Fully Understanding a Sequence Model of Tic-Tac-Toe

Andy Lo
Department of Computer Science
University of Cambridge
`cyal4@cam.ac.uk`

Abstract

While Large Language Models (LLMs) demonstrate strong capabilities from simple sequence modelling, its inner workings remain as a black box. As a step towards bridging this gap, this project meticulously studies TicTacGPT, a toy sequence model trained on Tic-Tac-Toe game sequences. By deconstructing the internals of TicTacGPT, down to the level of individual neurons, I build a complete understanding of how TicTacGPT solves the task. The results show that TicTacGPT correctly learns the rules of the game and makes its predictions logically. It however does not learn to respect every symmetry of the task, revealing the limits of next-token prediction models. The repository is available at `https://github.com/andylolu2/tic-tac-gpt`.

1 Introduction

While the success of deep learning in practice is ubiquitous, interpreting and explaining the behaviour of neural networks remains a difficult problem. The inability to understand the models' internal components is a significant obstacle to building reliable machine learning systems since typical evaluation methods provide no guarantee on out-of-distribution inputs. Mechanistic interpretability is an emerging approach to explainability by focusing on tractable, low-level components of a model, using them as an entry point to build an overall understanding of a model's behaviour. The aim is to determine whether neural networks only "appear" to be correct by stacking statistical correlations or truly reason about the task in a logical manner. This project aims to build upon this line of work, showcasing another example where models are not black boxes and can be made human-interpretable with sufficient effort [9, 17, 24].

There has been particular interest in understanding sequence models trained with next-token prediction, specifically on whether next-token prediction is sufficient for learning accurate world models. On one hand, there exist theoretical arguments [3, 15] that such models merely learn statistical correlations and thus cannot understand the causal relationships of the underlying data generation process. There is also empirical evidence that supports this view [13]. On the other hand, some empirical studies reveal opposing evidence where models appear to build accurate world models internally [14, 24]. Unfortunately, models used in practice are often too complex to be understood fully as many factors like data selection and training optimisations are not controlled for. Several works have thus shifted their focus to simpler sequence models on synthetic datasets [11, 25] where the task is well-understood.

This project studies autoregressive transformers trained on synthetic game sequences of Tic-Tac-Toe. The high-level goal is to understand how a next-token prediction model learns to solve this task. The simplistic nature of the task permits the use of a one-layer transformer, TicTacGPT, that still solves the task perfectly. This greatly simplifies the analysis and allows us to fully reverse engineer the model, down to the level of understanding what features of the inputs activate which neurons and their effects on the model's predictions.

2 Background

This section covers the necessary background to understand the contributions of this project. Section 2.1 provides an overview of the most relevant works and how they relate to this project. Section 2.2 describes the framework and the notation used later in this paper to describe the components of a transformer.

2.1 Related works

There are vast amount of literature on demystifying how neural networks make their predictions. One direction of study attempts to interpret the meaning of neurons in a network and has found many examples where neurons capture human-interpretable concepts such as objects and features in images [2, 4, 29], tense, language and gender in natural language [1, 8], or context in code [10]. While such analysis can identify data features captured by the model, it does not characterise how the neurons are related to the inputs or affect the outputs and thus does not fundamentally solve the black-box problem. Mechanistic interpretability is an emerging field that aims to find the underlying mechanism of how a model makes its predictions. Similar to this project, most work in mechanistic interpretability focuses on simplified models as they are the most tractable to analyse [5, 7, 17, 20].

Toshniwal et al. [25] were the first to probe for internal world presentations of next-token prediction models. The authors reveal that models trained to predict the next moves in chess track the state of the board internally. Li et al. [11] extend this idea by studying a next-move prediction model trained on Orthello. They demonstrate that not only do the models track the board state, interventions on the internal representation also have the expected causal effect on the model's outputs. This provides strong evidence that models trained on sequence modelling can in fact "understand" the nature of the task rather than simply memorising statistical patterns. Nanda et al. [18] further reveals that the board state can be expressed as a linear projection of the internal representation. While such works identify core properties of the model like its internal representations, they still fail to explain the model behaviour in full. For example, no explanation is offered as to *how* the internal board representations are computed. This project aims to close this gap by narrowing it down to even simpler tasks and models.

2.2 Mathematical framework of transformers

To conveniently reason about various components of the transformer [26], it is helpful to set up a mathematical description of it [5]. While the following description may be verbose, several simplifications can be made when combined with empirical evidence (Section 4). The notation will only cover one-layer transformers since that is the main focus of this project.

Let $\mathbf{E} \in \mathbb{R}^{s\times v}$ be the one-hot encoded input tokens, where s is the sequence length and v is the vocabulary size. The embedding layer with learnt positional embeddings can be expressed as:

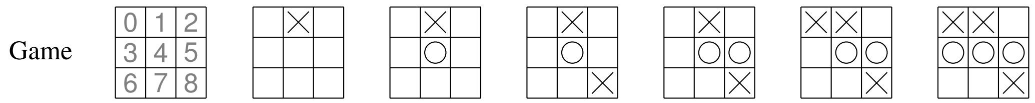
$$\text{Embed}(\mathbf{E}) = \mathbf{E}\mathbf{W}_E + \mathbf{W}_P \tag{1}$$

where $\mathbf{W}_E \in \mathbb{R}^{v\times d}$, $\mathbf{W}_P \in \mathbb{R}^{s\times d}$ and d is the hidden dimension of the model.

The core component of the transformer is the attention layer. In this project, we focus on the multi-head, casual, self-attention variant. Each attention head i is parameterised by the query $\mathbf{W}_Q^{(i)} \in \mathbb{R}^{d\times h}$, $\mathbf{b}_Q^{(i)} \in \mathbb{R}^{1\times h}$, key $\mathbf{W}_K^{(i)} \in \mathbb{R}^{d\times h}$, $\mathbf{b}_K^{(i)} \in \mathbb{R}^{1\times h}$, value $\mathbf{W}_V^{(i)} \in \mathbb{R}^{d\times h}$, $\mathbf{b}_V^{(i)} \in \mathbb{R}^{1\times h}$ and output $\mathbf{W}_O^{(i)} \in \mathbb{R}^{h\times d}$, $\mathbf{b}_O^{(i)} \in \mathbb{R}^{1\times d}$ projections, where h is the head dimension. The attention operator is defined as:

$$\begin{aligned}\text{Attn}(\mathbf{X}) &= \sum_i \mathbf{A}^{(i)}(\mathbf{X}\mathbf{W}_V^{(i)} + \mathbf{1}_s\mathbf{b}_V^{(i)})\mathbf{W}_O^{(i)} + \mathbf{1}_s\mathbf{b}_O^{(i)} \\ \mathbf{A}^{(i)} &= \text{Softmax}\left(\text{Tril}\left(\left(\mathbf{X}\mathbf{W}_Q^{(i)} + \mathbf{1}_s\mathbf{b}_Q^{(i)}\right)\left(\mathbf{X}\mathbf{W}_K^{(i)} + \mathbf{1}_s\mathbf{b}_K^{(i)}\right)^T, -\infty\right)\right)\end{aligned} \tag{2}$$

where $\mathbf{1}_s \in \mathbb{R}^{s\times 1}$ is a vector of ones and $\text{Tril}(\mathbf{X}, y)$ fills the upper triangular matrix of $\mathbf{X}$ with y. The typical scaling by a factor of $1/\sqrt{d}$ in the attention matrix is ignored as it is a linear operation and thus can be folded into the weights.



Input	`[B]`	`1`	`4`	`8`	`5`	`0`	`3`
Target	`1`	`4`	`8`	`5`	`0`	`3`	`[O]`

Figure 1: Example of a Tic-Tac-Toe game and its corresponding inputs and next-token prediction targets. The numbering of the board positions are shown in the first board (top left). `[B]` is the special beginning-of-sequence token and `[O]` denotes the end of the game with the result "`O` wins".

The other building block is the MLP block which is given by:

$$\text{MLP}(\mathbf{X}) = \phi(\mathbf{X}\mathbf{W}_{in} + \mathbf{1}_s\mathbf{b}_{in})\mathbf{W}_{out} + \mathbf{1}_s\mathbf{b}_{out}$$

where $\mathbf{W}_{in} \in \mathbb{R}^{d\times 4d}$, $\mathbf{b}_{in} \in \mathbb{R}^{1\times 4d}$, $\mathbf{W}_{out} \in \mathbb{R}^{4d\times d}$, $\mathbf{b}_{out} \in \mathbb{R}^{1\times d}$ and ϕ is a non-linearity.

Putting everything together, the transformer output $\mathbf{Y} \in \mathbb{R}^{s\times v}$ can be expressed as:

$$\begin{aligned} \mathbf{H}_0 &= \text{Embed}(\mathbf{E}) \\ \mathbf{H}_1 &= \mathbf{H}_0 + \text{Attn}(\text{LN}(\mathbf{H}_0)) \\ \mathbf{H}_2 &= \mathbf{H}_1 + \text{MLP}(\text{LN}(\mathbf{H}_1)) \\ \mathbf{Y} &= \text{LN}(\mathbf{H_2})\mathbf{W}_U + \mathbf{1}_s\mathbf{b}_U \end{aligned} \tag{3}$$

where $\mathbf{W}_U \in \mathbb{R}^{d\times v}$, $\mathbf{b}_U \in \mathbb{R}^{1\times v}$ and $\text{LN}(\mathbf{X})$ represents a LayerNorm which normalises $\mathbf{X}$ column-wise by subtracting the mean and dividing by the standard deviation. The scale and bias parameters of LayerNorm are ignored as they can be folded into the linear projections that proceed them.

3 TicTacGPT

3.1 Synthetic task and dataset

Following [11], each game is represented as the sequence of moves played, where each of the 9 possible moves is encoded as a separate token. The special beginning-of-sequence token `[B]` is prepended to every game sequence. A result token also is appended, it is one of `[X]`, `[O]` or `[D]`, representing "`X` wins", "`O` wins" and "Draw" respectively. An example of an encoded game is shown in Fig. 1. Ignoring rotational and reflectional symmetries there are 255,168 unique games of Tic-Tac-Toe. The train and test dataset is constructed by randomly splitting all the games into two sets of equal size.

One might naively think that the task only involves learning what moves are legal and deciding the final outcome of the game. However, there are nuances to the training objective. Since the model is trained on all possible game sequences, the true task is to learn the proportion of possible continuations for each possible next move. For example, the optimal model should assign a lower probability to immediately winning moves (one that creates a three-in-a-row) as there is only one continuation (either "`X` wins" or "`O` wins") while non-winning moves have more possible continuations. As shown later, the trained model captures such complexities of the task as well.

3.2 Model and experiment details

The training of TicTacGPT mostly follows standard practices [21] for typical prefix LLMs. TicTacGPT is a one-layer transformer with 128 hidden dimensions, 2 attention heads, 512 MLP dimensions, learnt positional encodings and uses the pre-norm architecture [28]. The only non-standard architectural choice is the use of the Softmax Linear Units (SoLU) [6] non-linearity in the MLP:

$$\text{SoLU}(\mathbf{X}) = \text{LN}(\mathbf{X} \cdot \text{Softmax}(\mathbf{X}))$$

Elhage et al. [6] demonstrated that SoLU increases the interpretability of MLP neurons such as by reducing polysemanticity[1] [19] while having a negligible impact on performance. Empirically I found

[1] Polysemanticity is the phenomenon where neurons respond to multiple unrelated inputs.

Table 1: A one-layer transformer solves the Tic-Tac-Toe task optimally. KL div. is the KL divergence between the optimal distribution and that predicted by the model, averaged across all game sequences. Legal move acc. measures whether the top-predicted move is a legal move, averaged across all *prefixes*. Top move acc. measures whether the top-predicted move is also (one of) the top move(s) under the optimal model, averaged across all prefixes.

	KL div. (nats)	Legal move acc. (%)	Top move acc. (%)
TicTacGPT	**0.0042**	**100**	**99.56**
Two-layer transformer	0.0060	99.88	99.33
Chance level	1.0664	13.58	11.25
TicTacGPT (MLP only)	0.0105	100	98.73

that models trained with SoLU produced slightly "cleaner" activation patterns than standard choices like GeLU, though the effect is not substantial. The model is trained for 40000 steps with batch size 512 using the AdamW optimizer [12] with learning rate 0.0003, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and weight decay 0.1 on non-bias and non-embedding parameters.

4 Three simplifying properties

This section lays down and provides empirical evidence for several appealing properties of the trained model that make it particularly friendly to mechanistic analysis. The remainder of this report will base its argument on such simplifications.

4.1 TicTacGPT solves Tic-Tac-Toe

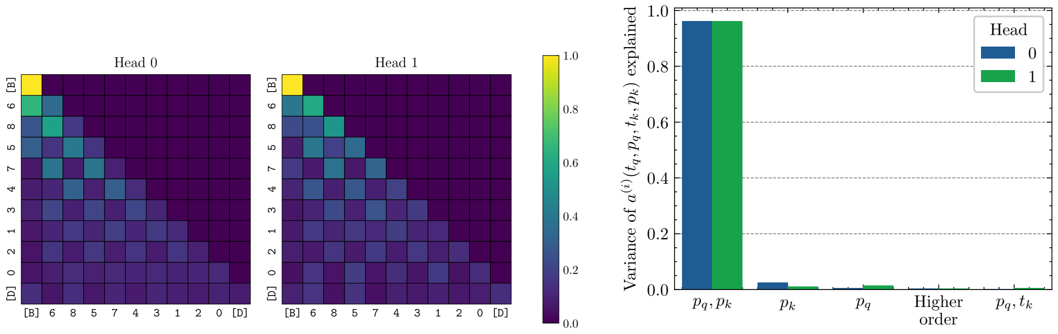
The first observation is that a one-layer transformer is sufficient to solve the task almost optimally. One of the advantages of working with a toy problem like Tic-Tac-Toe is that the optimal policy can be computed exactly, by enumerating all game sequences. This makes evaluation much simpler as we can directly compare the trained models to the optimal model. Table 1 shows such comparisons with the optimal model for TicTacGPT, a two-layer counterpart[2] and a baseline constant output model. TicTacGPT achieves much better performance than random and approaches 100% accuracy on other classification metrics. Surprisingly, it even outperforms the two-layer transformer which may be attributed to reduced overfitting. I thereby conclude that TicTacGPT solves Tic-Tac-Toe and thus the investigation of *"How does a sequence model solve Tic-Tac-Toe?"* can be limited to *"How does TicTacGPT solve Tic-Tac-Toe?"*.

4.2 Attention is only a function of position

The second observation is that the model appears to have learnt attention patterns that are independent of the input sequence. From the example attention map in Fig. 2a, we see that both attention heads use chequerboard-like attention patterns. Heuristically, it appears that `Head0` is performing "attend to all moves played by the opponent" and `Head1` is doing the same but for the current player. Since the player order in Tic-Tac-Toe always alternates between `X` and `O`, it is very likely that the attention pattern is derived from positional information alone.

To confirm the above intuition analytically, I perform sensitivity analysis of the attention pattern to identify which inputs have the most effect. To do so, I study a function $a^{(i)}$ that calculates the unnormalised attention paid by token t_q at position p_q on another token t_k at position p_k, for attention head i. This can be done by unfolding the embedding and attention calculation as described by

[2] The two-layer model uses the same hyperparameters, except having two layers and is trained with a lower weight decay of 0.03 (as it failed to converge with weight decay 0.1).



(a) A sample attention pattern of TicTacGPT. Other inputs generally have the same pattern too.

(b) Sobol decomposition of the unnormalised attention logits. Most of the variance can be explained by second order interactions between positional embeddings.

Figure 2: The attention patterns of TicTacGPT is approximately constant.

Eqs. (1) and (2), giving the following definition:

$$a^{(i)}(t_q, p_q, t_k, p_k) = \begin{cases} \left(\mathbf{x}_q \mathbf{W}_Q^{(i)} + \mathbf{b}_Q^{(i)}\right) \left(\mathbf{x}_k \mathbf{W}_K^{(i)} + \mathbf{b}_K^{(i)}\right)^T & \text{if } p_k \leq p_q, \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathbf{x}_q = \mathrm{LN}((\mathbf{W}_E)_{t_q} + (\mathbf{W}_P)_{p_q})^T$$

$$\mathbf{x}_k = \mathrm{LN}((\mathbf{W}_E)_{t_k} + (\mathbf{W}_P)_{p_k})^T$$

Treating t_q, p_q, t_k, p_k as independent, uniformly distributed random variables, one can decompose the variance of $a^{(i)}$ into those explained by first-order terms (i.e. each input individually), second-order terms (i.e. interactions between pairs of inputs) and higher-order terms using Sobol's method [22, 23]. As shown in Fig. 2b, nearly all of the variance can be explained by second-order interactions between p_1 and p_2. This matches our intuition since chequerboard attention patterns require second-order interactions yet are purely positional. Thus, TicTacGPT uses attention patterns that are input-independent and so *the attention matrices* $\mathbf{A}^{(0)}, \mathbf{A}^{(1)}$ *are constant, chequerboard-like matrices*.

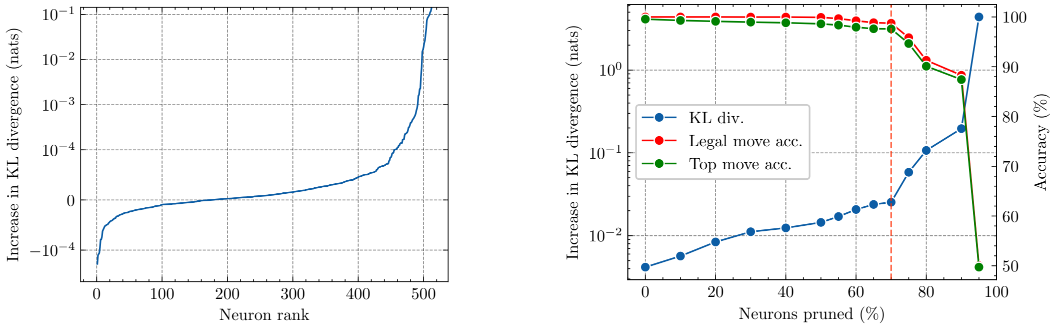
4.3 Output is only a function of MLP activations

The last observation is that the skip connection around the MLP layer has minimal effect on the output logits, and thus can be ignored in our subsequent analysis. This property can be shown empirically by artificially removing the skip connection during inference (i.e. setting $\mathbf{H}_2 = \mathrm{MLP}(\mathrm{LN}(\mathbf{H}_1))$) and studying its effects on the performance of the model. The comparison shown in Table 1 confirms such assertion, as the top move accuracy of the modified network only drops by 0.83% and remains significantly better than chance.

The insignificance of the MLP skip connection means that we can narrow down the analysis to the MLP activations only. If we can understand how the MLP neurons respond to inputs and characterise their corresponding effects on the output logits, we can explain most of the behaviour of TicTacGPT.

5 Zooming in: MLP neurons

Given the simplifications from Section 4, this section dives into the two circuits around the MLP layer that define the model's behaviour: The "input circuit"—how do the MLP neurons respond to inputs, and the "output circuit"—how do the activated neurons affect the output logits. The analysis will be done in two steps. Firstly, Section 5.1 identifies the neurons that contribute the most to the model's performance. In the process, I discover that most of the behaviour can be explained by a small subset of neurons. Then, Section 5.2 characterises the role of each of such neurons on the behaviour of the model.



(a) Effect of removing each neuron on the KL divergence, ordered from lowest to highest. Notice the y-axis is in log-scale.

(b) The retained performance of pruning the bottom $n\%$ of neurons, ordered according to Fig. 3a. The dotted line marks the 70% threshold where there appears to be a clear cut-off.

Figure 3: Approximately 30% of the neurons explain most of the performance of TicTacGPT.

5.1 Neurons that matter

TicTacGPT has 512 MLP neurons, which even in this toy problem is too many to be human comprehensible. To proceed, it is necessary to narrow down the number of MLP neurons for in-depth analysis.

However, identifying which neurons are the most "important" is a non-trivial task. A naive approach might measure how often a neuron is "active" (e.g., larger than some threshold) across a diverse sample of game sequences. Unfortunately, this neglects the importance of neurons that only fire in response to particular inputs yet have a strong influence on the output logits. Another method might be to identify the neurons with the largest output logits influence by inspecting the output circuit $(\mathbf{W}_{out} + \mathbf{1}_s \mathbf{b}_{out})\mathbf{W_U}$. This however assumes each neuron fires with equal probability and equal magnitude, which is not guaranteed. For example, some neurons might never be activated at all yet appear to have a large effect according to the output circuit.

Instead, I propose to take an empirical approach to measure neuron importance. To measure the significance of neuron n, I evaluate the performance of a patched version of TicTacGPT where neuron n is removed entirely. Concretely, the importance of a neuron is given by the increase in KL divergence from the optimal model of the patched model versus the original model, evaluated on a sample of 4096 games. The method is inspired by activation patching [14, 27] which is commonly used to identify core components of a network.

Fig. 3a shows that only a small proportion of neurons have a significant impact on the model's performance. Speculatively, this might be attributed to the use of SoLU non-linearity and weight decay which promotes sparse activations. To further quantify how many neurons can be ignored while still capturing most of the model's behaviour, I evaluate pruned versions of TicTacGPT where $k\%$ of the least important neurons are removed. The results in Fig. 3b reveal that 70% of the neurons can be removed by preserving nearly most of the performance. Interestingly, Fig. 3a suggests that there are approximately 30% of neurons that *negatively* impact the model's performance. I believe this is due to estimation error from only using a subset of games, since pruning such neurons does not actually result in improved performance as shown in Fig. 3b.

5.2 Understanding the neurons

Given the core neurons identified in Section 5.1, we proceed to analyse each neuron's role.

5.2.1 Input circuits

To conceptualise the "rule" implemented by each neuron, we must first understand under what situations the neuron activates, i.e. the input circuit. To simplify the following analysis, I ignore LayerNorm operations which I argue to have minimal effect on the analysis since it only applies

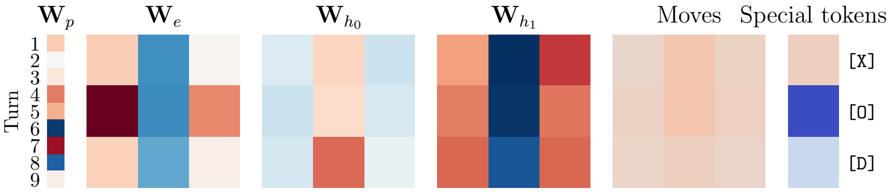


Figure 4: Visualisation of input and output circuits of N508. The 9x1 vertical line visualises positional quantities (i.e. turns of the game), ordered from turn 1 to turn 9 from top to bottom. The 3x3 grids visualise quantities that are tied to board positions. The first four columns visualises the input circuits. The last two columns visualises the effect of this neuron on the output logits. Blue denotes strong positive contribution while red denotes strong negative contribution. The visualisation suggests that this neuron implements the rule "If it is currently turn 6 or 8 (from the $\mathbf{W}_p$ circuit) and the currently player has filled the middle column (from the $\mathbf{W}_{h_1}$ circuit), the next token is [O], i.e. O wins."

affine transformations. Referring to Eqs. (2) and (3), the input to the MLP non-linearity is given by:

$$\begin{aligned}\mathbf{X} &= \mathbf{H}_1\mathbf{W}_{in} + \mathbf{1}_s\mathbf{b}_{in} \\ &= \left(\mathbf{H_0} + \sum_{i=0}^{1}\mathbf{A}^{(i)}(\mathbf{H}_0\mathbf{W}_V^{(i)} + \mathbf{1}_s\mathbf{b}_V^{(i)})\mathbf{W}_O^{(i)} + \mathbf{1}_s\mathbf{b}_O^{(i)}\right)\mathbf{W}_{in} + \mathbf{1}_s\mathbf{b}_{in} \\ &= \mathbf{H}_0\mathbf{W}_{in} + \mathbf{A}^{(0)}(\mathbf{H}_0\mathbf{W}_V^{(0)}\mathbf{W}_O^{(0)}\mathbf{W}_{in}) + \mathbf{A}^{(1)}(\mathbf{H}_0\mathbf{W}_V^{(1)}\mathbf{W}_O^{(1)}\mathbf{W}_{in}) + \mathbf{1}_s\mathbf{b}\end{aligned}$$

where $\mathbf{b} = \mathbf{b}_{in} + \sum_{i=0}^{1}\mathbf{b}_V^{(i)}\mathbf{W}_O^{(i)}\mathbf{W}_{in} + \mathbf{b}_O^{(i)}\mathbf{W}_{in}$. Factoring the bias terms out is possible since each row of the attention matrices always sums to one.

Furthermore, we have $\mathbf{H}_0 = \mathbf{E}\mathbf{W}_E + \mathbf{W}_P$ (Eq. (1)). Since the attention matrices are constants (Section 4.2), we can rewrite the above as a sum of one position-dependent term and three data-dependent terms (plus a constant):

$$\mathbf{X} = \mathbf{W}_p + \mathbf{E}\mathbf{W}_e + \mathbf{A}^{(0)}\mathbf{E}\mathbf{W}_{h_0} + \mathbf{A}^{(1)}\mathbf{E}\mathbf{W}_{h_1} + \mathbf{1}_s\mathbf{b} \tag{4}$$

where

$$\begin{aligned}\mathbf{W}_p &= \left(\mathbf{W}_P + \mathbf{A}_0\mathbf{W}_P\mathbf{W}_V^{(0)}\mathbf{W}_O^{(0)} + \mathbf{A}_1\mathbf{W}_P\mathbf{W}_V^{(1)}\mathbf{W}_O^{(1)}\right)\mathbf{W}_{in} \\ \mathbf{W}_e &= \mathbf{W}_E\mathbf{W}_{in} \\ \mathbf{W}_{h_0} &= \mathbf{W}_E\mathbf{W}_V^{(0)}\mathbf{W}_O^{(0)}\mathbf{W}_{in} \\ \mathbf{W}_{h_1} &= \mathbf{W}_E\mathbf{W}_V^{(1)}\mathbf{W}_O^{(1)}\mathbf{W}_{in}\end{aligned}$$

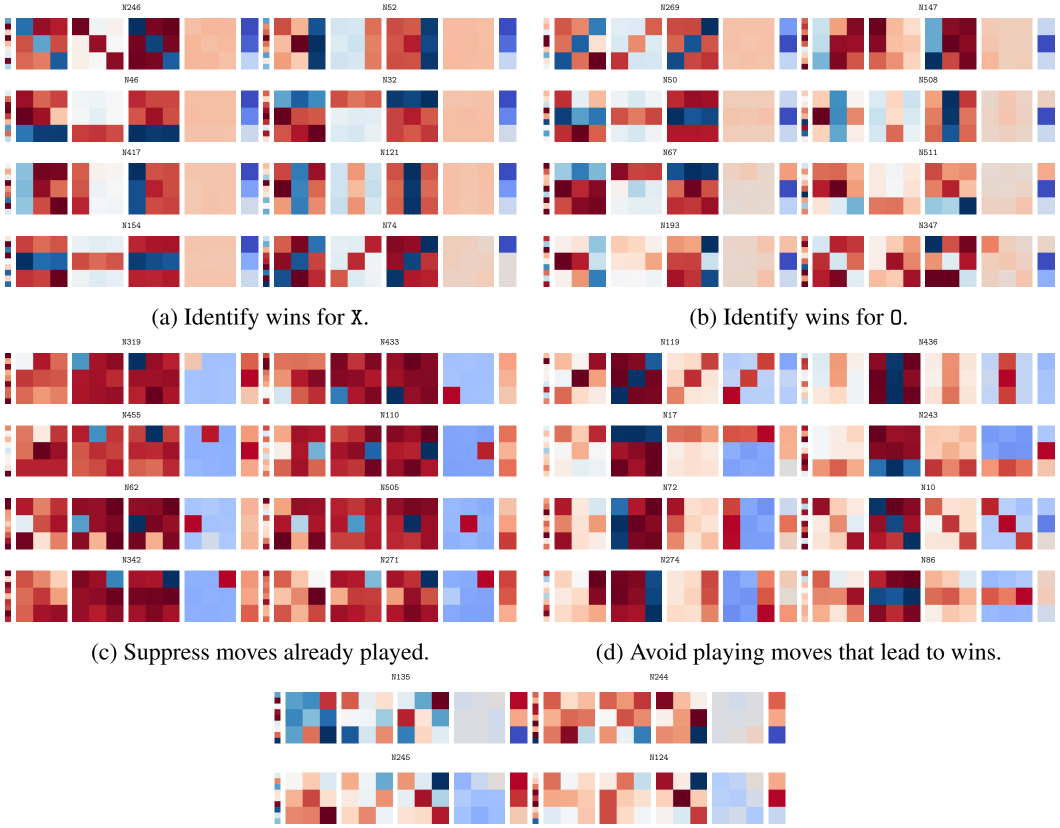
Decomposing the pre-activations into circuits is useful as they each have an interpretable meaning. Referring to Eq. (4), the $\mathbf{W}_p$ circuit encapsulates all positional information, thus it captures how the neurons react to the current turn number. The $\mathbf{W}_e$ circuit only involves position-wise operators, thus it measures how the current move affects the neurons. The $\mathbf{W}_{h_0}$ circuits trace how inputs at other positions affect each neuron via attention Head0. From Section 4.2, attention Head0 roughly serves the purpose of "attend to the opponent's moves so far" and thus we may interpret the $\mathbf{W}_{h_0}$ circuits as measuring "how do the opponent's moves affect each neuron?". Similarly, attention Head1 approximately attends to the current player's moves so far, and so the $\mathbf{W}_{h_1}$ circuits measure "how do the current player's moves so far affect each neuron?".

5.2.2 Output circuit

The output circuit is considerably simpler since Section 4.3 has established that only the direct contributions from the MLP layer matter. Ignoring LayerNorm, each neuron's effect on the output logits is given by $\mathbf{W}_{out}\mathbf{W}_U$.

5.2.3 Visualisation and interpretation

For each neuron, the data-dependent input circuits $\mathbf{W}_e$, $\mathbf{W}_{h_0}$ and $\mathbf{W}_{h_1}$ each give a scalar for how much a neuron reacts to a given move on the board. Such values are best visualised as 3x3 grids. The



(a) Identify wins for X.

(b) Identify wins for O.

(c) Suppress moves already played.

(d) Avoid playing moves that lead to wins.

(e) Predict draws.

Figure 5: Samples of neurons from TicTacGPT grouped by their function.

output logits are composed of two components: The move logits and the special token logits. Since they conceptually represent different quantities, I plotted them separately. An example visualising the 508th neuron (denoted as N508) is shown in Fig. 4. The visualisation strategy is successful as it is immediately clear what this neuron is performing: It identifies three-in-a-rows down in the middle column for player O.

We can now scale up the visualisation strategy detailed in Fig. 4 to the rest of the neurons identified in Section 5.1. Manually inspecting the visualisations reveals that many neurons are interpretable as they implement certain "rules" of the task. The full visualisation can be found in Appendix A.

The neurons with the most obvious interpretation are the "X wins" (Fig. 5a) and the "O wins" (Fig. 5b) neurons. Such neurons react the most strongly to moves via the $\mathbf{W}_{h_1}$ circuit (i.e. moves played by the current player) to determine whether the next token should be a winner's token. It also makes use of the turn number via the $\mathbf{W}_p$ circuit to determine whether the winner should be X for odd-numbered turns, or O for even-numbered turns. Surprisingly, there appear to be exactly 16 of such neurons in TicTacGPT, corresponding to the 8 three-in-a-row combinations for each player.

Some other neurons mainly affect the predicted move distributions. For example, I identified "single move suppression" neurons that prevent moves that have been played before to be played again on the board (Fig. 5c). These neurons react to inputs from both the $\mathbf{W}_{h_0}$ and $\mathbf{W}_{h_1}$ circuits which can interpreted as "no matter who played the move before, the same move cannot be played again".

Another interesting class of neurons are those that affect groups of moves. One outstanding collection of neurons are the "winning move suppression" neurons which discourage the probability of playing moves that immediately result in a win. The existence of such neurons matches our intuition as the true task of the model is to predict the game tree distributions, where immediately winning moves have the least number of possible continuations. Unlike the "X wins" or "O wins" neurons that mainly respond to the $\mathbf{W}_{h_1}$ circuit, these neurons react strongly to the $\mathbf{W}_{h_0}$ circuit as it needs to attend to the moves played by the opponent. One might argue that the existence of such neurons suggests that TicTacGPT has learned to "think one step ahead" as the penalty of predicting a winning move only comes into play after two steps. I tend not to believe in such a claim as I observed no other evidence that TicTacGPT utilises the min-max nature of the game to make its predictions.

5.3 What has TicTacGPT learned?

Our analysis in Section 5.2 shows that TicTacGPT has undoubtedly learned some structure about Tic-Tac-Toe. In is section, I give a qualitative assessment of to what extent has TicTacGPT fully "understood" Tic-Tac-Toe, where I define "understanding" as using solutions that respect the structure and symmetries of the game.

Tic-Tac-Toe has several symmetries in the time domain. For instance, the player always alternates between `X` and `O` on each move. Another example is that Tic-Tac-Toe is a Markovian process—the order of moves does not matter if they lead to the same game state. The fixed attention patterns (Fig. 2a) is a good indicator that TicTacGPT utilises both of these rules: The chequerboard-like patterns clearly match with the alternating nature of the game. We also see each query generally attends equally to all of its previous moves, suggesting that it is not the order of moves that matters, but rather the combination of moves.

The board of Tic-Tac-Toe also contains many symmetries geometrically, such as reflectional and rotational symmetries. One might also argue for translational symmetry (e.g. A three-in-a-row in the left column has the same effect as one in the middle/right.) Our inspection of the neurons Section 5.2 do *not* reveal any evidence that TicTacGPT has learned such structures. For example, TicTacGPT uses a distinct neuron (8 of them in total) for each of the winning combinations while there are only 5 cases if it respects reflectional symmetry or 3 cases if it respects rotational symmetry. There are also distinct "move suppression" neurons (Fig. 5c) for each of the 9 board positions. One explanation for why TicTacGPT fails to learn geometric symmetries is that the output predictions are *not* geometrically invariant. To leverage such structures the model would have to encode *and decode* the symmetries which is a more complicated algorithm than simply ignoring them.

6 Conclusion

The experiments of this paper show that a one-layer transformer solves Tic-Tac-Toe in an interpretable way. By virtue of the simplicity of the task, I discovered that TicTacGPT implements a particularly simple solution to the task which is friendly to mechanistic analysis. This not only allowed me to identify the most important neurons in the network, but also be able to characterise the behaviour of each neuron to the level of input features and their effects on the output logits, effectively fully dissecting the behaviour of the model.

Understanding TicTacGPT so thoroughly allows us to reason about what the model has or has not learned. For example, TicTacGPT has learned the alternating nature of the game but has not learned the geometric symmetries of the board, possibly due to the lack of geometric invariances in the output predictions. This demonstrates that simply doing next-token prediction does not necessarily lead to the most general solution, even if the model is capable of solving the task perfectly. I believe this project is a step towards understanding the fundamental limits of next-token prediction.

This project only focuses on the interpretability of a toy model. Future work could take the insights gained here and test them on more complex models such as OrthelloGPT [11] or simple text generation models [20]. The techniques used in this project may not scale well to larger models (e.g. there may be too many neurons to analyse manually) so more automated methods such as sparse probing [8] or attribution patching [16] can be incorporated.

References

[1] Anthony Bau, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. Identifying and controlling important neurons in neural machine translation. *arXiv preprint arXiv:1811.01157*, 2018.

[2] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 117(48):30071–30078, 2020.

[3] Emily M Bender and Alexander Koller. Climbing towards nlu: On meaning, form, and understanding in the age of data. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 5185–5198, 2020.

[4] Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, Michael Petrov, Ludwig Schubert, Chelsea Voss, Ben Egan, and Swee Kiat Lim. Thread: Circuits. *Distill*, 2020. doi: 10.23915/distill.00024. https://distill.pub/2020/circuits.

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

[6] Nelson Elhage, Tristan Hume, Catherine Olsson, Neel Nanda, Tom Henighan, Scott Johnston, Sheer ElShowk, Nicholas Joseph, Nova DasSarma, Ben Mann, Danny Hernandez, Amanda Askell, Kamal Ndousse, Andy Jones, Dawn Drain, Anna Chen, Yuntao Bai, Deep Ganguli, Liane Lovitt, Zac Hatfield-Dodds, Jackson Kernion, Tom Conerly, Shauna Kravec, Stanislav Fort, Saurav Kadavath, Josh Jacobson, Eli Tran-Johnson, Jared Kaplan, Jack Clark, Tom Brown, Sam McCandlish, Dario Amodei, and Christopher Olah. Softmax linear units. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/solu/index.html.

[7] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

[8] Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. Finding neurons in a haystack: Case studies with sparse probing. *arXiv preprint arXiv:2305.01610*, 2023.

[9] Stefan Heimersheim and Jett Janiak. A circuit for python docstrings in a 4-layer attention-only transformer. *URL: https://www. alignmentforum. org/posts/u6KXXmKFbXfWzoAXn/acircuit-for-python-docstrings-in-a-4-layer-attention-only*, 2023.

[10] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[11] Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. *arXiv preprint arXiv:2210.13382*, 2022.

[12] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[13] R Thomas McCoy, Shunyu Yao, Dan Friedman, Matthew Hardy, and Thomas L Griffiths. Embers of autoregression: Understanding large language models through the problem they are trained to solve. *arXiv preprint arXiv:2309.13638*, 2023.

[14] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372, 2022.

[15] William Merrill, Yoav Goldberg, Roy Schwartz, and Noah A Smith. Provable limitations of acquiring meaning from ungrounded form: What will future language models understand? *Transactions of the Association for Computational Linguistics*, 9:1047–1060, 2021.

[16] Neel Nanda. Attribution patching: Activation patching at industrial scale. *URL: https://www. neelnanda. io/mechanistic-interpretability/attribution-patching*, 2023.

[17] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*, 2023.

[18] Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent linear representations in world models of self-supervised sequence models. *arXiv preprint arXiv:2309.00941*, 2023.

[19] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

[20] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

[21] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[22] Ilya M Sobol. Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and computers in simulation*, 55(1-3):271–280, 2001.

[23] IM Soboĺ. Sensitivity estimates for nonlinear mathematical models. *Math. Model. Comput. Exp.*, 1:407, 1993.

[24] Curt Tigges, Oskar John Hollinsworth, Atticus Geiger, and Neel Nanda. Linear representations of sentiment in large language models. *arXiv preprint arXiv:2310.15154*, 2023.

[25] Shubham Toshniwal, Sam Wiseman, Karen Livescu, and Kevin Gimpel. Learning chess blindfolded: Evaluating language models on state tracking. *arXiv preprint arXiv:2102.13249*, 2, 2021.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[27] Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. *Advances in neural information processing systems*, 33:12388–12401, 2020.

[28] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR, 2020.

[29] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.

A Anatomy of TicTacGPT

Table 2: Manual classification of the top 30% of neurons in TicTacGPT.

Category	Neurons
Wins for [X]	246, 52, 46, 32, 417, 121, 154, 74
Wins for [O]	269, 147, 50, 508, 67, 511, 193
Draws	135, 244, 245, 124
Anti-draw	376
Anti-win	119, 436, 17, 243, 72, 10, 274, 86, 45, 241, 92, 137, 118, 73, 157, 70
Winner by position	266, 472
Move suppression (single)	319, 433, 455, 110, 62, 505, 342, 271, 83, 151, 277, 20, 354, 97, 339, 499, 323, 183, 177, 456, 171, 91, 240, 224, 501, 397, 94, 392, 192, 367, 216, 450
Move suppression (multiple)	478, 322, 482, 262, 370, 428, 203, 174, 18, 340, 419, 310, 459, 502, 336, 57, 47, 14, 101, 453, 377, 506, 172, 105, 371, 148, 509, 429, 102, 489
Move suppression and winner by position	194, 324, 270, 161, 152, 130, 469, 248, 22, 61, 420, 51, 19, 284, 169, 90, 424
Multi-purpose	487, 209, 483, 6, 167
Others	104, 412, 155, 507, 237, 117, 256, 414, 120, 34, 88, 408, 470, 82, 349, 49, 201, 422, 85, 280, 359, 335, 129, 287, 264, 156, 461

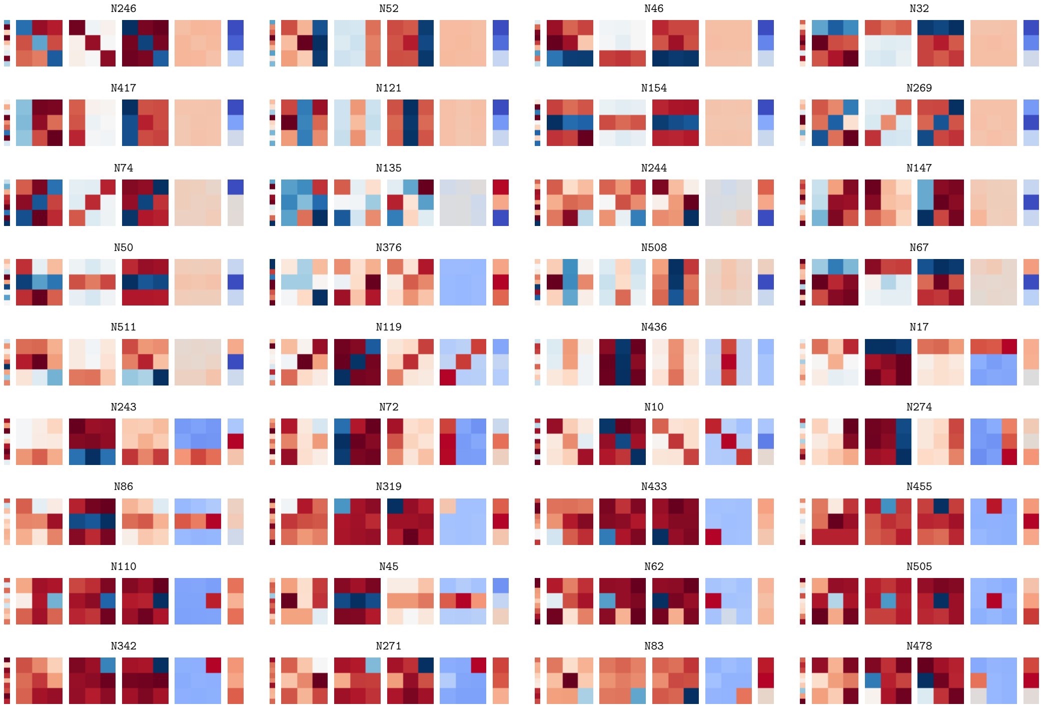
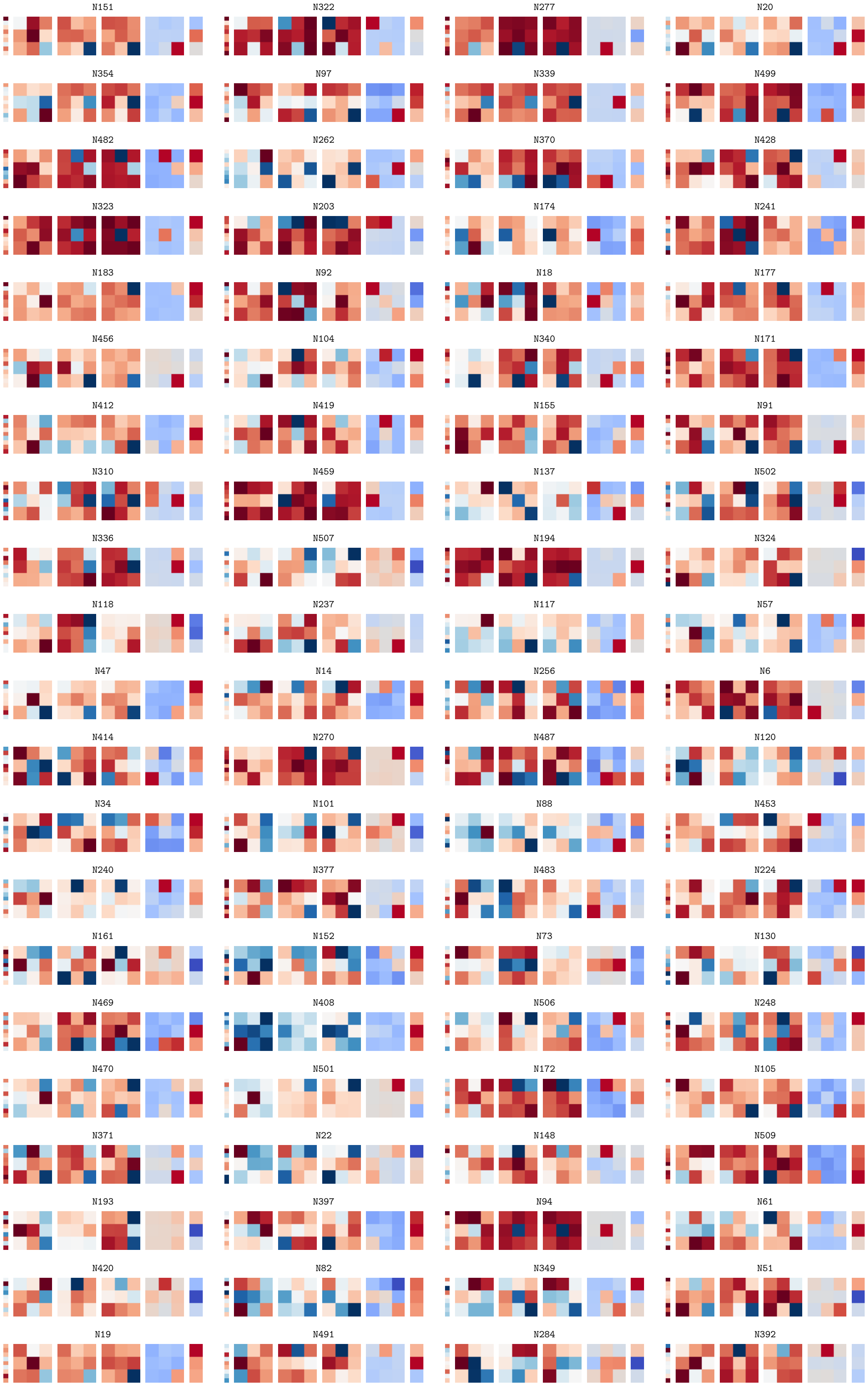
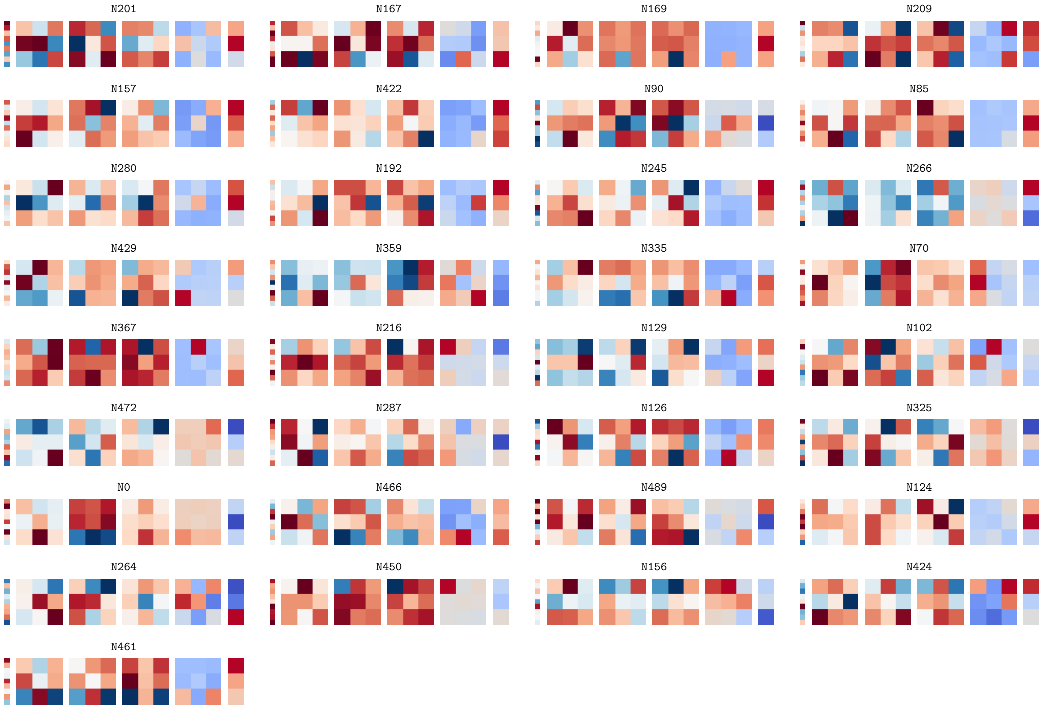


Figure 6: Top neurons of TicTacGPT visualised, ordered from the most the least important.



Top neurons of TicTacGPT visualised, ordered from the most the least important. (cont.)



Top neurons of TicTacGPT visualised, ordered from the most the least important. (cont.)