\mathbf{m}}_1$. When presented with a long sequence of ones prepended by a zero, the transformer should find it difficult to differentiate between the two states. This, however, rarely occurs in a randomly sampled dataset. We show in Figure 5 the probabilities the transformer assigns to each position of the sequence 011..11 of length 100.

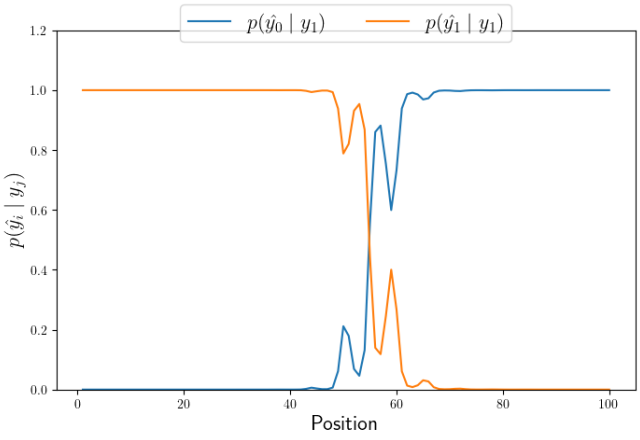


Figure 5. Probabilities assigned to each position of the sequence 011..11 of length 100 by a transformer trained on Ones.

Indeed, the transformer confuses state q_1 with state q_0 starting at position 54. We observe that the transformer is unable to recognise the symbol that is of importance due to the compounded effect of the other symbols. We can say the attention mechanism has been saturated.

Random sampling also explains the apparent perfect F1 score of the transformers trained on Ones for sequences with lengths up to 1000. The test set rarely contains an example with more than 53 consecutive ones and therefore the failure case is seldom encountered.

D.4. First

The language First is interesting as it is another simple language that suffers from the implications of Hahn's Lemma, which is the reason why Chiang & Cholak (2022) has studied it before. For this language, however, no reset sequences exist. The only symbol determining the correct label is the *first symbol*. We show the extracted machine and the similarity matrix in Figure 6 and the similarities between the activations on the returning suffixes and their mean in Figure 7.

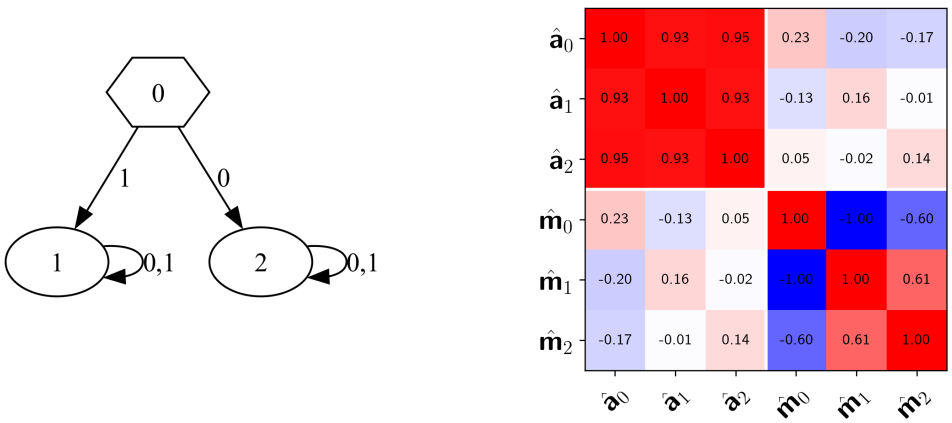


Figure 6. The extracted Moore machine and the similarity matrix for a transformer trained on First.

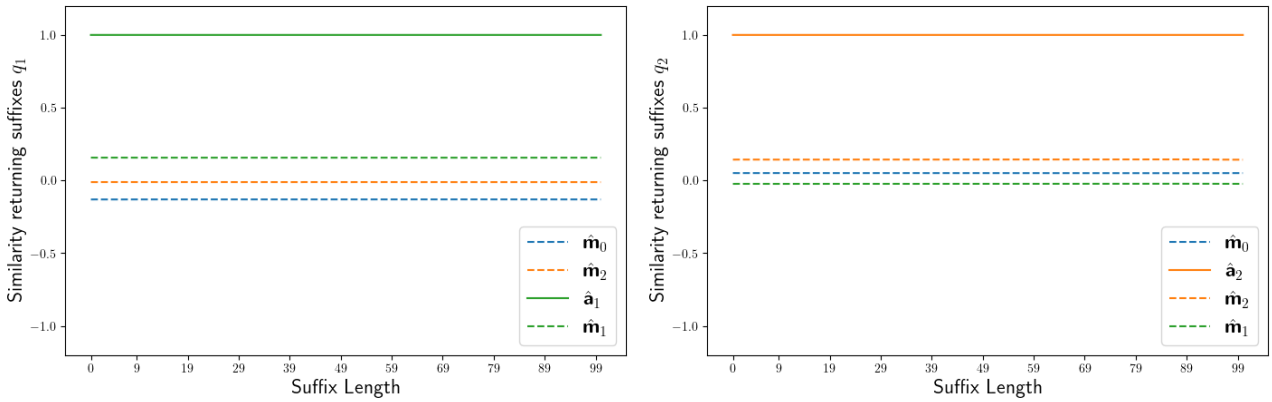
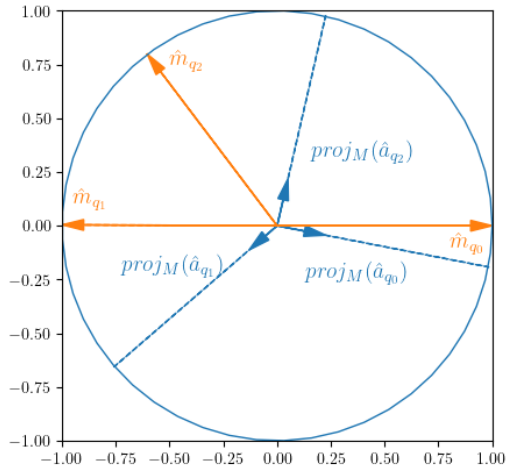
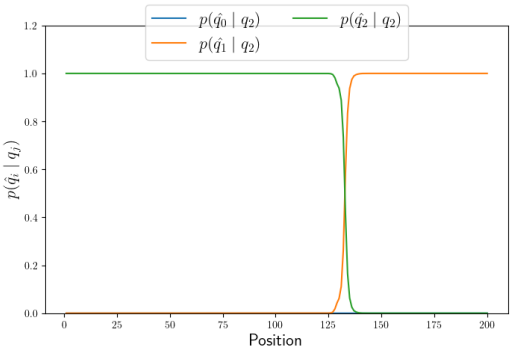


Figure 7. The cosine similarities between the activations on the returning suffixes and their average as well as the m-directions for the language First.

From the similarity matrix, we see the m-directions lie almost exactly in the same plane, which we call M. Indeed, the angles between the three vectors sum to 360 degrees, with a relative error of only 0.005. Between the activations on the returning suffixes and their means, we find a cosine similarity of $0.999+$ for both states, which justifies calling these common directions $\hat{\mathbf{a}}_1$ and $\hat{\mathbf{a}}_2$. As the target machine never returns to q_0, we take the activation of the transformer on the beginning-of-sequence symbol as $\hat{\mathbf{a}}_0$. However, as seen in the similarity matrix, all three a-directions are very similar to each other. To understand this better, we show the a-directions projected on the plane M in Figure 8a.



(a) The projection of the a-directions on the plane M spanned by the m-directions for a transformer trained on First.



(b) Probabilities assigned to each position of the sequence 011..11 of length 200 by a transformer trained on First.

The components of the a-directions in M are roughly evenly spaced. While they do not align well with the m-directions, each a-direction has a large component along the corresponding m-direction and a smaller or negative component along the other two m-directions. The correct label will thus be assigned a high probability. As the projected components are small, there must be a large component perpendicular to the plane. Although this perpendicular component does not alter the most probable symbol, it indirectly reduces the transformer's prediction confidence as the component in the plane is smaller. Similar to Ones, this component must exist. Input sequences that only differ in the first symbol and therefore end in different states must be represented similarly by a one-layer soft-attention transformer.

Again, we identify a sequence of consecutive ones prepended by a zero must be a failure case for this transformer. We show the probabilities assigned by the transformer at each position of the sequence $011\ldots11$ of length 200 in Figure 8b. The transformer confuses q_2 for q_1 starting from position 132, as the attention mechanism gets saturated.

D.5. $\mathcal{G}_1$ and $\mathcal{G}_2$

The gridworld languages are important as they have been studied before by Liu et al. (2023). They found that transformers can learn particularly shallow solutions for them. Furthermore, we have shown that it are these languages that transformers learn under the task setting of Bhattamishra et al. (2020) instead of the intended depth-bounded Dyck languages.

We first look at $\mathcal{G}_1$. Figure 9 shows the extracted Moore machine and the similarity matrix and Figure 10 shows the similarities of the activations on the returning suffixes compared to their mean and the maximal probability directions.

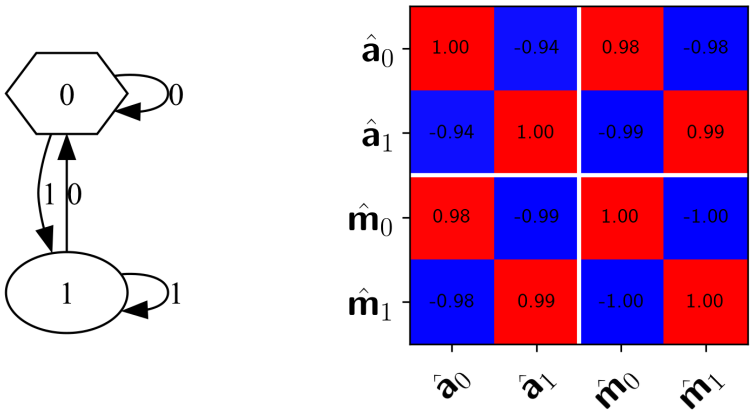


Figure 9. The extracted Moore machine and the similarity matrix for a transformer trained on $\mathcal{G}_1$.

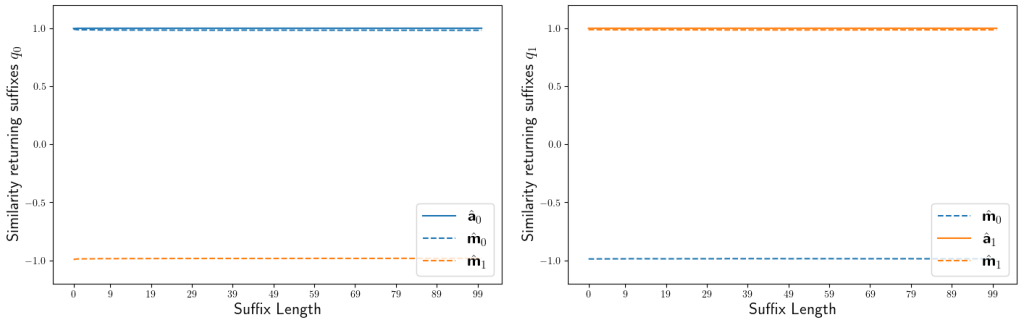


Figure 10. The cosine similarities between the activations on the returning suffixes and their average as well as the m-directions for the language $\mathcal{G}_1$.

Again, we see that the m-directions are exact opposites. We also observe that the activations on the returning suffixes for both states always lie in the same direction. The average cosine similarity is $+0.999$ for both states and does not diminish with longer sequence lengths.

The machine for $\mathcal{G}_1$ has the property that there is a finite $n \in \mathbb{N}$, where in this case $n = 1$, such that all sequences $s \in \Sigma^n$ are reset sequences. This abundance of reset sequences allows for the great alignment we observe. However, a failure case can be found again. We look at sequences containing only zeros but ending in one. We show in Figure 11 the probabilities the transformer assigns to q_0 and q_1 on the final position of such sequences of increasing length. We observe that when the sequence length increases, the transformer confuses q_1 with q_0. Again, this is because the symbol that matters is unable to change the output of the transformer sufficiently.

Next, we look at $\mathcal{G}_2$. The similarities between the activations on the returning suffixes, their average and the m-directions are shown in Figure 12. The target machine and similarity matrix are shown in Figure 13.

The average cosine similarity for q_0 is 0.98, for q_1 it is 0.977 and for q_2 it is 0.960. We can see some fluctuations in the orientation of the activations on returning suffixes of the same state. However, as there is no apparent decrease in similarity when the sequence length increases, speaking of common a-directions for each state is justified. The similarity matrix shows

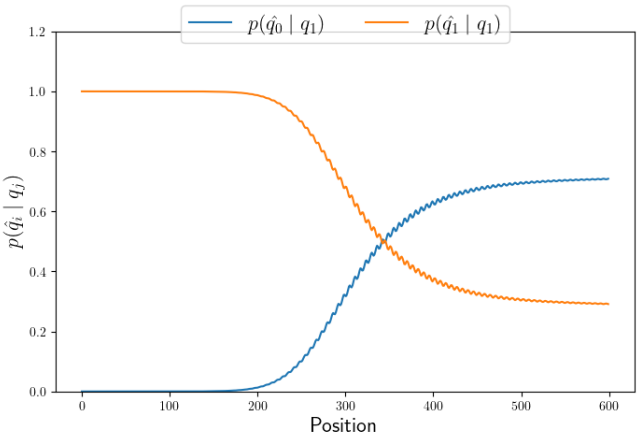


Figure 11. The probabilities assigned by a transformer trained on $\mathcal{G}_1$ at the final position of increasingly long sequences of zeros with a single one at the end.

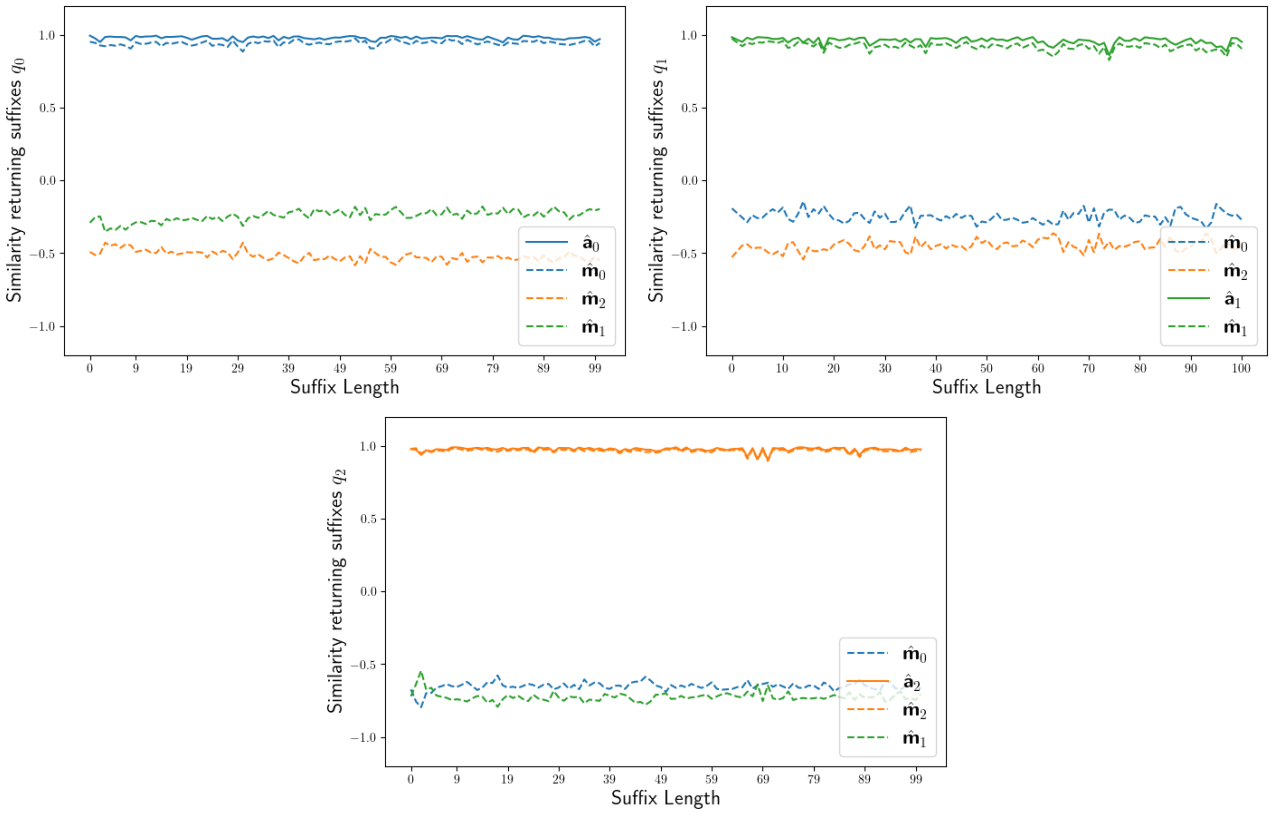


Figure 12. The cosine similarities between the activations on the returning suffixes and their average as well as the m-directions for the language $\mathcal{G}_2$.

both the m-directions and a-directions lie in a plane as the angles between them sum to 360 degrees with a relative error of less than 0.003. This is also the case for any triplet of m- and a-directions; therefore, they lie in *the same* plane. We show the projection of the a-directions on the plane spanned by the m-directions in Figure 14a.

The alignment between the a-directions and m-directions is not perfect, but the component of every a-direction is the largest along its corresponding m-direction and will therefore cause the probability of the correct output to be large.

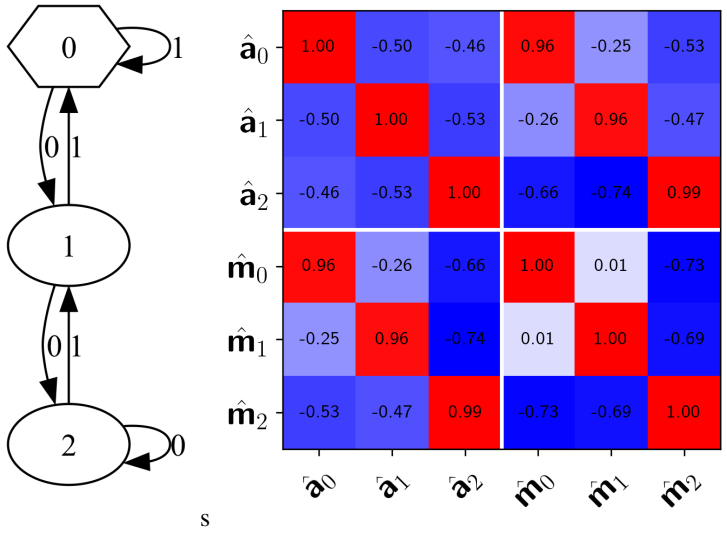
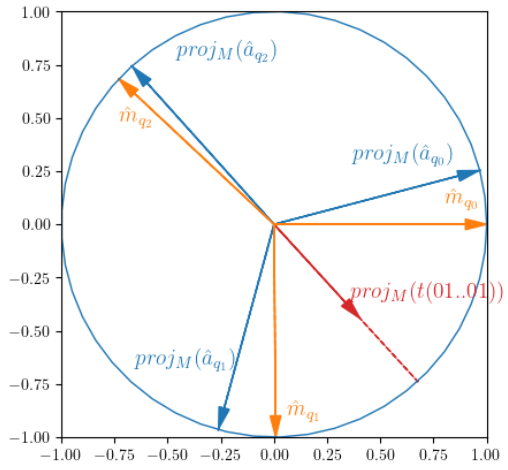
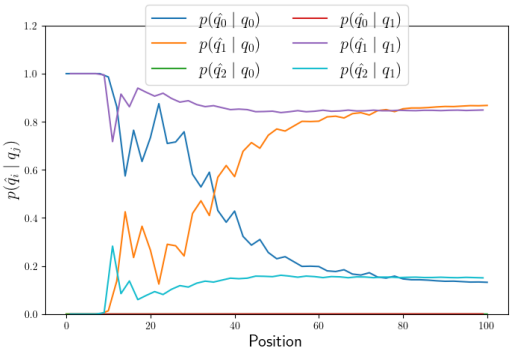


Figure 13. The extracted Moore machine and the similarity matrix for a transformer trained on $\mathcal{G}_2$.



(a) The projection of the a-directions on the plane spanned by the m-directions for $\mathcal{G}_2$.



(b) Probabilities assigned to each position of the sequence 0101…0101 of length 100 by a transformer trained on $\mathcal{G}_2$.

We try to explain the observed fluctuations in the similarity. The machine for $\mathcal{G}_2$ no longer has a finite $n \in \mathbb{N}$, such that all sequences of that length are reset sequences. For $n = 2$ the sequences 00 and 11 are reset sequences, but the sequences 01 and 10 are not. We again reason that a returning suffix containing no consecutive zeros or ones could end in any of the three states, depending on which symbols are prepended to it. For example, 010101 ends in q_0, 0010101 ends in q_1 and 00010101 ends in q_2, yet the one-layer soft-attention transformer must represent these sequences similarly, due to the implications of Hahn's lemma. These are the fluctuations we observe: some of the randomly sampled returning suffixes contain no reset sequences, which necessarily results in an activation vector that is less well aligned with the correct m-direction. Whenever a reset sequence occurs, the transformer can again align the activation vector with the correct m-direction.

To illustrate this, we show that our transformer eventually confuses the states if it does not encounter a reset sequence. Figure 14a shows the activation of the transformer on the sequence 0101..0101 of length 40, projected in the plane M spanned by the m-directions. The transformer orients the activation with a significant component perpendicular to the plane M. Again, this component does not contribute to the output probabilities but makes the component in the plane smaller, which results in less confident predictions. The component of the activation in the plane lies between $\hat{\mathbf{m}}_0$ and $\hat{\mathbf{m}}_1$ and thus the transformer will give both a high probability. We also show the predictions that the transformer assigns to each position on input sequence 0101..0101 in Figure 14b. Again, we observe the transformer confuses q_0 for q_1 before position 40.

D.6. $\mathcal{D}_1$ and $\mathcal{D}_2$

We now look at the depth-bounded Dyck languages. These languages are very similar to the gridworld languages studied before, but they have a garbage state.

First, we look at $\mathcal{D}_1$. Figure 16 shows the extracted Moore machine and the similarity matrix and Figure 15 shows the cosine similarities of the activations on the returning suffixes compared to their mean and the maximal probability directions.

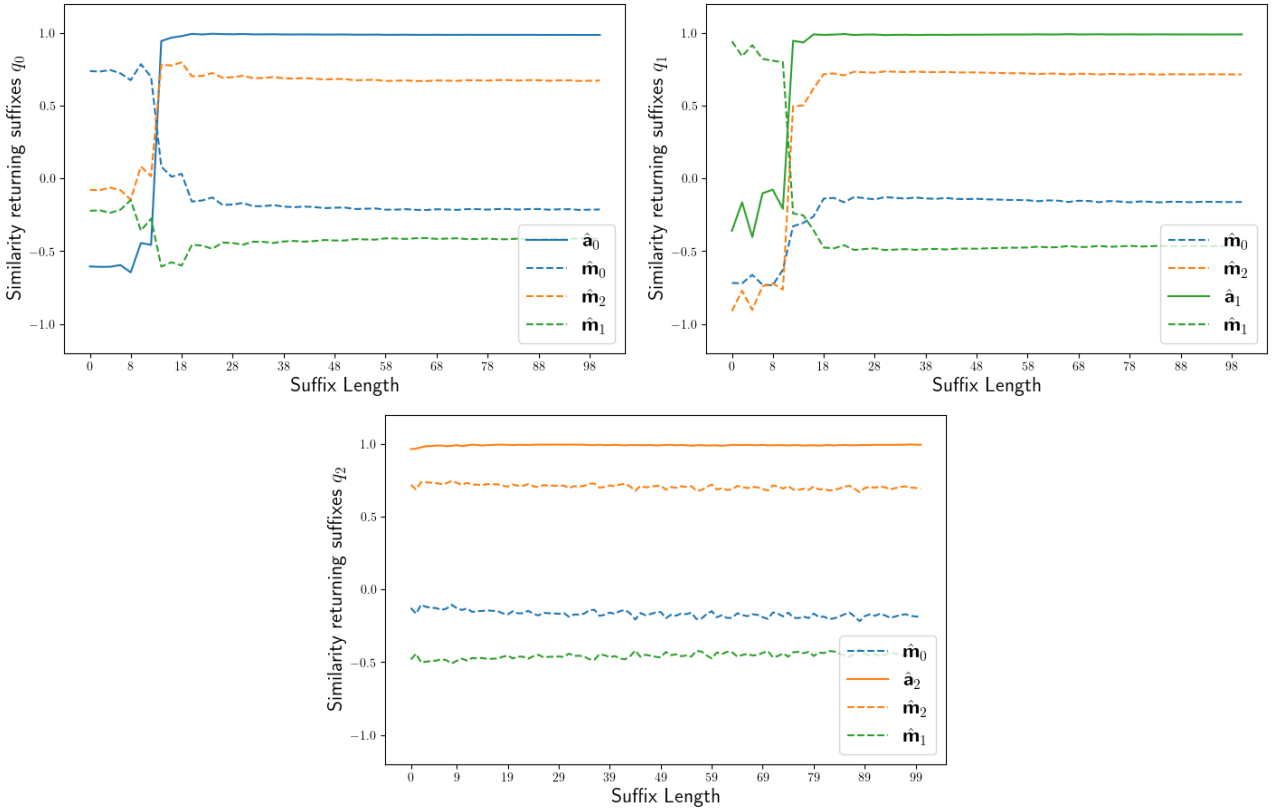


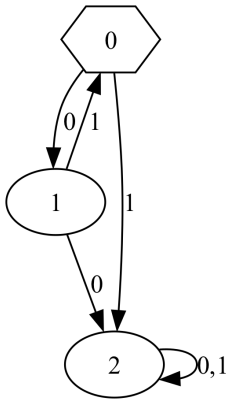
Figure 15. The cosine similarities between the activations on the returning suffixes and their average as well as the m-directions for the language $\mathcal{D}_1$.

As before, we see that the m-directions lie in a plane. The angles sum to 360 with a relative error of less than 0.001. The average cosine similarity between the activations on the returning suffixes and their mean is 0.774 for q_0, 0.845 for q_1 and 0.992 for q_2. For q_0 and q_1, this is significantly lower than what we have seen on the previous languages.

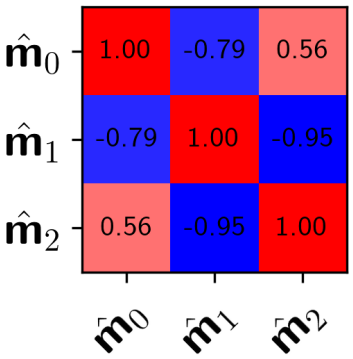
Using the plots, we can interpret what is going on. The transformers orient the activations on returning suffixes for q_0 and q_1 correctly, roughly until length 10. Indeed, until then, the similarity with $\hat{\mathbf{m}}_0$ and $\hat{\mathbf{m}}_1$ is high. Beyond this point, the transformer aligns the activations on the returning suffixes of both of these states along $\hat{\mathbf{m}}_2$. We can again explain this by reasoning over reset sequences. The transformer easily models q_2 as it has both 00 and 11 as reset sequences. It has difficulty with the other two states, which have no reset sequences. However, differently than before, there are no transitions from the state with reset sequences to the other states. Therefore, a sequence that ends in q_0 or q_1, never contains any reset sequences, resulting in the misalignment we observe.

Next, we look at $\mathcal{D}_2$. The similarities between the activations on the returning suffixes, their mean and the m-directions are shown in Figure 17. The target machine and similarity matrix are shown in Figure 18.

The four m-directions again seem to be geometrically arranged in such a way that the angle between any two directions is large. Indeed, the cosine similarity between the two most similar directions is 0.3. The average cosine similarity between the activations on the returning suffixes and their means is 0.905 for q_0, 0.890 for q_1, 0.911 for q_2 and 0.991 for q_3. This is



(a) Extracted Automaton



(b) Similarity matrix

Figure 16. The extracted Moore machine and the similarity matrix for a transformer trained on $\mathcal{D}_1$.

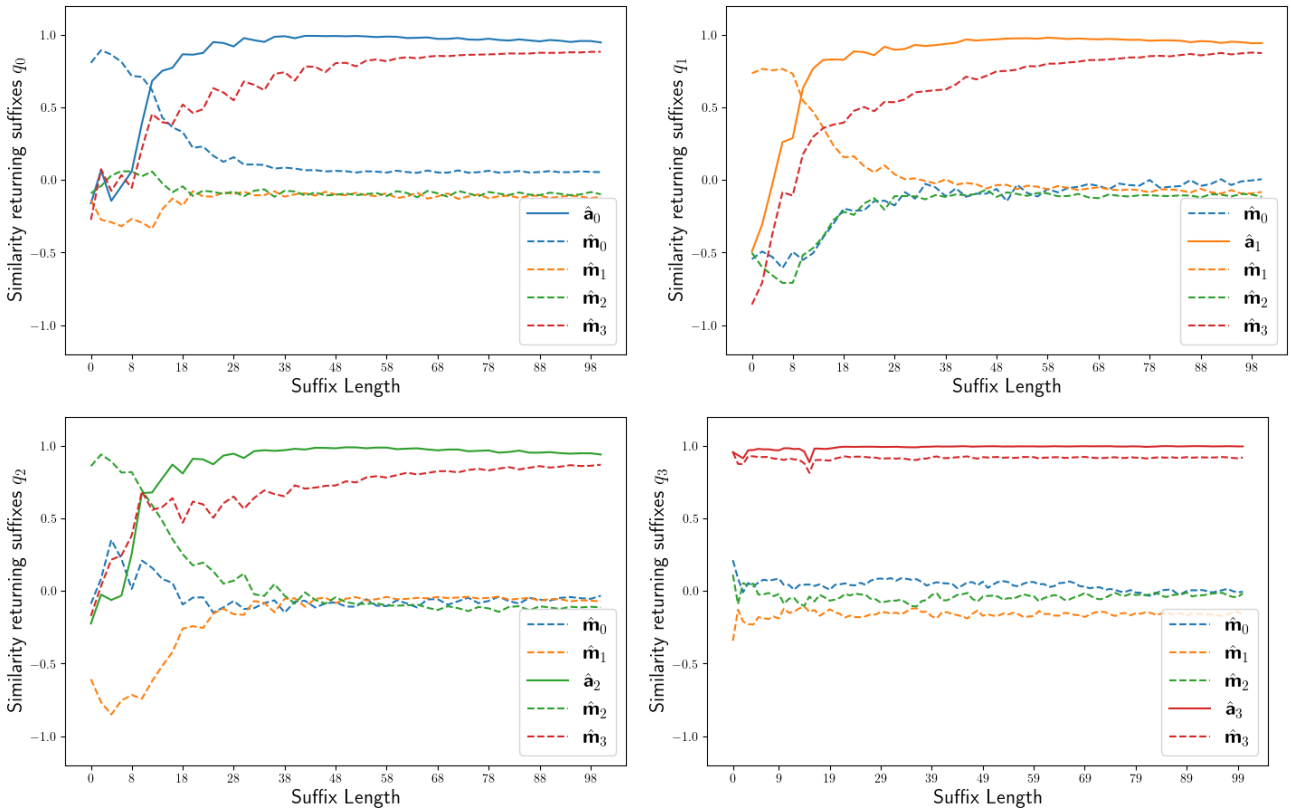
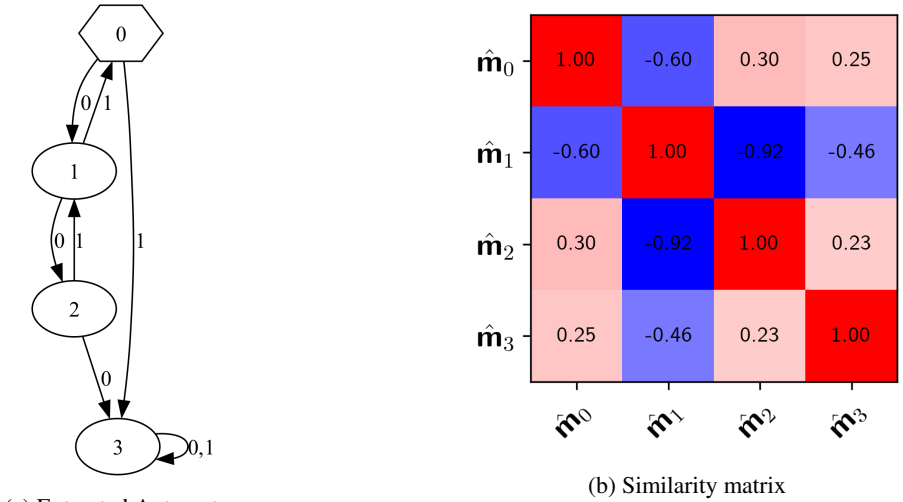


Figure 17. The cosine similarities between the activations on the returning suffixes and their average as well as the m-directions for the language $\mathcal{D}_2$.

rather low for q_0, q_1 and q_2 to speak of a common direction and we see from the plot that the transformer again confuses the correct states for the garbage state except for the early positions. This seems to be similar to what we have seen for $\mathcal{D}_1$ and we propose the same explanation.

We show in Figure 19 the probabilities transformers trained on $\mathcal{D}_1$ and $\mathcal{D}_2$ assigned to the different positions of a sequence 0101...0101 of length 24.

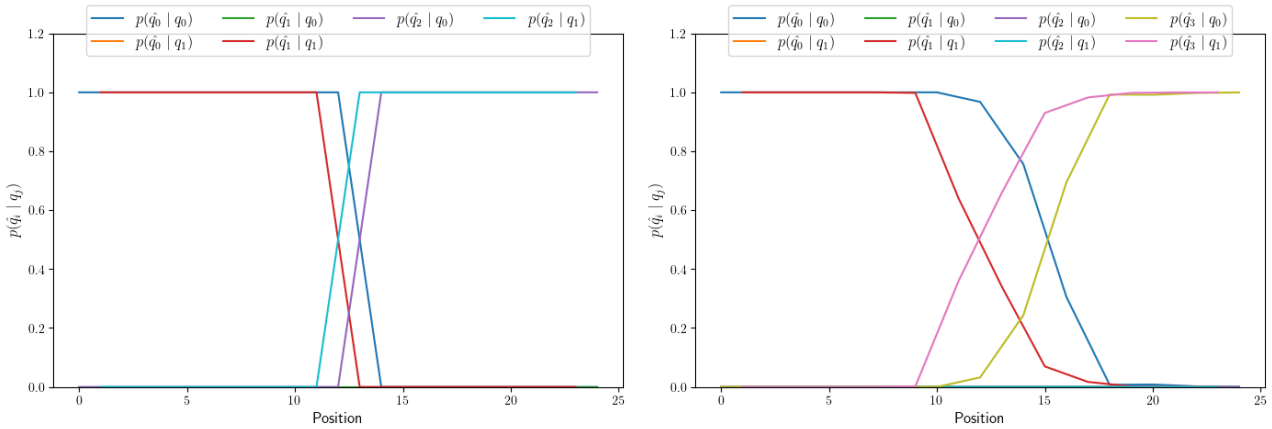
We observe that the transformer trained on $\mathcal{D}_2$ suddenly confuses the correct states, starting from position 13. This again confirms that transformers fail when reset sequences are absent. We note that for this language, a large number of sequences in the training data are labelled with the garbage state in later positions. This imbalance might explain why these transformers fail even before the training length.



(a) Extracted Automaton

(b) Similarity matrix

Figure 18. The extracted Moore machine and the similarity matrix for a transformer trained on $\mathcal{D}_2$.



(a) $\mathcal{D}_1$

(b) $\mathcal{D}_2$

Figure 19. The probabilities assigned by a transformer trained on $\mathcal{D}_1$ and $\mathcal{D}_2$ on sequences on all positions of a sequence of alternating 0's and 1's of length 24.

D.7. $\mathcal{C}_2$, $\mathcal{C}_3$, and Parity

Finally, we look at the non-star-free languages. First, we focus on $\mathcal{C}_2$ and $\mathcal{C}_3$, which are both in AC^0. They are unary languages and the transformer must therefore rely entirely on the beginning-of-sequence symbol and rotary positional encodings. We show the similarities between the activations on the returning suffixes, their mean and the m-states in Figures 20 and 21. We again observed the m-states to be opposed to each other for $\mathcal{C}_2$ and to lie in the same plane for $\mathcal{C}_3$.

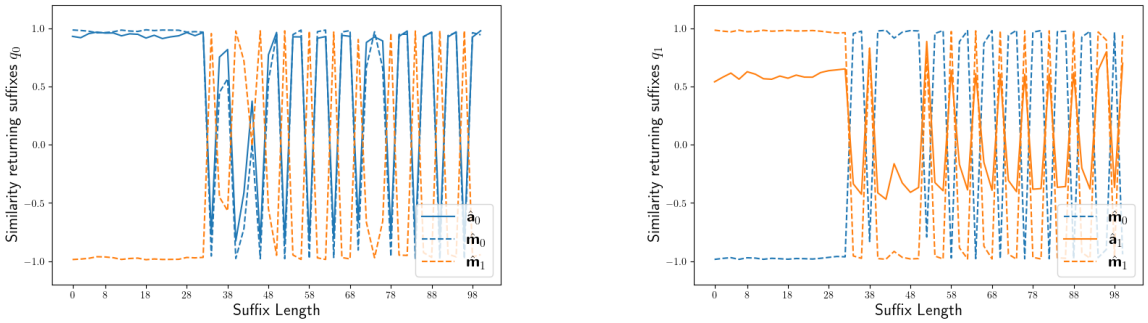


Figure 20. Cosine similarities of activations on returning suffixes for $\mathcal{C}_2$.

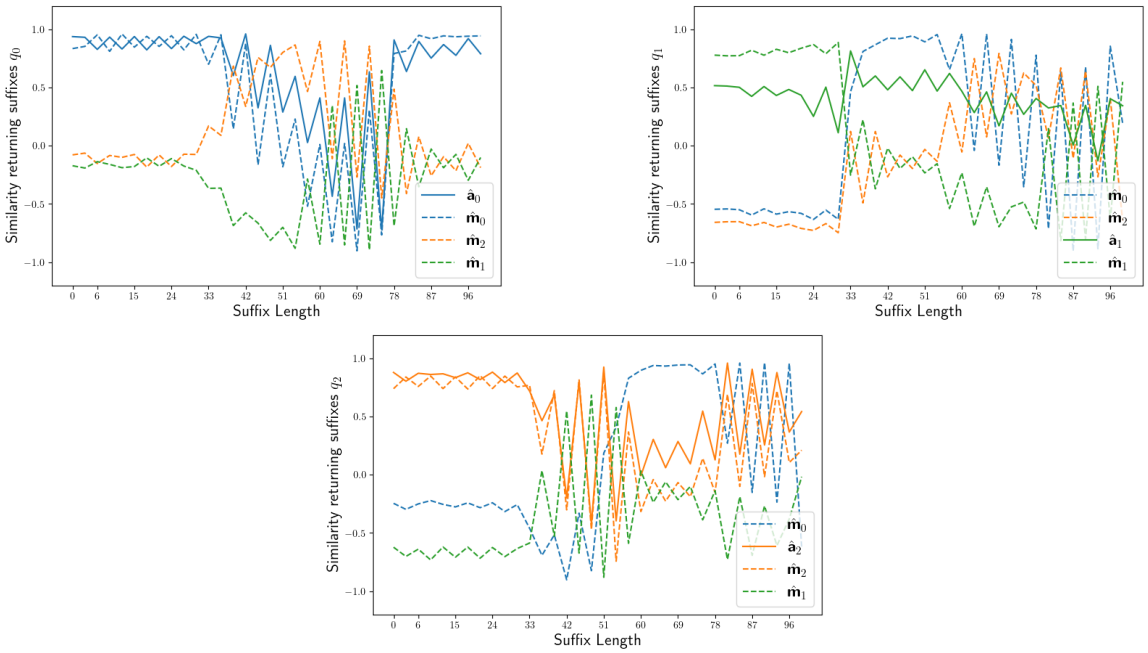


Figure 21. Cosine similarities of activations on returning suffixes for $\mathcal{C}_3$.

For these languages, the transformer still finds a common direction to represent the states of the extracted machine. However, beyond the length seen in training, it quickly loses this ability.

Finally, for the language Parity, we show the similarities between the activations on the returning suffixes, their mean and the m-directions in Figure 22.

The transformer trained on this language fails to align the activations on sequences ending in the same state along a common direction. As the sequences become longer, the activations lie in a direction almost completely perpendicular to the line the two m-directions are aligned with. At this point, the transformer does not differentiate between the two states at all.

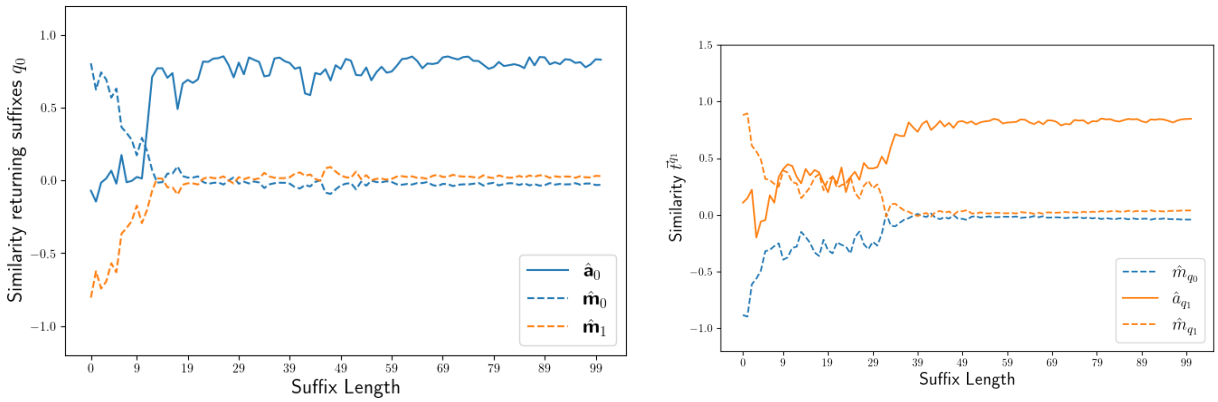
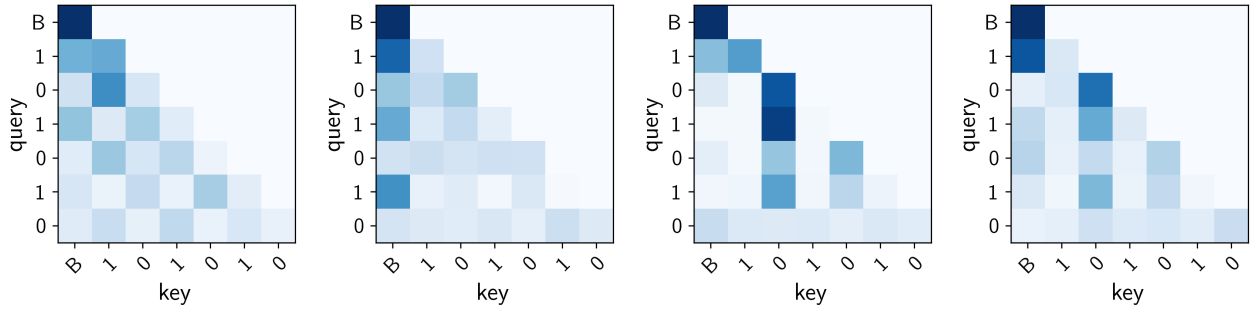


Figure 22. Cosine similarities of activations on returning suffixes for Parity.

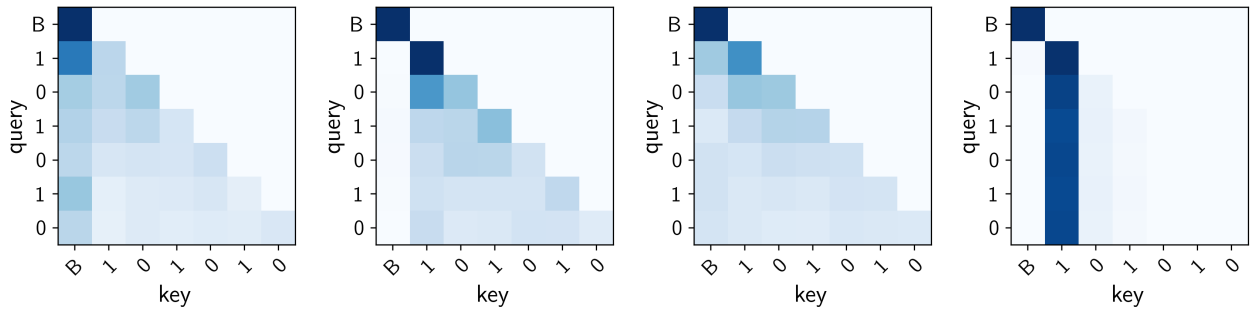
D.8. Inspecting Attention Heads

Here we investigate for the languages Ones, First, $\mathcal{G}_1$ and $\mathcal{G}_2$ the attention patterns that the transformers learned. For each language, we show the attention patterns of the four attention heads on the sequence $B010101$, where B is the beginning-of-sentence symbol. The results are shown in Figures 23,24,25 and 26. A darker shade of blue indicates a higher attention score.



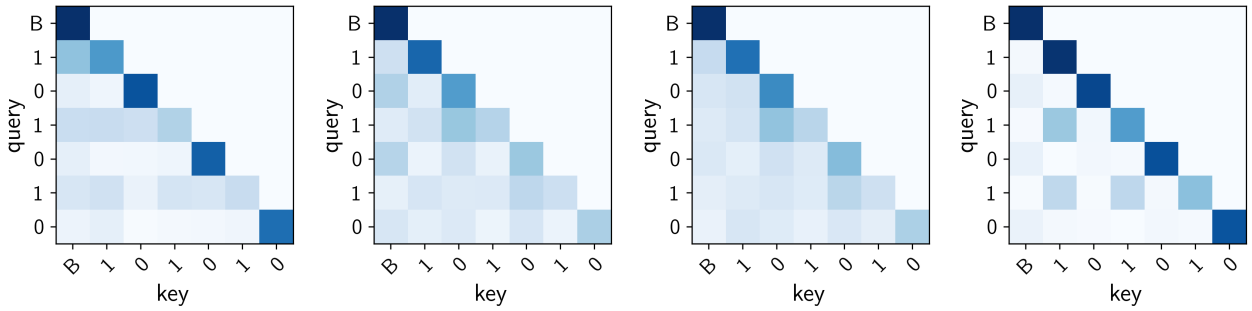
(a) Attention head 1. (b) Attention head 2. (c) Attention head 3. (d) Attention head 4.

Figure 23. The attention patterns of a transformer trained on Ones on the sequence $B010101$.



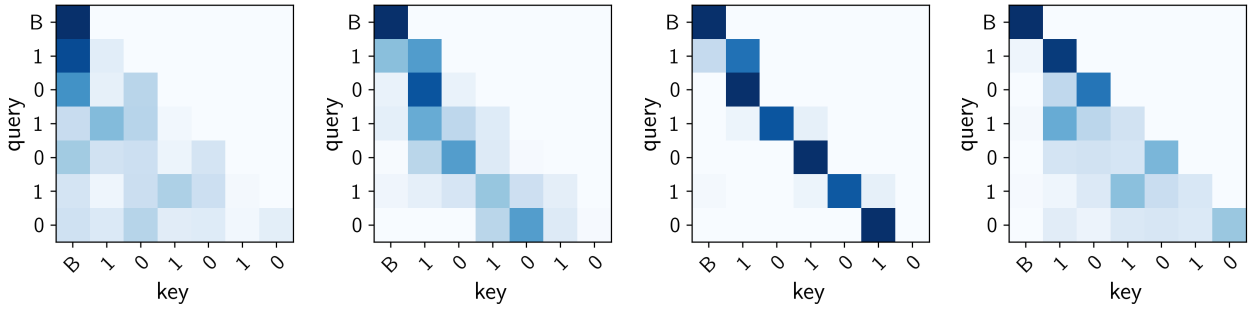
(a) Attention head 1. (b) Attention head 2. (c) Attention head 3. (d) Attention head 4.

Figure 24. The attention patterns of a transformer trained on First on the sequence B010101.



(a) Attention head 1. (b) Attention head 2. (c) Attention head 3. (d) Attention head 4.

Figure 25. The attention patterns of a transformer trained on $\mathcal{G}_1$ on the sequence B010101.



(a) Attention head 1. (b) Attention head 2. (c) Attention head 3. (d) Attention head 4.

Figure 26. The attention patterns of a transformer trained on $\mathcal{G}_2$ on the sequence B010101.

We observe that head 3 and 4 of the transformer trained on Ones pay strong attention to zero symbols. The same is true for head 1, whenever the current symbol is a one. We observe that head 4 of the transformer trained on First pays almost exclusive attention to the symbol at the first position. All heads of the transformer trained on $\mathcal{G}_1$ pay strong attention to the position corresponding to the current symbol. The transformer trained on $\mathcal{G}_2$ pays stronger attention to the current symbol and the two previous symbols, having one head for each of these three positions.

These observations support the hypothesis proposed in Section 7.