Automata in Space: Formal Language Theory meets Neural Computation[*]

J. Llarena
Independent
jose.llarena@gmail.com

December 2024

Abstract

Understanding neural systems, both natural and artificial, lags far behind their remarkable performance. Current techniques for explaining their inner workings are often criticised for lacking theoretical grounding or causal structure, for being incomplete, labour-intensive or limited to specific models or tasks. In this work, I address this challenge through an empirical study of the hidden representations of Simple RNNs , GRUs, LSTMs, MLPs, CNNs and encoder-Transformers after being trained or constructed to recognise the binary languages XOR/PARITY, OR and AND. Using the weight vectors of the linear presentation of Weighted Finite State Automata (WFSA) learned with Spectral Learning, I develop a reference reverse engineering model to guide the characterisation of the networks' internal mechanisms in terms of algorithms, representational primitives and implementing operations within Marr's conceptual framework of levels of analysis. I find that the sequences of hidden states are homomorphic to the unweighted automaton corresponding to the WFSA, either in their raw state -for recurrent models-, or -for feedforward models- through a simulation created by unrolling linear transforms into a sequence of atomic operations. I conclude that elucidating the meaning of specific dimensions or functional roles for neurons is not necessary for a mathematically grounded, intuitive and complete causal explanation for the models' ability to solve the task. Finally, I discuss potential extensions to more expressive automata, more complex models, harder tasks and learning dynamics. Code repository available at https://github.com/josellarena/ais

1 Introduction

Despite their remarkable performance on symbolic tasks, our understanding of the algorithmic principles underlying neural systems —rooted in a subsymbolic substrate[1] (Smolensky 1988)— remains limited (Doshi-Velez and Kim 2017).

Artificial Neural Networks' (NNs) complex architectures, opacity of learned representations, non-intuitive feature extraction and lack of transparency in decision-making have earned them a reputation for being "black boxes". Yet as NN-based models are increasingly deployed across diverse applications, it is crucial to unravel their inner workings. Of particular urgency is addressing notable problems, such as "hallucinations" and toxic outputs (Ji et al. 2023), to ensure that these models are fair, reliable, robust, and capable of earning user trust (He et al. 2024). Efforts to date have attempted to explain models' behaviour either

[*]This title is a nod to the excellent "Frege in Space: A Program for Compositional Distributional Semantics" (Baroni et al. 2014) and "Formal language theory meets modern NLP" (Merrill 2021), both inspirations for this work.

[1]"Substantive progress in subsymbolic cognitive science requires that systematic commitments be made to vectorial representations for individual cognitive domains. . . . Unlike symbolic tokens, these vectors lie in a topological space in which some are close together and others far apart" (Smolensky 1988).

through methods that treat NNs as black boxes (e.g., input attribution and synthesis, probing, distillation, …) or by dissecting their internals and identifying critical components (weights (Frankle and Carbin 2019), units (Sajjad, Durrani, and Dalvi 2022), circuits (Elhage et al. 2021)), learned algorithms (functions (Todd et al. 2024), modular addition (Zhong et al. 2024), group operations (Chughtai, Chan, and Nanda 2023 & al.'23)) and representations (words (Mikolov et al. 2013), syntax (Hewitt and Manning 2019; Diego-Simón et al. 2024), dimensions (Elhage et al. 2022)). Nonetheless, criticisms have put a question mark on their generalisability, completeness and validity (Vilas et al. 2024; Arvan, n.d.; Räuker et al. 2023).

Cognitive neuroscience faces analogous challenges: the task of aligning abstractions in linguistics (morphemes, phrases, concatenation, merge,…) with biological mechanisms (neurons, cortical columns, oscillations,…), the so-called *mapping problem* (Poeppel 2012), remains a significant difficulty (Poeppel and Embick 2005; Poeppel 2012; Embick and Poeppel 2015; Hale et al. 2021); which linguistics units are stored and how it is still a mystery (Poeppel and Idsardi 2022); and there is no consensus on the meaning of "mechanism" (Bree 2022). Work thus far has attempted to characterise the underlying computations in continuous spaces through dynamical systems (Vyas et al. 2020; Smith, Linderman, and Sussillo 2021; Fitz, Hagoort, and Petersson 2024) and geometry (Chung and Abbott 2021; Bernardi et al. 2020). However, a unified account has yet to materialise (Lillicrap and Kording 2019).

In this paper, I propose to use automata as the primary construct for understanding both artificial and biological neural systems' inner workings. Automata are universally applicable, in that, if a function is computable at all, it will be amenable to being modelled by them. This is a consequence of the Church-Turing Thesis[2] (Barak 2022), and it offers the potential to be a unifying perspective across diverse tasks and model architectures. Furthermore, explanations of model decisions based on automata are inherently causal, as they break down the effective steps that give rise to the model's behaviour in the form of a path computation algorithm (see appendix 3.4).

Unlike most previous work, where automata and NNs are connected indirectly, through theoretical (expressivity or complexity results) or empirical equivalences (extraction, distillation, benchmarks), here I study the detailed mappings between the network's operations in vector spaces and the elements of the automata corresponding to the target tasks (recursive XOR, OR and AND). This leads to a departure from currently popular approaches, as I analyse *computational state* rather *input feature* representations[3]. Also in contrast to prior work, which investigates specific, often non-standard, topologies, I examine all major architectures: Simple RNN (SRNN), GRU, LSTM, MLP, CNN and Transformer (TFM). In order to theoretically ground this work within the larger efforts in the field, I frame it in terms of the algorithmic and primitive levels of analysis in Marr's multi-level framework (Marr and Poggio 1976).

Further diverging from earlier methods, I use automata not just as abstract models of the tasks' solutions but, through the application of the linear representation of WFSAs learned through Spectral Learning (SL), I am able to construct a normative reverse engineering model for the NN's hidden representations themselves (cf. *Representation Engineering*, Zou et al. '23)

This paper's main contributions are:

- **A conceptual shift**: a call to use automata as the conceptual anchor around which algorithmic explanations of all neural models ought to be built.

[2]Automata come in many shapes and forms: hybrid (Henzinger 1996), weighted, accepting, transducing, etc… (Droste, Kuich, and Vogler 2009) which allows them to model every type of task. There are other kinds of classical computational models such as logic circuits, transition systems or semi-automata, as well as other models of computation: the lambda calculus, rewrite systems, cellular automata, Post systems or combinatory logic amongst others, but all of them are equivalent to some kind of automaton (Linz 2001; Sipser 2006).

[3]These two views can be unified by interpreting states as context-dependent inputs.

- **A novel methodology**: a widely applicable method to study the intermediate representations of neural models, using WFSAs as normative models to motivate simple interventions.
- **An empirical study**: investigating how different model architectures solve the language acceptance task (binary classification) for recursive boolean functions XOR (aka PARITY), OR and AND.

The remainder of the paper is structured as follows: Section 2 describes existing work and how this paper differs from it. Section 3 establishes the theoretical preliminaries necessary to understand the rest of the paper. Section 4 describes the experimental setup. Section 5 discusses the experimental results. Section 6 discusses the broader implications of the findings and suggests future lines of work. Section A provides additional details.

2 Related Work

The first works relating neural networks with automata connected Finite State Automata (FSA) (Minsky 1967) and Regular Expressions (Kleene 1956) with McCulloch & Pitts' nerve nets (Mcculloch and Pitts 1943), equivalent to present-day's (single-layer) Perceptrons. These works were theoretical in nature and didn't study learned models or make detailed analyses of hidden spaces.

Early work relating automata to modern NN architectures began during the Connectionist era, almost all of it on RNNs. (Servan-Schreiber, Cleeremans, and McClelland 1988) train RNNs to learn a regular language defined by a Reber grammar (Reber 1967). They then proceed to compare the hidden states against the corresponding FSA states, during and after training, through cluster analyses. (Sun et al. 1990) train 2nd-order RNNs to learn regular languages and, with an external stack, context-free languages. They extract FSAs and PDAs but don't analyse hidden states. (Pollack 1991; Zeng, Goodman, and Smyth 1993) trained RNNs to learn Tomita grammars (Tomita 1982), study the hidden spaces, extract FSAs and compare them with the reference ones. (Siegelmann and Sontag 1992) theoretically analyse RNNs and prove that under ideal circumstances they are equivalent to Turing Machines, but do no empirical work. (Casey 1996) theoretically analyse RNNs, train RNNs on regular languages, analyse their hidden states and extract FSAs. (Tiňo et al. 1998) train RNNs on regular languages, analyse their hidden states and compare them with the reference FSAs. (Omlin and Giles 2000) train RNNs on regular languages analyse their hidden states, extract WFSAs and compare them with the reference WFSAs. (Rodriguez 2001) empirically study how RNNs learn CFLs and CSLs, analysing hidden state trajectories.

More recent work has extended and deepened the connections between automata and RNNs to other architectures. The research closest to this paper features detailed analyses of hidden representations. (Michalenko et al. 2019) train RNNs on emails generated by Regular Expressions and analyse the learned hidden representations and compare them with the corresponding FSA. They also extract an FSA from the models. (Liu et al. 2022, 2024) establish learnability conditions for regular languages through automata and group theory and connect them with the layered architecture of TFMs. They train TFMs and analyse their hidden states, comparing them with the corresponding FSAs. (Adriaensen and Maene 2024) extract DFAs from TFMs train on formal languages using Angluin's L* algorithm (Angluin 1987). They study the internal representations of the learned models.

Other work also makes connections between (W)FSAs and NNs but doesn't analyse the learned/constructed representations. (Weiss, Goldberg, and Yahav 2024) extract DFAs from RNNs train on languages generated by Tomita grammars using Angluin's L* algorithm. (Peng et al. 2018) make theoretical connections between different types of RNNs and WFSAs, train new WFSA-inspired architectures. (Ayache, Eyraud, and Goudian 2019; Eyraud and Ayache 2021) extract WFSAs from trained RNNs, treated as black boxes, using SL. (Merrill 2021) draws theoretical links between WFSAs and RNNs, amongst other architectures, using Formal Language Theory (FML). (Zhang, Wang, and Giles 2021) use FML to explain the learnability of different Tomita grammars by RNNs, backed up by experiments. (Li, Precup, and

Rabusseau 2022) prove an equivalence between WFSA and linear 2nd-order RNNs, and run experiments to compare SL and alternative learning strategies. (Svete and Cotterell 2023) study the equivalence between probabilistic FSAs and simple RNNs from a theoretical point of view. (Rizvi-Martel et al. 2024) theoretically prove and empirically confirm that modified TFMs can simulate WFSAs and Tree Automata.

Less closely related are works that present formal models other than automata to understand neural representation. (Weiss, Goldberg, and Yahav 2021; Lindner et al. 2023) compile tasks as high-level programs into TFMs and then train TFMs on the same tasks and compare their internal representations. The tasks are compiled into high level computational models using operations like "copy" or "select", which are higher in the algorithmic abstraction than automata. Relatedly, (Friedman, Wettig, and Chen 2023) train TFMs on formal languages and then turn them into high-level programs. (Geiger et al. 2021) use causal models in the form of graphs to extract explanations for how RNNs and TFMs solve tasks. (McCoy et al. 2019) show that trained RNNs' hidden spaces approximate Tensor Product Representations (TPRs), distributed embeddings of symbol structures in the form of filler-role bindings (Smolensky 1990). (Chughtai, Chan, and Nanda 2023) analyse how trained MLP's solve algebraic group composition. (Zhong et al. 2024) show how TFMs solve modular arithmetic tasks. (Kaushik and Martin 2022) propose a framework based on categorial grammar and TPRs to model language comprehension. (Kim and Bassett 2023) introduce a programming language for RNNs to perform symbolic computations.

3 Methods

3.1 Levels of Analysis[4]

I situate the research in this paper within the algorithmic and primitives level of Marr's 4-level framework of analysis (Marr and Poggio 1976), which originated in neuroscience and has previously been applied to Machine Learning (Hamrick and Mohamed 2020). As a multi-level framework, it provides a common frame-of-reference and vocabulary to compare arguments and evidence, while establishing consistency, constraints and complementation relationships between studies. It can help examine models regardless of the complexity of their architecture, training data, or task performed. Finally, and crucially for the present work, it helps explain how learning systems may simultaneously implement computations with both symbolic and distributed representations at different levels. (Hamrick and Mohamed 2020; Vilas et al. 2024; He et al. 2024)

Marr & Poggio (1976) proposed four levels of analysis to study machines that solve an information processing problem[5] (Vilas et al. 2024):

- The computational level: gives a functional specification of the capacity underlying the observed behaviour, i.e., it describes what the system is doing.
- The algorithmic level: explains the system's ability to solve the problem by spelling out a sequence of human-understandable steps
- The primitives level: specifies how the algorithm is executed using primitive operations and representations, the building blocks of the system
- The implementation level: characterizes how the primitives are implemented in the model

At the computational level, the models can be understood as having the ability to perform basic boolean logic inference on infinite binary sequences through binary classification: those with an odd number of 1s (XOR), those with at least one 1 (OR) and those with no 0s (AND). Alternatively, they can be

[4]They can also be understood as levels of explanation (Vilas et al. 2024).

[5]Marr (Marr 2010) later revised the number of levels to 3 (Vilas et al. 2024), but here I use the original 4 levels as they help to clarify the mapping between the basic explanatory concepts of automata theory and those of neural systems.

characterised as having the ability to recognise binary languages by accepting (resp. rejecting) strings in (resp. out of) XOR, OR and AND input spaces. The former view is typical of Machine Learning and the latter of FLT.

At the algorithmic level, explanations depend on the model, but in this paper they all have the form of computing a path through an FSA plus a mapping from FSA states to outputs. As we'll see in section 5, recurrent models keep an intermediate representation that switches between points/regions of hidden space, homomorphic to the states of the corresponding FSA, as inputs are consumed, and which is then mapped to the binary output by the decoder. The feed-forward networks (FFNs) count 1s and 0s and their decoders map these counts to the binary output by slicing the hidden space into decision regions.

At the primitives level, the models are analysed as constructing a graph in hidden space, with vertices corresponding to FSA states, and edges corresponding to FSA transitions. This graph data structure then serves as the foundation for explanations at the algorithmic level.

At the implementation level, explanations describe how the models' atomic operations, linear and non-linear functions, define the vertices/states and edges/transitions corresponding to the graph at the primitives level.

Levels of analysis are orthogonal to levels of organisation of the models. The latter refer to units/neurons, weights, layers and submodules, all of which can be part of explanations at the lower 3 levels.

Both levels of explanation and organisation can also be contrasted with levels of abstraction, which refer to the amount of detail at a given level of organisation that is used in explanations. This work focuses on basic computational abstractions (i.e., automata) embedded in simplified geometric spaces, de-emphasising the architectural specifics of a model[6].

3.2 Forward + Reverse Engineering

A novel methodological contribution of this paper is to forward-engineer reference representations followed by reverse- engineering target representations by comparing them with the reference ones.

The reference representations are constructed from weight vectors computed in WFSAs learned through SL (though they can be constructed too). Under the right conditions (see section 3.5.1, spectral learning will find the minimal automaton, in terms of the number of states, accepting the language in the training set. This allows this representation to be an optimal one, against which all other models' representations can be contrasted. Also, as we will see in section 4, these representations are suitable as baselines because they are particularly simple and homomorphic to the FSA underlying the spectral WFSA. Altogether, these traits justify their use as normative models.

3.3 Minimal Tasks, Minimal Models

With a view to finding explanations that are thorough and faithful, I chose tasks and models that minimise the number of confounding variables. This approach is inspired by the natural sciences[7] (Hutson 2018).

The tasks are recursive versions of the boolean functions XOR, OR and AND, some of the simplest

[6]This is the so-called *Hopfieldian* view of the relation between the brain/hardware and cognition/computation, which explains cognition as the result of transformations between or movement within representational spaces. It stands in contrast with the *Sherringtonian* view, which interprets cognition as the result of operations on signals performed at nodes in networks and circuits (Barack and Krakauer 2021). A considerable amount of current work on interpretability (e.g., (elhelo2024inferring; Nikankin et al. 2024; Elhage et al. 2021)) can be cast as part of this effort.

[7]"AI needs to borrow from physics, where researchers often shrink a problem down to a smaller 'toy problem'... Physicists are amazing at devising simple experiments to root out explanations for phenomena" (Ben Recht)".

functions[8]. At the same time, they are some of the simplest formal languages -regular languages-, defined as those that can be recognised by FSA. For all tasks, there are well-known expressively equivalent[9] optimal solutions (Liu et al. 2023; Wang et al. 2018) as illustrated in Figures 1, 2 and 3.

All models except the constructed XOR-TFMs have one layer. No model except the constructed XOR-MLP has biases. Only the XOR-MLP has more than 3 hidden dimensions. All models have been learned or constructed to have 0 training, validation and test losses to ensure that found representations genuinely represent optimal (in terms of accuracy) solutions to the tasks.

$$S \to 0S \mid 1T$$
$$T \to 0T \mid 1S \mid \epsilon$$

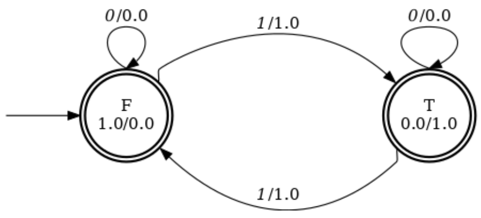
(a)

$$0^*(10^*10^*)^*10^*$$

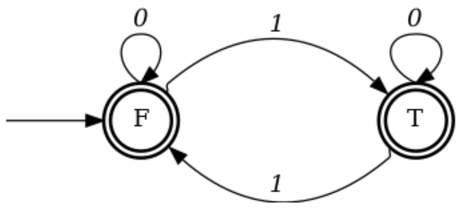
(b)

$$\boldsymbol{\alpha} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \mathbf{A}_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{A}_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

(c)



(d)



(e)

Input / State	*0*	*1*
→ F	F	T
T	T	F

(f)

Figure 1: Equivalent descriptions of the XOR language: a) right-linear grammar, b) regular expression, c) linear representation, d) WFSA state-transition diagram, e) FSA state-transition diagram, f) state-transition table

[8]They all are also "permutation-invariant", i.e., the order of the input tokens doesn't affect the output, one of the traits that make them particularly easy to model.

[9]Equivalent here means they all describe the same language. The formalisms themselves are not equivalent to each other, though they are translatable into each other (Linz 2001): grammars and regular expressions are generative models equivalent to each other, whereas algebraic (see section 3.4), graphical (state-transition diagrams), tabular (state-transition tables) and linear representations of FSAs are discriminative models that are also equivalent to each other.

$S \to 0S \mid 1T$

$T \to 0T \mid 1T \mid \epsilon$

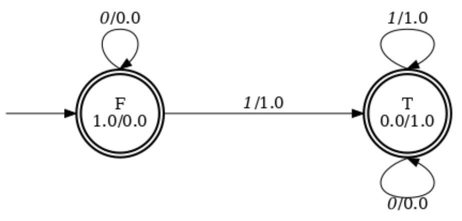
(a)

$0^*1[01]^*$

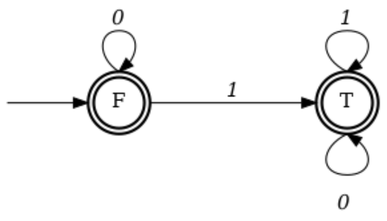
(b)

$$\boldsymbol{\alpha} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \mathbf{A}_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{A}_1 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

(c)



(d)



(e)

State \ Input	*0*	*1*
→ F	F	T
T	T	T

(f)

Figure 2: Equivalent descriptions of the OR language: a) right-linear grammar, b) regular expression, c) linear representation, d) WFSA state-transition diagram, e) FSA state-transition diagram, f) state-transition table

$S \to 1S \mid \epsilon$

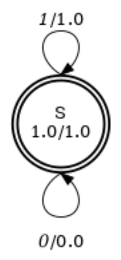
(a)

1^*

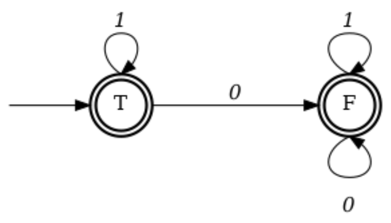
(b)

$$\boldsymbol{\alpha} = \begin{bmatrix} 1 \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} 1 \end{bmatrix} \quad \mathbf{A}_0 = \begin{bmatrix} 0 \end{bmatrix} \quad \mathbf{A}_1 = \begin{bmatrix} 1 \end{bmatrix}$$

(c)



(d)



(e)

State \ Input	*0*	*1*
F	F	F
→ T	F	T

(f)

Figure 3: Equivalent descriptions of the AND language: a) right-linear grammar, b) regular expression, c) linear representation, d) WFSA state-transition diagram, e) FSA state-transition diagram, f) state-transition table

3.4 FSA

FSAs are the simplest models of sequential computation, lying at the bottom of the hierarchy of automata, below Pushdown Automata, Linear Bounded Automata and Turing Machines. This hierarchy mirrors the

Chomsky hierarchy of formal languages, defined by the automata at the corresponding level (Jäger and Rogers 2012). FSAs represent computations for which the only memory needed is a variable keeping track of the current state. These tasks are exactly the set of regular languages.

The simplest of FSAs are acceptors[10], whose formal definition is a 5-tuple $\langle \Sigma, Q, q_0, F, \delta \rangle$ where:

$$
\begin{aligned}
\Sigma \quad & \text{is a finite set of input symbols, the vocabulary or alphabet} \\
Q \quad & \text{is a finite set of states} \\
q_0 \in Q \quad & \text{is the start or initial state} \\
F \subseteq Q \quad & \text{is the set of accepting or final states} \\
\delta : Q \times \Sigma \to Q \quad & \text{is a transition function, mapping pairs of states and symbols to states}
\end{aligned}
$$

Q and δ implicitly define a graph where the states correspond to vertices and the transitions between them correspond to edges. Together with Σ, they define a computation as the tracing of a path through the graph while a sequence of input symbols is consumed. Formally, a path π is pair $\langle (q_0, q_1, \dots q_n), (\sigma_1, \sigma_2, \dots \sigma_n) \rangle$ where:

$$
\begin{aligned}
(q_0, q_1, \dots, q_n) \in Q^n \quad & \text{is an ordered set of states} \\
(\sigma_1, \sigma_2, \dots, \sigma_n) \in \Sigma^n \quad & \text{is an ordered set of symbols, i.e., a string} \\
\forall\ \sigma_i\ , \forall\ q_i,\ q_{i+1},\ (q_i, \sigma_i, q_{i+1}) \in \delta \quad & \text{there is an edge from each pair of states labelled with that symbol}
\end{aligned}
$$

A path is complete if it starts at the initial state and ends in one of the final states. Formally:

$$
\begin{gathered}
(q_0, \sigma_1, q_1) \in \delta \text{ and} \\
q_n \in F
\end{gathered}
$$

An FSA then accepts a string if there is a complete path through it for that string. The sets of all strings accepted by an FSA is called its language (Gorman and Sproat 2022).

The description above is for *deterministic* FSAs. There's also another class of FSAs, nondeterministic FSAs, whose definition, $\langle \Sigma, Q, I, F, \delta^* \rangle$ is similar to deterministic FSAs, with the following differences (Kozen 1997):

$$
\begin{aligned}
\delta^* : Q \times (\Sigma \cup \{\epsilon\})^+ \to 2^Q \quad & \text{maps pairs of states and strings to subsets of states} \\
I \subseteq Q \quad & \text{is a set of initial states instead of a single one}
\end{aligned}
$$

Nondeterministic FSAs represent computations where multiple paths are computed in parallel and the first valid one is returned. They are able to recognise the same languages as deterministic FSA, but they allow much more concise representations of some tasks. Nondeterministic FSAs are important for our purposes, because WFSAs are equivalent to them.

[10]In this paper I let FSA stand for both Finite State Automaton and Finite State Acceptor. There are other Finite State Automata, such as Finite State Transducers.

3.5 WFSA

WFSAs[11] are an extension of FSAs where state and transitions are augmented with weights. There are two equivalent formal definitions of WFSAs, one based on semirings[12] and one based on linear algebra (Droste, Kuich, and Vogler 2009). The former defines a WFSA as being over a semiring $(K, \oplus, \otimes, \overline{0}, \overline{1})$ and is a 7-tuple $\langle \Sigma, Q, I, F, \Delta, \alpha, \beta \rangle$ (Balle and Mohri 2015; Mohri 2009) where:

Σ	vocabulary/alphabet
Q	states
$I \subseteq Q$	start/initial states
$F \subseteq Q$	final/accepting states
$\Delta : Q \times (\Sigma \cup \{\epsilon\}) \to K \times Q$	transition function
$\alpha : Q \to K$	initial weight function
$\beta : Q \to K$	final weight function

A path through a WFSA is an extension to a path through FSA's with a weight sequence $(w_1, w_2, ...w_n) \in K^n$). A WFSA path is complete if $\beta(q_n) \neq 0$ (Gorman and Sproat 2022), and the total weight for a given string is .

$$\text{WFSA}(\sigma_1 \dots \sigma_n) = \bigoplus_{\pi \in \mathcal{P}^n} [\alpha(q_0) \otimes \Delta(q_0, \sigma_1), \Delta(q_1, \sigma_2), \dots, \Delta(q_{n-1}, \sigma_{n-1}) \otimes \beta(q_n)]_\pi$$

where $\mathcal{P}^n$ is the set of all paths of length n in the WFSA. The other definition is a 3-tuple $\langle \boldsymbol{\alpha}, \boldsymbol{\beta}, \{\mathbf{A}_{\sigma \in \Sigma}\} \rangle$ where:

$\boldsymbol{\alpha} \in \mathbb{R}^{\lvert Q \rvert}$	initial vector
$\boldsymbol{\beta} \in \mathbb{R}^{\lvert Q \rvert}$	final vector
$\mathbf{A}_\sigma \in \mathbb{R}^{\lvert Q \rvert \times \lvert Q \rvert}$	the transition matrix associated to each alphabet symbol σ

$\{\mathbf{A}_\sigma\}$ can also be represented as rank-3 tensor $\mathbf{T} \in \mathbb{R}^{|Q| \times |Q| \times |\Sigma|}$, where the mapping from symbol to transition matrix is implemented as a tensor-vector product $\mathbf{T}\mathbf{x}_\sigma$, where $\mathbf{x}_\sigma$ is a one-hot vector with the single 1 in the dimension corresponding to the symbol σ at that index, assuming Σ has been sorted into an ordered set. The output of the WFSA, the weight of a string, is computed as a sequence of matrix-vector products:

$$\text{WFSA}(\sigma_1 \dots \sigma_n) = \boldsymbol{\beta}^\top \mathbf{A}_{\sigma_n} \dots \mathbf{A}_{\sigma_1} \boldsymbol{\alpha}$$

The linear format is the basis for the bridge between symbolic and non-symbolic/vectorial representations that's central to this paper's approach. It's exactly equivalent to the semiring format under the assumption that in the latter, all states are both initial and final states. Addtionally, there must be a transition from every state to every other state, with a 0-weight for those transitions missing from the transition relation or equivalently from the implicit unweighted graph.

[11]in some works they are defined as Weighted Finite Automata (WFA). Here I use WFSA for consistency with other works that use WFST for Weighted Finite State Transducer (Mohri 2001).

[12]There are multiple alternative semiring-based definitions of a WFSA (Mohri 2001; Balle and Mohri 2015; Gorman and Sproat 2022). Here, I use the one that more clearly shows the equivalence to its linear representation.

3.5.1 Spectral Learning

Spectral Learning (Bailly, Denis, and Ralaivola 2009; Hsu, Kakade, and Zhang 2009; Balle et al. 2014) is a technique for estimating WFSAs using the Singular Value Decomposition (SVD) of a matrix containing the statistics of prefixes and suffixes in sequential data, the Hankel matrix. SL is an alternative to Expectation-Maximisation based algorithms that is efficient and consistent (Li, Precup, and Rabusseau 2022). Provided a number of conditions are met[13], SL will find the WFSA with the minimum number of states, as well as initial, final and transition weights minimising the ℓ_2 loss on the training data (Quattoni and Carreras 2019). The only hyperparameter is the Hankel basis, the set of prefixes and suffixes used to index the Hankel matrix's rows and columns respectively.

The output of SL is the linear representation of a WFSA, with the initial and the final weights as vectors and the per-symbol weights in the form of matrices.

3.5.2 Stochastic Gradient Descent

WFSAs can also be trained with an SGD-based setup identical to standard supervised NN training. In this paper I use SGD-training as a way of probing the impact of architecture vs training scheme on a WFSA's hidden representation. It also serves as a debugging method to ensure SL is properly implemented.

3.6 Recurrent Neural Networks

RNNs, like WFSAs, are autoregressive sequential models. In fact, WFSAs have been proven to be equivalent to 2-order linear RNNs with one-hot inputs (Li, Precup, and Rabusseau 2022). As such, they are arguably the closest models to compare WFSAs against. In section 4.2 I compare the main recurrent architectures: Simple/Elman RNNs (SRNNs), GRUs and LSTMs.

3.7 Feedforward Neural Networks

Comparing automata to FFNs, MLPs, CNNs and TFMs, is not as straightforward as with RNNs. FFNs lack RNNs' serial, stateful nature. An alternative computational model for them are circuits, which are non-sequential, parallel and fixed-length. However, circuits can be made equivalent to length-limited FSAs (Savage 1997) after adding a dummy input corresponding to an automata's initial state and serialise their computation using the recursive scan (aka prefix-sum) algorithm (Hillis and Steele Jr 1986; Blelloch 1990). It is this approach that I take in this work with a view to providing a unifying computational model for all architectures' hidden spaces.

4 Experiments

The experiments presented in this section are per-model. All models are trained on the AND[14] and OR tasks. The XOR task can only be learned by the recurrent models. Therefore, XOR-MLPs and -TFMs are constructed instead. CNNs don't have an XOR task experiment as I know of no construction for them. AND and OR are linearly separable, while XOR is non-linear. All of them describe regular languages.

For each task, I train (or construct) a model until it reaches 0 validation loss. For trained models, the training loss is cross-entropy for all models except for WFSAs where it's ℓ_2 (also for linear-output RNNs).

[13]The conditions can roughly be summarised as that the basis must be complete, i.e., the Hankel matrix block it indexes must have the same rank as the minimal WFSA. There is no known guaranteed procedure to find complete bases (but see (Quattoni, Carreras, and Gallé 2017)), however I've empirically found that using a validation set that is a superset of the training set works for the tasks in this paper as well as for all binary boolean functions (found in experiments not shown here). See appendix A.3.

[14]AND is identical to Tomita Grammar 1 and it's also known as *ones* (Adriaensen and Maene 2024).

The validation and test loss is a thresholded 0-1 loss $\mathbb{1}_\gamma(y, \hat{y}) = \mathbb{I}(|y - \hat{y}| > \gamma)$, where 1 is given to a model's output $\hat{y}$ when it's less than the threshold $\gamma = 1\%$ from its target y. All models are trained on all strings from lengths 0 to 12 and are validated with 100 strings of length 13 to 18. They are tested on 100 strings of lengths 19 to 24. This setup is a compromise between speed and accuracy.

After training, the hidden states for selected inputs are visualised as a 3D graph, with the vertices representing the hidden states for each timestep and the edges joining each pair of consecutive states. For feed forward models, the recursive scan version of their hidden state is shown. The output for each step in the prefix-sum is then input to the rest of the model and the output is also shown as a 3D graph.

Details of training are set out in appendix A.1 and visualisation setups in appendix A.2.

4.1 WFSAs

4.1.1 Spectral WFSAs

WFSAs were estimated following the standard SL algorithm with one difference: I used Non-Negative Matrix Factorisation (NMF) (Lee and Seung 1999) instead of SVD to factorise the Hankel matrix as NMF gives more interpretable weights[15]. The hyperparameters, including the Hankel basis and NMF initialisation options, were found through a grid search driven by minimising the 0-1 loss on the validation set. The full algorithms are set out in appendix A.3.

Figures 1, 2 and 3 show the canonical solutions to all tasks. Figures 4, 5 and 6 show the vector weights computed after each token in the example sequence 0101 has been consumed. As can be seen, they traced the same graph as the unweighted FSA.

4.1.2 SGD WFSAs

To test the impact of architecture vs training scheme on the Spectral WFSAs' internal representation I trained WFSAs with SGD and backpropagation to 0 validation loss. Figures 7, 8 and 9 depict the weight vectors computed for input 0101. As can be observed, the weight vectors trace the same graph as in the SL-trained ones, and so it appears that it's the architecture that is responsible for its structure.

4.2 RNNs

I trained SRNNs, GRUs and LSTMs with the same setup as the SGD WFSA. Figures 10, 11 and 12 visualise the SRNNs' hidden state for all tasks. Figures 13, 14 and 15 show the GRUs' hidden state for all tasks. Figures 16, 17 and 18 illustrate the LSTMs' hidden state for all tasks. As can be seen, and unlike trained WFSAs, there are often more vertices than there are states in the FSA. These vertices are in the correct decision region, which means that as far as the model is concerned they are output-relative equivalent[16]. For comparison, I repeated the same setup but with the softmax disabled, i.e., with a linear output. Figures 19 - 27 present the results, confirming that all output-equivalent vertices lie on the same line (aka 2-dimensional affine subspace)[17].

[15]This doesn't always guarantee non-negative weights or outputs. See (Glaude, Enderli, and Pietquin 2015) for a method to ensure outputs are always probabilities.

[16]This notion of state equivalence is different from the classic one in FLT (Linz 2001), where two states are considered equivalent if, for all strings in the language, they both lead to the same final state. I plan on applying this more complex test in future work.

[17]Another way of looking at it is that it's the isolines and decision regions that are themselves equivalent to the FSA's states.

4.3 FFNs

As feedforward networks don't keep a running state, including an initial one, they cannot be directly compared to WFSAs. Instead, we are going to make two changes that bring them closer: first we add a dummy initial token to the sequence inputs and take the computed hidden state to be the initial FSA state, as in recurrent models. Then, using the recursive scan algorithm, we decompose the compound hidden state that aggregates all per-input hidden states to compute the final classification decision. The result will be a sequence of states equivalent to the ones in a recurrent model.

For all trained networks, the training configuration is otherwise the same as for the RNNs.

4.3.1 MLPs

We start the study of MLPs' hidden states with the AND and OR tasks as they are simpler than the constructed XOR one.

The unbatched forward pass in the trained MLPs is given by the following equations:

$$\begin{aligned}
\mathbf{H} &= \max(0, [\mathbf{W}_e \mathbf{X}^\top]^\top) \\
\mathbf{z} &= [z_d] = \sum_{t=1}^{T} \mathbf{H}_{t,d} \\
\mathbf{y} &= \sigma(\mathbf{W}_z \mathbf{z})
\end{aligned}$$

where $\mathbf{X} \in \mathbb{R}^{T\times|V|}$ is the matrix of one-hot input vectors, $\mathbf{W}_e \in \mathbb{R}^{D\times|V|}$ are the embedding weights, $\mathbf{H} \in \mathbb{R}^{T\times D}$ is the hidden layer, $\mathbf{z} \in \mathbb{R}^{D}$ is the reduction (aka pooling or aggregation) layer, $\mathbf{W}_z \in \mathbb{R}^{2\times D}$ is the logit weights, σ is the softmax function and $\mathbf{y} \in \mathbb{R}^2$ is the output layer. $\mathbf{z}$ is where the inputs from different timesteps are aggregated, and it's therefore the level where it makes sense to intervene to compute a per-timestep running total similar recurrent models' hidden state. We therefore modify the equations as follows:

$$\begin{aligned}
\mathbf{Z} = [z_{t,d}] &= \sum_{\tau\leq t} \mathbf{H}_{\tau,d} \\
\mathbf{Y} &= \sigma([\mathbf{W}_z \mathbf{Z}^\top]^\top)
\end{aligned}$$

where $\mathbf{Z} \in \mathbb{R}^{T\times D}$ contains the prefix-sums for each timestep in its rows, and $\mathbf{Y} \in \mathbb{R}^{T\times 2}$ contains the output for each prefix-sum in its rows. Figure 28 shows $\mathbf{Z}$ in the lower level and $\mathbf{Y}$ in the upper level for input 1010 to the OR-model. Vertices in $\mathbf{Z}$ space reflect the number of 1s. Zero counts of 1s are in the negative decision region and larger counts are all on the positive decision region. This is consistent with the fact that the presence of one or more 1s that classifies a binary string as belonging to the OR language. Figure 29 shows the same probe but for the AND-model. As with OR, the vertices in $\mathbf{Z}$ space reflect the counts of 0s, with zero counts in the positive decision region and one or more in the negative region. Again, this matches a count-based algorithmic description of the task. The $\mathbf{Y}$ space simulates the FSA by tracing its state-diagram.

MLPs cannot apparently learn the length-extrapolating version of XOR[18] and so here I study the con-

[18] I know of no proof that they can('t) but all my attempts at getting them to do so have failed so far. It is conceivable that a different training scheme and/or a deeper architecture could succeed. In experiments not shown in this paper, I did manage to get an MLP to learn XOR by just replacing $\mathbf{z} = \sum_{t=1}^{T} \mathbf{H}_{t,d}$ with $\mathbf{z} = \prod_{t=1}^{T} \mathbf{H}_{t,d}$. This leads me to speculate that it might be the non-linearity of XOR that prevents successful learning, rather than the training scheme or the general architecture.

struction by Rumelhart et al. (1986) for the non-length-extrapolating version of XOR[19], following the exposition in Chiang & Cholak (2022). Rumelhart et al. (1986) showed that MLPs can be constructed that solve XOR for inputs of length N provided the hidden dimension is also N. The equations defining this MLP are:

$$\mathbf{z} = H(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$y = H(\mathbf{W}_2\mathbf{z} + b_2)$$

where $\mathbf{x} \in \mathbb{R}^N$ is a binary input, $\mathbf{W}_1$ is the N-dimensional unity matrix:

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

$\mathbf{b}_1$ is the hidden bias vector defined as:

$$\begin{bmatrix} -1 + .5 \\ -2 + .5 \\ \vdots \\ -n + .5 \end{bmatrix}$$

H is the heaviside activation function:

$$H(x) = \begin{cases} 0, & x < 0, \\ 1, & x \geq 0. \end{cases}$$

$\mathbf{z} \in \mathbb{R}^N$ is the hidden layer, $\mathbf{W}_2$ is the single-row logit weight matrix:

$$\begin{bmatrix} -1^0 & -1^1 & \cdots & (-1)^{n-1} \end{bmatrix}$$

and b_2 is the scalar logit bias $-.5$.

Note that in this construction, y is a scalar instead of the 2-dimensional vector we've used in all other models. In order to make them comparable we redefine $\mathbf{W}_2$ as:

$$\begin{bmatrix} -1^1 & -1^2 & \cdots & (-1)^n \\ -1^0 & -1^1 & \cdots & (-1)^{n-1} \end{bmatrix}$$

[19]This can also be learned, but I use the construction for simplicity and because it helps understand a similar construction for TFMs (section TFMs).

and $\mathbf{b}_2$ as $[.5 \quad -.5]$. Probing then proceeds as for the learned MLPs, applying prefix-sum to simulate a running state. This we achieve by "unrolling" the matrix-vector multiplication between $\mathbf{W}_1$ and $\mathbf{x}$:

$$\mathbf{Z} = [z_{t,d}] = H(\sum_{\tau \le t} (\mathbf{W}_1 \circ x)_{\tau,d} \oplus \mathbf{b}_1)$$
$$\mathbf{Y} = H([\mathbf{W}_2 \mathbf{Z}^\top]^\top \oplus \mathbf{b}_2)$$

where $\circ$ is the broadcast Hadamard product and $\oplus$ the broadcast matrix addition.[20] $\mathbf{Z} \in \mathbb{R}^{N \times N}$ and $\mathbf{Y} \in \mathbb{R}^{N \times 2}$ are now matrices with their rows being the prefix-sums for each timestep and the output for each prefix-sum respectively. Figure 30 shows the probes for input \$10\$1. As can be observed, $\mathbf{Z}$ counts the number of 1s as for the OR task, and $\mathbf{Y}$ traces the XOR-FSA as a state-transition diagram.

For all models and just like with RNNs, we can choose to map the concept of an FSA state to either the decision regions, or to the vertices under the understanding that all of those in the same decision regions are output-equivalent.

4.3.2 CNNs

CNNs apparently can't learn the length-extrapolating version of XOR[21] so I only present results for AND and OR. The unbatched equations for the model are:

$$\mathbf{X}' = \begin{bmatrix} \mathbf{0} \\ \mathbf{X} \end{bmatrix}$$
$$\mathbf{C} = [c_{h,t}] = \sum_{d=1}^{|V|} \mathbf{W}_{h,d} * [\mathbf{X}'^\top]_{d,t}$$
$$\mathbf{p} = [p_h] = \max_t \mathbf{C}_{:,t}$$
$$\mathbf{z} = \mathbf{W}_z \mathbf{p}$$
$$\mathbf{y} = \sigma(\mathbf{z})$$

where $\mathbf{X}' \in \mathbb{R}^{T+1 \times |V|}$ is the input matrix $\mathbf{X} \in \mathbb{R}^{T \times |V|}$ augmented with a zero vector to ensure a hidden state is computed for the initial dummy input, as the convolution reduces the dimension by 1, $*$ is the 1D convolution operator with a kernel size 2, $\mathbf{C} \in \mathbb{R}^{D \times N}$ is the convolutional layer, *max* is the 1D *maxpool* operator, $\mathbf{p} \in \mathbb{R}^D$ is the pooling layer, $\mathbf{W}_\mathbf{z} \in \mathbb{R}^{2 \times D}$ is the logits weight matrix, σ is the softmax function and $\mathbf{y} \in \mathbb{R}^2$ is the output. For probing, we use the recursive scan algorithm as with MLPs:

[20]The broadcast Hadamard product is defined as:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} a\alpha & b\beta \\ c\alpha & d\beta \end{bmatrix}$$

and the broadcast matrix addition is similarly defined as:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} a+\alpha & b+\beta \\ c+\alpha & d+\beta \end{bmatrix}$$

[21]I know of no proof that they can('t) but all my attempts at getting them to do so have failed so far. (Merrill 2019) proves that 1-layer CNNs with a tanh non-linearity are subregular, but it's unclear how that would extend to the simpler architecture in this paper.

$$\begin{aligned}\mathbf{P} &= [p_{h,t}] = \max_{\tau \leq t} C_{:,\tau} \\ \mathbf{Z} &= [\mathbf{W_z P}]^\top \\ \mathbf{Y} &= \sigma(\mathbf{Z})\end{aligned}$$

where max is the cumulative max function whose output $\mathbf{P} \in \mathbb{R}^{D \times T}$ is a matrix simulating a running state in its columns. $\mathbf{Z} \in \mathbb{R}^{T \times D}$ is the per-timestep logit and $\mathbf{Y} \in \mathbb{R}^{2 \times D}$ is the per-timestep output. Figure 31 shows $\mathbf{P}$ in the lower level and $\mathbf{Y}$ in the upper level for input 0101 to the OR-model. Similar to MLPs, the vertices in $\mathbf{P}$ space reflect the counts of 1s though it's not a homomorphism.[22] Zero counts of 1s are still in the negative decision region and greater counts are all on the positive decision region. Figure 32 shows the same probe but for the AND-CNN. Just as with OR, the vertices in $\mathbf{P}$ space reflect the counts of 0s, with zero counts in the positive decision region and one or more in the negative region. The $\mathbf{Y}$ space simulates the FSA by tracing its state-transition diagram.

4.3.3 TFMs

We start the study of TFMs' hidden states with the AND and OR tasks as they are simpler than the XOR construction. The TFMs used here are the encoder-TFMs with no attention mask (Phuong and Hutter 2022) and an initial token that plays the role of the CLS token traditionally used in BERT-like models, and which also servers a dummy input for simulating an initial state. The unbatched equations at inference time for a 1-layer 1-head model are:

$$\begin{aligned}\mathbf{T} &= \mathbf{W}_e\mathbf{X} \\ \mathbf{P} &= \mathbf{W}_p[1, 2, ...T] \\ \mathbf{E} &= \mathbf{T} + \mathbf{P} \\ \mathbf{L}_e &= LayerNorm(\mathbf{E}; \boldsymbol{\gamma}_e, \boldsymbol{\beta}_e) \\ \mathbf{A}_K &= \mathbf{K}\mathbf{L}_e \\ \mathbf{A}_Q &= \mathbf{Q}\mathbf{L}_e \\ \mathbf{A}_V &= \mathbf{V}\mathbf{L}_e \\ \mathbf{A} &= \sigma(\frac{\mathbf{A}_Q\mathbf{A}_K}{\sqrt{D}}) \\ \mathbf{A}_y &= \mathbf{A}\mathbf{A}_V \\ \mathbf{A}_{out} &= \mathbf{W}_{att}\mathbf{A}_y \\ \mathbf{A}_{res} &= \mathbf{E} + \mathbf{A}_{out}\end{aligned}$$

[22] I speculate this is due to the convolution operation, which, similarly to RNNs, aggregates embeddings over time steps (2 in this case, as that's the size of the kernel).

$$\begin{aligned}
\mathbf{L}_{att} &= LayerNorm(\mathbf{A}_{res}; \boldsymbol{\gamma}_{att}, \boldsymbol{\beta}_{att}) \\
\mathbf{FF}_{h} &= \max(0, \mathbf{W}_{z}\mathbf{L}_{att}) \\
\mathbf{FF}_{out} &= \mathbf{W}_{h}\mathbf{FF}_{h} \\
\mathbf{FF}_{res} &= \mathbf{L}_{att} + \mathbf{FF}_{out} \\
\mathbf{L} &= LayerNorm(\mathbf{FF}_{res}; \boldsymbol{\gamma}, \boldsymbol{\beta}) \\
\mathbf{f} &= \mathbf{L}_{1,:} \\
\mathbf{z} &= \mathbf{W}\mathbf{f} \\
\mathbf{y} &= \sigma(\mathbf{z})
\end{aligned}$$

In this architecture, the attention matrix $\mathbf{A} \in \mathbb{R}^{D\times D}$ and the attention-weighted matrix $\mathbf{A}_y \in \mathbb{R}^{D\times D}$ are the first and last points in the inference (forward) pass where the token embeddings interact, and it's therefore the obvious place to start probing for counting behaviour as in MLPs (section 4.3.1) and CNNs (section 4.3.2).

Table 1 shows the attention matrices for the input sequence 0101 to the OR-TFM (see appendix A.5 for the attention matrices for all sequences of length 4). It shows that all non-zero weights for the CLS token[23] are placed on the 1-tokens. This makes intuitive sense, as the presence of a 1 turns the classification decision from *True/1* to *False/0*. Table 2 shows the same attention matrices but with the weights accumulated over timesteps. These weights stand in a 1-to-1 correspondence with the number of 1s in the input up to that time index[24]. Next, we probe the $\mathbf{A}_y$ matrix by simulating a running state with the prefix-sum algorithm. Replacing it with:

$$\mathbf{A}_y^u = [a_{b,t,d}] = [\sum_{\tau\leq t'}(\mathbf{A}_{tt'} \circ_{t'} \mathbf{A}_{V\ t'd})_{t,\tau,d}]_{t,\tau,d\to\tau,t,d}$$

where the expression in brackets is the broadcast Hadamard product for each column in $\mathbf{A}_V$, the expression in square brackets is the cumulative sum over the timesteps and indices b, t, τ and d represent the batch, timestep, cumulative timestep and dimension axes respectively. The reason why we have to turn each unrolled sequence of tokens into a different batch is that the TFM head picks only the first token (CLS), which would leave us with a single token if we only kept all first tokens in a single sequence. As we did with the learned MLPs, the matrix/tensor multiplication is unrolled by breaking it down into a Hadamard product plus a cumulative sum. A further change needs to be done to the embedding matrix $\mathbf{E}$ before adding it to the attention out-projection to match the latter's new dimensions after unrolling:

$$\begin{aligned}
\mathbf{E}^u_{b=t,t,d} &= \mathbf{E}_{1,t,d} \\
\mathbf{A}^u_{res} &= \mathbf{E}^u + \mathbf{A}^u_{out}
\end{aligned}$$

Figure 33's lower level shows the result matches the probing results for the OR-MLP (and to a lesser extent the OR-CNN) showing that they count the number of 1s. Now, unlike with MLPs and CNNs, this alone is insufficient to simulate the FSA at the output layer. The reason is that the token embeddings

[23]A reminder that the BERT-like encoder-TFM's head only keeps the first token for downstream processing.

[24]The only exception is the attention matrix for the 0000 input where it counts the number of 0s (1s for 1111 to the AND-TFM). This is expected as attention weights always have to sum up to 1.

added in the attention-residual channel $\mathbf{A}_{res}$ have a different scale than the counts computed in $\mathbf{A}_y^u$[25], which upsets the downstream operations. The reweighted residual equation[26] is:

$$\mathbf{A}_{res}^u = 100\mathbf{E}^u + -.1\mathbf{A}_{out}^u$$

Figure 33's upper level shows the result, which is the correct FSA state-transition diagram.

For the AND-task, we follow exactly the same procedure. The results are depicted in Figures 3, 4 and 34.

TFMs cannot reliably learn the XOR function[27] (Hahn and Rofin 2024). Instead, we probe a construction by Chiang & Cholak (2022) which solves the task, albeit with increasing entropy as the length of the input increases. Chiang & Cholak do provide a workaround in the form of a modified normalisation layer, but here, for ease of exposition, I use a heuristic that is simpler and has a similar effect. The constructed XOR is an encoder-TFM that has 2 layers and 2 heads, without any layer normalisation, defined by the following unbatched equations for the embedding layer:

$$\begin{aligned}
\mathbf{T} &= \mathbf{W}_e\mathbf{X} \\
\mathbf{P} &= \mathbf{W}_p[1, 2, ...T] \\
\mathbf{E} &= \mathbf{T} + \mathbf{P}
\end{aligned}$$

The first block:

$$\begin{aligned}
\mathbf{A}_K^{1,1} &= \mathbf{K}^{1,1}\mathbf{E} \\
\mathbf{A}_Q^{1,1} &= \mathbf{Q}^{1,1}\mathbf{E} \\
\mathbf{A}_V^{1,1} &= \mathbf{V}^{1,1}\mathbf{E} \\
\mathbf{A}^{1,1} &= \sigma(\frac{\mathbf{A}_Q^{1,1}\mathbf{A}_K^{1,1}}{\sqrt{D}}) \\
\mathbf{A}_y^{1,1} &= \mathbf{A}^{1,1}\mathbf{A}_V^{1,1} \\
\mathbf{A}_K^{1,2} &= \mathbf{K}^{1,2}\mathbf{E}^1 \\
\mathbf{A}_Q^{1,2} &= \mathbf{Q}^{1,2}\mathbf{E}^1 \\
\mathbf{A}_V^{1,2} &= \mathbf{V}^{1,2}\mathbf{E}^1
\end{aligned}$$

[25] In experiments not shown here, I trained the same TFMs but with the attention residual channel ablated:
$\mathbf{A}_{res} = \mathbf{E} + \mathbf{A}_{out} \rightarrow \mathbf{A}_{res}^{ablated} = \mathbf{A}_{out}$.
With this modification, the unrolled $\mathbf{A}\mathbf{A}_V$ product is enough to simulate the FSA without the need for rescaling.

[26] This I found through trial and error. I plan on searching for a general formula in future work.

[27] In a similar fashion to MLPs, I did manage to get a TFM to learn XOR by replacing $\mathbf{f} = \mathbf{L}_{1,:}$ with $\mathbf{f} = \prod_{t=1}^{T} \mathbf{L}_{t,d}$. However, because this head does all the work, the TFM body itself ends up being an over-parameterised embedding layer (in further experiments, I ablated both the attention and the feedforward layers without negative impact in learning or inference ability). Additionally, it's unclear to what extent this could still be considered a TFM.

$$
\begin{aligned}
\mathbf{A}^{1,2} &= \sigma(\frac{\mathbf{A}_Q^{1,2}\mathbf{A}_{\mathbf{K}}^{1,2}}{\sqrt{D}}) \\
\mathbf{A}_y^1 &= [\mathbf{A}^{1,1}\mathbf{A}_{\mathbf{V}}^{1,1}|\mathbf{A}^{1,2}\mathbf{A}_{\mathbf{V}}^{1,2}] \\
\mathbf{A}_{out}^1 &= \mathbf{W}_{att}^1\mathbf{A}_y^1 \\
\mathbf{A}_{res}^1 &= \mathbf{E} + \mathbf{A}_{out}^1 \\
\mathbf{FF}_h^1 &= \max(0, \mathbf{W}_z^1\mathbf{A}_{res}^1) \\
\mathbf{FF}_{out}^1 &= \mathbf{W}_h^1\mathbf{FF}_h^1 \\
\mathbf{FF}_{res}^1 &= \mathbf{A}_{res}^1 + \mathbf{FF}_{out}^1
\end{aligned}
$$

The second block:

$$
\begin{aligned}
\mathbf{A}_K^{2,1} &= \mathbf{K}^{2,1}\mathbf{FF}_{res}^1 \\
\mathbf{A}_Q^{2,1} &= \mathbf{Q}^{2,1}\mathbf{FF}_{res}^1 \\
\mathbf{A}_V^{2,1} &= \mathbf{V}^{2,1}\mathbf{FF}_{res}^1 \\
\mathbf{A}^{2,1} &= \sigma(\frac{\mathbf{A}_{\mathbf{Q}}^{2,1}\mathbf{A}_{\mathbf{K}}^{2,1}}{\sqrt{D}}) \\
\mathbf{A}_y^{2,1} &= \mathbf{A}^{2,1}\mathbf{A}_V^{2,1} \\
\mathbf{A}_K^{2,2} &= \mathbf{K}^{2,2}\mathbf{FF}_{res}^1 \\
\mathbf{A}_Q^{2,2} &= \mathbf{Q}^{2,2}\mathbf{FF}_{res}^1 \\
\mathbf{A}_V^{2,2} &= \mathbf{V}^{2,2}\mathbf{FF}_{res}^1 \\
\mathbf{A}^{2,2} &= \sigma(\frac{\mathbf{A}_Q^{2,2}\mathbf{A}_K^{2,2}}{\sqrt{D}}) \\
\mathbf{A}_y^2 &= [\mathbf{A}^{2,1}\mathbf{A}_V^{2,1}|\mathbf{A}^{2,2}\mathbf{A}_V^{2,2}] \\
\mathbf{A}_{out}^2 &= \mathbf{W}_{att}^2\mathbf{A}_y^2 \\
\mathbf{A}_{res}^2 &= \mathbf{FF}_{res}^1 + \mathbf{A}_{out}^2 \\
\mathbf{FF}_h^2 &= \max(0, \mathbf{W}_z^2\mathbf{A}_{res}^2) \\
\mathbf{FF}_{out}^2 &= \mathbf{W}_h^2\mathbf{FF}_h^2 \\
\mathbf{FF}_{res}^2 &= \mathbf{A}_{res}^2 + \mathbf{FF}_{out}^2
\end{aligned}
$$

And the head:

$$
\begin{aligned}
\mathbf{f} &= \mathbf{FF}_{res\ 1,:}^2 \\
\mathbf{z} &= \mathbf{Wf} \\
\mathbf{y} &= \sigma(\mathbf{z})
\end{aligned}
$$

The approach to probing is the same as with the learned AND- and OR-TFMs: unrolling the product of the attention matrix and the value matrix. In fact, because in the XOR construction the embeddings are added to a different set of dimensions from those containing the effective weights in the weighted value matrix, they don't interfere with the prefix-sum. Thus, reweighting is unnecessary. The modified first layer's attention-weighted value matrix $\mathbf{A}_y^1$ is then:

$$\mathbf{A}_y^u = [a_{b,t,d}] = [\sum_{\tau \leq t'} (\mathbf{A}_{htt'} \circ_{t'} \mathbf{A}_{V\ ht'g})_{h,t_{1:T \to T:1},\tau,g}]_{(h,t_{flip},\tau,g) \to (\tau,t_{flip},hg)}$$

Where h, t_{flip}, τ, g and d are the head, upside-down timestep[28], cumulative timestep, attention dimension and embedding dimension axes respectively.

Figures 35, 36, 37 show $\mathbf{A}_y^{u\ 1}$ space in the lower level and $\mathbf{Y}$ space in the upper level, for inputs 01, 0101 and 01010101. It is apparent that the lower level represents the number of 1s and upper level traces the FSA. Also, as described in Chiang & Cholak (2022), the distance between *True/1* and *False/0*, corresponding to the inverse of the entropy, becomes smaller as the sequence length increases. This increase in entropy can be countered by increasing the magnitude of the multiplier in the logits $\mathbf{z}$ layer from the original value of 1 to, for instance, 10^6. This constant will move the positive and negative logit values further apart the larger it gets, compensating for the increasing entropy of the attention weights.

Table 1: OR-TFM's attention matrix for input 1010 plus the initial dummy/CLS token. Each row contains the attention weights for each token rounded to 2 decimals. 0 weights are left out for clarity.

Tokens	CLS	0	1	0	1
CLS			.50		.50
0			.50		.50
1	.33	.33		.33	
0			.50		.50
1	.33	.33		.33	

Table 2: OR-TFM's cumulative attention matrix for input 1010 plus the initial dummy/CLS token. The first row shows the cumulative number of 1s. 0 weights are left out for clarity.

Tokens	CLS	0	1	0	1
#1s	(0)	(0)	(1)	(1)	(2)
CLS			.50	.50	1.0
0			.50	.50	1.0
1	.33	.67	.67	1.0	1.0
0			.50	.50	1.0
1	.33	.67	.67	1.0	1.0

[28]This is correct for the fact that the Chiang & Cholak construction puts the CLS token at the end of the sequence instead of at the beginning.

Table 3: AND-TFM's attention matrix for input 1010 plus the initial dummy/CLS token. Each row contains the attention weights for each token rounded to 2 decimals. 0 weights are left out for clarity.

Tokens	CLS	0	1	0	1
CLS		.50		.50	
0	.33		.33		.33
1		.50		.50	
0	.33		.33		.33
1		.50		.50	

Table 4: AND-TFM's cumulative attention matrix for input 1010 plus the initial dummy/ CLS token. The first row shows the cumulative number of 0s. 0 weights are left out for clarity.

Tokens	CLS	0	1	0	1
#0s	(0)	(1)	(1)	(2)	(2)
CLS		.50	.50	1.0	1.0
0	.33	.33	.67	.67	1.0
1		.50	.50	1.0	1.0
0	.33	.33	.67	.67	1.0
1		.50	.50	1.0	1.0

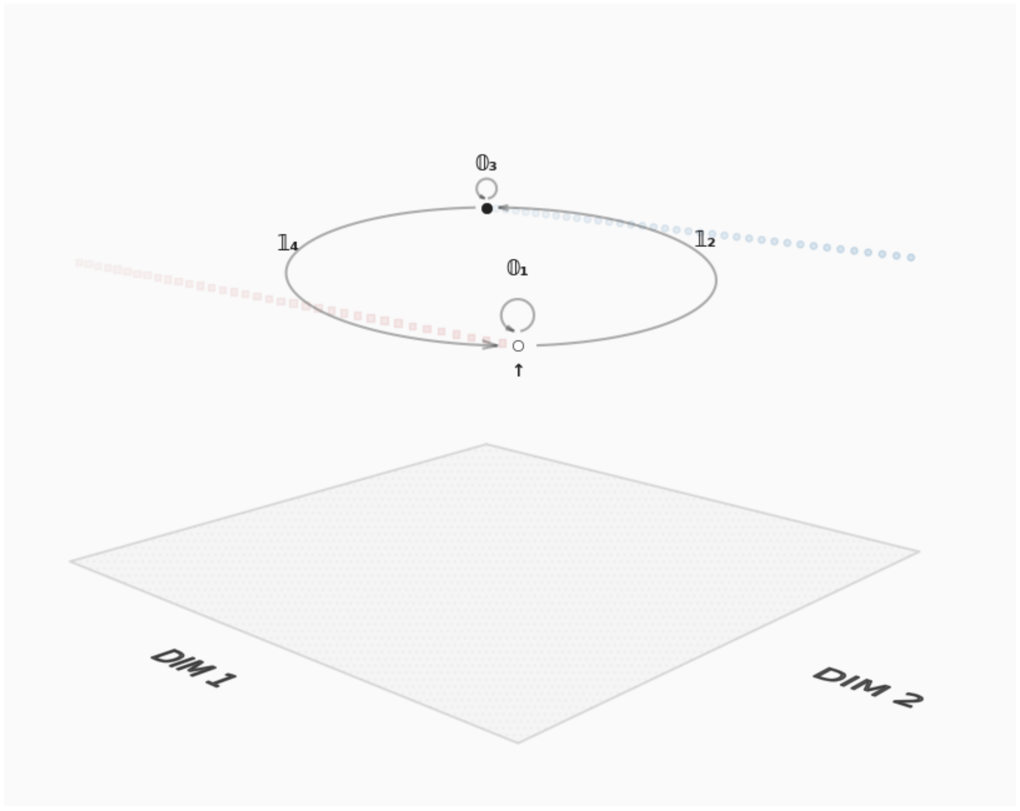


Figure 4: Visualisation of hidden space for an SL-trained WFSA on the XOR task for the 0101 input. Vertices represent weight vectors, coloured according to the value of their output (in $[0, 1]$), with darker colours representing higher values. Edges are drawn between consecutive vertices, labelled with the corresponding input tokens, subscripted by timesteps. The small up-arrow under a vertex indicates it's the initial vector. Red squares represent probing points that result in outputs close to zero, within 10% of the maximum possible distance between $[0, 0]$ and $[1, 1]$, the extreme points of the 2-dimensional output space. Blue circles represent probing points close to one within the same distance threshold.

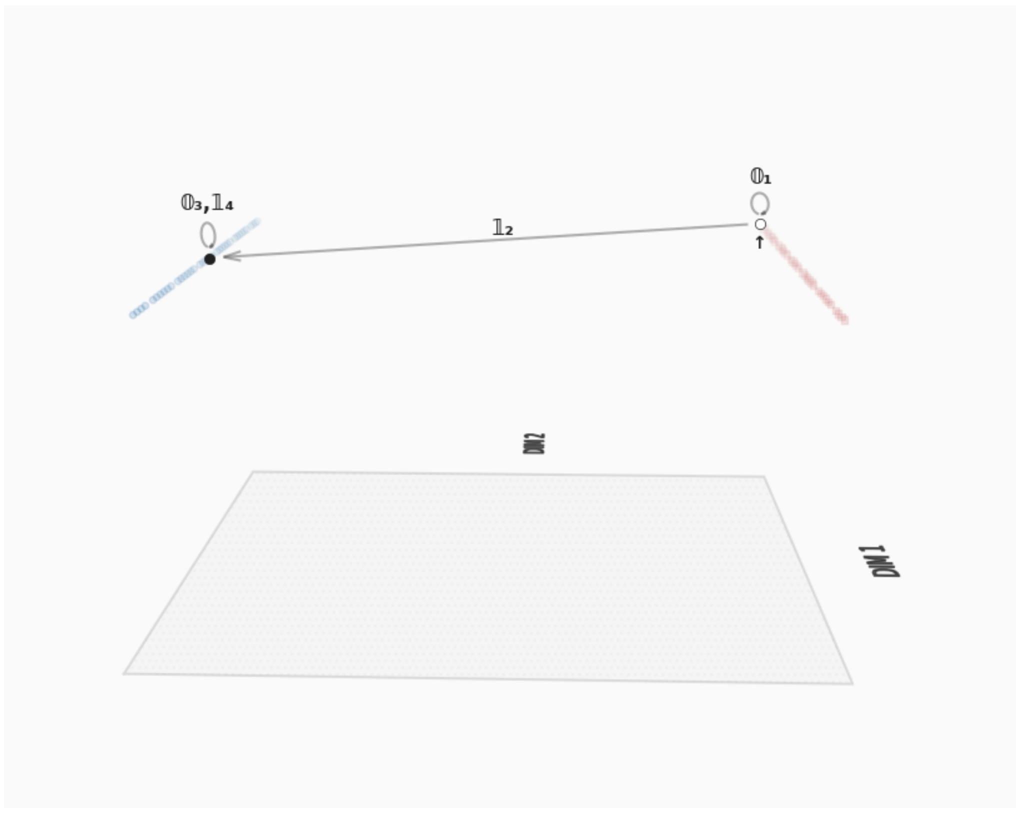


Figure 5: Visualisation of hidden space for an SL-trained WFSA on the OR task for the 0101 input. Conventions as per Figure 4.

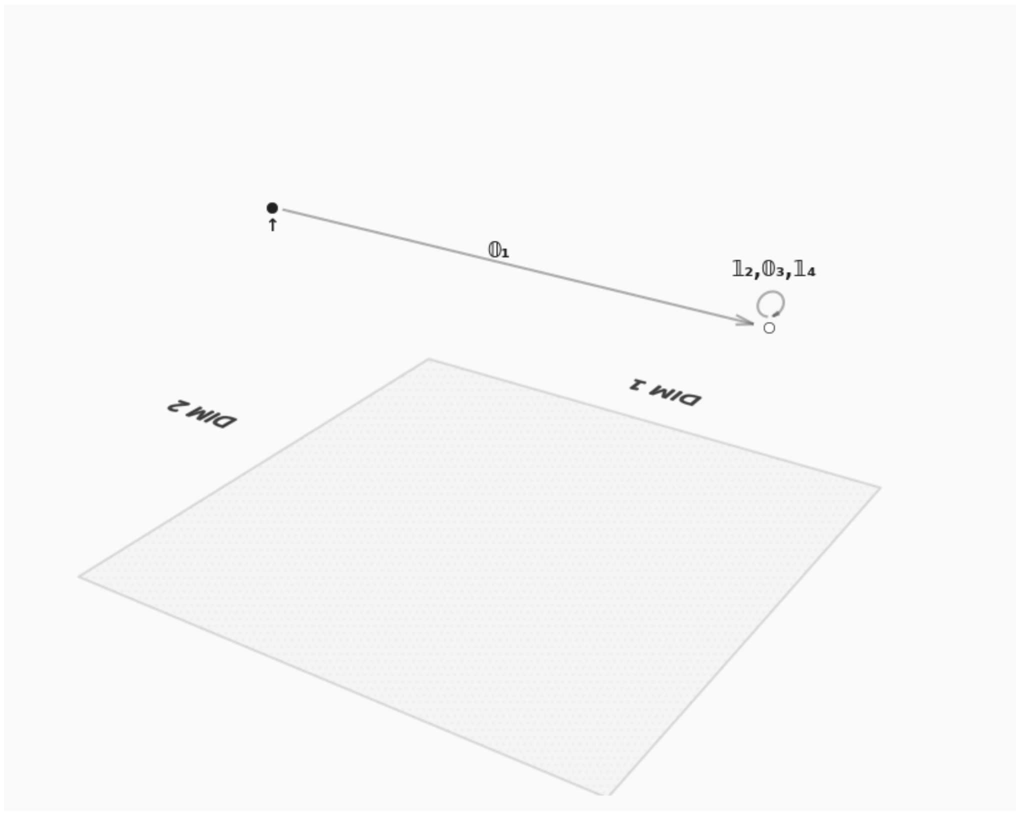


Figure 6: Visualisation of hidden space for an SL-trained WFSA on the AND task for the 0101 input. Conventions as per Figure 4.

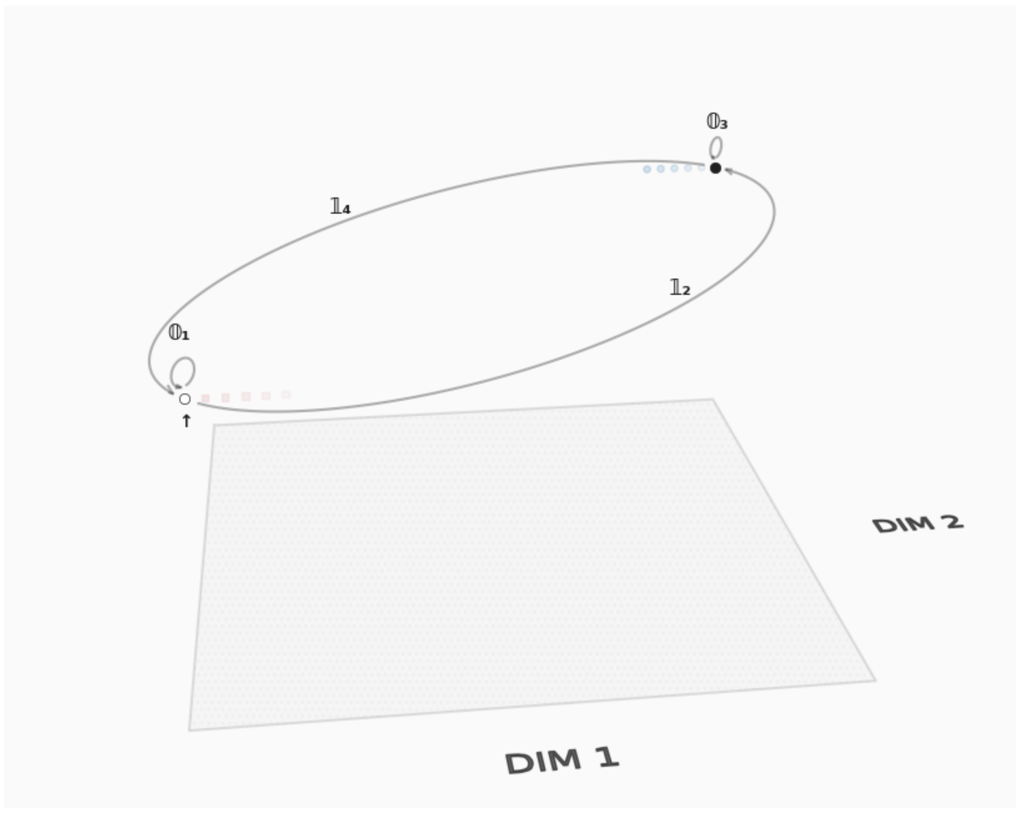


Figure 7: Visualisation of hidden space for an SGD-trained WFSA on the XOR task for the 0101 input. Conventions as per Figure 4.

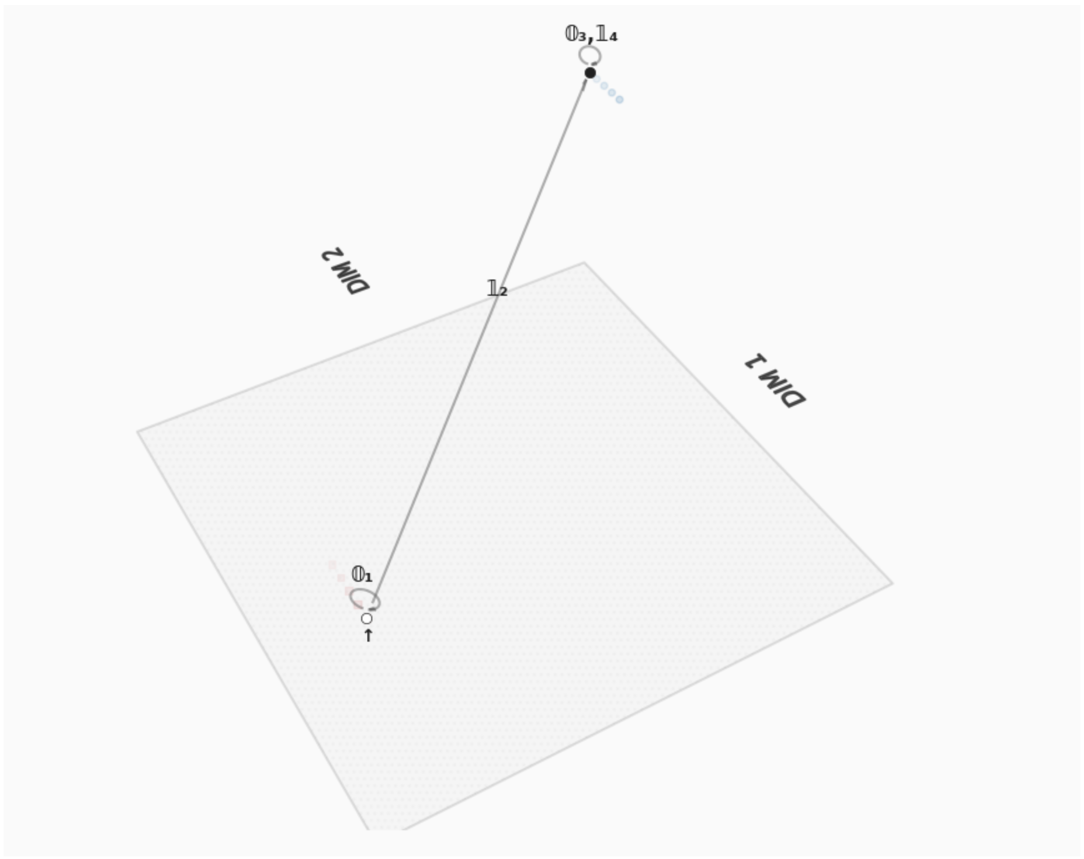


Figure 8: Visualisation of hidden space for an SGD-trained WFSA on the OR task for the 0101 input. Conventions as per Figure 4.

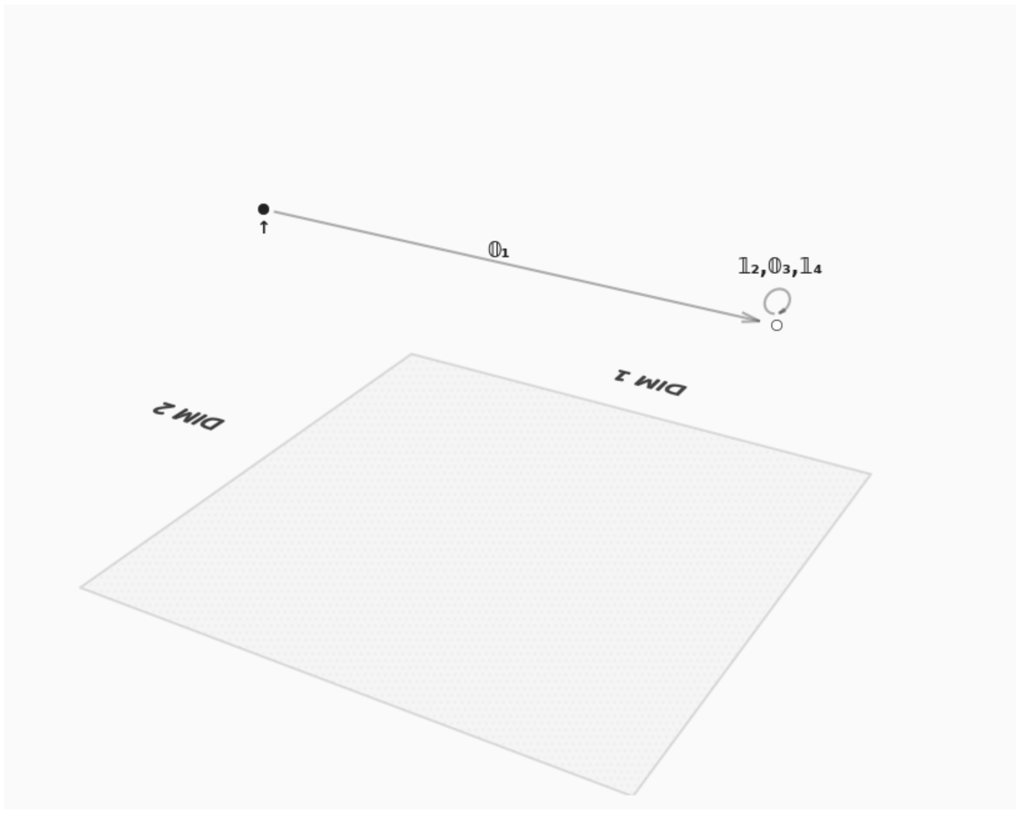


Figure 9: Visualisation of hidden space for an SGD-trained WFSA on the AND task for the 0101 input. Conventions as per Figure 4.

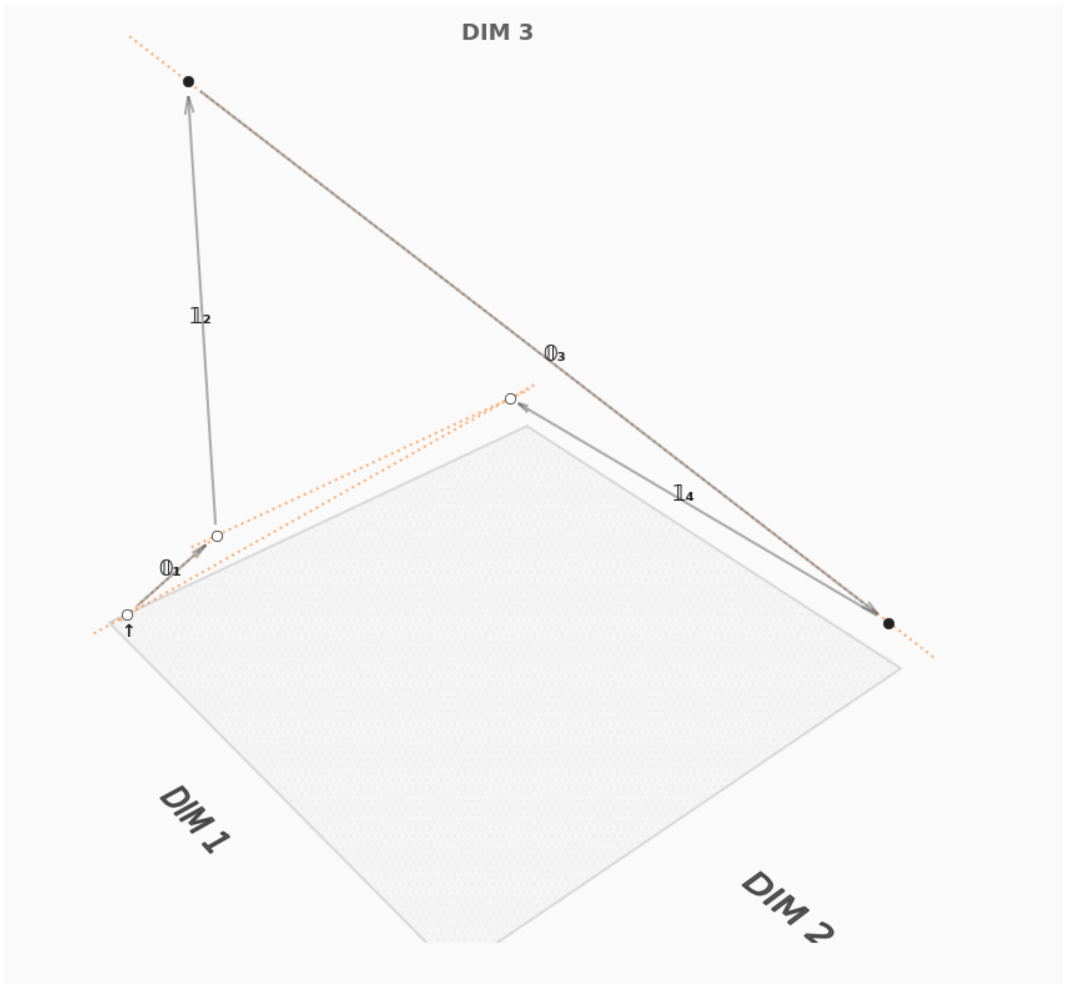


Figure 10: Softmax-output SRNN's hidden space for 0101 input. Vertices represent hidden layer vectors, coloured according to the value of their output (in $[0, 1]$), with darker colours representing higher values. Edges are drawn between consecutive vertices, labelled with the corresponding input tokens, subscripted by timesteps. The small up-arrow under a vertex indicates it's the initial vector. For 2-dimensional hidden spaces, red squares represent probing points that result in outputs close to zero, within 5% of the maximum possible distance between $[0, 0]$ and $[1, 1]$, the extreme points of the 2-dimensional output space. Blue circles represent probing points close to one within the same distance threshold. For 3-dimensional hidden spaces, the dotted orange lines join vertices lying in lines where every point shares the same output value (isolines). This is done as an alternative to probing points to avoid visual clutter.

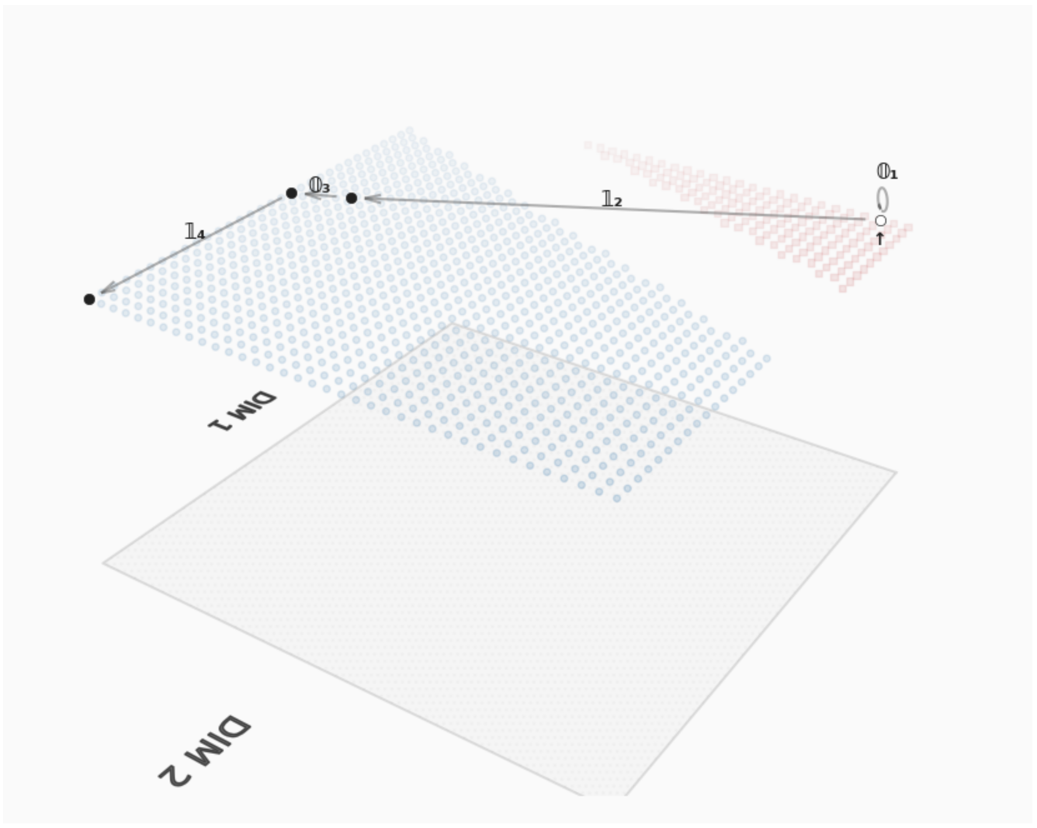


Figure 11: Visualisation of hidden space for an SRNN on the OR task for the 0101 input. Conventions as per Figure 10.

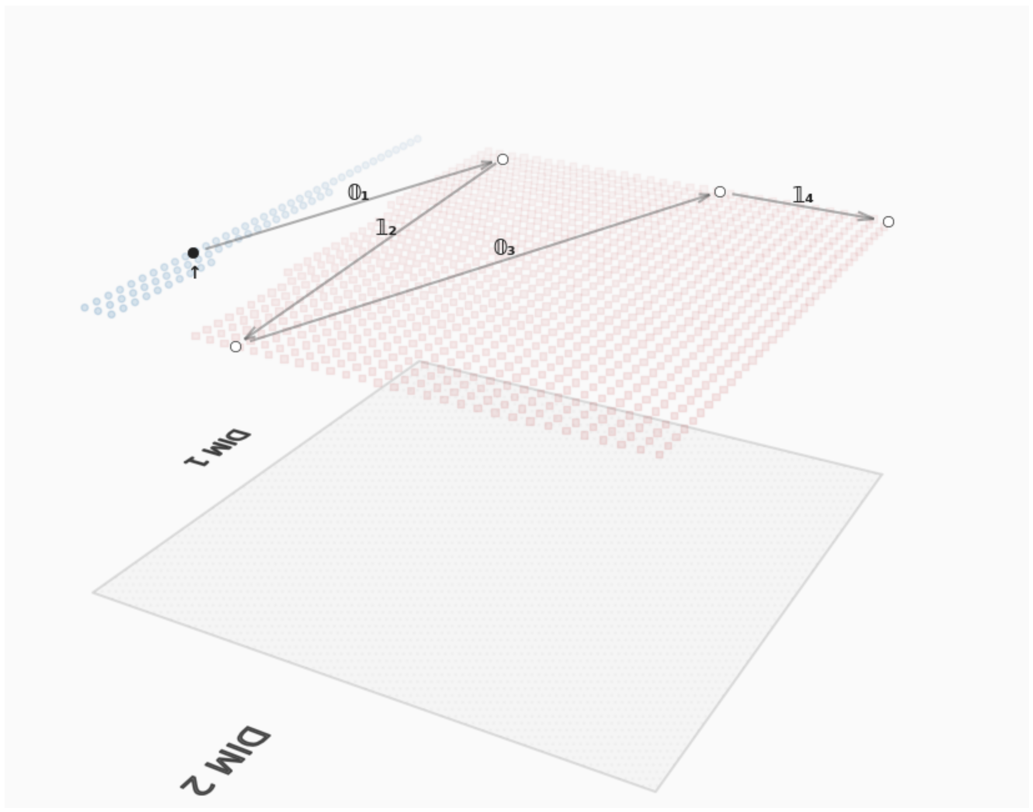


Figure 12: Visualisation of hidden space for an SRNN on the AND task for the 0101 input. Conventions as per Figure 10.

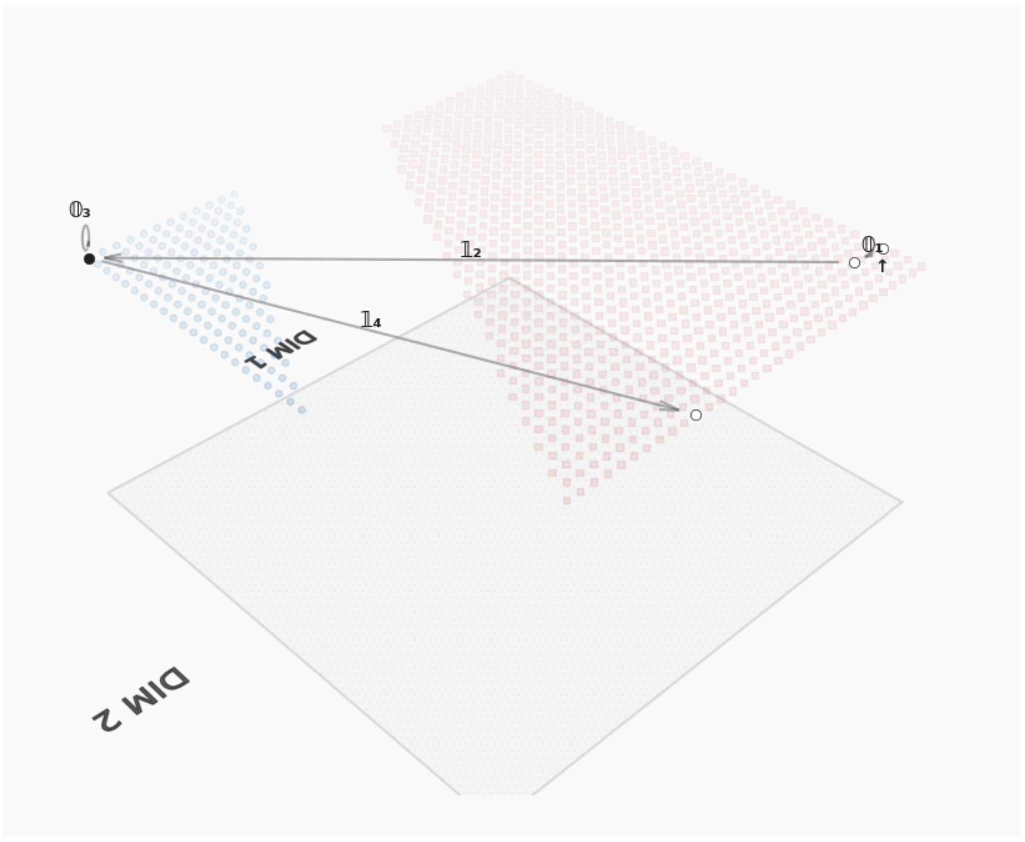


Figure 13: Visualisation of hidden space for an GRU on the XOR task for the 0101 input. Conventions as per Figure 10.

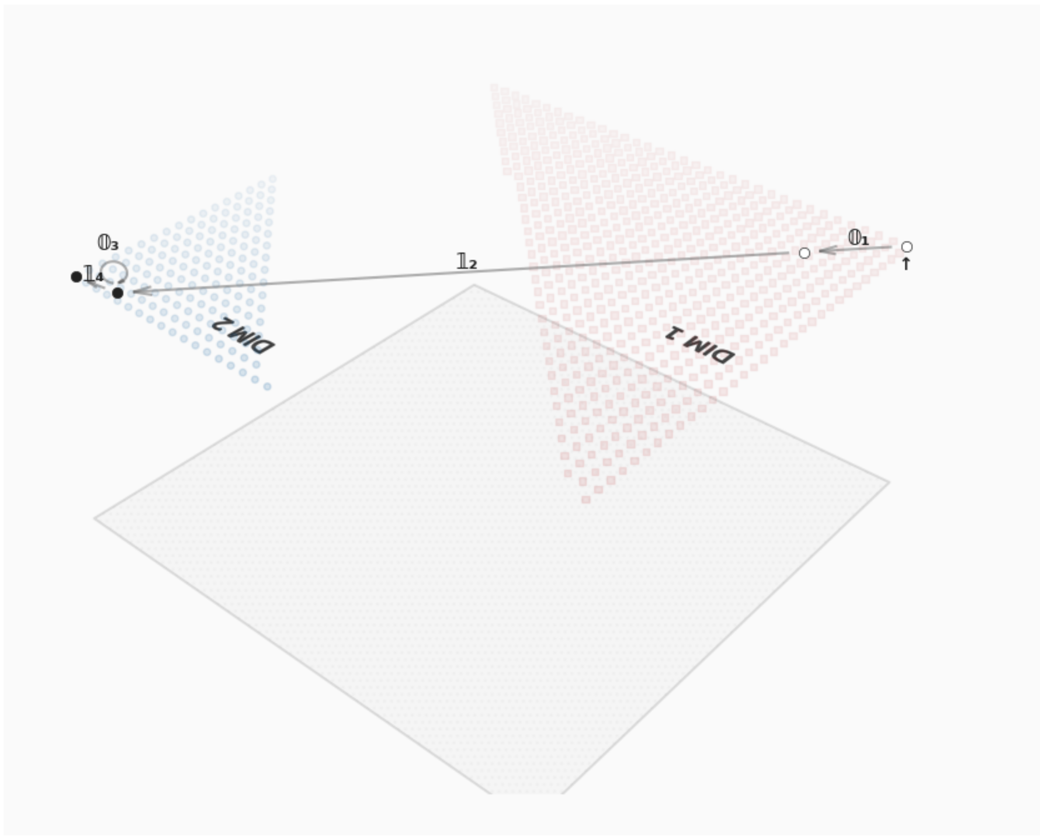


Figure 14: Visualisation of hidden space for an GRU on the OR task for the 0101 input. Conventions as per Figure 10.

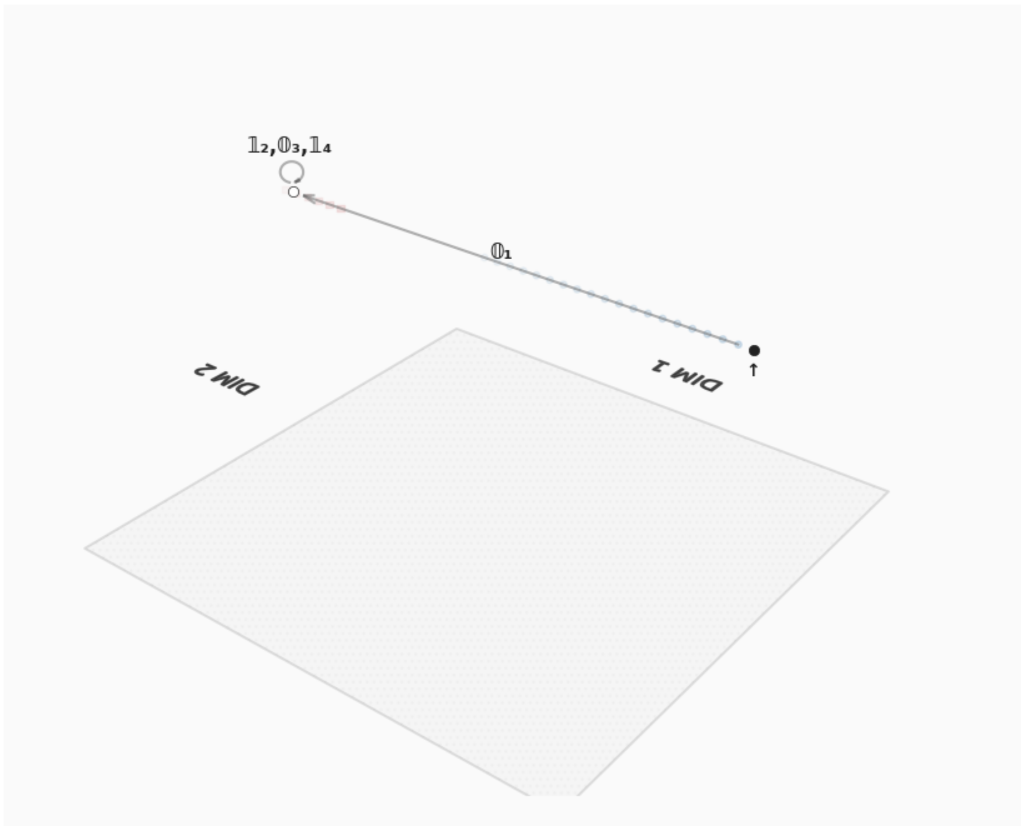


Figure 15: Visualisation of hidden space for an GRU on the AND task for the 0101 input. Conventions as per Figure 10.

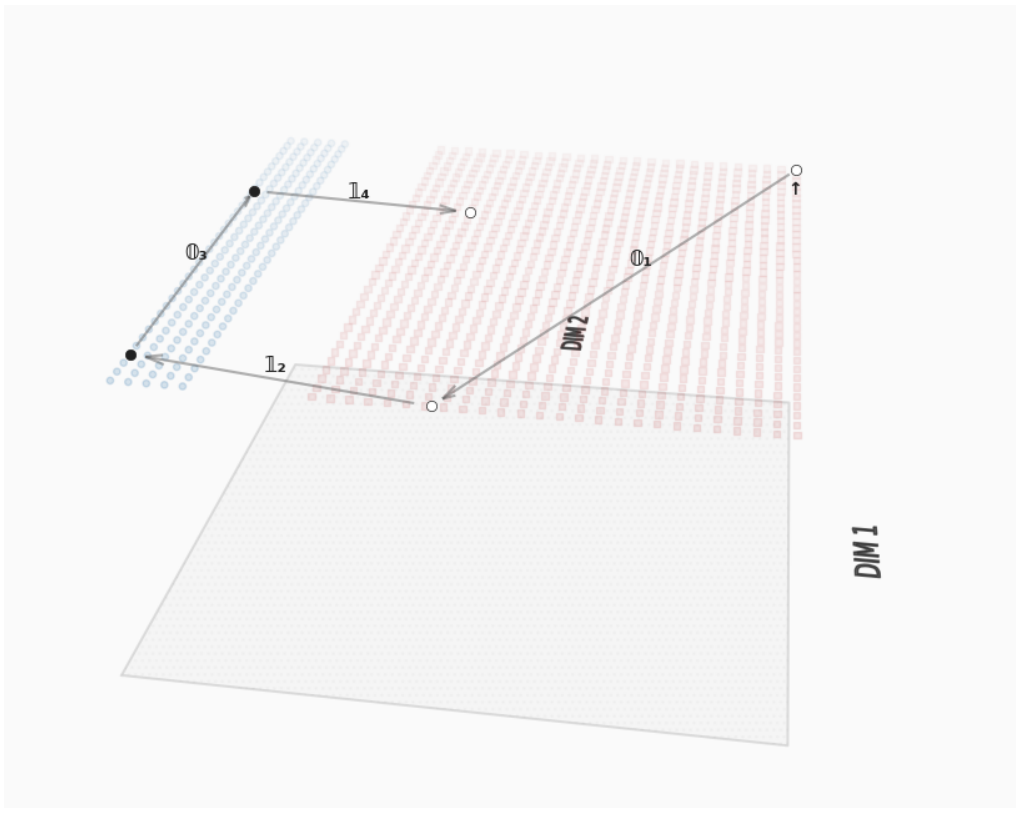


Figure 16: Visualisation of hidden space for an LSTM on the XOR task for the 0101 input. Conventions as per Figure 10.

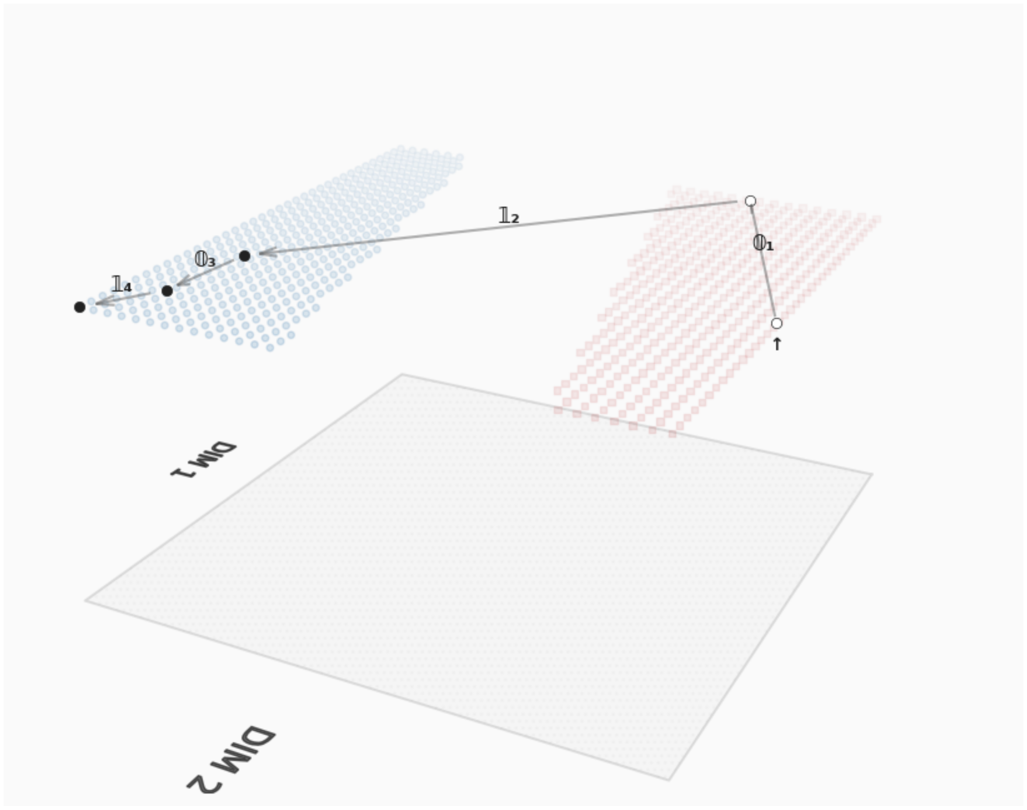


Figure 17: Visualisation of hidden space for an LSTM on the OR task for the 0101 input. Conventions as per Figure 10.

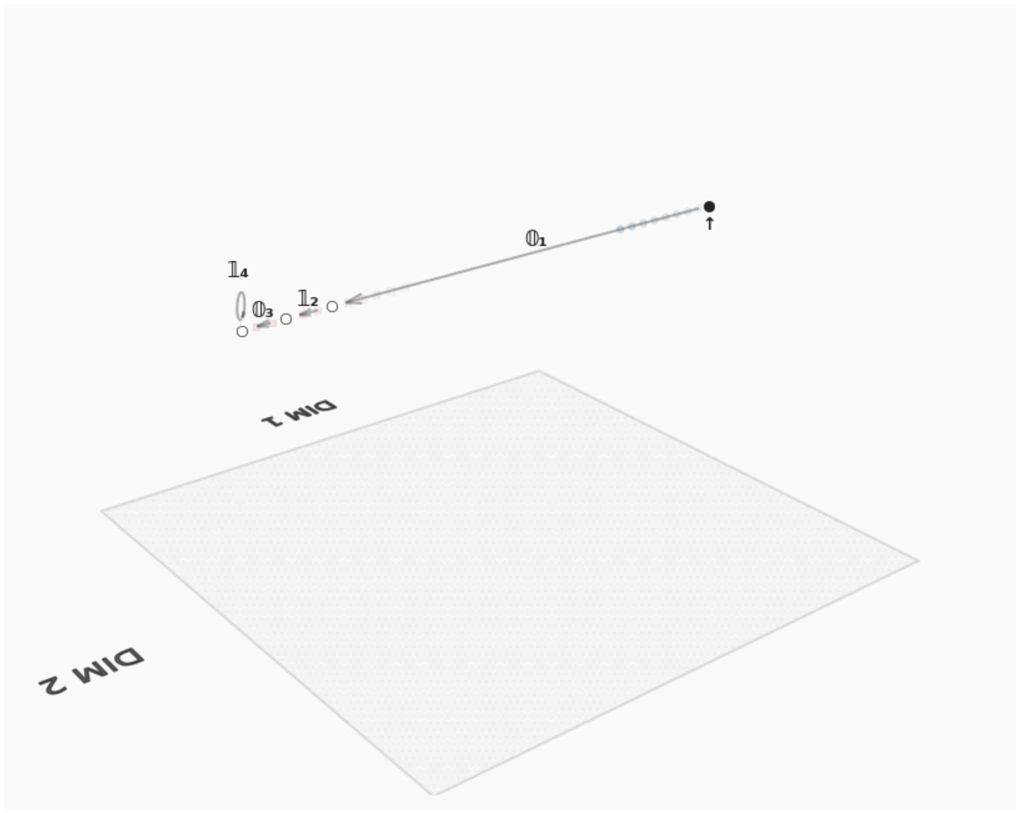


Figure 18: Visualisation of hidden space for an LSTM on the AND task for the 0101 input. Conventions as per Figure 10.

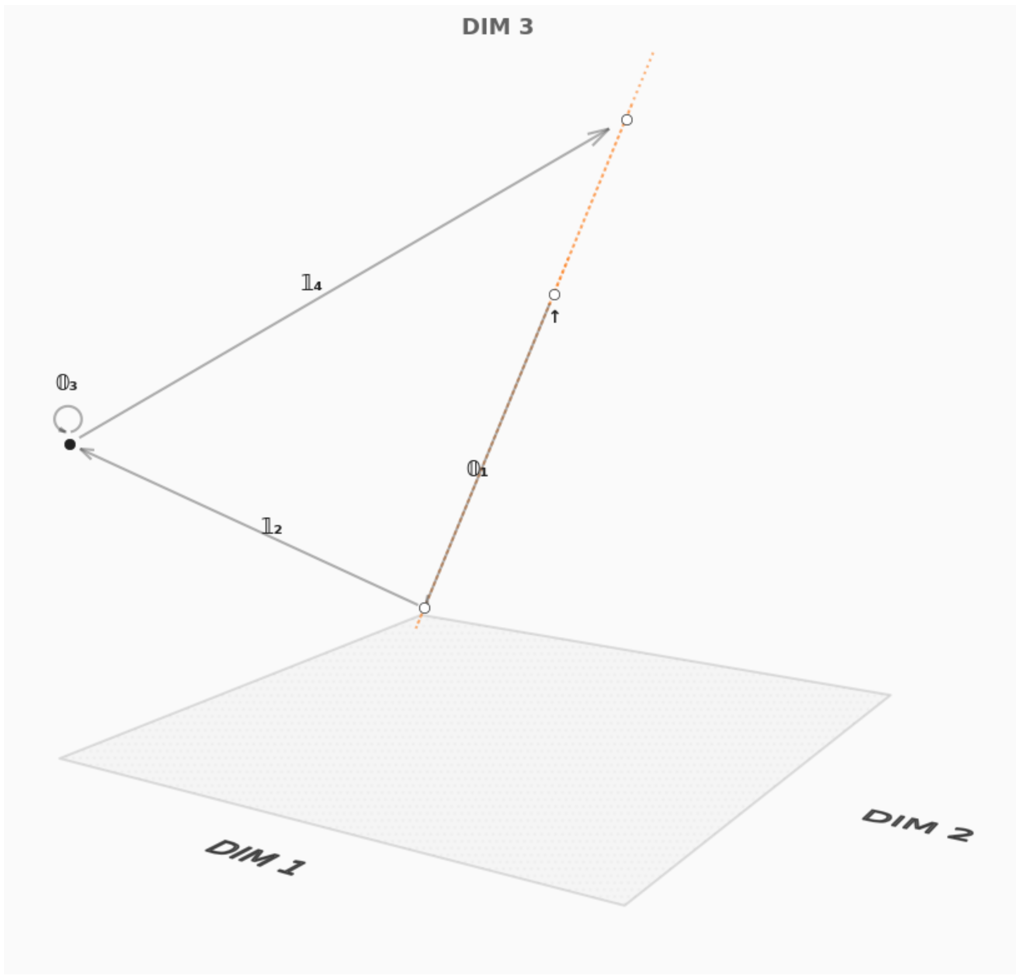


Figure 19: Visualisation of hidden space for a linear-output SRNN on the XOR task for the 0101 input. Conventions as per Figure 10.

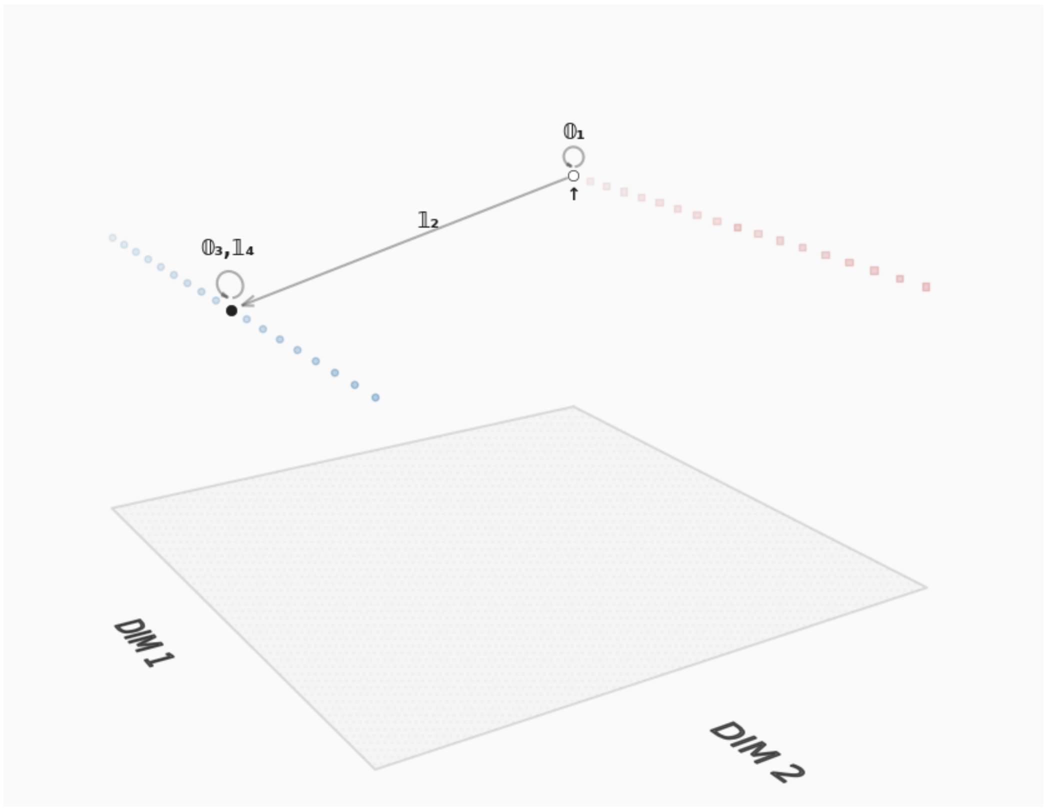


Figure 20: Visualisation of hidden space for a linear-output SRNN on the OR task for the 0101 input. Conventions as per Figure 10.

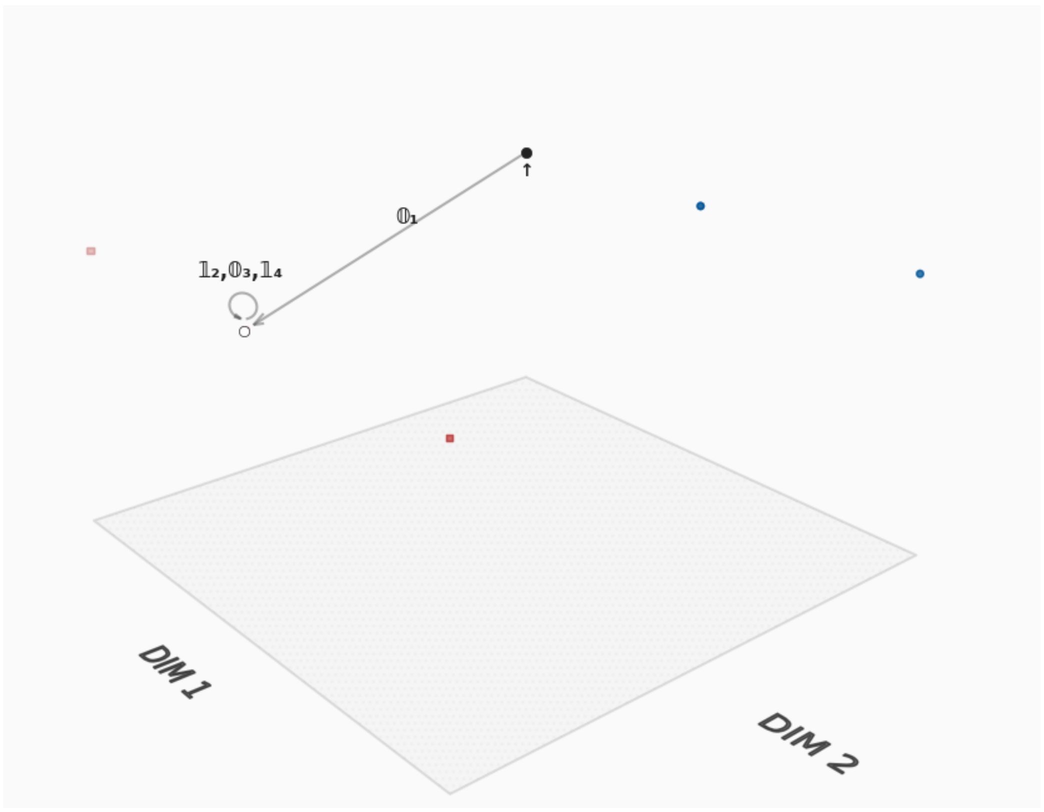


Figure 21: Visualisation of hidden space for a linear-output SRNN on the AND task for the 0101 input. Conventions as per Figure 10.

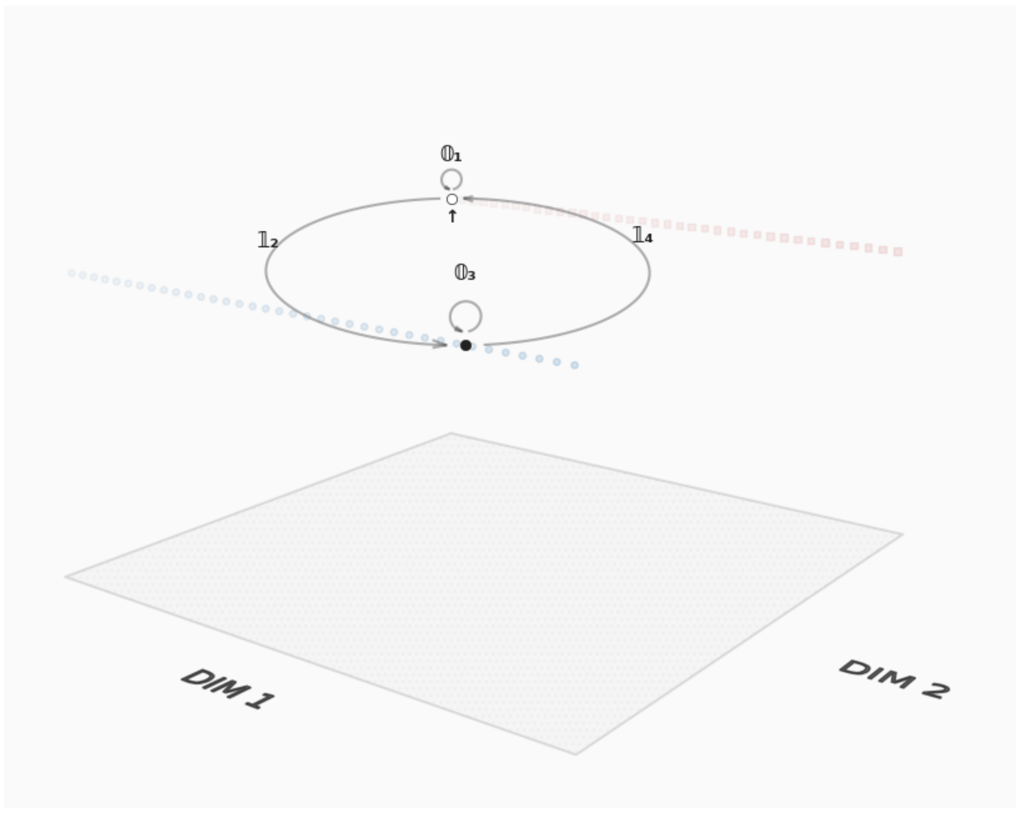


Figure 22: Visualisation of hidden space for a linear-output GRU on the XOR task for the 0101 input. Conventions as per Figure 10.

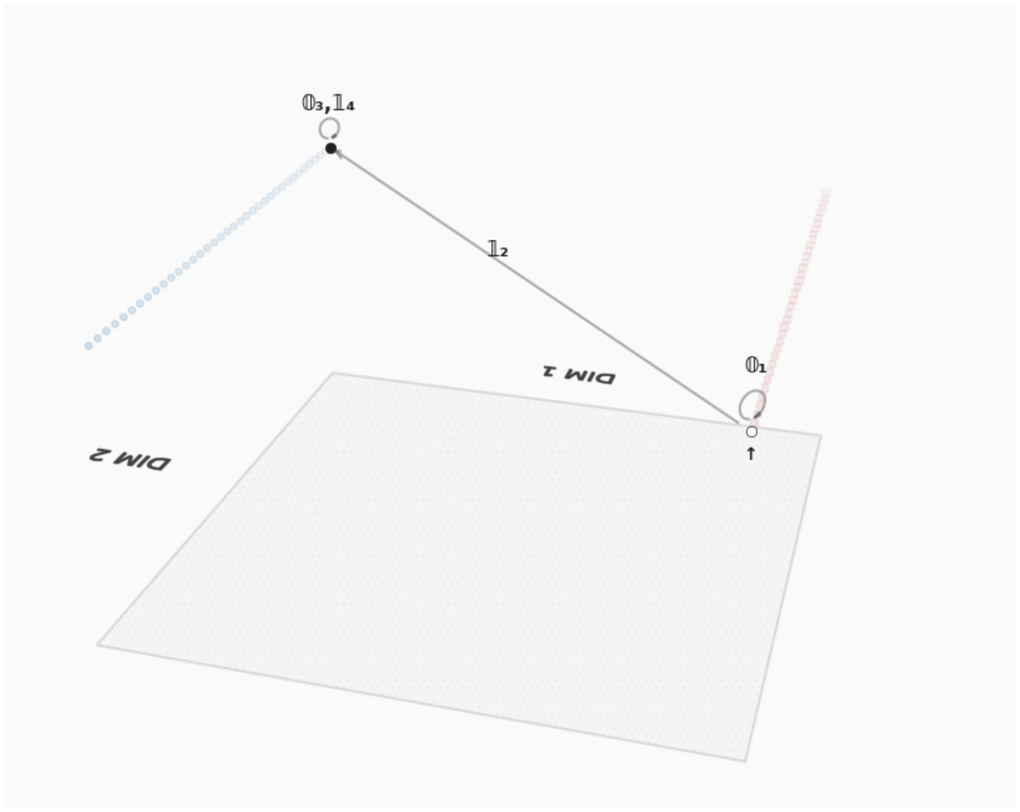


Figure 23: Visualisation of hidden space for a linear-output GRU on the OR task for the 0101 input. Conventions as per Figure 10.

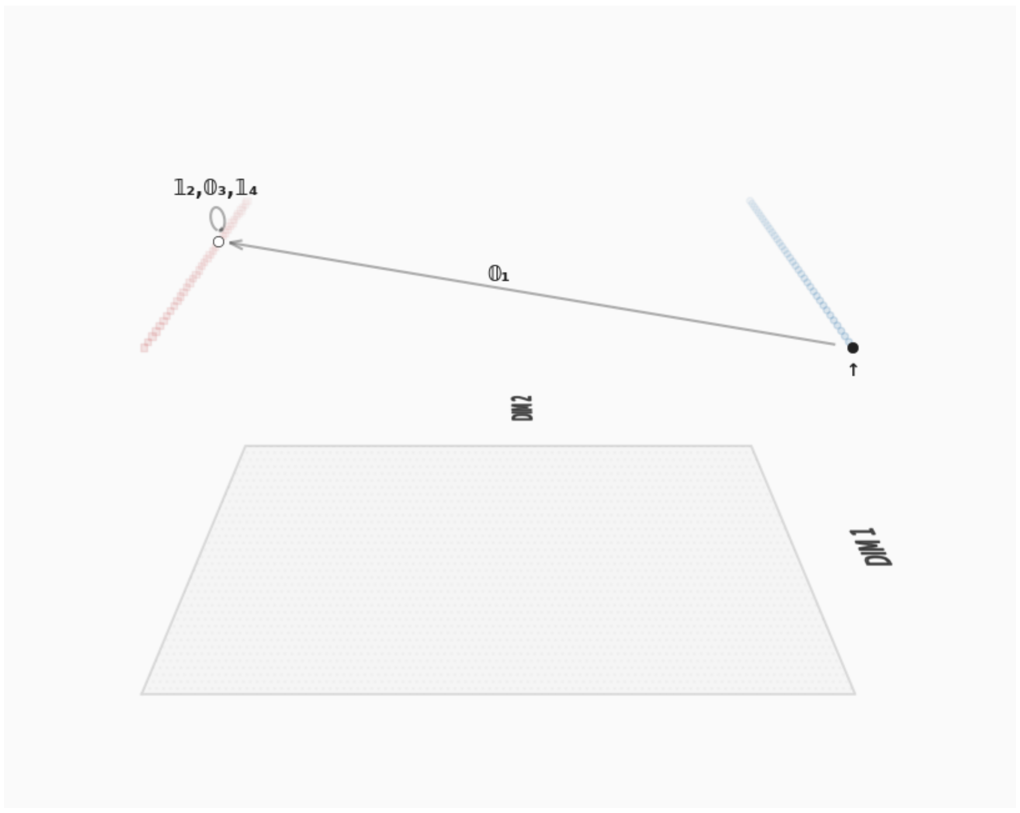


Figure 24: Visualisation of hidden space for a linear-output GRU on the AND task for the 0101 input. Conventions as per Figure 10.

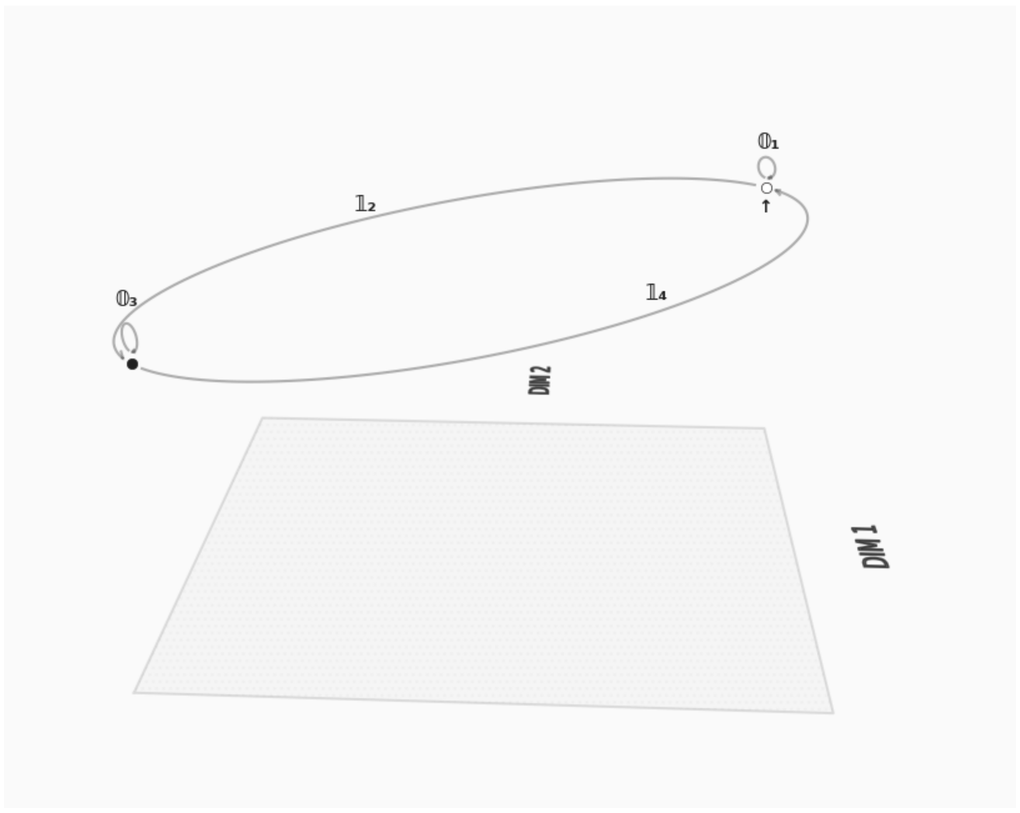


Figure 25: Visualisation of hidden space for an linear-output LSTM on the XOR task for the 0101 input. Conventions as per Figure 10.

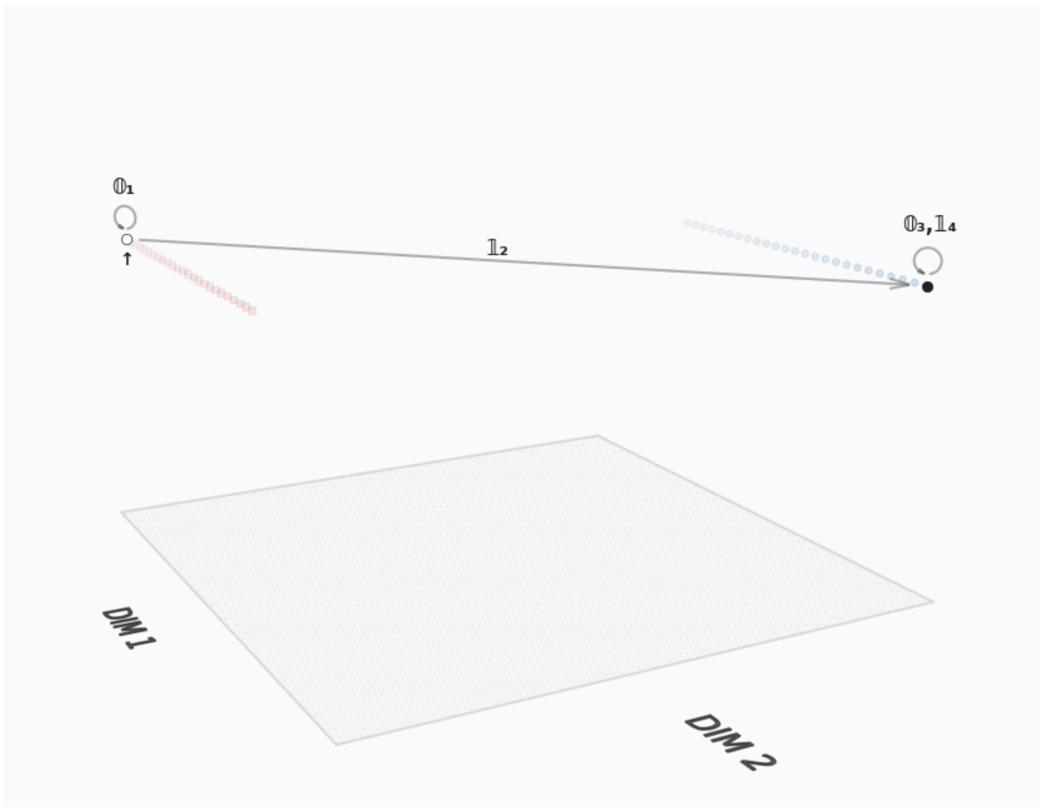


Figure 26: Visualisation of hidden space for an linear-output LSTM on the OR task for the 0101 input. Conventions as per Figure 10.

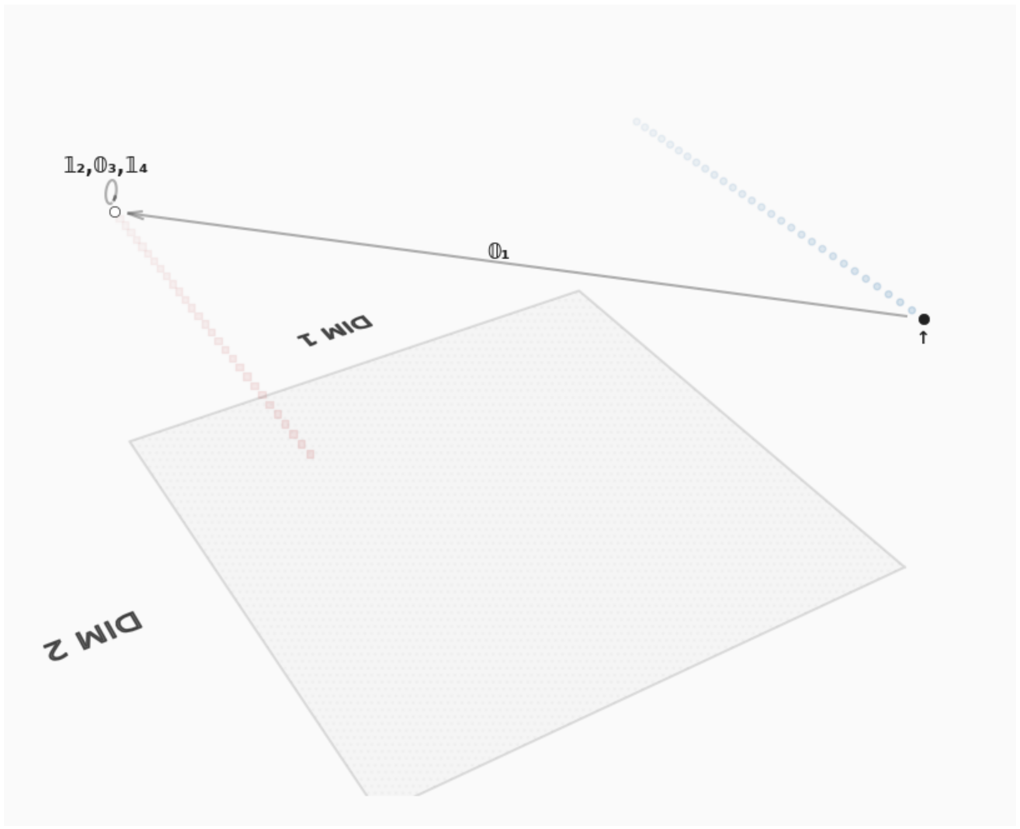


Figure 27: Visualisation of hidden space for an linear-output LSTM on the AND task for the 0101 input. Conventions as per Figure 10.

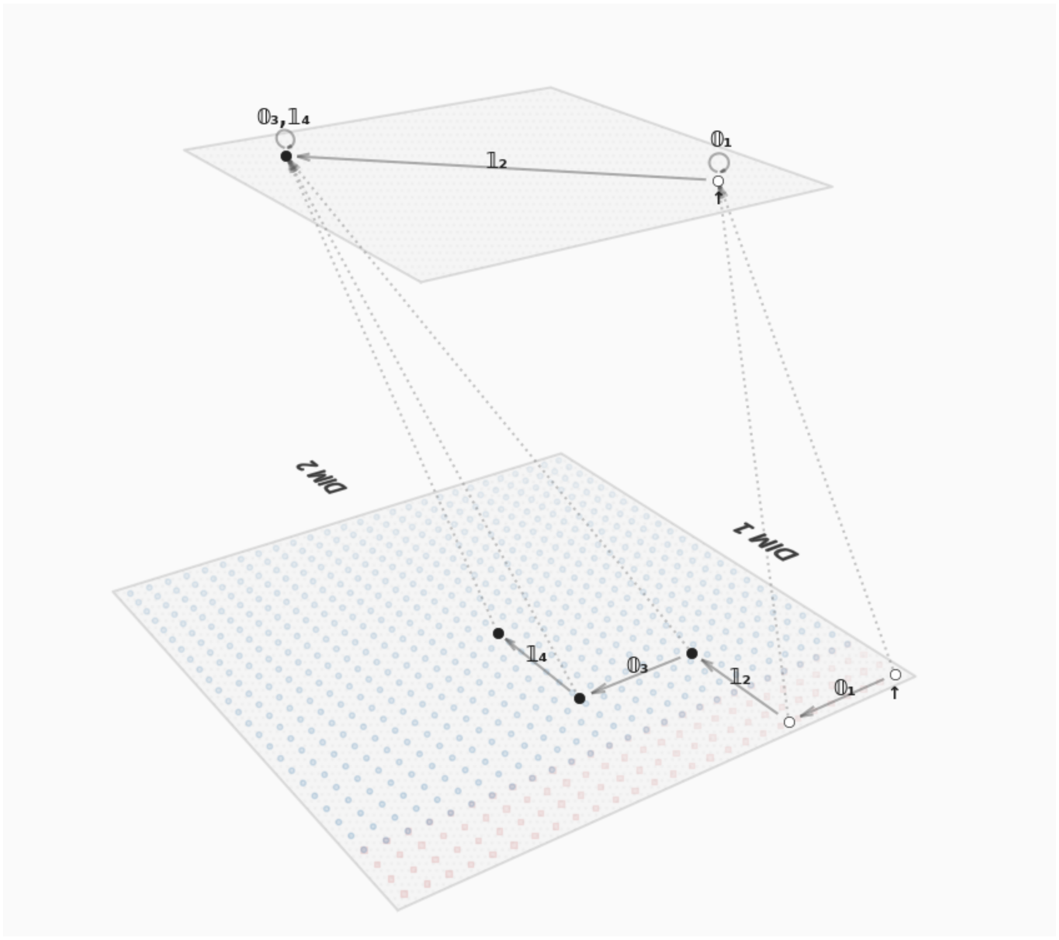


Figure 28: Hidden (lower plane) and output (upper plane) spaces for input 0101 to the OR-MLP. Vertices represent hidden and output layer vectors, coloured according to the value of their output (in $[0, 1]$), with darker colours representing higher values. Edges are drawn between consecutive vertices, labelled with the corresponding input tokens, subscripted by timesteps. The small up-arrow under a vertex indicates it's the dummy initial vector. Red squares represent probing points that result in outputs close to zero, within 1% of the maximum possible distance between $[0, 0]$ and $[1, 1]$, the extreme points of the 2-dimensional output space. Blue circles represent probing points close to one within the same distance threshold. Light-grey dotted lines join the hidden vectors (counts) to the corresponding output vectors (FSA states).

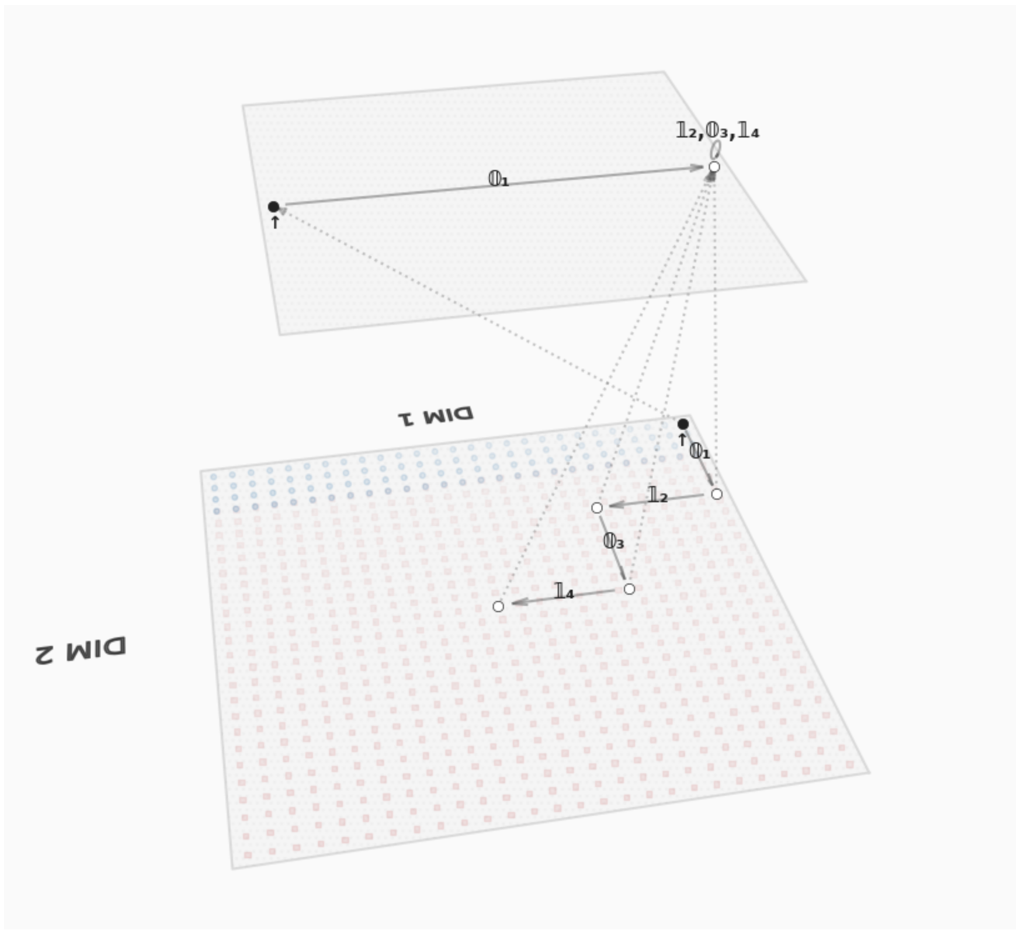


Figure 29: Hidden (lower plane) and output (upper plane) spaces for input 0101 to the AND-MLP. Visualisation Conventions as per Figure 28.

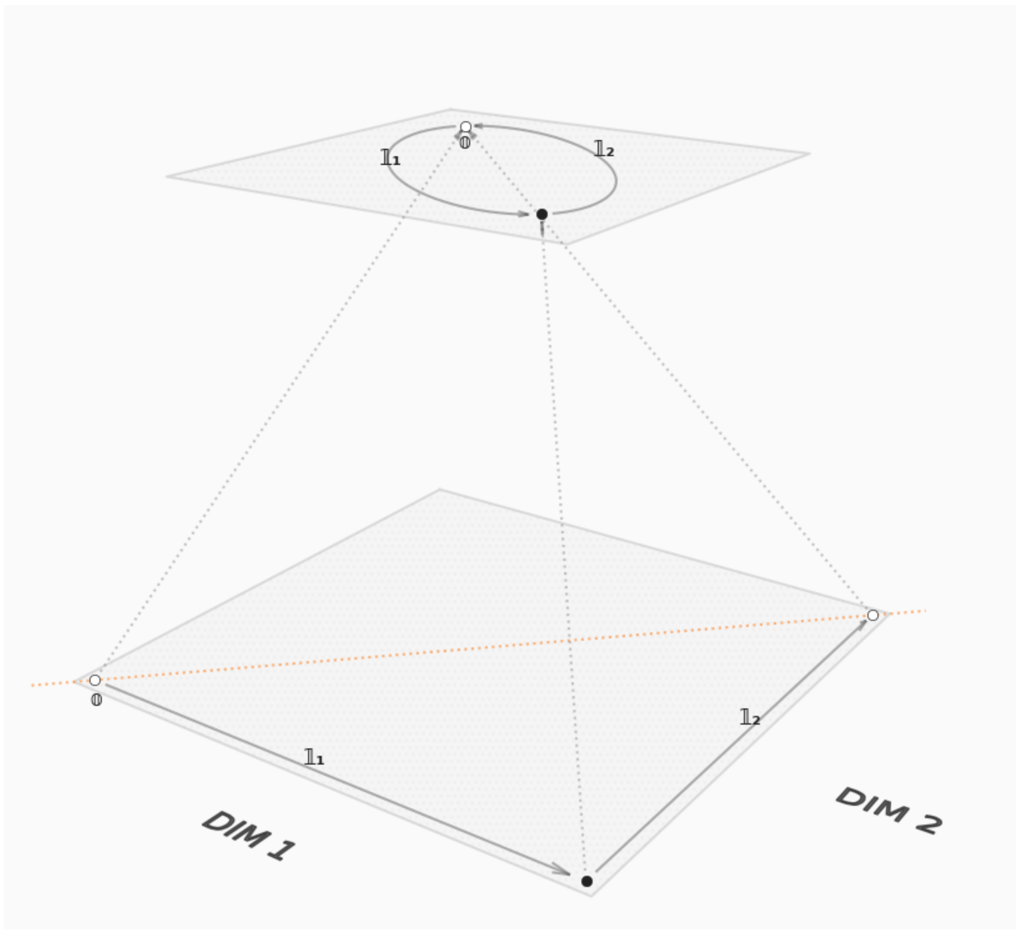


Figure 30: Hidden (lower plane) and output (upper plane) spaces for input 101 the XOR-MLP. Conventions similar to Figure 28 except the structure of the hidden space is shown through dotted orange lines joining vertices lying in lines where every point shares the same output value (isolines).

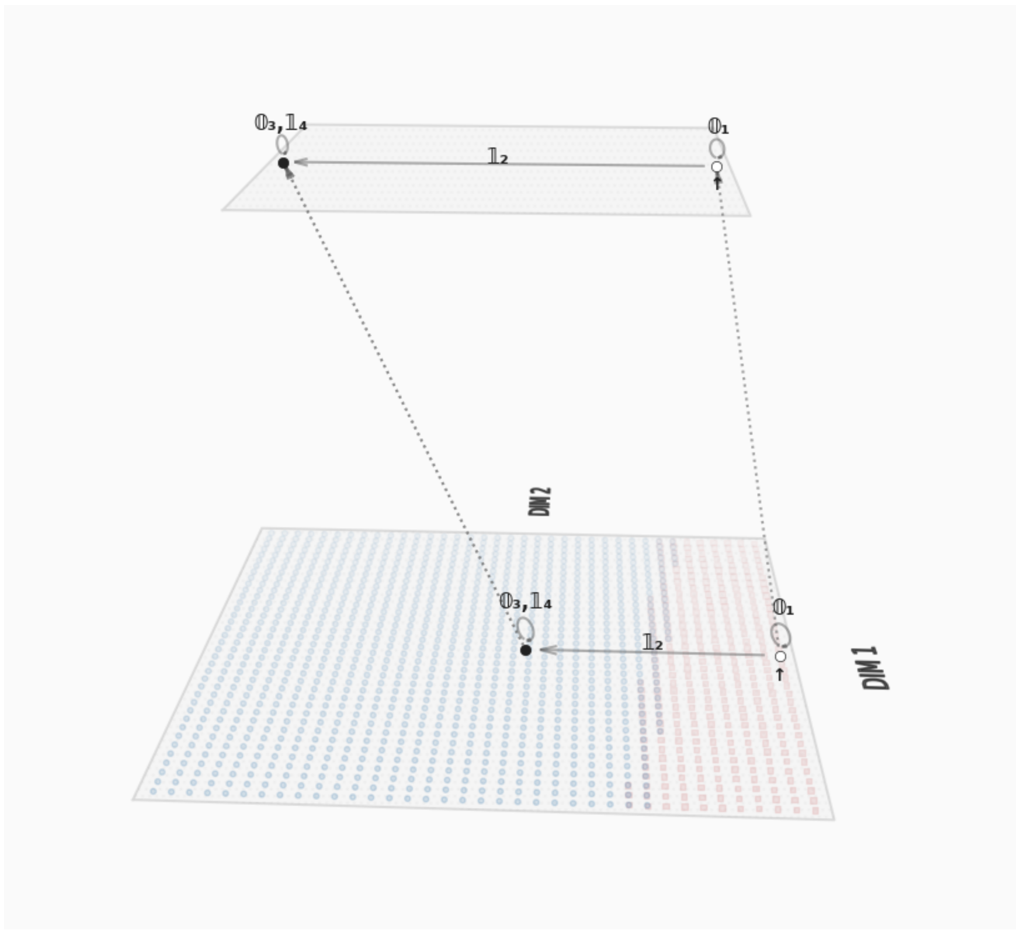
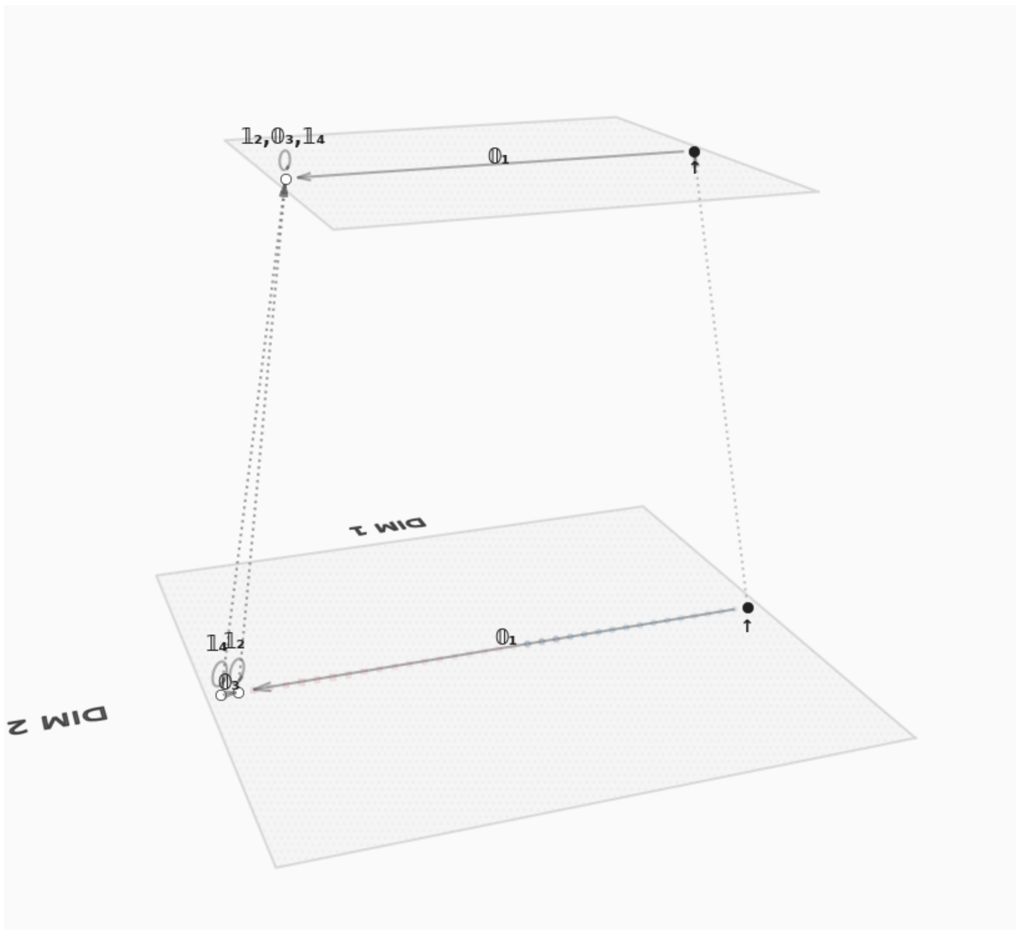


Figure 31: Hidden (lower plane) and output (upper plane) spaces for input 0101 to the OR-CNN. Conventions as per Figure 28.



$\mathbb{1}_2, \mathbb{0}_3, \mathbb{1}_4$
$\mathbb{0}_1$
DIM 1
DIM 2
$\mathbb{1}_4 \mathbb{1}_2$
$\mathbb{0}_3$
$\mathbb{0}_1$

Hidden (lower plane) and output (upper plane) spaces for input 0101 to the AND-CNN. Conventions as per Figure 28.

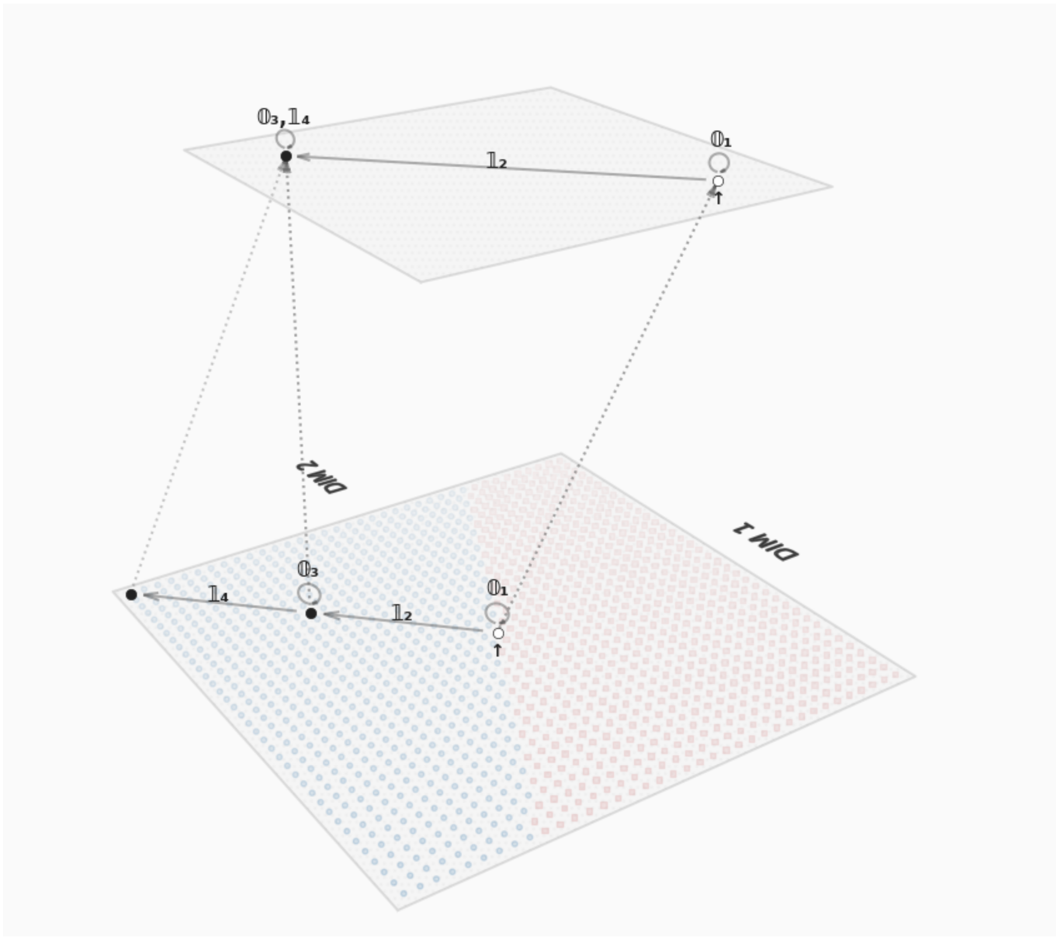
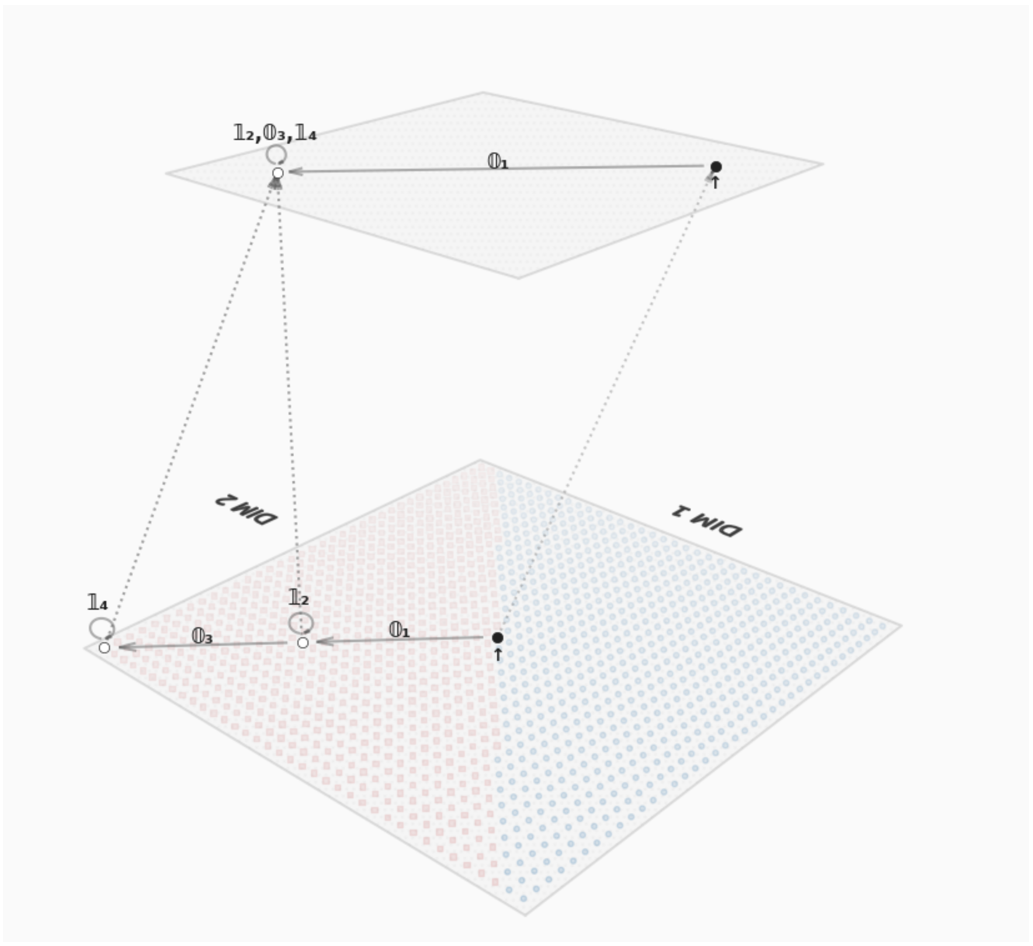


Figure 33: Hidden (lower plane) and output (upper plane) spaces for input 0101 to the OR-TFM. Conventions as per Figure 28.



Hidden (lower plane) and output (upper plane) spaces for input 0101 to the AND-FM. The hidden layer's vectors are originally 10-dimensional but have been reduced to 2 dimensions with PCA keeping 99.9% of the variance. Conventions as per Figure 28.

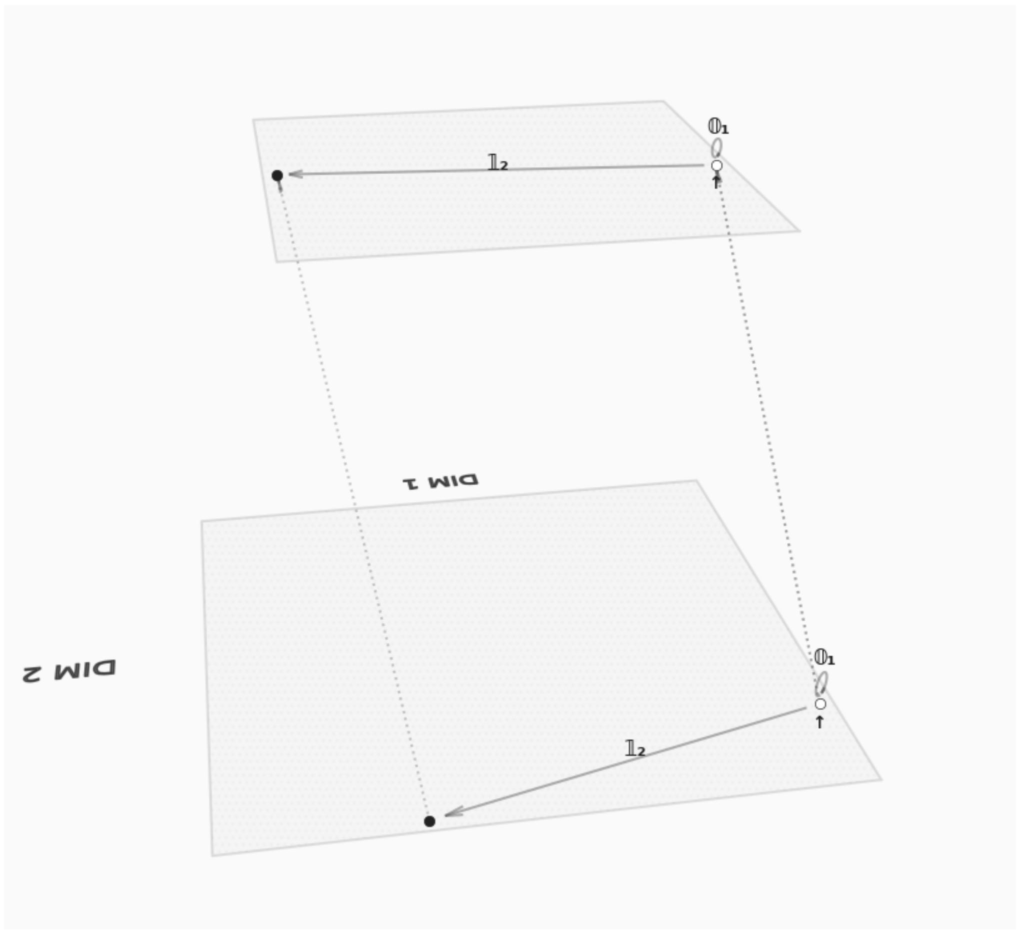


Figure 35: Hidden (lower plane) and output (upper plane) spaces for input 01 to the XOR-TFM. Conventions as per Figure 28.

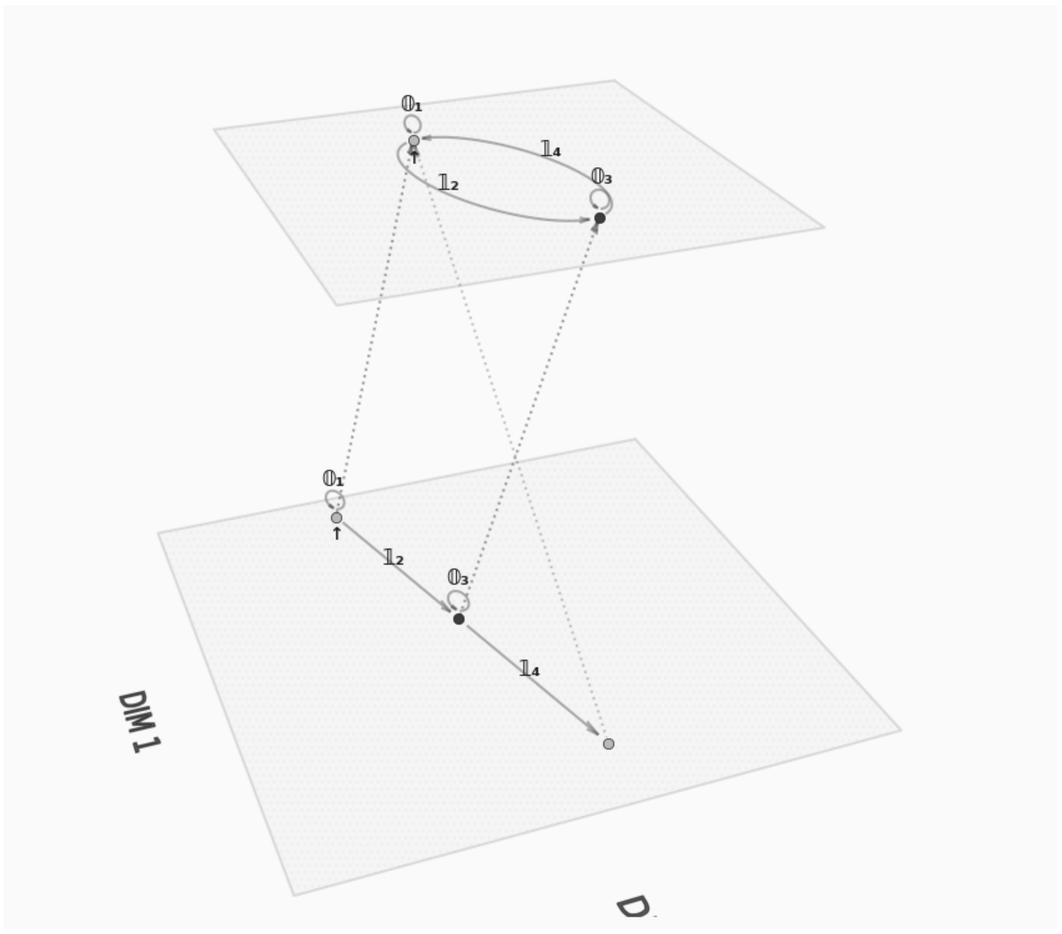


Figure 36: Hidden (lower plane) and output (upper plane) spaces for input 0101 to the XOR-TFM. Conventions as per Figure 28.

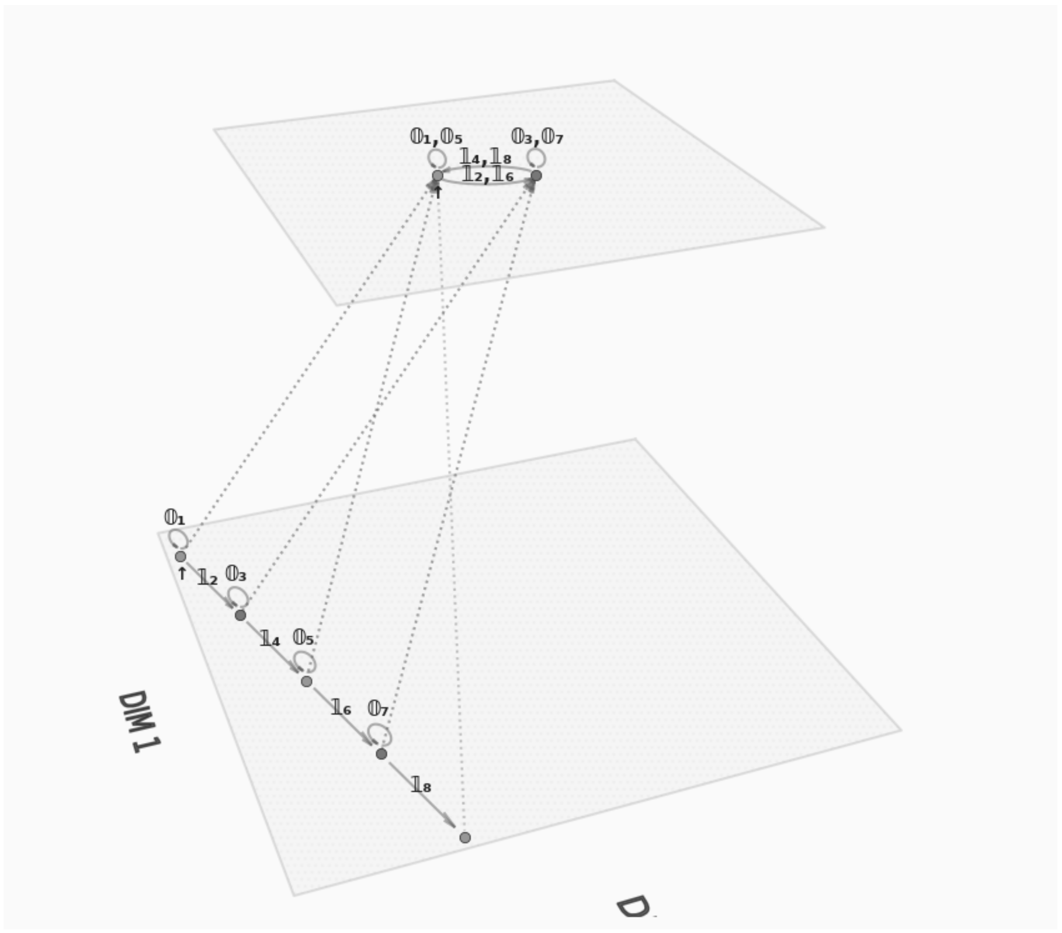


Figure 37: Hidden (lower plane) and output (upper plane) spaces for input 01010101 to the XOR-TFM. Conventions as per Figure 28.

5 Discussion

5.1 WFSAs

We've seen that the hidden representations of WFSAs trace the state-transition diagram of the underlying unweighted FSA. This is especially clear with the AND-task, whose optimal WFSA has 1 dimension but its optimal FSA has 2 dimensions, which matches the number of vertices in its hidden space. This makes intuitive sense because the number of different vertices must be at least be equal to the number of output classes.[29]

5.2 RNNs

We have confirmed that WFSAs hidden space graph matches its FSA (Table 5). We then found that linear-output RNNs' vertices lie on 2D affine subspaces with the same output (isolines), while softmax-output vertices lie within decision regions. This appears to hint to an explanation of the structure of the hidden space in WFSAs and RNN, namely that the more non-linear operations, the larger the number of vertices, and the less confined they are to a region of the hidden space. I speculate that the reason the vertices don't match the states is that they are result from the specifics of SGD-training and, since they are all made equivalent by the decoder, there is no incentive to map vertices to FSA states one-to-one. Figure VI illustrates how the fraction of the hidden space that is mapped to the same output is smaller, as expected, since linear transforms don't squash their inputs like softmax functions do, and thus we would expect that vertices would end up being closer together and, in the limit, becoming one and the same point in space (1-D affine subspace), as in WFSAs.

5.3 FFNs

I have shown that unrolling the first forward-pass operation that aggregates embeddings from different tokens - along with the addition of an initial dummy token - allows us to simulate a recurrent model inside an FFN, and then compare it against a reference FSA. This unrolling is in fact a decomposition of an atomic operation with the recursive scan algorithm. For MLPs, the operation is the pooling sum before the logit computation. For CNNs it's the max-pooling layer. For TFMs, it is the matrix product between the attention and the value matrices. The result of this decomposition is a sequence of embeddings homomorphic to a token count, 0s for the AND-task and 1s for the OR- and XOR-tasks. Downstream operations divide this hidden counting space into two decision regions, corresponding to the binary outputs. At the output (and also in some earlier layers) the reference FSA is simulated.

Table 5: Homomorphisms between hidden space structure and FSA structure for all model architectures.

Architecture	FSA state	FSA transition
WFSA	1-D affine subspace (point)	linear transform
linear-output RNN	2D affine subspace (line)	non-/linear transforms
softmax-output RNN	output-equivalent (decision) region	non-/linear transforms
MLP/CNN/TFM	recursive scan output (point)	recursive scan + non-/linear transforms

[29]In fact, XOR can be solved with a 1-dimensional WFSA if we make the output polar $[-1, 1]$ instead of binary $[0, 1]$, though the number of vertices remains 2. This implies that for WFSAs, the output format, not just the input language, influences the number of states.

5.4 Implementation-level to Primitives-level Mapping

In this paper I view the basic parts of a model, embeddings and linear/non-linear operations, as the ontological atoms of its implementation level of analysis.[30] The corresponding atoms at the primitives levels I take to be FSA states and transitions, plus counts for FFNs models.[31] The relationships between those two levels is summarised in 6. Different model architectures implement these mappings at different levels of abstraction. WFSAs' atoms stand in one-to-one correspondence to each other, I call this type of models *iso-representational*. In RNNs, multiple implementation elements map to a single primitive atom: these are *sub-representational* models. FFNs' single implementation abstraction maps to multiple primitives atoms: these are *super-representational* models.

Table 6: Mappings between implementation and primitives level atoms for all model architectures.

Architecture	Mapping type	FSA state	FSA transition
WFSA	iso-representational	weight-vector	matrix-vector multiplication
RNN	sub-representational	hidden state	non-/linear operations
FFN	super-representational	unrolled output	unrolled non-/linear operations

5.5 Primitives-level to Algorithmic-level Mapping

Recurrent models' task-solving algorithm can be described as a recursive application of a binary operation (FSA-transition) mapping the current input (FSA-symbol) and the current running state (FSA-state) (implementations 1-4 in Figures 38, 39, 40).

Feed-forward models can be seen as using a per-timestep token (FSA-input)-counting algorithm followed (layerwise) by a binary operation (FSA-transition) at each timestep mapping the current count and a comparison term - 0 or 1 - to the current output (FSA-state) (implementations 5-6 in Figures 38, 39, 40).

However, as the python code examples show, these atomic operations themselves can be implemented in different ways and further, more detailed research will be needed to uncover which ones.

5.6 Abstraction- vs Architectural-based Explanations

The type of explanation at the core of this paper puts abstract representations and their transformations at its center. This falls within the *Hopfieldian* view (Barack and Krakauer 2021) of the relationship between brain/machine and cognition/computation. The previous experiments confirm that this way of explaining at the algorithmic and primitives levels allows us to achieve a cohesive integrated understanding of multiple (if related) tasks across different architectures. Though we've mostly used visual explanations so far, this can also be approached analytically. As an example, here's the OR-WFSA parameters (to 3 decimal points) learned by SGD, shown in Figure 8:

$$\boldsymbol{\alpha} = \begin{bmatrix} .910 \\ .866 \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} .841 \\ -.884 \end{bmatrix} \quad \mathbf{A}_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{A}_1 = \begin{bmatrix} 1.018 & -.128 \\ .142 & -.018 \end{bmatrix}$$

Focussing on the transition matrix for the 1 input, $\mathbf{A}_1$, we know that it must be equivalent to the canonical solution in Figure 2. This implies that it must also be a projection matrix, which we can

[30]Firmware and hardware details are also part of the (lower) implementation level, but for the purposes of explanation of general models (as opposed to specific deployments) they can be ignored.

[31]I didn't investigate the lower-level operations in RNNs, which could reveal counting behaviour. I'll leave this for future work. See (El-Naggar et al. 2023).

confirm by multiplying it by itself: $\mathbf{A}_1\mathbf{A}_1 = \mathbf{A}_1$. Geometrically, projection matrices implement absorbing states, ones which the FSA cannot leave once it enters them (this is also the case with $\mathbf{A}_0$ as the identity matrix is also a projection matrix). This explanation has led us from a numerical entity (non-symbolic), meaningless by itself, through a geometric entity (sub-symbolic) to a computational one (symbolic). Yet another example is the transition matrix for 1, learned by the SGD-trained XOR-WFSA depicted in Figure 7:

$$\boldsymbol{\alpha} = \begin{bmatrix} .670 \\ -.376 \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} .640 \\ 1.142 \end{bmatrix} \quad \mathbf{A}_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{A}_1 = \begin{bmatrix} 1.011 & .061 \\ -.355 & -1.011 \end{bmatrix}$$

We know that it must be an involutory matrix, like the canonical one in Figure 1, which we can verify by multiplying it by itself: $\mathbf{A}_1\mathbf{A}_1 = \mathbf{I}$. In a vector space, involutory matrices implement a two-state or "switch" in an FSA.

In contrast to a substantial amount of current research seeking to understand neural networks, we've been able to avoid resorting to explanations centered on architectural components, the *Sherringtonian* view (Barack and Krakauer 2021), such as neurons/units (Sajjad, Durrani, and Dalvi 2022) and circuits (Elhage et al. 2021), thus avoiding the criticisms that have been levelled at this approach (Vilas et al. 2024). However, I do believe that a complete theory of neural computation must be able to explain why units/neurons and circuits appear to have specialised roles.[32]

[32]On the other hand, it is conceivable that any apparent semantic role for units might turn out to be due to statistics of the data fed to the model, similarly to how pixels in the centre of a TV screen might be found to be correlated with, say faces, given the typical images in TV programs, without actually being specialised for them. See (Jonas and Kording 2017; Posani et al. 2024).

```python
from collections.abc import Iterable
from functools import reduce

from itertools import product, accumulate
from numpy import eye, stack

def xor1(inputs: Iterable[bool]) -> bool:  # recursive
    return reduce(lambda context, input: context ^ input, inputs)


def xor2(inputs: Iterable[bool]) -> bool:  # recursive
    return bool(reduce(lambda context, input: (context + input) % 2, inputs))


def xor3(inputs: Iterable[bool]) -> bool:  # recursive
    return reduce(lambda context, input: (context or input) and not (context and input), inputs)


def xor4(inputs: Iterable[bool]) -> bool:  # recursive
    I = eye(2)
    trans = stack([I, I[:, ::-1]], axis=-1)
    return bool(reduce(lambda weight, input: trans @ I[int(input)] @ weight, inputs, I[0]) @ I[-1])


def xor5(inputs: Iterable[bool]) -> bool:  # non-recursive
    return bool(list(inputs).count(True) % 2)


def xor6(inputs: Iterable[bool]) -> bool:  # non-recursive
    return bool([ones % 2 for ones in accumulate(inputs, initial=0)][-1])
```

Figure 38: Functionally identical implementations of the XOR-task as Python functions. The first 4 are recursive, the 5th and 6th ones are parallel (though the 6th relies on a recursive count of 0s/1s).

```python
from collections.abc import Iterable
from functools import reduce

from itertools import product, accumulate
from numpy import eye, stack


def or1(inputs: Iterable[bool]) -> bool:  # recursive
    return reduce(lambda context, input: context or input, inputs)


def or2(inputs: Iterable[bool]) -> bool:  # recursive
    return reduce(lambda context, input: max(context, input), inputs)


def or3(inputs: Iterable[bool]) -> bool:  # recursive
    return bool(reduce(lambda context, input: min(context + input, True), inputs))


def or4(inputs: Iterable[bool]) -> bool:  # recursive
    I = eye(2)
    trans = stack([I, stack([I[0], I[0]])], axis=-1)
    return bool(reduce(lambda weight, input: trans @ I[int(input)] @ weight, inputs, I[0]) @ I[-1])


def or5(inputs: Iterable[bool]) -> bool:  # non-recursive
    return tuple(inputs).count(True) > 0


def or6(inputs: Iterable[bool]) -> bool:  # non-recursive
    return bool([ones > 0 for ones in accumulate(inputs, initial=0)][-1])
```

Figure 39: Functionally identical implementations of the OR-task as Python functions.

```python
from collections.abc import Iterable
from functools import reduce

from itertools import product, accumulate
from numpy import eye, stack

def and1(inputs: Iterable[bool]) -> bool:  # recursive
    return reduce(lambda context, input: context and input, inputs)


def and2(inputs: Iterable[bool]) -> bool:  # recursive
    return reduce(lambda context, input: min(context, input), inputs)


def and3(inputs: Iterable[bool]) -> bool:  # recursive
    return bool(reduce(lambda context, input: context * input, inputs))


def and4(inputs: Iterable[bool]) -> bool:  # recursive
    I = eye(2)
    trans = stack([I[1:, :1], I[:1, :1]], axis=-1)
    return bool(reduce(lambda weight, input: trans @ I[int(input)] @ weight, inputs, I[:1, :1]) @ I[:1, :1])


def and5(inputs: Iterable[bool]) -> bool:  # non-recursive
    return list(inputs).count(False) == 0


def and6(inputs: Iterable[bool]) -> bool:  # non-recursive
    return [not zeroes for zeroes in accumulate(inputs, lambda c, token: c + (not token), initial=0)][-1]
```

Figure 40: Functionally identical implementations of the AND-task as Python functions.

6 Conclusion and Future Work

In this work, I set out to test the idea that automata could be the centerpiece of our understanding of the inner workings of neural systems. Driving the exploration with a normative model - a WFSA's linear representation - for what a hidden representation should look like, and contrasting it with the actual hidden states of neural networks, has helped me find techniques for establishing homomorphisms between the models' operations and the structure of the corresponding FSA. This suggests that neural networks can be conceptualised as automata in a vector space, and thus, that formal language theory and automata theory could be key to unravelling the complexities of computation in a distributed continuous medium. For neuroscience, automata could be a potential missing piece for the solution of the mapping problem.[33]

Automata-based explanations have a number of traits that avoid criticisms levelled at the field of explainability, especially its reliance on neuron-level analyses, which in this paper I forgo altogether. I believe this makes it a complementary avenue of research to current methods such as causal interventions, feature attribution and mechanistic interpretability (Räuker et al. 2023). Within neuroscience, there have been efforts to characterise the internal representations of models with representational similarity techniques (Kornblith et al. 2019). The same effort could be approached with automata-similarity techniques (Mohri 2003; Cortes et al. 2008).

The limitations of this study motivate further work to fully address partially answered questions, as well as to ask new questions.

The purely empirical nature of my findings could be strengthened by theoretical work that might yield guarantees of where and how to find automata in any model, especially larger ones, and regardless of task, especially the complex ones. Representation theory (Serre 1977; Steinberg 2016) (appendix A.4), the algebraic theory of automata (Ginzburg 2014) or distributed automata (Reiter 2018) are some examples of theoretical starting points.

Experiments with more complex regular languages (e.g. Tomita grammars, Reber grammar, depth-bounded Dyck (Hewitt et al. 2020)) and languages higher up the Chomsky hierarchy (context-free, context-sensitive, recursively enumerable or counter languages) could help extend the scope of the current techniques to more challenging tasks. Spectral techniques for continuous inputs (Recasens and Quattoni 2013; Li, Precup, and Rabusseau 2022), for multi-class classifiers (Rabusseau, Balle, and Pineau 2017), for Weighted Pushdown Automata (Raphaël Bailly et al. 2013; Butoi et al. 2022; Cohen et al. 2014; Labai and Makowsky 2016) and for WFSTs (Raphael Bailly, Carreras, and Quattoni 2013; Recasens and Quattoni 2013) could help extend the scope of my approach to further tasks. One limitation of spectral techniques is that the number of model parameters grows as a product of the vocabulary size and the square of the number of states,[34] which limits its applicability, and therefore the applicability of this paper's approach. Although there are techniques for post-training WFSA parameter reduction (Jiang et al. 2020), they won't solve the training-time bottleneck. Another avenue of research could therefore be to use approaches like VQ-VAE (Van Den Oord, Vinyals, and others 2017) and Gumbel-softmax (Jang, Gu, and Poole 2017) to induce lower-dimensional but still discrete embeddings.

An intentional limitation of this work is the simplicity of the models. However, now that a foundation has been laid I can move onto more complex, deeper models and even further architectures like state-space models (Sarrof, Veitsman, and Hahn 2024). Another intentional limitation is the focus on inference at the expense of learning, which I plan on tackling future work. Yet another restriction is that I haven't studied how models *fail* to learn a task, which could provide a different type of insight.

[33]It is my belief that the issue at the root of the mapping problem is the skipping of a crucial level of abstraction. To draw an analogy from the natural sciences: it is like attempting to map physics (neuroscience) to biology (linguistics) without going through chemistry (the theory of automata in vector spaces) first.

[34]The number of parameters of a WFSA with k states and vocabulary size $|V|$ is $|V|k^2 + 2k$ vs an SRNN's $|V|k + k^2 + 2k$

7 Acknowledgements

I would like to thank Fahim Dalvi, Nadine El-Naggar, César González, Andreas Opedal and Raphaël Bailly for early feedback on this paper's ideas.

8 References

Adriaensen, Rik, and Jaron Maene. 2024. "Extracting Finite State Machines from Transformers." https://arxiv.org/abs/2410.06045.

Angluin, Dana. 1987. "Learning Regular Sets from Queries and Counterexamples." *Information and Computation* 75 (2): 87–106.

Arvan, Marcus. n.d. " 'Interpretability' and 'Alignment' Are Fool's Errands: A Proof That Controlling Misaligned Large Language Models Is the Best Anyone Can Hope for." *AI and Society*. https://doi.org/10.1007/s00146-024-02113-9.

Ayache, Stéphane, Rémi Eyraud, and Noé Goudian. 2019. "Explaining Black Boxes on Sequential Data Using Weighted Automata." In *Proceedings of the 14th International Conference on Grammatical Inference 2018*, edited by Olgierd Unold, Witold Dyrka, and Wojciech Wieczorek, 93:81–103. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v93/ayache19a.html.

Bailly, Raphaël, Xavier Carreras, Franco M. Luque, and Ariadna Quattoni. 2013. "Unsupervised Spectral Learning of WCFG as Low-Rank Matrix Completion." In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, edited by David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard, 624–35. Seattle, Washington, USA: Association for Computational Linguistics. https://aclanthology.org/D13-1059.

Bailly, Raphael, Xavier Carreras, and Ariadna Quattoni. 2013. "Unsupervised Spectral Learning of Finite State Transducers." In *Advances in Neural Information Processing Systems*, edited by C. J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Vol. 26. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2013/file/e836d813fd184325132fca8edcdfb40e-Paper.pdf.

Bailly, Raphaël, François Denis, and Liva Ralaivola. 2009. "Grammatical Inference as a Principal Component Analysis Problem." In *Proceedings of the 26th Annual International Conference on Machine Learning*, 33–40.

Balle, Borja, Xavier Carreras, Franco M. Luque, and Ariadna Quattoni. 2014. "Spectral Learning of Weighted Automata." *Machine Learning* 96 (1): 33–63. https://doi.org/10.1007/s10994-013-5416-x.

Balle, Borja, and Mehryar Mohri. 2012. "Spectral Learning of General Weighted Automata via Constrained Matrix Completion." In *Advances in Neural Information Processing Systems*, edited by F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/700fdb2ba62d4554dc268c65add4b16e-Paper.pdf.

———. 2015. "Learning Weighted Automata." In *Algebraic Informatics*, edited by Andreas Maletti, 1–21. Cham: Springer International Publishing.

Barack, David L., and John W. Krakauer. 2021. "Two Views on the Cognitive Brain." *Nature Reviews Neuroscience* 22 (6): 359–71. https://doi.org/10.1038/s41583-021-00448-6.

Barak, Boaz. 2022. "Introduction to Theoretical Computer Science." *Under Preparation*.

Baroni, Marco, Raffaella Bernardi, Roberto Zamparelli, and others. 2014. "Frege in Space: A Program for Compositional Distributional Semantics." *Linguistic Issues in Language Technology* 9: 241–346.

Bernardi, Silvia, Marcus K. Benna, Mattia Rigotti, Jérôme Munuera, Stefano Fusi, and C. Daniel Salzman. 2020. "The Geometry of Abstraction in the Hippocampus and Prefrontal Cortex." *Cell* 183 (4): 954–967.e21. https://doi.org/10.1016/j.cell.2020.09.031.

Blelloch, Guy E. 1990. "Prefix Sums and Their Applications."

Bree, Sander van. 2022. "A Critical Perspective on Neural Mechanisms in Cognitive Neuroscience: Towards Unification." *Perspectives on Psychological Science*, 17456916231191744.

Butoi, Alexandra, Brian DuSell, Tim Vieira, Ryan Cotterell, and David Chiang. 2022. "Algorithms for Weighted Pushdown Automata." In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, edited by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, 9669–80. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.emnlp-main.656.

Casey, Mike. 1996. "The Dynamics of Discrete-Time Computation, with Application to Recurrent Neural Networks and Finite State Machine Extraction." *Neural Comput.* 8 (6): 1135–78. https://doi.org/10.1162/neco.1996.8.6.1135.

Chughtai, Bilal, Lawrence Chan, and Neel Nanda. 2023. "A Toy Model of Universality: Reverse Engineering How Networks Learn Group Operations." In *International Conference on Machine Learning*, 6243–67. PMLR.

Chung, SueYeon, and L. F. Abbott. 2021. "Neural Population Geometry: An Approach for Understanding Biological and Artificial Neural Networks." *Current Opinion in Neurobiology* 70 (October): 137–44. https://doi.org/10.1016/j.conb.2021.10.010.

Cohen, Shay B., Karl Stratos, Michael Collins, Dean P. Foster, and Lyle Ungar. 2014. "Spectral Learning of Latent-Variable Pcfgs: Algorithms and Sample Complexity." *Journal of Machine Learning Research* 15 (69): 2399–2449. http://jmlr.org/papers/v15/cohen14a.html.

Cortes, Corinna, Mehryar Mohri, Ashish Rastogi, and Michael Riley. 2008. "On the Computation of the Relative Entropy of Probabilistic Automata." *International Journal of Foundations of Computer Science* 19 (01): 219–42.

Diego-Simón, Pablo, Stéphane D'Ascoli, Emmanuel Chemla, Yair Lakretz, and Jean-Rémi King. 2024. "A Polar Coordinate System Represents Syntax in Large Language Models." *arXiv Preprint arXiv:2412.05571.*

Doshi-Velez, Finale, and Been Kim. 2017. "Towards a Rigorous Science of Interpretable Machine Learning." *arXiv Preprint arXiv:1702.08608.*

Droste, Manfred, Werner Kuich, and Heiko Vogler. 2009. *Handbook of Weighted Automata.* 1st ed. Springer Publishing Company, Incorporated.

Elhage, Nelson, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, et al. 2022. "Toy Models of Superposition." *Transformer Circuits Thread.*

Elhage, Nelson, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, et al. 2021. "A Mathematical Framework for Transformer Circuits." *Transformer Circuits Thread.*

El-Naggar, Nadine, Andrew Ryzhikov, Laure Daviaud, Pranava Madhyastha, and Tillman Weyde. 2023. "Formal and Empirical Studies of Counting Behaviour in Relu Rnns." In *Proceedings of 16th Edition of the International Conference on Grammatical Inference*, edited by François Coste, Faissal Ouardi, and Guillaume Rabusseau, 217:199–222. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v217/el-naggar23a.html.

Embick, David, and David Poeppel. 2015. "Towards a Computational (Ist) Neurobiology of Language: Correlational, Integrated and Explanatory Neurolinguistics." *Language, Cognition and Neuroscience* 30 (4): 357–66.

Epstein, David B A. 1992. *Word Processing in Groups.* A K Peters/CRC Press.

Eyraud, Rémi, and Stéphane Ayache. 2021. "Distillation of Weighted Automata from Recurrent Neural Networks Using a Spectral Approach." *Machine Learning*, April. https://doi.org/10.1007/s10994-021-05948-1.

Fitz, Hartmut, Peter Hagoort, and Karl Magnus Petersson. 2024. "Neurobiological Causal Models of Language Processing." *Neurobiology of Language* 5 (1): 225–47. https://doi.org/10.1162/nol_a_00133.

Frankle, Jonathan, and Michael Carbin. 2019. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks." https://arxiv.org/abs/1803.03635.

Friedman, Dan, Alexander Wettig, and Danqi Chen. 2023. "Learning Transformer Programs." https://arxiv.org/abs/2306.01128.

Geiger, Atticus, Hanson Lu, Thomas Icard, and Christopher Potts. 2021. "Causal Abstractions of Neural Networks." In *Advances in Neural Information Processing Systems*, edited by M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan, 34:9574–86. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2021/file/4f5c422f4d49a5a807eda27434231040-Paper.pdf.

Ginzburg, Abraham. 2014. *Algebraic Theory of Automata.* San Diego, CA: Academic Press.

Glaude, Hadrien, Cyrille Enderli, and Olivier Pietquin. 2015. "Spectral Learning with Proper Probabilities for Finite State Automation." In *ASRU 2015-Automatic Speech Recognition and Understanding Workshop.* IEEE.

Gorman, Kyle, and Richard Sproat. 2022. *Finite-State Text Processing.* Springer Nature.

Gutmann, Michael, and Aapo Hyvärinen. 2010. "Noise-Contrastive Estimation: A New Estimation Principle for Unnormalized Statistical Models." In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, edited by Yee Whye Teh and Mike Titterington, 9:297–304. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR. https://proceedings.mlr.press/v9/gutmann10a.html.

Hahn, Michael, and Mark Rofin. 2024. "Why Are Sensitive Functions Hard for Transformers?" *arXiv Preprint arXiv:2402.09963.*

Hale, John Tracy, Luca Campanelli, Jixing Li, Christophe Pallier, and Jonathan Brennan. 2021. "Neurocomputational Models of Language Processing." *Annual Review of Linguistics.* https://api.semanticscholar.org/CorpusID:237235743.

Hamrick, Jessica, and Shakir Mohamed. 2020. "Levels of Analysis for Machine Learning." https://arxiv.org/abs/2004.05107.

He, Zhonghao, Jascha Achterberg, Katie Collins, Kevin Nejad, Danyal Akarca, Yinzhu Yang, Wes Gurnee, et al. 2024. "Multilevel Interpretability of Artificial Neural Networks: Leveraging Framework and Methods from Neuroscience." *arXiv Preprint arXiv:2408.12664.*

Henzinger, T. A. 1996. "The Theory of Hybrid Automata." In *Proceedings 11th Annual Ieee Symposium on Logic in Computer Science*, 278–92. https://doi.org/10.1109/LICS.1996.561342.

Hewitt, John, Michael Hahn, Surya Ganguli, Percy Liang, and Christopher D Manning. 2020. "RNNs Can Generate Bounded Hierarchical Languages with Optimal Memory." *arXiv Preprint arXiv:2010.07515*.

Hewitt, John, and Christopher D. Manning. 2019. "A Structural Probe for Finding Syntax in Word Representations." In *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)*, edited by Jill Burstein, Christy Doran, and Thamar Solorio, 4129–38. Minneapolis, Minnesota: Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1419.

Hillis, W Daniel, and Guy L Steele Jr. 1986. "Data Parallel Algorithms." *Communications of the ACM* 29 (12): 1170–83.

Holcombe, W. M. L. 2004. *Algebraic Automata Theory*. Cambridge Studies in Advanced Mathematics. Cambridge University Press. https://books.google.es/books?id=xCtthlOQUfYC.

Hsu, Daniel, Sham M. Kakade, and Tong Zhang. 2009. "A Spectral Algorithm for Learning Hidden Markov Models." In *Proceedings of the Annual Conference on Computational Learning Theory (Colt)*.

Hutson, Matthew. 2018. "Has Artificial Intelligence Become Alchemy?" *Science* 360 (6388): 478–78. https://doi.org/10.1126/science.360.6388.478.

Jäger, Gerhard, and James Rogers. 2012. "Formal Language Theory: Refining the Chomsky Hierarchy." *Philosophical Transactions of the Royal Society B: Biological Sciences* 367: 1956–70. https://api.semanticscholar.org/CorpusID:8765815.

Jang, Eric, Shixiang Gu, and Ben Poole. 2017. "Categorical Reparameterization with Gumbel-Softmax." https://arxiv.org/abs/1611.01144.

Ji, Ziwei, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. "Survey of Hallucination in Natural Language Generation." *ACM Comput. Surv.* 55 (12). https://doi.org/10.1145/3571730.

Jiang, Chengyue, Yinggong Zhao, Shanbo Chu, Libin Shen, and Kewei Tu. 2020. "Cold-Start and Interpretability: Turning Regular Expressions into Trainable Recurrent Neural Networks." In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (Emnlp)*, edited by Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, 3193–3207. Online: Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.258.

Jonas, Eric, and Konrad Paul Kording. 2017. "Could a Neuroscientist Understand a Microprocessor?" *PLOS Computational Biology* 13 (1): 1–24. https://doi.org/10.1371/journal.pcbi.1005268.

Kaushik, Karthikeya R, and Andrea E Martin. 2022. "A Mathematical Neural Process Model of Language Comprehension, from Syllable to Sentence." PsyArXiv. https://doi.org/10.31234/osf.io/xs5kr.

Kim, Jason Z., and Dani S. Bassett. 2023. "A Neural Machine Code and Programming Framework for the Reservoir Computer." *Nature Machine Intelligence* 5 (6): 622–30. https://doi.org/10.1038/s42256-023-00668-8.

Kingma, Diederik P., and Jimmy Ba. 2017. "Adam: A Method for Stochastic Optimization." https://arxiv.org/abs/1412.6980.

Kleene, S. C. 1956. "Representation of Events in Nerve Nets and Finite Automata." In *Automata Studies*, edited by C. E. Shannon and J. McCarthy, 3–42. Princeton: Princeton University Press. https://doi.org/doi:10.1515/9781400882618-002.

Kornblith, Simon, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. 2019. "Similarity of Neural Network Representations Revisited." In *International Conference on Machine Learning*, 3519–29.

PMLR.

Kozen, Dexter. 1997. *Automata and Computability*. 1st ed. Undergraduate Texts in Computer Science. New York, NY: Springer.

Labai, Nadia, and Johann A. Makowsky. 2016. "Hankel Matrices for Weighted Visibly Pushdownautomata." In *Language and Automata Theory and Applications*, edited by Adrian-Horia Dediu, Jan Janoušek, Carlos Martín-Vide, and Bianca Truthe, 464–77. Cham: Springer International Publishing.

Lee, Daniel D., and H. Sebastian Seung. 1999. "Learning the Parts of Objects by Non-Negative Matrix Factorization." *Nature* 401 (6755): 788–91. https://doi.org/10.1038/44565.

Li, Tianyu, Doina Precup, and Guillaume Rabusseau. 2022. "Connecting Weighted Automata, Tensor Networks and Recurrent Neural Networks Through Spectral Learning." https://arxiv.org/abs/2010.10029.

Lillicrap, Timothy P, and Konrad P Kording. 2019. "What Does It Mean to Understand a Neural Network?" *arXiv Preprint arXiv:1907.06374*.

Lindner, David, Janos Kramar, Sebastian Farquhar, Matthew Rahtz, Tom McGrath, and Vladimir Mikulik. 2023. "Tracr: Compiled Transformers as a Laboratory for Interpretability." In *Advances in Neural Information Processing Systems*, edited by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, 36:37876–99. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2023/file/771155abaae744e08576f1f3b4b7ac0d-Paper-Conference.pdf.

Linz, P. 2001. *An Introduction to Formal Languages and Automata*. Jones and Bartlett Computer Science. Jones; Bartlett. https://books.google.es/books?id=Cgooanwdo9AC.

Liu, Bingbin, Jordan Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. 2024. "Exposing Attention Glitches with Flip-Flop Language Modeling." *Advances in Neural Information Processing Systems* 36.

Liu, Bingbin, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. 2022. "Transformers Learn Shortcuts to Automata." *arXiv Preprint arXiv:2210.10749*.

Liu, Bingbin, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. 2023. "Transformers Learn Shortcuts to Automata." https://arxiv.org/abs/2210.10749.

Marr, David. 2010. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. MIT press.

Marr, David, and Tomaso Poggio. 1976. "From Understanding Computation to Understanding Neural Circuitry."

McCoy, R. Thomas, Tal Linzen, Ewan Dunbar, and Paul Smolensky. 2019. "RNNs Implicitly Implement Tensor Product Representations." https://arxiv.org/abs/1812.08718.

Mcculloch, Warren S., and Walter Pitts. 1943. "A Logical Calculus of the Ideas Immanent in Nervous Activity." *Journal of Symbolic Logic* 9 (2): 49–50. https://doi.org/10.2307/2268029.

Merrill, William. 2019. "Sequential Neural Networks as Automata." *arXiv Preprint arXiv:1906.01615*.

———. 2021. "Formal Language Theory Meets Modern Nlp." *arXiv Preprint arXiv:2102.10094*.

Michalenko, Joshua J., Ameesh Shah, Abhinav Verma, Richard G. Baraniuk, Swarat Chaudhuri, and Ankit B. Patel. 2019. "Representing Formal Languages: A Comparison Between Finite Automata and Recurrent Neural Networks." https://arxiv.org/abs/1902.10297.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. "Distributed Representations of Words and Phrases and Their Compositionality." *Advances in Neural Information Processing Systems* 26.

Minsky, Marvin L. 1967. *Computation: Finite and Infinite Machines.* USA: Prentice-Hall, Inc.

Mohri, Mehryar. 2001. "Language Processing with Weighted Transducers." In *Actes de La 8ème Conférence Sur Le Traitement Automatique Des Langues Naturelles. Conférences Invitées*, 5–14.

———. 2003. "Edit-Distance of Weighted Automata." In *Implementation and Application of Automata*, edited by Jean-Marc Champarnaud and Denis Maurel, 1–23. Berlin, Heidelberg: Springer Berlin Heidelberg.

———. 2009. "Weighted Automata Algorithms." *Handbook of Weighted Automata*, 213–54.

Nikankin, Yaniv, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. 2024. "Arithmetic Without Algorithms: Language Models Solve Math with a Bag of Heuristics." *arXiv Preprint arXiv:2410.21272*.

Omlin, Christian W., and C. Lee Giles. 2000. "Symbolic Knowledge Representation in Recurrent Neural Networks: Insights from Theoretical Models of Computation." In *Knowledge-Based Neurocomputing*. The MIT Press. https://doi.org/10.7551/mitpress/4070.003.0005.

Peng, Hao, Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018. "Rational Recurrences." https://arxiv.org/abs/1808.09357.

Phuong, Mary, and Marcus Hutter. 2022. "Formal Algorithms for Transformers." *arXiv Preprint arXiv:2207.09238*.

Poeppel, David. 2012. "The Maps Problem and the Mapping Problem: Two Challenges for a Cognitive Neuroscience of Speech and Language." *Cognitive Neuropsychology* 29 (1-2): 34–55. https://doi.org/10.1080/02643294.2012.710600.

Poeppel, David, and William Idsardi. 2022. "We Don't Know How the Brain Stores Anything, Let Alone Words." *Trends in Cognitive Sciences* 26 (12): 1054–5. https://doi.org/10.1016/j.tics.2022.08.010.

Poeppel, D, and D Embick. 2005. "Defining the Relation Between Linguistics and Neuroscience." In *Twenty-First Century Psycholinguistics: Four Cornerstones*, edited by A Cutler, 103–18. Lawrence Erlbaum Associates Publishers.

Pollack, Jordan B. 1991. "The Induction of Dynamical Recognizers." *Machine Learning* 7 (2): 227–52. https://doi.org/10.1007/BF00114845.

Posani, Lorenzo, Shuqi Wang, Samuel Muscinelli, Liam Paninski, and Stefano Fusi. 2024. "Rarely Categorical, Always High-Dimensional: How the Neural Code Changes Along the Cortical Hierarchy." *bioRxiv*, 2024–11.

Quattoni, Ariadna, and Xavier Carreras. 2019. "Interpolated Spectral Ngram Language Models." In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 5926–30.

Quattoni, Ariadna, Xavier Carreras, and Matthias Gallé. 2017. "A Maximum Matching Algorithm for Basis Selection in Spectral Learning." In *Artificial Intelligence and Statistics*, 1477–85. PMLR.

Rabusseau, Guillaume, Borja Balle, and Joelle Pineau. 2017. "Multitask Spectral Learning of Weighted Automata." In *Advances in Neural Information Processing Systems*, edited by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/e655c7716a4b3ea67f48c6322fc42ed6-Paper.pdf.

Räuker, Tilman, Anson Ho, Stephen Casper, and Dylan Hadfield-Menell. 2023. "Toward Transparent Ai: A Survey on Interpreting the Inner Structures of Deep Neural Networks." In *2023 Ieee Conference on Secure and Trustworthy Machine Learning (Satml)*, 464–83. IEEE.

Reber, Arthur S. 1967. "Implicit Learning of Artificial Grammars." *Journal of Verbal Learning and Verbal Behavior* 6 (6): 855–63. https://doi.org/https://doi.org/10.1016/S0022-5371(67)80149-X.

Recasens, Adria, and Ariadna Quattoni. 2013. "Spectral Learning of Sequence Taggers over Continuous Sequences." In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 289–304. Springer.

Reiter, Fabian. 2018. "Distributed Automata and Logic." https://arxiv.org/abs/1805.06238.

Rizvi-Martel, Michael, Maude Lizaire, Clara Lacroce, and Guillaume Rabusseau. 2024. "Simulating Weighted Automata over Sequences and Trees with Transformers." In *International Conference on Artificial Intelligence and Statistics*, 2368–76. PMLR.

Rodriguez, Paul. 2001. "Simple Recurrent Networks Learn Context-Free and Context-Sensitive Languages by Counting." *Neural Computation* 13 (9): 2093–2118. https://doi.org/10.1162/089976601750399326.

Sajjad, Hassan, Nadir Durrani, and Fahim Dalvi. 2022. "Neuron-Level Interpretation of Deep Nlp Models: A Survey." *Transactions of the Association for Computational Linguistics* 10: 1285–1303.

Sarrof, Yash, Yana Veitsman, and Michael Hahn. 2024. "The Expressive Capacity of State Space Models: A Formal Language Perspective." https://arxiv.org/abs/2405.17394.

Savage, John E. 1997. *Models of Computation: Exploring the Power of Computing*. 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc.

Serre, Jean-Pierre. 1977. *Linear Representations of Finite Groups*. Graduate Texts in Mathematics. New York, NY: Springer.

Servan-Schreiber, David, Axel Cleeremans, and James McClelland. 1988. "Learning Sequential Structure in Simple Recurrent Networks." *Advances in Neural Information Processing Systems* 1.

Siegelmann, Hava T, and Eduardo D Sontag. 1992. "On the Computational Power of Neural Nets." In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 440–49.

Sipser, M. 2006. *Introduction to the Theory of Computation*. Thomson Course Technology. https://books.google.es/books?id=VJ1mQgAACAAJ.

Smith, Jimmy T. H., Scott W. Linderman, and David Sussillo. 2021. "Reverse Engineering Recurrent Neural Networks with Jacobian Switching Linear Dynamical Systems." https://arxiv.org/abs/2111.01256.

Smith, Leslie N., and Nicholay Topin. 2018. "Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates." https://arxiv.org/abs/1708.07120.

Smolensky, Paul. 1988. "On the Proper Treatment of Connectionism." *Behavioral and Brain Sciences* 11 (1): 1–23.

———. 1990. "Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems." *Artificial Intelligence* 46 (1): 159–216. https://doi.org/https://doi.org/10.1016/0004-3702(90)90007-M.

Steinberg, Benjamin. 2016. *Representation Theory of Finite Monoids*. Universitext. Springer International Publishing.

Sun, G. Z., H. H. Chen, Y. C. Lee, and C. L. Giles. 1990. "Recurrent Neural Networks, Hidden Markov Models and Stochastic Grammars." In *1990 Ijcnn International Joint Conference on Neural Networks*, 729–34 vol.1. https://doi.org/10.1109/IJCNN.1990.137655.

Svete, Anej, and Ryan Cotterell. 2023. "Recurrent Neural Language Models as Probabilistic Finite-State Automata." In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, edited by Houda Bouamor, Juan Pino, and Kalika Bali, 8069–86. Singapore: Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.emnlp-main.502.

Tiňo, Peter, Bill G. Horne, C. Lee Giles, and Pete C. Collingwood. 1998. "Chapter 6 - Finite State Machines and Recurrent Neural Networks — Automata and Dynamical Systems Approaches." In *Neural Networks and Pattern Recognition*, edited by Omid Omidvar and Judith Dayhoff, 171–219. San Diego: Academic Press. https://doi.org/https://doi.org/10.1016/B978-012526420-4/50007-0.

Todd, Eric, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. 2024. "Function Vectors in Large Language Models." https://arxiv.org/abs/2310.15213.

Tomita, Masaru. 1982. "Dynamic Construction of Finite-State Automata from Examples Using Hill-Climbing." In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, 105–8.

Van Den Oord, Aaron, Oriol Vinyals, and others. 2017. "Neural Discrete Representation Learning." *Advances in Neural Information Processing Systems* 30.

Vilas, Martina G., Federico Adolfi, David Poeppel, and Gemma Roig. 2024. "Position: An Inner Interpretability Framework for Ai Inspired by Lessons from Cognitive Neuroscience." https://arxiv.org/abs/2406.01352.

Vyas, Saurabh, Matthew D Golub, David Sussillo, and Krishna V Shenoy. 2020. "Computation Through Neural Population Dynamics." *Annual Review of Neuroscience* 43 (1): 249–75.

Wang, Qinglong, Kaixuan Zhang, Alexander G Ororbia II, Xinyu Xing, Xue Liu, and C Lee Giles. 2018. "A Comparative Study of Rule Extraction for Recurrent Neural Networks." *arXiv Preprint arXiv:1801.05420*.

Weiss, Gail, Yoav Goldberg, and Eran Yahav. 2021. "Thinking Like Transformers." In *Proceedings of the 38th International Conference on Machine Learning*, edited by Marina Meila and Tong Zhang, 139:11080–90. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v139/weiss21a.html.

———. 2024. "Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples (Extended Version)." *Machine Learning* 113 (5): 2877–2919. https://doi.org/10.1007/s10994-022-06163-2.

Zeng, Zheng, Rodney M. Goodman, and Padhraic Smyth. 1993. "Learning Finite State Machines with Self-Clustering Recurrent Networks." *Neural Computation* 5 (6): 976–90. https://doi.org/10.1162/neco.1993.5.6.976.

Zhang, Kaixuan, Qinglong Wang, and C. Lee Giles. 2021. "An Entropy Metric for Regular Grammar Classification and Learning with Recurrent Neural Networks." *Entropy* 23 (1). https://doi.org/10.3390/e23010127.

Zhong, Ziqian, Ziming Liu, Max Tegmark, and Jacob Andreas. 2024. "The Clock and the Pizza: Two Stories in Mechanistic Explanation of Neural Networks." *Advances in Neural Information Processing Systems* 36.

A Appendix

A.1 Training Setup

All models achieve 0 training, validation and test thresholded 0-1 loss, with thresholds at 99.9% of the target value (0 or 1). Spectral WFSAs also achieve 0 ℓ_2 training loss. For SGD models, the cross-entropy and ℓ_2 training losses are shown in their corresponding tables.

Table 7 shows the hyperparameters for the SL-trained WFSAs, found through the gridsearch algorithm 7 and which maximise ternariness.

Table 8 lists the SGD-training setup for all tasks and models. The optimiser is Adam (Kingma and Ba 2017), the learning rate scheduler is one-cycle (Smith and Topin 2018). The batch size is actually the minimum between the length of the input sequence and the number shown. This is because, in order to avoid using a padding token (for simplicity's sake), batches are length-based. SRNNs all use ReLU as a non-linearity. TFMs' feedforward layer is 4 times the embedding dimension and uses ReLU instead of GELU as a non-linearity, for simplicity. TFMs' context size is 24. All models are trained with the cross-entropy loss except WFSAs and the linear-output RNNs, which are trained with ℓ_2. Hyperparameters were found with a gridsearch driven by the validation loss.

Table 9 lists the training setup for linear-output RNNs, identical to that in Table 8, except that the training loss is ℓ_2.

Table 7: SL training setup for WFSAs shown in Figures 1, 2 and 3. The NMF is optimised with the Frobenius loss. The ternariness threshold is always .01 either side of 0 or ± 1.

Task	dimensions	# prefixes	# suffixes	initialisation	shuffle	ternariness
XOR	2	1	1	nndsvd	True	.8333
OR	2	1	8	svd	False	.9167
AND	1	1	1	nndsvd	False	1.000

Table 8: Hyperparamters and losses for the XOR, OR and AND tasks

Architecture	dimensions	epochs	training loss	batch size	learning rate	max learning rate	init
WFSA	2	100	$1.60e^{-12}$	64	$1e^{-5}$	$1e^{-1}$	xavier uniform
SRNN	3	750	$1.93e^{-5}$	32	$1e^{-1}$	$1e^{-1}$	xavier uniform
GRU	2	200	$1.29e^{-5}$	128	$1e^{-5}$	$1e^{-0}$	orthogonal
LSTM	2	200	$8.42e^{-5}$	128	$1e^{-4}$	$1e^{-0}$	orthogonal
MLP	–	–	–	–	–	–	–
CNN	–	–	–	–	–	–	–
TFM	–	–	–	–	–	–	–

Architecture	dimensions	epochs	training loss	batch size	learning rate	max learning rate	init
WFSA	2	100	$2.55e^{-11}$	32	$1e^{-1}$	$1e^{-0}$	xavier uniform
SRNN	2	750	$9.01e^{-8}$	32	$1e^{-3}$	$1e^{-0}$	orthogonal
GRU	2	200	$1.82e^{-5}$	128	$1e^{-5}$	$1e^{-0}$	orthogonal
LSTM	2	200	$2.06e^{-5}$	64	$1e^{-5}$	$1e^{-0}$	xavieru
MLP	2	1000	$1.86e^{-6}$	32	$1e^{-5}$	$1e^{-0}$	xavier normal
CNN	2	200	$3.54e^{-6}$	128	$1e^{-5}$	$1e^{-0}$	orthogonal
TFM	2	201	$1.21e^{-5}$	128	$1e^{-3}$	$1e^{-0}$	orthogonal

Architecture	dimensions	epochs	training loss	batch size	learning rate	max learning rate	init
WFSA	1	100	$1.15e^{-10}$	128	$1e^{-5}$	$1e^{-1}$	orthogonal
SRNN	2	250	$6.11e^{-6}$	16	$1e^{-5}$	$1e^{-0}$	xavier normal
GRU	1	200	$1.28e^{-4}$	128	$1e^{-3}$	$1e^{-0}$	xavier normal
LSTM	1	200	$6.60e^{-5}$	32	$1e^{-5}$	$1e^{-0}$	xavier uniform
MLP	2	1000	$8.41e^{-6}$	128	$1e^{-5}$	$1e^{-1}$	xavier uniform
CNN	1	200	$1.03e^{-5}$	128	$1e^{-5}$	$1e^{-0}$	xavier normal
TFM	2	201	$5.77e^{-4}$	128	$1e^{-4}$	$1e^{-1}$	xavier normal

Table 9: Hyperparamters and losses for XOR, OR and AND tasks solved by RNNs with linear output.

Architecture	dimensions	epochs	training loss	batch size	learning rate	max learning rate	init
SRNN	3	750	$1.08e^{-14}$	16	$1e^{-1}$	$1e^{-1}$	orthogonal
GRU	2	1000	$1.72e^{-7}$	16	$1e^{-3}$	$1e^{-0}$	orthogonal
LSTM	2	1000	$2.27e^{-7}$	8	$1e^{-4}$	$1e^{-0}$	xavier uniform

Architecture	dimensions	epochs	training loss	batch size	learning rate	max learning rate	init
SRNN	2	750	$2.58e^{-14}$	32	$1e^{-1}$	$1e^{-1}$	orthogonal
GRU	2	1000	$1.97e^{-7}$	32	$1e^{-5}$	$1e^{-1}$	xavier uniform
LSTM	2	200	$8.44e^{-13}$	16	$1e^{-0}$	$1e^{-0}$	xavier normal

Architecture	dimensions	epochs	training loss	batch size	learning rate	max learning rate	init
SRNN	2	250	$1.28e^{-16}$	16	$1.0e^{-5}$	$1.0e^{-1}$	orthogonal
GRU	2	1000	$1.96e^{-8}$	4	$1.0e^{-4}$	$1.0e^{-1}$	xavier normal
LSTM	2	500	$4.17e^{-9}$	32	$1.0e^{-3}$	$1.0e^{-0}$	xavier normal

A.2 Visualisation Algorithms

Algorithm 1 converts a model's embeddings into a format fit for 3D visualisation. The entire pipeline is run on all embeddings, or every token in every sequence of the same length. This is done to ensure that they occupy the same representational space and thus can be compared both within and between sequences. When the embedding dimension is less than three, the probing points are appended to the embeddings. These probing points are used to visualise the decision regions with the algorithm 3. When the embedding dimension is 3 or more, the hidden space is visualised with isolines created with algorithm 2. For feedforward models, the unrolled output is converted into a visual representation using algorithm 4. It is then combined with the unrolled hidden state created with algorithm 1 through algorithm 5.

A.3 Spectral Learning

Algorithm 6 describes the Spectral Learning method, loosely based on (Balle and Mohri 2012).

Algorithm 7 is the hyperparameter search procedure for algorithm 6. The Hankel basis is a hyperparameter for both SVD and NMF. The tolerance, which controls the minimum size of valid singular values and therefore the number of WFSA states, is also a hyperparameter. However, its value (10%) was found through preliminary testing rather than grid-search. For NMF, there are two other hyperparameters: initialisation method and dimension shuffling. Ternariness is a constraint that selects the WFSA whose parameters are close to ± 1 or 0, for better interpretability. The validation data is a superset of the training data, enabling unsupervised learning where the absence of a sequence $w = p \cdot s$ is considered a negative example.[35]

[35] Strictly speaking, this mode of learning is actually two separate induction tasks: matrix completion first, followed by parameter estimation (Balle and Mohri 2012). However, for expediency, I assumed that missing cells in the Hankel matrix represent negative examples. This SL-with-missing-entries approach resembles NCE (Gutmann and Hyvärinen 2010). In NCE, negative examples are created by concatenating words sampled from a unigram or uniform distribution, whereas here the negative examples are created by concatenating prefixes and suffixes that are not in the training data.

Algorithm 1: embedding_to_view(states, radius)

```
   Input: $\mathbf{X} \in \mathbb{R}^{N \times D}, rad \in \mathbb{R}^{+}$
   Output: $\mathbf{Y} \in \mathbb{R}^{N \times 3}$
 1 if $d > 3$ then
 2     $\mathbf{X}^{N \times D} \leftarrow PCA(\mathbf{X}^{N \times D})$ ; // reduces with PCA set to preserve $99.9\%$ of the explained variance
 3 if $d > 3$ then
 4     $\mathbf{X}^{N \times D} \leftarrow sparsePCA(\mathbf{X}^{N \times D})$ ; // reduces with sparse PCA to 3 dimensions
 5 $\mathbf{X}^{N \times D} \leftarrow quantise(\mathbf{X}^{N \times D}, rad)$ ; // merges embeddings within a distance $rad$ of each other
 6 if $d < 3$ then
 7     $\mathbf{X}^{N \times 3} \leftarrow [\mathbf{0}^{N \times (3-D)} \mid \mathbf{X}^{N \times D}]$ ; // pads to 3 dimensions with 0s
 8 $\mathbf{X}^{N \times 3} \leftarrow normalise(\mathbf{X}^{N \times 3})$ ; // normalises each dimension to [-1, 1]
 9 return $\mathbf{Y}^{N \times 3} \leftarrow \mathbf{X}^{N \times 3}$
```

Algorithm 2: isolines(states, decoder, tol, steps)

```
   Input: $\mathbf{X}_{m,n} \in \mathbb{R}^{N \times D}, decoder() : \mathbb{R}^{D} \rightarrow \mathbb{R}^{2}, tol \in \mathbb{R}^{+}, steps \geq 2 \quad steps \in \mathbb{N}^{+}$
   Output: $Y = (\mathcal{P}(\{1, \dots, m\}))$
   Data: $coaffinity \leftarrow False, done \leftarrow \emptyset, coaffines : s \in \{1, \dots, m\} \rightarrow \mu \in \mathcal{P}(\{1, \dots, m\})$
 1 $\mathcal{R} \leftarrow \binom{m}{2}$ ; // finds all unique pairs of embedding indices
 2 for $\{i, j\} \in \mathcal{R}$ do
 3     $\mathcal{P} \leftarrow \binom{\{decoder((1-t)X_{i,:} + tX_{j,:}) \mid t \in \{\frac{0}{steps-1}, \frac{1}{steps-1} \dots \frac{steps-1}{steps-1}\}\}}{2}$ ; // computes probing steps along each dimension
 4     $coaffinity \leftarrow True$
 5     for $\{p_a, p_b\} \in \mathcal{P}$ do
 6         if $similar(p_a, p_b, tol) = False$ then
 7             $coaffinity \leftarrow False$
 8             break
 9     if $coaffinity = True$ and $j \notin done$ then
10         $coaffines \leftarrow coaffines \cup [i \rightarrow \{e, \dots\} \cup \{j\}]$
11         $done \leftarrow done \cup \{i, j\}$
12 return $Y \leftarrow sort(\{s_q\} \cup \{\mu_q\} \mid coaffines : s_q \rightarrow \mu_q)$ ; // sets of coaffine embedding indices
```

Algorithm 3: decision_regions(states, decoder, classes, min_dist, steps)

```
    Input: $\mathbf{X} \in \mathbb{R}^{N\times D}, decoder() : \mathbb{R}^D \rightarrow \mathbb{R}^2, classes = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_C \mid; c \in \mathbb{R}^2\}, min\_dist \in \mathbb{R}^+, steps \in \mathbb{N}^+$
    Output: $R = (M_{c_1}, M_{c_2}, \ldots),\ M \in \mathbb{R}^{R_{c_k} \times D}$
1   $L \leftarrow (\{l_j, u_j\} \mid l_j \le x_{:,j} \le u_j),\ x_{i,j} \in X$ ; // computes limits
2   $S \leftarrow (l + \frac{i-1}{steps-1} \cdot (u - l) \mid \{l, u\} \in L,\ i \in \{1, 2, \ldots, steps\})$ ; // computes probing steps along each dimension
3   $\mathbf{G} \leftarrow decoder(S \times S \times \cdots \times S)$ ; // builds grid of probing points
4   $R \leftarrow ([\mathbf{x}_1; \mathbf{x}_2; \ldots; \mathbf{x}_r]_{c_k} \mid \|\mathbf{x}_i - \mathbf{c}_k\| \le \text{min\_dist} \cdot \sqrt{2}), \quad k \in \{1, 2, \ldots, C\},\ \mathbf{x}_* \in \mathbf{G}_{i,:}$ ; // finds points within $min\_dist$ of a class
5   return $R$
```

Algorithm 4: output_to_view(outputs, radius)

```
    Input: $\mathbf{X} \in \mathbb{R}^{N\times 2}, rad \in \mathbb{R}^+$
    Output: $\mathbf{Y} \in \mathbb{R}^{N\times 3}$
1   $\mathbf{X}^{N\times 2} \leftarrow quantise(\mathbf{X}^{N\times 2}, rad)$ ; // merges embeddings within a distance $rad$ of each other
2   $\mathbf{X}^{N\times 3} \leftarrow [\mathbf{0}^{N\times 1} \mid \mathbf{X}^{N\times 2}]$ ; // pads to 3 dimensions with 0s
3   $\mathbf{X}^{N\times 3} \leftarrow normalise(\mathbf{X}^{N\times 3})$ ; // normalises each dimension to [-1, 1]
4   return $\mathbf{Y}^{N\times 3} \leftarrow \mathbf{X}^{N\times 3}$
```

Algorithm 5: combine_levels(counter, fsa)

```
    Input: $\mathbf{C} \in \mathbb{R}^{N\times 3}, \mathbf{F} \in \mathbb{R}^{N\times 3}$
    Output: $\mathbf{C} \in \mathbb{R}^{N\times 3}, \mathbf{F} \in \mathbb{R}^{N\times 3}$
1   $\mathbf{F} \leftarrow [\mathbf{F}_{:,:2} \mid \mathbf{Z}^{N\times 1}],\ \mathbf{Z} = [z_i + g],\ g \ge \max_i(c_{i,3} - f_{i,3}) + .5$ ; // shifts FSA level above counter level
2   $\mathbf{C} \leftarrow [\mathbf{C}_{:,:2} \mid \mathbf{Z}^{N\times 1}],\ \mathbf{Z} = [z_i] = -1 + (C_{i,3} - \min_i C_{i,3}) \cdot \frac{.75+1}{\max_i F_{i,3} - \min_i C_{i,3}}$ ; // normalises $\mathbf{C}_{:,3}$ to [-1, .75]
3   $\mathbf{F} \leftarrow [\mathbf{F}_{:,:2} \mid \mathbf{Z}^{N\times 1}],\ \mathbf{Z} = [z_i] = -1 + (F_{i,3} - \min_i C_{i,3}) \cdot \frac{.75+1}{\max_i F_{i,3} - \min_i C_{i,3}}$ ; // normalises $\mathbf{F}_{:,3}$ to [-1, .75]
4   $\mathbf{R}^* \leftarrow \arg\min_{\mathbf{R} \in \mathbb{R}^{3\times 3}} \frac{1}{2} \sum_{i=1}^{N} \|\mathbf{C}_{i,:} - \mathbf{R}\mathbf{F}_{i,:}\|^2$ ; // calculates best rotation with Kabsh algorithm
5   $\mathbf{F} \leftarrow \mathbf{R}\mathbf{F}$ ; // rotates FSA level to maximise vertex alignment and minimise crossing linking lines
6   return $\mathbf{C}, \mathbf{F}$
```

Algorithm 6: learn_wfsa(vocab, training_data, method, num_prefixes, num_suffixes, init, shuffle)

```
    Input: $\Sigma = \{0, 1\}, T \subset \Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, ...\}$, $method = \{$SVD, NMF$\}$, $n_p$, $n_s$, $init$, $shuffle$
    Output: WFSA
 1  $P \leftarrow \arg\,\text{top}_n\{freq(p)\}$ , $p \in U = \{u|x = u \cdot v\}$ , $x \in \Sigma^*$   ; // picks the $n_p$-most frequent prefixes as row indices to the
      Hankel matrix
 2  $S \leftarrow \arg\,\text{top}_n\{freq(s)\}$ , $s \in V = \{v|x = u \cdot v\}$ , $x \in \Sigma^*$   ; // picks the $n_s$-most frequent suffixes as column indices to
      the Hankel matrix
 3  $P \leftarrow P' \cdot \Sigma \cup \epsilon$   ; // appends root prefixes with vocabulary plus empty string to give p-closed prefixes
 4  $\mathbf{H}(p, s) \leftarrow \min(freq(p \cdot s), 1)$ , $\mathbf{H} \in \mathbb{R}^{|P| \times |S|}$   ; // fills Hankel matrix
 5  $\mathbf{H}_\epsilon \leftarrow \mathbf{P}^{|P| \times k}\mathbf{S}^{k \times |S|}$   ; // computes a k-rank factorization, k being the number of singular values that are at least
      10% of the largest singular value
 6  if $method = SVD$ then
 7      $\mathbf{H}_\epsilon \leftarrow \mathbf{U\Sigma V}^\top = \text{SVD}(\mathbf{H}_\epsilon)$
 8      $\mathbf{P} \leftarrow \mathbf{U\Sigma}^{\frac{1}{2}}$
 9      $\mathbf{S} \leftarrow \mathbf{\Sigma}^{\frac{1}{2}}\mathbf{V}^\top$
10  else
11      $\mathbf{H}_\epsilon \leftarrow \mathbf{WH} = \text{NMF}(\mathbf{H}_\epsilon, init, shuffle)$
12      $\mathbf{P} \leftarrow \mathbf{W}$
13      $\mathbf{S} \leftarrow \mathbf{H}$
14  estimate k-state WFSA:
15    $\boldsymbol{\alpha} \leftarrow (\mathbf{S}^+)^\top \mathbf{H}_{\epsilon\,1,:}$   (initial vector) ;
16    $\boldsymbol{\beta} \leftarrow \mathbf{H}_{\epsilon\,:,1}(\mathbf{P}^+)^\top$   (final vector) ;
17    $\mathbf{A}_\sigma \leftarrow (\mathbf{P}^+\mathbf{H}_\epsilon\mathbf{S}^+)^\top$   (transition matrices) ;
18  return $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\mathbf{T}_{i,j,k} = (\mathbf{A}_{\sigma_k})_{i,j}$
```

Algorithm 7: sl_search(vocab, validation_data, training_data, hyperparameter_sets, ternariness_threshold)

```
**Input:** $\Sigma = \{0,1\}, D \subset \Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, ...\}, T \subset D, H = \{h_k\}, \gamma \in [.5, 1]$
**Output:** $\mathrm{WFSA}^*$
**Data:** $L^* \leftarrow \infty, \mathrm{WFSA}^* \leftarrow \emptyset, \tau^* \leftarrow 0$
for $c_k \in C$ do
    $C \leftarrow h_1 \times h_2 \times \cdots \times h_n \ , \ h_k \in H$    ; // computes hyperparameter combinations
    $\mathrm{WFSA} \leftarrow \alpha,\ \beta,\ T_{i,j,k} = \mathrm{learn\_wfsa}(\Sigma, T, \mathrm{NMF}, c_k = \{n_p, n_s, init, shuffle\})$    ; // (Algorithm 6)
    $L \leftarrow \ell_2(\mathrm{WFSA}, D)$    ; // computes validation loss
    if $L < L^*$ then
        $L^* \leftarrow L$
        $\mathrm{WFSA}^* \leftarrow \mathrm{WFSA}$
    else
        if $L = L^*$ then
            $\tau \leftarrow \frac{\sum_{i=1}^{n} (\mathbb{1}_{\{|p_i| \leq \gamma\}} + \mathbb{1}_{\{1-\gamma \leq |p_i| \leq +\gamma\}})}{n} \quad \forall\ p_i \in \mathrm{WFSA},\ n = |\mathrm{WFSA}|$    ; // computes ternariness
            if $\tau > \tau^*$ then
                $\tau^* \leftarrow \tau$
                $\mathrm{WFSA}^* \leftarrow \mathrm{WFSA}$
    if $L < 1e^{-6}$ then
        return $\mathrm{WFSA}^*$
return $WFSA^*$
```

A.4 Representation Theory

Representation Theory is the branch of mathematics that provides the foundation for the linear representation of WFSAs. It studies the homomorphisms between algebraic structures like groups, monoids and semigroups, and linear transformations in vector spaces. In what follows I present only the bare minimum needed to understand its application to automata.

A.4.1 Groups

A group is an algebraic structure formally defined as a tuple $\langle S, \circ \rangle$ where S is a set of elements and $\circ$ a binary operation with the following properties:

- **Closure:** $a \circ b \in S$ The result of the operation is also a member of the same set.
- **Associativity:** $a \circ (b \circ c) = (a \circ b) \circ c$ The order of the operation in a chain of operations is irrelevant.
- **Identity Element:** $a \circ e = e$ A unique element that gives back the left/right symbol when operated on.
- **Inverses:** $a \circ a^{-1} = e$ For each element a, there exists another element a^{-1} such that when combined, it returns the identity element.

A monoid is like a group but without inverses.

A.4.2 General Linear Group

The General Linear Group over the field of reals of dimension n $GL(n, \mathbb{R})$ is an example of a group $\langle A, \cdot \rangle$ where $A \in \mathbb{R}^{n \times n}$ is the set of invertible matrices and $\circ$ is matrix multiplication with properties:

- **Closure:** $A \cdot A' \in S$
- **Associativity:** $A \cdot (A' \cdot A'') = (A \cdot A') \cdot A''$
- **Identity Element:** $A \cdot I = A$
- **Inverses:** $A \cdot A^{-1} = I$

The Matrix Monoid $M(n, \mathbb{R})$ is like $GL(n, \mathbb{R})$ but without inverses.

A.4.3 Free Group

The free group $F(\Sigma)$ over Σ (Epstein 1992) is an example of a group $\langle F(\Sigma), \cdot \rangle$ where $\Sigma = \{\sigma, \sigma^{-1}, \sigma' \ldots\}$ is a set of symbols called the vocabulary, alphabet or generator; $\Sigma^* = \{\epsilon, \sigma, \sigma^{-1}, \sigma \cdot \sigma, \ldots\}$ is the set of strings generated from Σ via concatenation; and $\cdot$ is the concatenation operator with properties:

- **Closure:** $\sigma \cdot \sigma' \in \Sigma^*$
- **Associativity:** $\sigma \cdot (\sigma' \cdot \sigma'') = (\sigma \cdot \sigma') \cdot \sigma''$
- **Identity Element:** $\sigma \cdot \epsilon = \sigma$ (the empty string)
- **Inverses:** $\sigma \cdot \sigma^{-1} = \epsilon$ (the formal inverse)

The inverses in this group are defined by an external criterion independent of the group operation, which in our case is the action of the free group on the states on an FSA (see section A.4.8).

A free monoid is like a free group but without inverses.

A.4.4 Group Actions

A Group action α is a mapping from the cartesian product of a group $\langle G, \circ \rangle$ and an underlying set S to S:

$\alpha : G \times S \rightarrow S$

with properties:

- **Identity:** $\alpha(\epsilon, x) = x$
- **Compatibility:** $\alpha(g, \alpha(h, x)) = \alpha(g \circ h, x)$

for $\epsilon, g, h \in G$ and $x \in S$.

A Monoid Act is formally identical to a Group Action when G is replaced with a monoid M.

A.4.5 Linear Actions

A linear action is an example of a group action where the group is $GL(n, \mathbb{R})$ and the underlying set is a vector space V. Then:

- **Identity:** $\alpha(\epsilon, x) = I \cdot x = x$
- **Compatibility:** $\alpha(g, \alpha(h, x)) = A(A' \cdot x) = (A \cdot A')x$

for $A, A' \in \mathbb{R}^{n \times n}$ and $x \in \mathbb{R}^n$.

A linear act is formally identical to a linear action when $GL(n, \mathbb{R})$ is replaced with $M(n, \mathbb{R})$.

A.4.6 Automaton Transformation Group

The automaton transformation group (Holcombe 2004) is an example of a group action where the group is $F(\Sigma)$ and the underlying set Q contains the states of a deterministic FSA. Then:

- **Identity:** $\alpha(\epsilon, x) = \epsilon \cdot q = q$
- **Compatibility:** $\alpha(g, \alpha(h, x)) = \sigma(\sigma' \cdot q) = (\sigma \cdot \sigma)q$

for $\epsilon, \sigma, \sigma' \in \Sigma^*$ and $q \in Q$.

The computation of a path in an DFSA can be modelled by repeated application of the action, starting with the first input symbol until we either get to a final state (valid path) or the computation is halted if there's no existing transition for the current input-state pair.

An automaton transformation monoid is formally equivalent to an automaton transformation group where strings don't have formal inverses.

A.4.7 Group Representation

A representation of a group $\langle G, \circ \rangle$ on a vector space V over a real field $\mathbb{R}$ is a group homomorphism f from G to $GL(n, \mathbb{R})$. That is, a map such that:

$$\begin{aligned} f(g \circ h) &= f(g) \cdot f(h) \\ f(g^{-1}) &= f(g)^{-1} \\ f(e) &= I \end{aligned}$$

for $e, g, h \in G \textit{ and } I, f(g), f(h) \in GL(n, \mathbb{R})$.

A.4.8 Linear Representation of an FSA

An FSA's linear representation is a type of group representation where the mapped group is the free group $F(\Sigma)$. I will illustrate it with an example for inputs $\{\epsilon, 0, 1, 00, 01, 10, 11\}$ to the XOR-FSA:

$$\begin{aligned}
\Sigma\ &= \{0, 1\}\\
\Sigma^* &= \{\epsilon, 0, 1, 00, 01, 10, 11, ...\}
\end{aligned}$$

$$\begin{aligned}
f(\epsilon) &= I\\
f(0) &= I\\
f(1) &= F
\end{aligned}$$

$$\begin{aligned}
f(\epsilon^{-1}) &= f(\epsilon)^{-1} = I^{-1} = I\\
f(0^{-1}) &= f(0)^{-1} = I^{-1} = I\\
f(1^{-1}) &= f(1)^{-1} = F^{-1} = F
\end{aligned}$$

$$\begin{aligned}
f(00) &= f(0) \cdot f(0)\ = I \cdot I = I\\
f(01) &= f(0) \cdot f(1) = I \cdot F = F\\
f(10) &= f(1) \cdot f(0) = F \cdot I = F\\
f(11) &= f(1) \cdot f(1) = F \cdot F = I
\end{aligned}$$

where I is the identity matrix and F is the (involutory) reflection or exchange matrix. It is apparent that $\epsilon, 0$ *and* 1 are their own formal inverses as determined by their representation as matrices.

The linear representation, together with the action of $F(\Sigma)$ on Q, and a mapping h from states to vectors, defines the computation both on the algebraic and vectorial sides:

$$\begin{aligned}
&Q = \{F, T\}\\
&V \in \mathbb{R}^2\\
&h : Q \to V
\end{aligned}$$

$$\begin{aligned}
h(F) &\to [1\ 0]\\
h(T) &\to [0\ 1]
\end{aligned}$$

$$\begin{aligned}
\alpha(0, F) &\to F\\
\alpha(0, T) &\to T\\
\alpha(1, F) &\to T\\
\alpha(1, T) &\to F
\end{aligned}$$

$$\begin{aligned}
\alpha(f(0), h(F)) &\to I \cdot [1\ 0] = [1\ 0]\\
\alpha(f(0), h(T)) &\to I \cdot [0\ 1] = [0\ 1]\\
\alpha(f(1), h(F)) &\to F \cdot [1\ 0] = [0\ 1]\\
\alpha(f(1), h(T)) &\to F \cdot [0\ 1] = [1\ 0]
\end{aligned}$$

Again, $\epsilon, 0$ and 1 are found to be their own formal inverses as determined by their effect on the state transitions.

Similarly for the OR-FSA, though in this case, it's based on a monoid, as here not all strings have inverses:

$$\begin{aligned} f(\epsilon) &= I \\ f(0) &= I \\ f(1) &= P \end{aligned}$$

$$\begin{aligned} f(00) &= f(0) \cdot f(0) = I \cdot I = I \\ f(01) &= f(0) \cdot f(1) = I \cdot P = P \\ f(10) &= f(1) \cdot f(0) = P \cdot I = P \\ f(11) &= f(1) \cdot f(1) = P \cdot P = P \end{aligned}$$

$$\begin{aligned} \alpha(0, F) &\to F \\ \alpha(0, T) &\to T \\ \alpha(1, F) &\to T \\ \alpha(1, T) &\to T \end{aligned}$$

$$\begin{aligned} \alpha(f(0), h(F)) &\to I \cdot [1\ 0] = [1\ 0] \\ \alpha(f(0), h(T)) &\to I \cdot [0\ 1] = [0\ 1] \\ \alpha(f(1), h(F)) &\to P \cdot [1\ 0] = [1\ 0] \\ \alpha(f(1), h(T)) &\to P \cdot [0\ 1] = [1\ 0] \end{aligned}$$

where I is the identity matrix and P is the projection-onto-x matrix. Figure 41 shows a commutative diagram summarising the relationships between the algebraic structures underlying the linear representation of FSAs.

$$\begin{tikzcd} F(\Sigma)/M(\Sigma) \arrow[r, "f \text{ homomorphism}"] \arrow[d, "\alpha \text{ acts on}"'] & GL(n,\mathbb{R})/M(n,\mathbb{R}) \arrow[d, "\alpha \text{ acts on}"] \\ Q \arrow[r, "h \text{ maps to}"] & \mathbb{R}^n \end{tikzcd}$$

Figure 41

A.5 Attention matrices

Tables A.5 and A.5 show the attention matrices of the OR- and AND-TFMs for all inputs of length 4.

	CLS	0	0	0	0
CLS	.20	.20	.20	.20	.20
0	.20	.20	.20	.20	.20
0	.20	.20	.20	.20	.20
0	.20	.20	.20	.20	.20
0	.20	.20	.20	.20	.20

	CLS	0	0	0	1
CLS					1.0
0					1.0
0					1.0
0					1.0
1	.25	.25	.25	.25	

	CLS	0	0	1	0
CLS				1.0	
0				1.0	
0				1.0	
1	.25	.25	.25		.25
0				1.0	

	CLS	0	0	1	1
CLS				.50	.50
0				.50	.50
0				.50	.50
1	.33	.33	.33		
1	.33	.33	.33		

	CLS	0	1	0	0
CLS			1.0		
0			1.0		
1	.25	.25		.25	.25
0			1.0		
0			1.0		

	CLS	0	1	0	1
CLS			.50		.50
0			.50		.50
1	.33	.33		.33	
0				.50	.50
1	.33	.33		.33	

	CLS	0	1	1	0
CLS			.50	.50	
0			.50	.50	
1	.33	.33			.33
1	.33	.33			.33
0			.50	.50	

	CLS	0	1	1	1
CLS			.33	.33	.33
0			.33	.33	.33
1	.50	.50			
1	.50	.50			
1	.50	.50			

	CLS	1	0	0	0
CLS		1.0			
1	.25		.25	.25	.25
0		1.0			
0		1.0			
0		1.0			

	CLS	1	0	0	1
CLS		.50			.50
1	.33		.33	.33	
0		.50			.50
0		.50			.50
1	.33		.33	.33	

	CLS	1	0	1	0
CLS		.50		.50	
1	.33		.33		.33
0		.50		.50	
1	.33		.33		.33
0		.50		.50	

	CLS	1	0	1	1
CLS		.33		.33	.33
1	.50		.50		
0		.33		.33	.33
1	.50		.50		
1	.50		.50		

	CLS	1	1	0	0
CLS		.50	.50		
1	.33			.33	.33
1	.33			.33	.33
0		.50	.50		
0		.50	.50		

	CLS	1	1	0	1
CLS		.33	.33		.33
1	.50			.50	
1	.50			.50	
0		.33	.33		.33
1	.50			.50	

	CLS	1	1	1	0
CLS		.33	.33	.33	
1	.50				.50
1	.50				.50
1	.50				.50
0		.33	.33	.33	

	CLS	1	1	1	1
CLS		.25	.25	.25	.25
1	1.0				
1	1.0				
1	1.0				
1	1.0				

Table 10: OR-TFM's attention matrices for all inputs of length 4 plus the initial dummy/CLS token. Each row contains the attention weights for each token rounded to 2 decimals. 0 weights are left out for clarity.

	CLS	0	0	0	0
CLS		.25	.25	.25	.25
0	1.0				
0	1.0				
0	1.0				
0	1.0				

	CLS	0	0	0	1
CLS		.33	.33	.33	
0	.50				.50
0	.50				.50
0	.50				.50
1		.33	.33	.33	

	CLS	0	0	1	0
CLS		.33	.33		.33
0	.50			.50	
0	.50			.50	
1		.33	.33		.33
0	.50			.50	

	CLS	0	0	1	1
CLS		.50	.50		
0	.33			.33	.33
0	.33			.33	.33
1		.50	.50		
1		.50	.50		

	CLS	0	1	0	0
CLS		.33		.33	.33
0	.50		.50		
1		.33		.33	.33
0	.50		.50		
0	.50		.50		

	CLS	0	1	0	1
CLS		.50		.50	
0	.33		.33		.33
1		.50		.50	
0	.33		.33		.33
1		.50		.50	

	CLS	0	1	1	0
CLS		.50			.50
0	.33		.33	.33	
1		.50			.50
1		.50			.50
0	.33		.33	.33	

	CLS	0	1	1	1
CLS		1.0			
0	.25		.25	.25	.25
1		1.0			
1		1.0			
1		1.0			

	CLS	1	0	0	0
CLS			.33	.33	.33
1			.33	.33	.33
0	.50	.50			
0	.50	.50			
0	.50	.50			

	CLS	1	0	0	1
CLS			.50	.50	
1			.50	.50	
0	.33	.33			.33
0	.33	.33			.33
1			.50	.50	

	CLS	1	0	1	0
CLS			.50		.50
1			.50		.50
0	.33	.33		.33	
1			.50		.50
0	.33	.33		.33	

	CLS	1	0	1	1
CLS			1.0		
1			1.0		
0	.25	.25		.25	.25
1			1.0		
1			1.0		

	CLS	1	1	0	0
CLS				.50	.50
1				.50	.50
1				.50	.50
0	.33	.33	.33		
0	.33	.33	.33		

	CLS	1	1	0	1
CLS				1.0	
1				1.0	
1				1.0	
0	.25	.25	.25		.25
1				1.0	

	CLS	1	1	1	0
CLS					1.0
1					1.0
1					1.0
1					1.0
0	.25	.25	.25	.25	

	CLS	1	1	1	1
CLS	.20	.20	.20	.20	.20
1	.20	.20	.20	.20	.20
1	.20	.20	.20	.20	.20
1	.20	.20	.20	.20	.20
1	.20	.20	.20	.20	.20

Table 11: AND-TFM's attention matrices for all inputs of length 4 plus the initial dummy/CLS token. Each row contains the attention weights for each token rounded to 2 decimals. 0 weights are left out for clarity.