Handcrafting a Network to Predict Next Token Probabilities for the Random-Random-XOR Process

Rick Goldstein

June 3, 2024

Abstract

For this hackathon, we handcrafted a network to perfectly predict next token probabilities for the Random-Random-XOR process. The network takes existing tokens from the process's output, computes several features on these tokens, and then uses these features to calculate next token probabilities. These probabilities match those from a process simulator (also coded for this hackathon). The handcrafted network is our core contribution and we describe it in Section 3 of this writeup.

Prior work has demonstrated that a network trained on the Random-Random-XOR process approximates the 36 possible belief states [SMT+24]. Our network does not directly calculate these belief states, demonstrating that networks trained on Hidden Markov Models may not need to comprehend all belief states. Our hope is that this work will aid in the interpretability of neural networks trained on Hidden Markov Models by demonstrating potential shortcuts that neural networks can take.

1 Introduction

1.1 Overview

We hope to better understand the thinking that deep networks perform. For example, we'd like to know whether networks build a world model, and if so, how. The field of Mechanistic Interpretability tries to answer the question of how the parameters inside of neural networks work together to create its output. Additionally, recent work in Computational Mechanics tries to understand the probabilistic world models that neural networks trained on Hidden Markov Models internally build [SMT+24].

In this hackathon project, we combine the two, by handcrafting network weights to predict the next token in the Random-Random-XOR process (described below). While our network utilizes human interpretable features, its internal world model does not have the same internal structure as the network trained in [SMT+24].

Our hope is that this work will aid in the interpretability of neural networks trained on Hidden Markov Models by demonstrating potential shortcuts that neural networks can take. For example, the specific features from our handcrafted network are features that might be worth looking for if we decide to interpret model(s) trained the Random-Random XOR process. Additionally, if we are able to replace hard to understand network components with handcrafted components that we fully understand, there is less risk that the network will perform undesired computations.

1.2 Table of Contents

- In section 2, we overview the Random-Random-XOR process.
- **In section 3, we present the handcrafted network. This is the primary contribution of this project.**
- In section 4, we present our testing to confirm that the handcrafted network is accurate.
- In section 5, we present some short analysis we conducted to demonstrate that the handcrafted network does not have the same internal representation as the network from [SMT+24].
- In section 6, we conclude by discussing limitations of this project and potential future work.

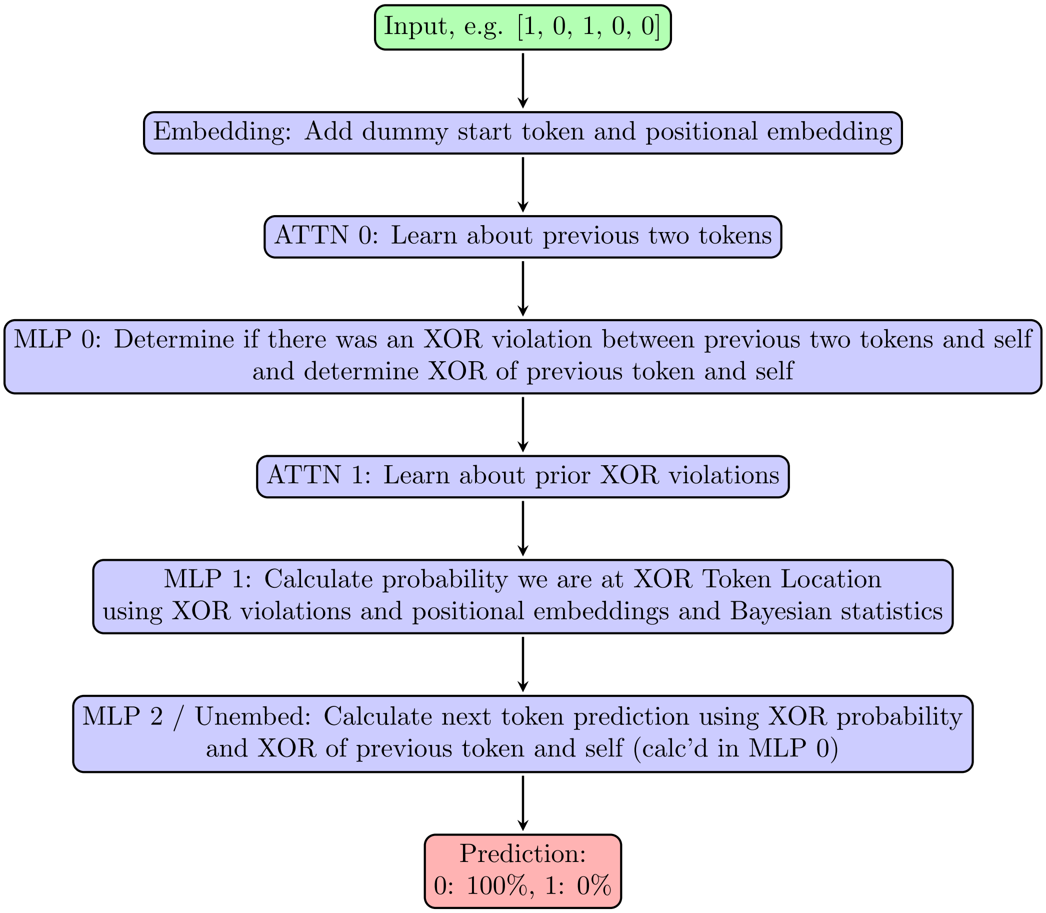

Input, e.g. [1, 0, 1, 0, 0]
Embedding: Add dummy start token and positional embedding
ATTN 0: Learn about previous two tokens
MLP 0: Determine if there was an XOR violation between previous two tokens and self and determine XOR of previous token and self
ATTN 1: Learn about prior XOR violations
MLP 1: Calculate probability we are at XOR Token Location using XOR violations and positional embeddings and Bayesian statistics
MLP 2 / Unembed: Calculate next token prediction using XOR probability and XOR of previous token and self (calc'd in MLP 0)
Prediction:
0: 100%, 1: 0%

Figure 1: Synthetic Network Diagram; Synthetic Network takes in existing tokens (green), runs network (blue) and outputs next token predictions (red). These steps are explained in additional detail in Section 3.

2 Random-Random-XOR Process

The primary goal of this hackathon was to handcraft a neural network to predict the next token of the Random-Random-XOR process. An image of this process, reproduced from [SMT+24], is presented below in Figure 2.

In this process, from the center point, S_s, the model outputs two random bits followed by a third bit that is the XOR of the first two bits. Then the cycle repeats. This is a Hidden Markov Model, meaning we cannot directly observe our state. We don't know our initial state (it need not be the center state S_s). Thus, after observing the first two bits, we do not know whether the third bit will be the XOR of these first two bits. It might be the XOR because we started at S_s and are now at S_T or S_F or it might be the XOR simply due to chance even though we didn't start at S_s. The third possibility is that it might be the other bit because we didn't start at S_s.

In the creation of the 2.5-layer neural network that follows, the overwhelming majority of the calculation goes into determining how likely we are currently at either S_T or S_F. These are the two nodes that are about to output the token that is the XOR of the previous two tokens. We will refer to these locations as the **XOR locations**, meaning that the token outputted from these locations must be the XOR of the previous two tokens (assuming that there are at least two previous tokens).

3 Handcrafted Network

In this section, we describe the creation of a handcrafted neural network to estimate the probabilities of the next token for the Random-Random-XOR process. As the name suggests, handcrafted means

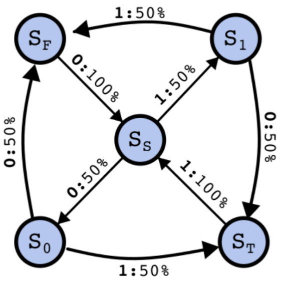


Figure 2: The Random-Random-XOR Process, from [SMT+24]

we manually set all network weights. The network takes in a string of already outputted tokens (which can be no tokens) and perfectly predicts the probabilities that the next token in the process will be either 0 or 1.

The handcrafted network is 2.5 transformer layers - consisting of an embedding, an attention layer, an mlp layer, a second attention layer, a second mlp layer, followed by a joint mlp/unembedding layer. This final layer outputs probabilities that the next token of the process will be 0 or 1. A diagram of the network is viewable above in Figure 1.

Our high level strategy will be for the network to determine how likely it is that our process is currently at the XOR location, meaning that the next token outputted must be the XOR of the previous two tokens. Along with whether the XOR token is 0 or 1, we will be able to predict the next token probabilities.

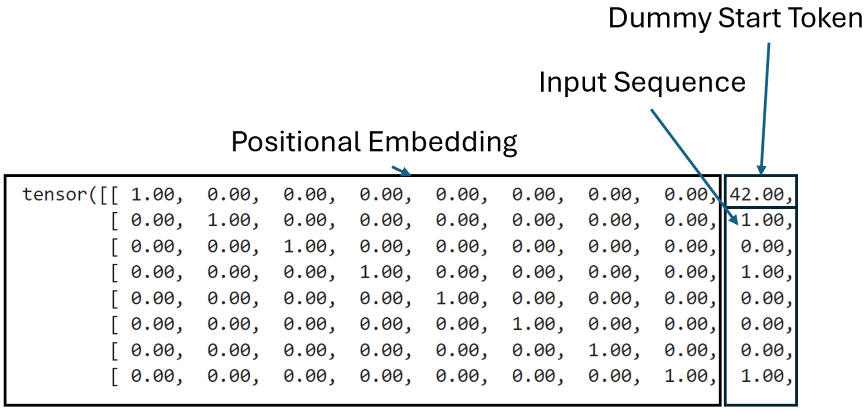
Code is viewable at https://github.com/freestylerick/comp_mech/tree/main

We now describe each step.

3.1 Embedding

The network takes in a string of all tokens that have been outputted by the process so far. This can be an empty string. This list of tokens is appended to a dummy start_of_text token (42), which is common in many LLMs; this way, we can output a guess for the first token in the process. This list of tokens is also appended (along a different axis) to a one-hot positional embedding which will be required for attention calculations.

Suppose we have the input string of tokens, $[1, 0, 1, 0, 0, 0, 1]$, our embedded input is presented in Figure 3.

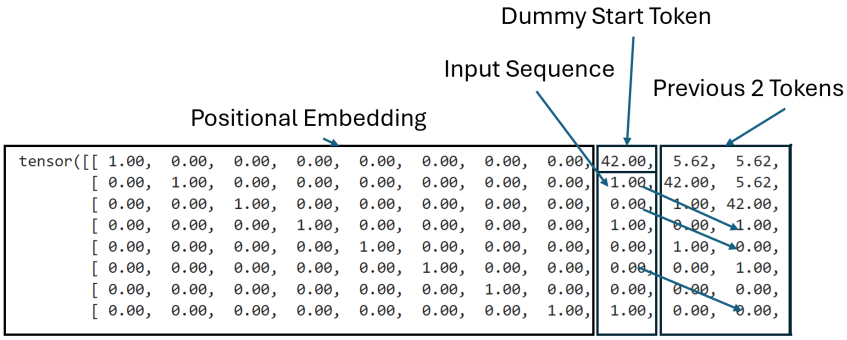


```
tensor([[ 1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00, 42.00,
        [ 0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,
        [ 0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,
        [ 0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,
        [ 0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  1.00,
```

Figure 3: Residual Stream After Embedding

3.2 Attention 0

Our strategy will be to build up important knowledge such that the network can predict how likely it is that the next token outputted must be an XOR. To do this we need to move information around. Specifically, we begin by moving information about the previous two tokens to every token so we can compute XORs in the next step. An image of our residual stream after this step is presented in Figure 4.

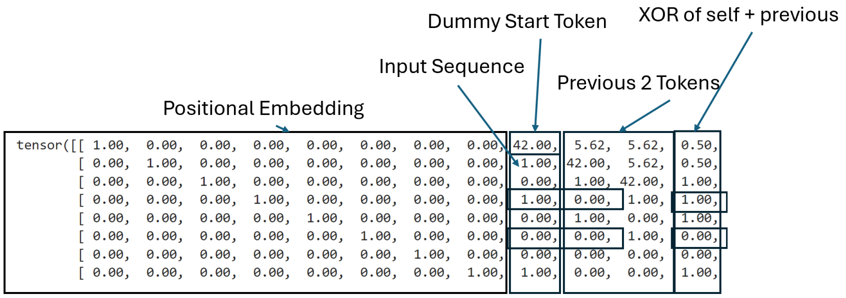


```
tensor([[ 1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00, 42.00,  5.62,  5.62,
        [ 0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00, 42.00,  5.62,
        [ 0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00, 42.00,
        [ 0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,
        [ 0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  1.00,  0.00,  0.00,
```

Figure 4: Residual Stream After ATTN 0

3.3 MLP 0

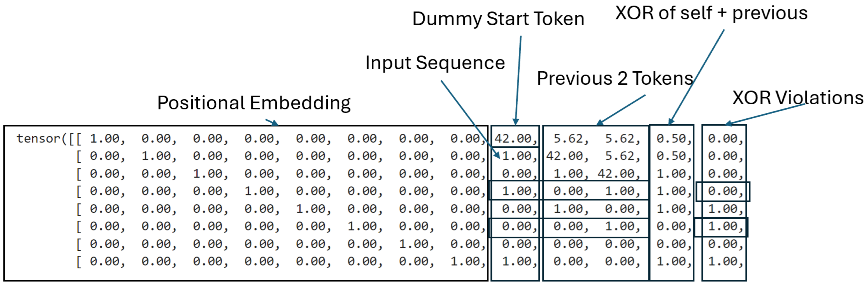
In this first MLP layer, we compute two features for each token position. We first compute the XOR of the current token with its previous token; this tells us that later, should we think we might be at the XOR location, which token should be outputted. We do not compute this for the dummy token or the first token since there is not enough information. Example outputs from this calculation can be seen in Figure 5.



```
tensor([[ 1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00, 42.00,  5.62,  5.62, 0.50,
        [ 0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00, 42.00,  5.62, 0.50,
        [ 0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00, 42.00, 1.00,
        [ 0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00, 1.00,
        [ 0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00, 1.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00, 0.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00, 0.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  1.00,  0.00,  0.00, 1.00,
```

Figure 5: Residual Stream For XOR calculation

Additionally, we compute whether the current token is actually the XOR of the previous two tokens. If it is the XOR, it means that the next two process steps are less likely to be XORs. If it is not the XOR, it means that the next two process steps are more likely to be XORs. We do not compute this for the dummy token or the first or second tokens since there is not enough information. Essentially, the sum of these 4 tokens must be even. We utilize 4 neurons (with ReLu activation) to calculate this. Example outputs from this calculation can be seen in Figure 6.



```
tensor([[ 1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00, 42.00,  5.62,  5.62,  0.50,  0.00,
        [ 0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00, 42.00,  5.62,  0.50,  0.00,
        [ 0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00, 42.00,  1.00,  0.00,
        [ 0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,  1.00,  0.00,
        [ 0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,  1.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  1.00,  0.00,  0.00,  1.00,  1.00,
```

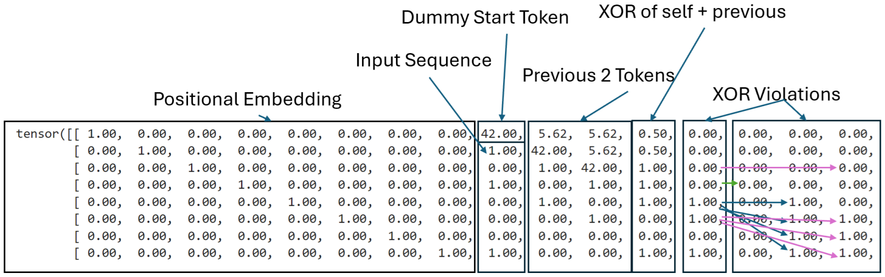
Figure 6: Residual Stream for XOR Violation Calculation

3.4 Attention 1

From the above calculation, we may have found some locations where an XOR did not occur, and thus, we know that this token location is not the XOR location. For example, suppose that the first three bits outputted by the Random-Random-XOR process are 0, 1, 0. The third token was not an XOR of the previous two tokens. This means that when we are at the second token, we were not at the XOR location. Since our Hidden Markov Model process has period 3, we also now know that when we are at the 5th, 8th, 11th ... tokens, we again will not be at the XOR output location. This information makes it more likely that other locations will be the XOR output location.

In this attention layer, tokens attend to earlier tokens that were not XORs, to learn whether XORs are possible at locations that are 2 mod 3 (5, 8, 11 ...), 0 mod 3 (3, 6, 9 ...), and 1 mod 3 (4, 7, 10 ...).

The example residual stream after applying this logic is shown in Figure 7. Note that the period 3 pattern where the third and sixth point populate the last column (pink arrows), the fourth point populates the earliest column (green arrows) and the fifth point populates the middle column (blue arrows). Not all arrows are shown.



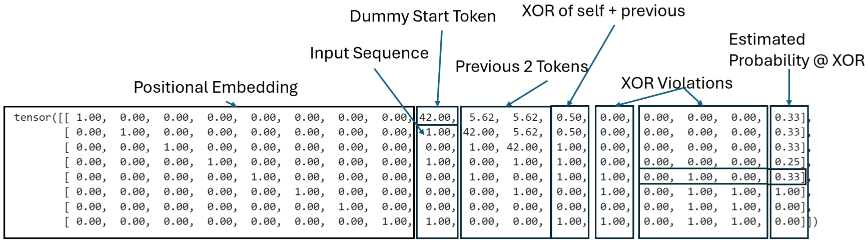
```
tensor([[ 1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00, 42.00,  5.62,  5.62,  0.50,  0.00,  0.00,  0.00,  0.00,
        [ 0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00, 42.00,  5.62,  0.50,  0.00,  0.00,  0.00,  0.00,
        [ 0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00, 42.00,  1.00,  0.00,  0.00,  0.00,  0.00,
        [ 0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,  1.00,  0.00,  0.00,  0.00,  0.00,
        [ 0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,  1.00,  0.00,  1.00,  0.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,  0.00,  1.00,  1.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  1.00,
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  1.00,  0.00,  0.00,  1.00,  1.00,  0.00,  1.00,  1.00,
```

Figure 7: Residual Stream After ATTN 1

3.5 MLP 1

The network uses knowledge of the impossible XOR locations to determine what the probability is that we are now at the XOR location. When this location has been eliminated, the probability is trivially 0. When this location is the only feasible location remaining, the calculation is trivially 1. When there are one or two other possible XOR locations remaining, we use conditional probabilities to calculate these values. Specific values of conditional probabilities are viewable in my code. Note that these values are different for each of 2 mod 3 (5, 8, 11 ...), 0 mod 3 (3, 6, 9 ...), and 1 mod 3 (4, 7, 10 ...).

After this step, we have conditional probabilities that we are at the XOR location given earlier tokens. Example outputs of this calculation are viewable in Figure 8.

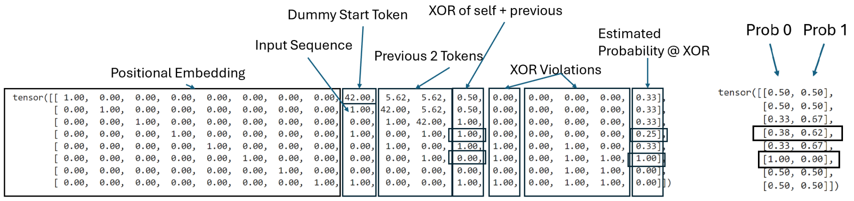


```
tensor([[ 1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00, 42.00,  5.62,  5.62,  0.50,  0.00,  0.00,  0.00,  0.00,  0.33],
        [ 0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00, 42.00,  5.62,  0.50,  0.00,  0.00,  0.00,  0.00,  0.33],
        [ 0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00, 42.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.33],
        [ 0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.25],
        [ 0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,  1.00,  0.00,  1.00,  0.00,  0.33],
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,  0.00,  1.00,  1.00,  1.00],
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  1.00,  0.00],
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  1.00,  0.00,  0.00,  1.00,  1.00,  0.00,  1.00,  1.00,  0.00]])
```

Figure 8: Residual Stream After MLP 1

3.6 MLP 2 / Unembed

We now combine the knowledge of how likely it is we are at the XOR location with the information of whether the XOR of the previous two tokens is 0 or 1 to calculate how likely it is that we should output 0 or 1. Example outputs of this calculation are viewable in Figure 9.



```
tensor([[ 1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00, 42.00,  5.62,  5.62,  0.50,  0.00,  0.00,  0.00,  0.00,  0.33],
        [ 0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00, 42.00,  5.62,  0.50,  0.00,  0.00,  0.00,  0.00,  0.33],
        [ 0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00, 42.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.33],
        [ 0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.25],
        [ 0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,  1.00,  0.00,  1.00,  0.00,  0.33],
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  1.00,  0.00,  1.00,  1.00,  1.00],
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  1.00,  0.00],
        [ 0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  0.00,  1.00,  1.00,  0.00,  0.00,  1.00,  1.00,  0.00,  1.00,  1.00,  0.00]])
```

```
tensor([[0.50, 0.50],
        [0.50, 0.50],
        [0.33, 0.67],
        [0.38, 0.62],
        [0.33, 0.67],
        [1.00, 0.00],
        [0.50, 0.50],
        [0.50, 0.50]])
```

Figure 9: Residual Stream After MLP 2

3.7 Wrap Up

This section presented the logic behind our handcrafted neural network for next token prediction. It calculates features that inform the network's internal world model of the likelihood that the HMM process is currently at the XOR location and thus should output the XOR of itself and the previous token.

4 Testing

We use normal (imperative) programming to write a simulator (nothing fancy, just a for-loop) for the Random-Random-XOR Process. The simulator has a variable (*prob_weight*) to keep track of the conditional probability that moves leaving an XOR location are more likely than other moves.

We examine all valid processes between length 0 and 14, inclusive. We compare the output of the simulator to the output of the handcrafted neural network and they all match.

5 Model Analysis

In this section, we present the ambiguous results of analyzing the residual stream of our handcrafted model.

In [Sha24], the authors analyze their deep network trained on a Random-Random-XOR Process. They run a regression between their residual stream and their belief states then use pca to plot the center of different belief states and obtain a simplex pattern. We run the same code on the residual stream of our model; it looks like there is some structure, and even some kinda-triangular-like structure in the right-most plot, but it is not the clear simplex structure from the author's network.

We leave additional understanding of my handcrafted network as future work.

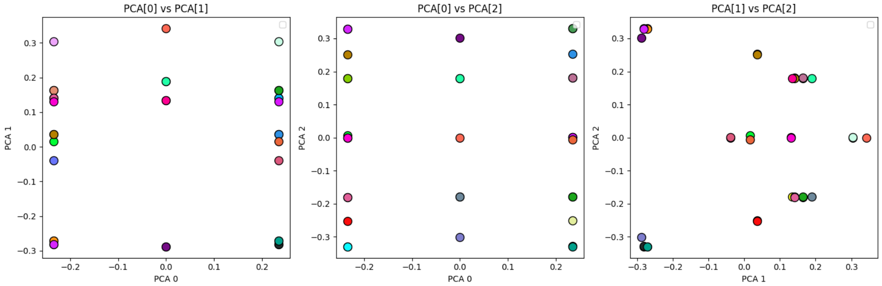


Figure 10: Plotting regression results, code modified from [Sha24]

6 Conclusion

For this project, we successfully handcrafted a neural network to perfectly predict the next token of a Random-Random-XOR Process. We validated that the network was accurate using an independent simulator and generated a few plots that seem to indicate some sort of structure.

Our hope is that the features calculated in this handcrafted model will inspire the search for features when analyzing models trained on Hidden Markov Models. Explicitly, can we train probes and/or find causal evidence that networks trained on the Random-Random XOR process utilize these features? Additionally, are there neurons or groups of neurons that conduct similar logic as the neurons in the synthetic network?

Additionally, there is the question of how much of a world model a network needs to build. Our synthetic model only calculates the probability of being at an XOR location, but we could have added additional features to the residual stream to calculate the probabilities of being at each of the five states. Do we expect models trained on HMMs to understand the full probabilistic state or only enough features to enable next token predictions?

6.1 Limitations

It should be noted that this is a quick hackathon project and thus errors and omissions are possible.

We did not dive deeply into understanding the network we handcrafted. Section 5 only began to understand the internal structure of how our network models possible HMM states; there is likely more that one could do.

6.2 Future Work

It would be interesting to dig into models trained on the Random-Random-XOR process to see how similar their internal logic is compared against the features and MLP logic of the synthetic network.

References

[Sha24] Adam S. Shai. compmech_rrxor.ipynb. https://colab.research.google.com/drive/19xJiYbU7kpDsMOvqYgWq4cOT9jA53l60, 2024. Accessed: 6/1/2024.

[SMT+24] Adam S. Shai, Sarah E. Marzen, Lucas Teixeira, Alexander Gietelink Oldenziel, and Paul M. Riechers. Transformers represent belief state geometry in their residual stream, 2024.