Experiments in Superposition[1]

Kunvar Thaman
1stuserhere@gmail.c
om

Alice Rigg
rigg.alice0@gmail.com

Narmeen Oozeer
nooze066@uottawa
.ca

Joshua David
joshuad93@gmail.com

Neel Nanda, Esben Kran, Fazl Barez

Abstract

We present work in 4 separate projects all related to superposition: examining attention head superposition in a toy model of addition; search for evidence of neuron superposition in TinyStories, implement Neuroscope for TinyStories, and reproduce the results of the toy models in superposition Anthropic paper, for computation in superposition for a variety of nonlinear functions.

Keywords:Mechanistic interpretability, AI safety

1. Introduction

Inspired by the 200 Concrete open problems in mechanistic interpretability, specifically in the superposition section, we set out to cover as much ground as possible and see what we could find. In particular, we were interested in both toy algorithmic models, as well as finding evidence in a real language model, focusing in on TinyStories 1L and 2L models, as they produce legible children's stories with few layers and parameters (21M and 33M, respectively). All of our code can be found at this github link here: https://github.com/firstuserhere/hackathon-attention-superposition

Project 1: Examining a 1-Layer, Attention-Only Multi-Digit Addition Transformer

The goal of this project was to examine behavior in the attention layer where there are more features than attention heads and so either a single attention head needs to individually attend to more than one feature, or groups of attention heads can cooperate to attend to a feature (there are more relevant features than attention heads, but there are more groups of attention heads than there are relevant features). The focus of this project was more strongly on the latter of these.

[1] Research conducted at the Apart Research Alignment Jam #10 (Interpretability 3.0), 2023 (see https://alignmentjam.com/jam/interpretability)

In order to, in a sense, “force” attention superposition to arise, we choose a toy model of addition with no MLP layer, only attention, and then gave it a task where it needed to attend to far more features than the product of the number of attention heads and the dimension of those heads.

2. Methods

Project 1: Examining a 1-Layer, Attention-Only Multi-Digit Addition Transformer

Data Shape

This model was trained on 6-digit base-10 addition with the least significant digit first. Examples of data points are

Data Point	**Meaning**
4321008765002196000	1234 + 5678 == 6912
4444445555559999990	444444 + 555555 == 999999
5444445555550000001	444445 + 555555 == 1000000
9006982803141909031	896009 + 413082 == 1309091

We chose this shape of input data so that the model could, in principle, determine which carries occurred by attending to only 3 tokens. For example, at the third output position, the model needs to attend to:

1. The third digit of the first number
2. The third digit of the second number
3. The second digit of the first number (for determining whether there was a carry)
4. The second digit of the second number (for determining whether there was a carry)
5. The second digit of the output (for determining whether there was a carry – differentiating between 4**44**0005**55**0009**9** (the next token would be **9**) and 5**44**0005**55**00000**0** (the next token would be **0**)

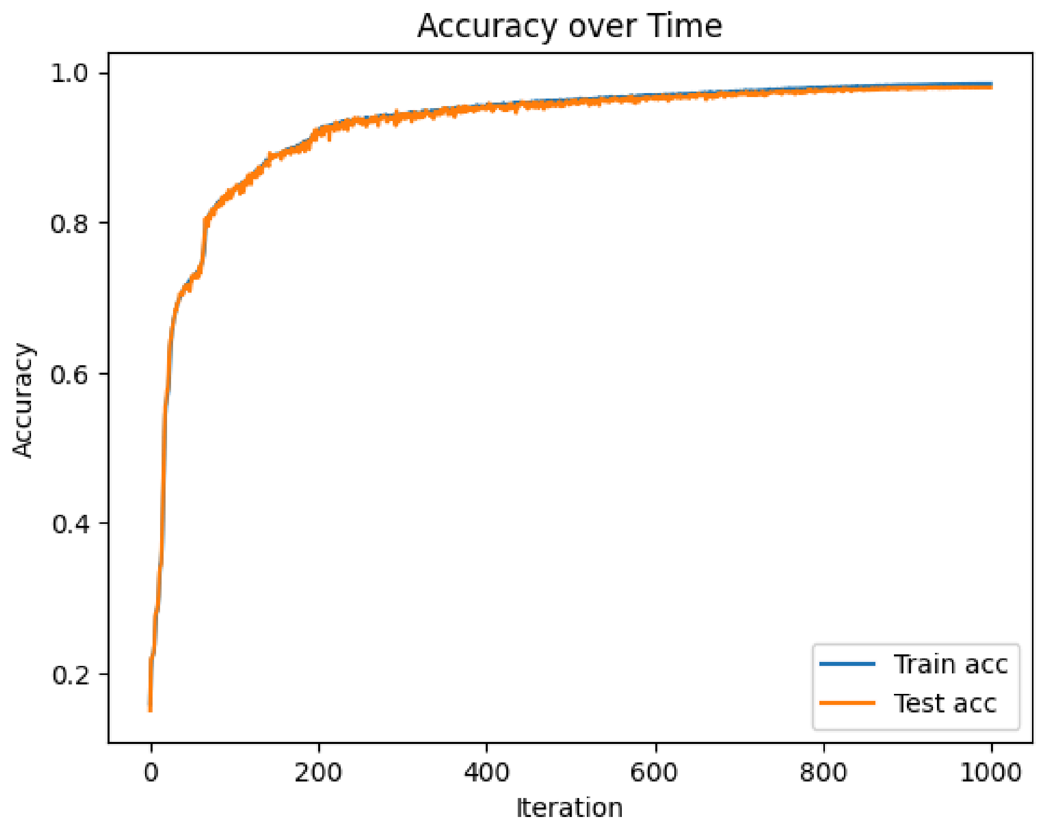
Model Architecture

This was a fairly standard transformer in most respects, except that we chose to use a large number of very small heads (n_heads=32, d_head=2). Since it is not possible to do very much in the way of superposition in only 2 dimensions, this ensured that we could examine how multiple heads within the same layer could usefully combine. Philosophically, we think it is useful to study degenerate cases when first investigating some phenomenon.

Training

We trained the model for 1000 epochs on a training set of size 25000 and a test/validation set of size 2000, with batch size 256.

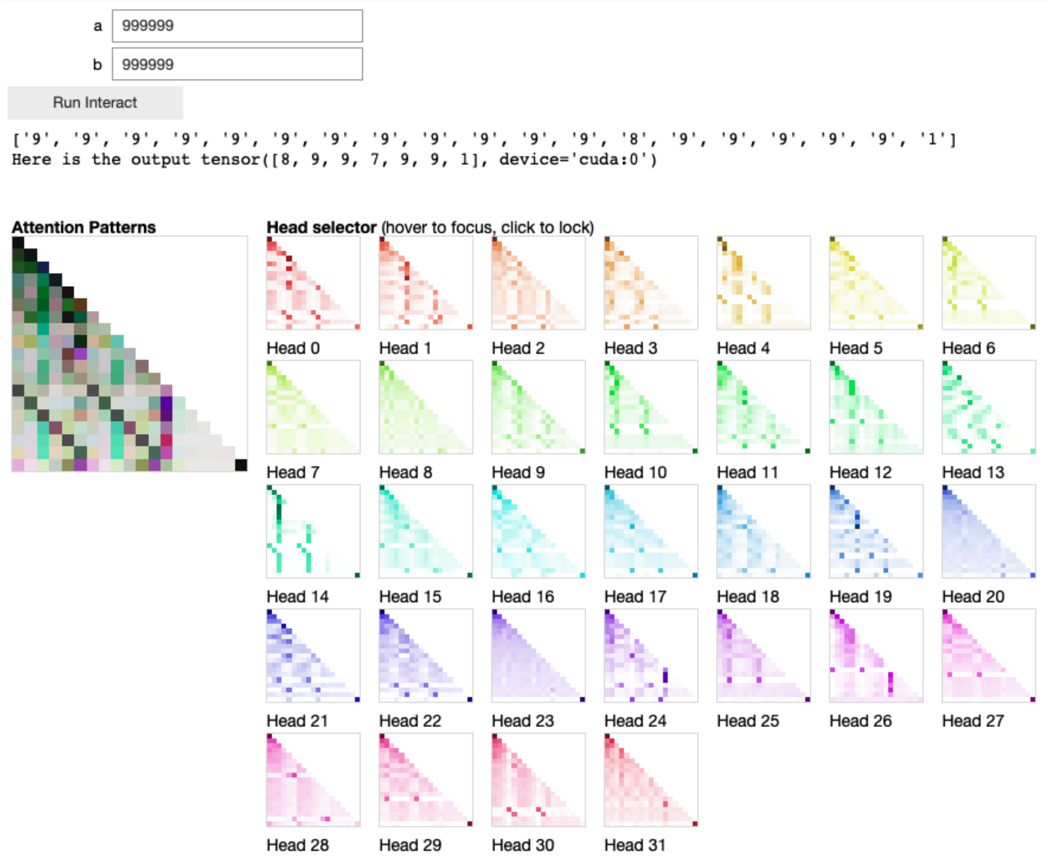
Accuracy improved in the way described in the grokking paper, and at 1000 epochs the accuracy was approximately 98% test, 98.5% train.



Exploration

Attention Pattern

The attention pattern was quite distinctive.


a 999999

b 999999

Run Interact

```
['9', '9', '9', '9', '9', '9', '9', '9', '9', '9', '9', '9', '8', '9', '9', '9', '9', '9', '1']
Here is the output tensor([8, 9, 9, 7, 9, 9, 1], device='cuda:0')
```

Visualizing positions in d_head for various tokens in/out

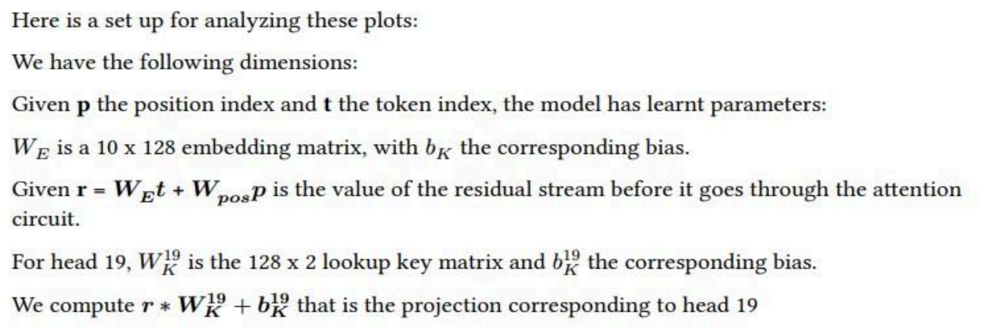
Description

Since d_head for the model is only 2, that means that we can look much more directly than usual at what an activation of a head looks like – it's 2 dimensions so you can just stick it in a chart to visualize it. We did this for every head and for each of the head parameters (Q, K, V, and O).

Where, holding head (h), position (p), and token (t) constant, we get, and looking specifically at attn.K, we get:

The Math

Variable	Shape/Type	Description
h	Integer	Head index
p	Integer	Position index
t	Integer	Token index
W_E	$[10,\ 128]$ float	Embedding matrix
W_{pos}	$[19,\ 128]$ float	Positional embedding matrix
emb	$[128]$ float	Residual stream value before attention
W_K	$[128,\ 2]$	The key matrix(learnt parameter)



Here is a set up for analyzing these plots:

We have the following dimensions:

Given **p** the position index and **t** the token index, the model has learnt parameters:

W_E is a 10 x 128 embedding matrix, with b_K the corresponding bias.

Given **r** = $\boldsymbol{W_E t} + \boldsymbol{W_{pos} p}$ is the value of the residual stream before it goes through the attention circuit.

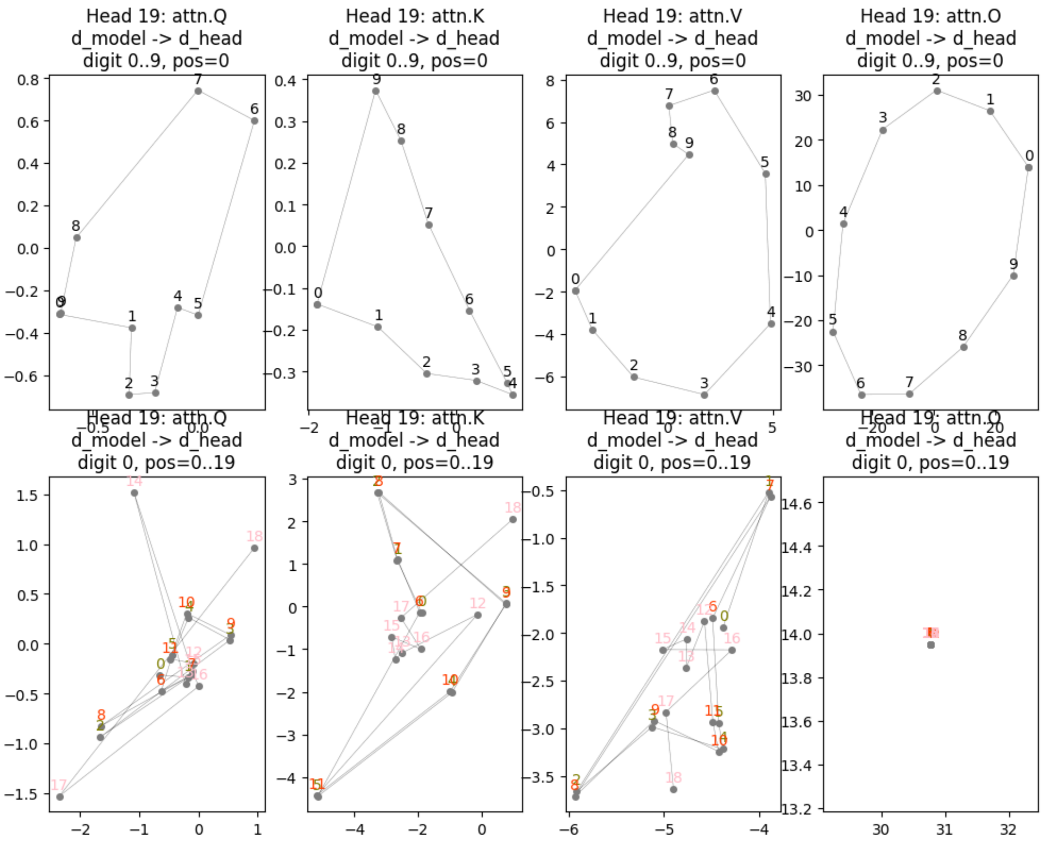
For head 19, W_K^{19} is the 128 x 2 lookup key matrix and b_K^{19} the corresponding bias.

We compute $\boldsymbol{r} * \boldsymbol{W_K^{19}} + \boldsymbol{b_K^{19}}$ that is the projection corresponding to head 19

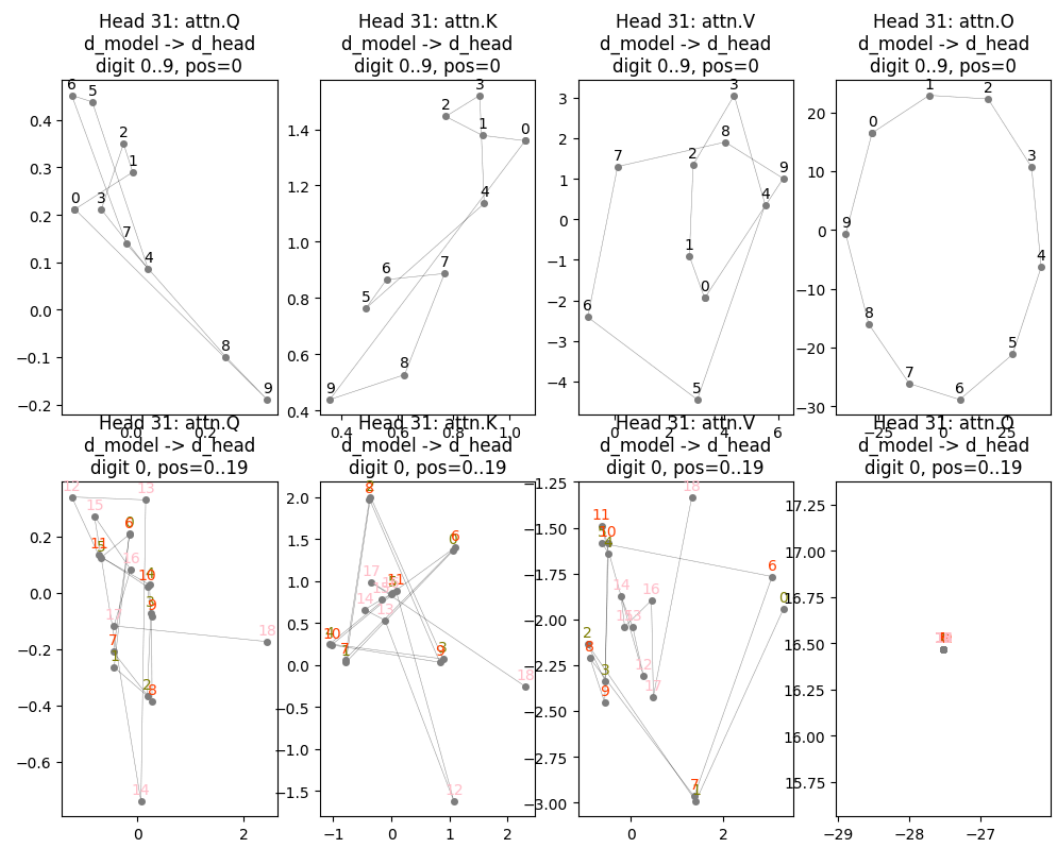
The Chart

Head 19 is an interesting example here:

- For the digits, it looks like it is trying to choose a projection that is approximately close to circular.
- For the positions, it appears that both input numbers project to nearly identical points relative to their position within the number.



An example that is notable for being not as great is head 31

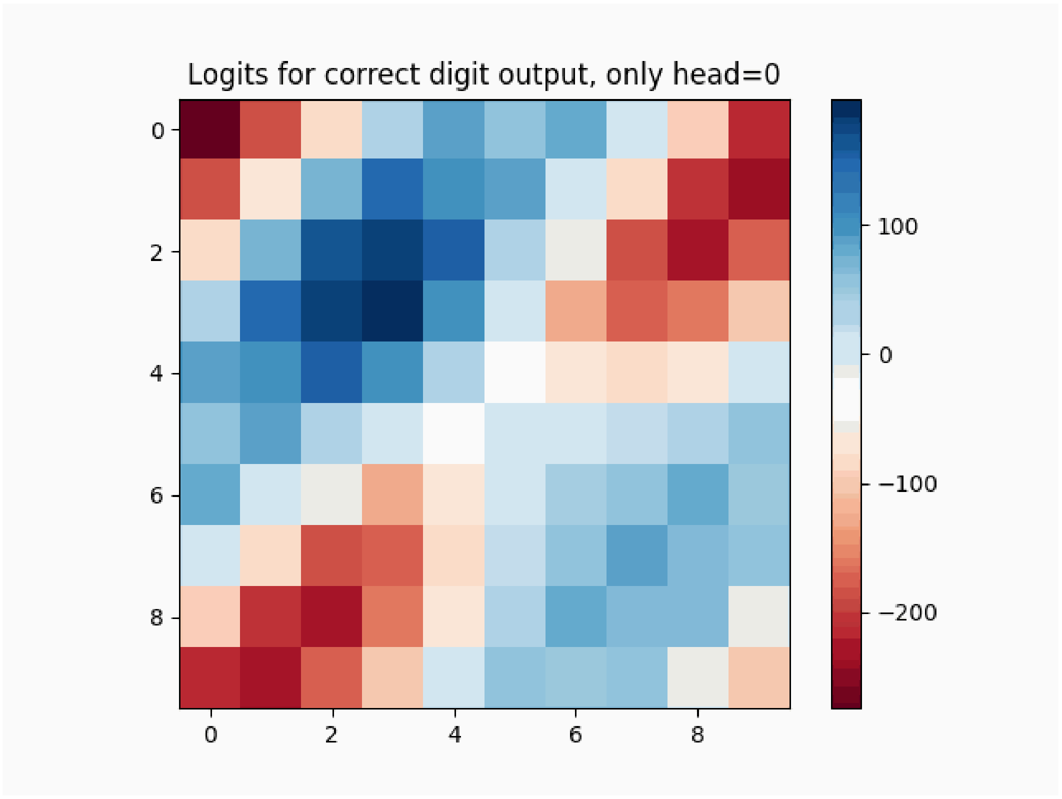


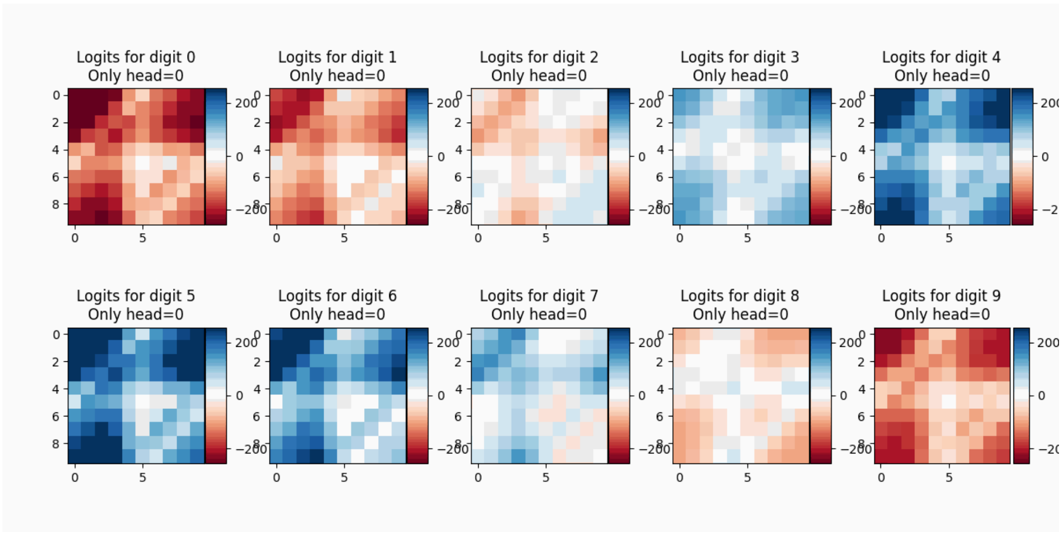
Ablating various heads (warning: animations)

We aim at showing that there are several attention circuits that distribute over attentional heads given we have more circuits that heads to do the task.

We talk about the trigrams (a b -> c) as our attention circuits and a and b corresponds to integer digits added together and c represents the correct output logit. Setting both a and b to vary in ranges of 0…9, we set 100 circuits while our model has 31 heads.

We tried to determine the effects that each individual head had on the logits. To do this, we zero-ablated every head except for the head of interest, and then plotted the logits for the correct digit in position 12 (the first output position) for every possible digit in positions 0 and 6 (first digit of first and second number, respectively. Note again that the numbers are written in least-significant-digit-first order, which is the opposite direction from how numbers are normally written).

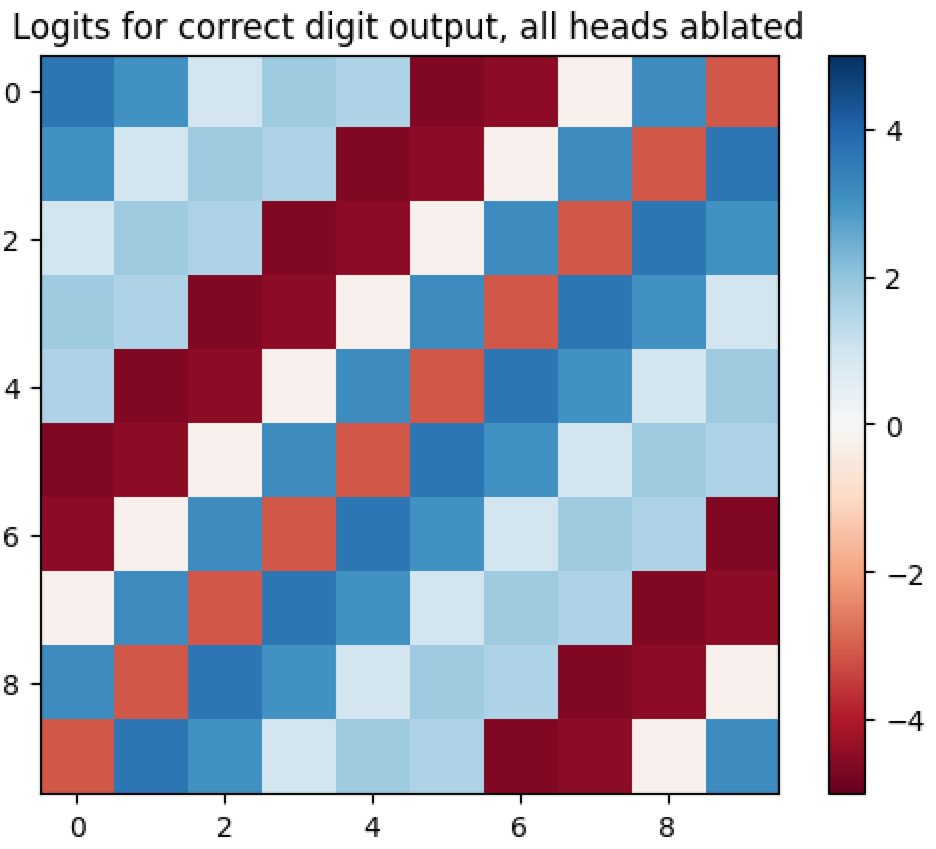


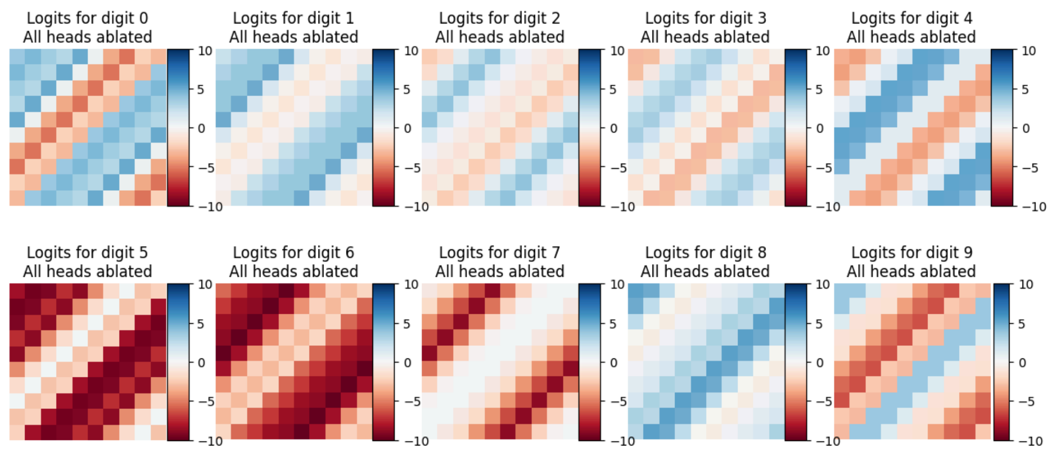


We see that for each given circuit, there is a combination of heads contribute positively to the logit of the correct prediction while some of them contributes negatively to the correct logits. The combination of heads contributing towards the correct prediction varies for each head. The magnitude of the logit contribution is demonstrated through the headmap.

The interactive diagrams above tells us that different combinations of heads work together at predicting the correct output for different tasks. (In other words circuits are stored over attention heads)

As a baseline, we also checked what this looked like with all heads zero-ablated:
The 2 blue lines above





Further Research Directions

More thoroughly understanding the horizontal composition attention heads are learning to compress data.

The attention heads need to cooperate in the sense that there are more possible output combinations than there are attention heads. For example, setting aside compound behavior such as carrying, if we restrict our focus to summing two 1-digit numbers to a value <= 9, there are (10 choose 2) = 45 solutions to x+y <= 9, with 0 <= x, y <= 9, and 32 heads here. Writing code to determine the precise compression the attention heads learned would be cool.

Another idea: finding attention head superposition in TinyStories-1L-21M. We spent a good amount of time setting this up but could not think of a way during the hackathon period. We think this is possible but hard to set up. We need a "fixed task" for the 16 attention heads to figure out (in a skip trigram?), and successfully distinguish between at least 17 possible next tokens (or "attentional features"). Not only this, but we would also need to distinguish between what are "attentional features" and memorized information from the MLP neurons. I think defining this is quite difficult and the human-level interpretation might be task specific and hard to describe in a vacuum.

Project 2: Searching for superposition in the MLP of tinystories-2L

Goal

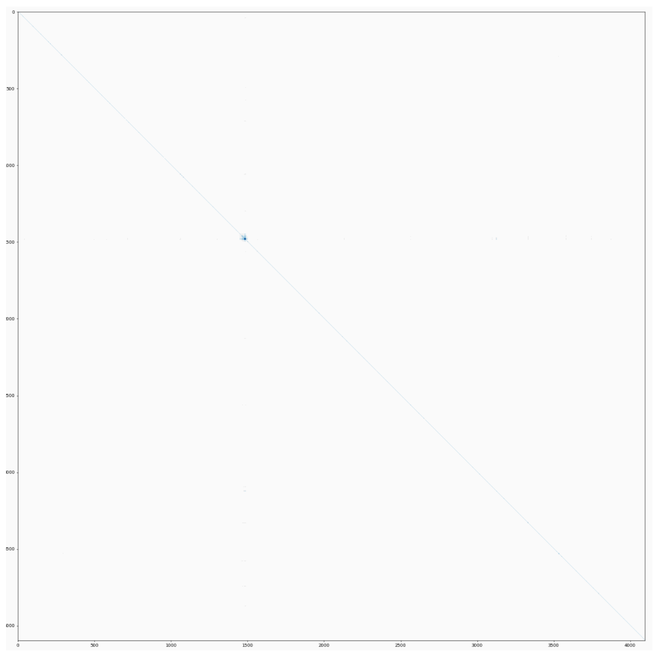
Just stare at the weights really hard and see if a meaningful basis falls out

Exploration

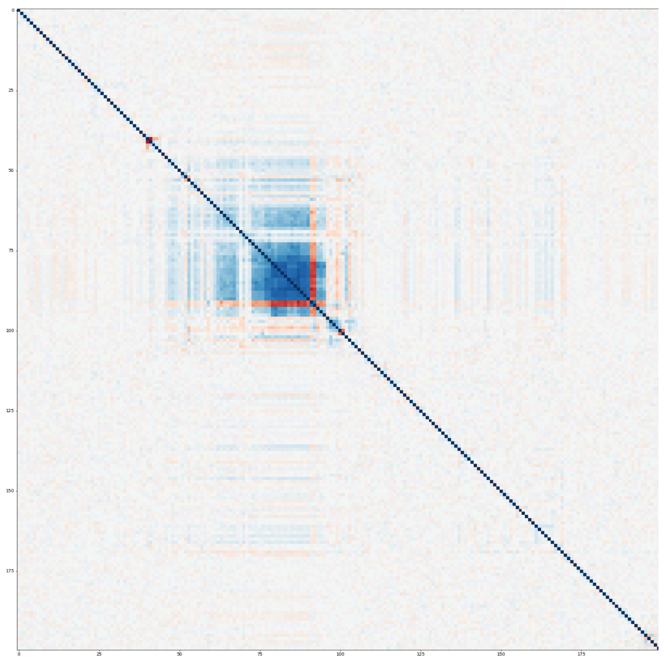
We had seen that the W^TW approach was pretty promising for finding meaningful directions, so we started out by grabbing one of those from the layer 1 MLP of tinystories-2L. We then normed the W^TW tensor to get cosine similarities between the directions the neurons pointed in activation space (d_mlp = 4096 so this was a 4096 x 4096 matrix).

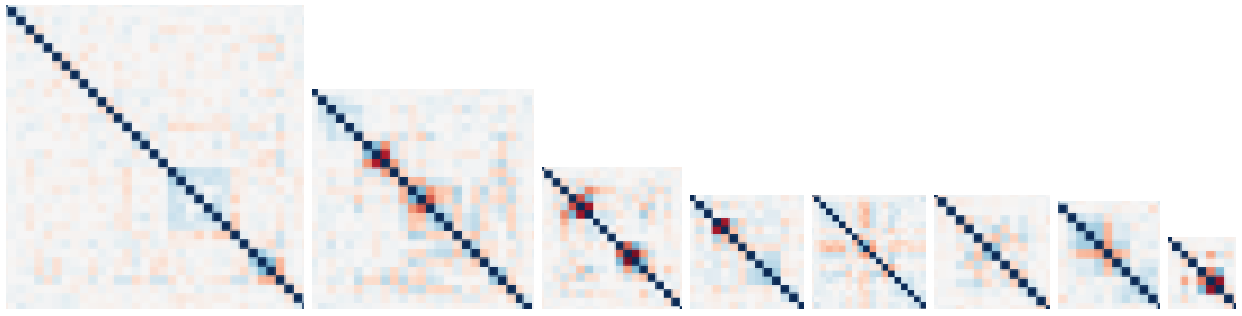
We then tried to find an ordering of the neurons that made it easiest to visualize patterns. We did this by constructing a path that maximized-ish the squared cosine similarity between adjacent neurons, and then plotting the permuted cosine similarity matrix. Our hypothesis was that, if there was any meaningful structure within the model weights themselves, it would show up on those graphs, and specifically along the main diagonal.

Zooming in on and moving along the diagonal

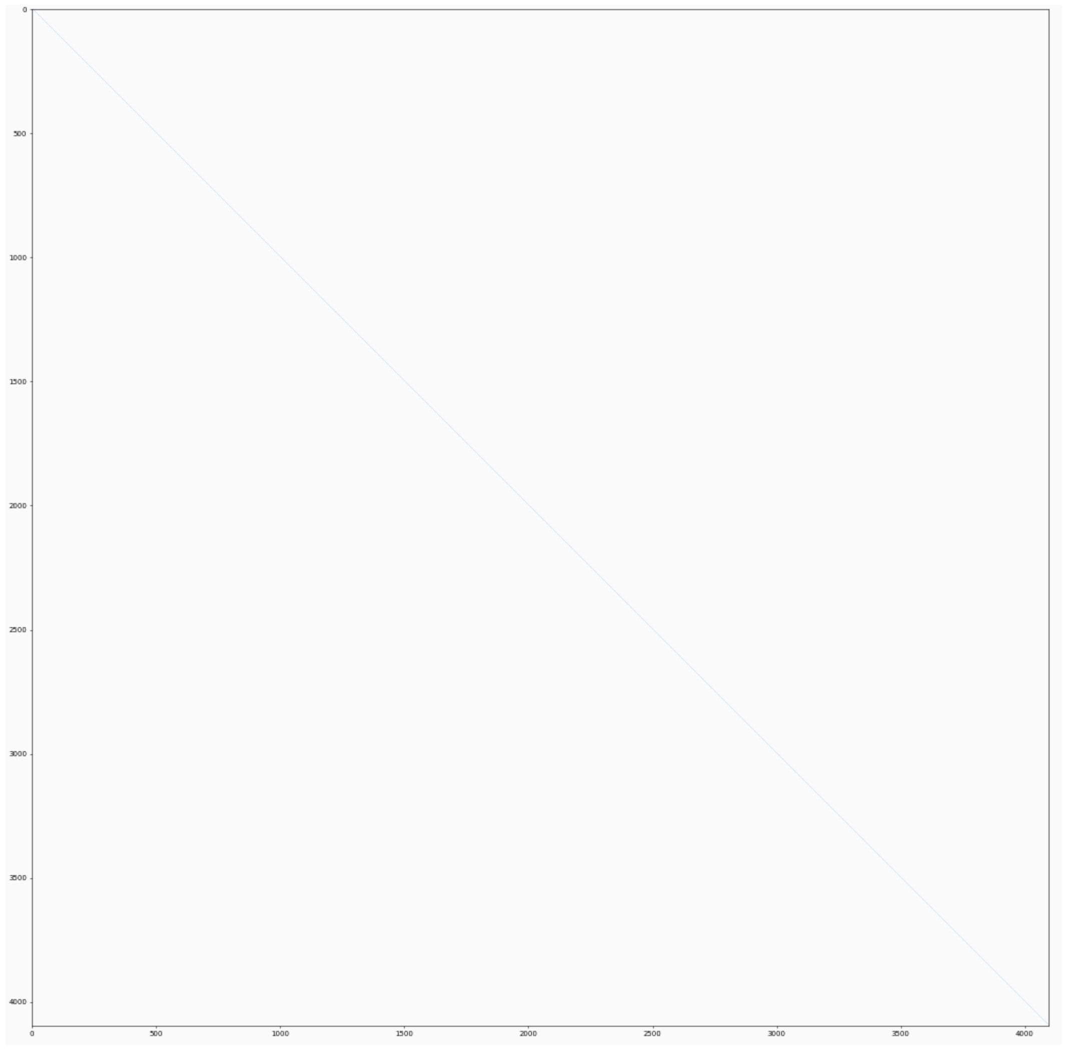


Interesting spots along the diagonal





So there does seem to be quite a bit of structure there. We verified that if we replaced the 4096x1024 MLP_out matrix with a randomized 4096x1024 matrix, and reran the same steps, we did not see similar structure



Further Research Directions

- Use an actually-principled clustering algorithm to find structure
- Look at the giant blob of structure. Stare at those neurons. See what they mean. Ideally in Neuroscope (see project 3)
- Do that for the other giant parameter matrices as well
- Do a similar thing, but cluster by activations on real-world data rather than by weights

Project 3: Neuroscope for TinyStories

Code here: https://github.com/joshuadavid/neuroscope

Goal

Get Neuroscope up and running with tiny-stories (both the model and the dataset)

Progress

Hopefully soon. Forked the repo here and have been hacking away at it.

Further Directions

Once it's up and running, maybe add activation histograms for each neuron and also link to the neurons with the highest and lowest cosine similarity

Project 4 : Understanding superposition in toy models

Structure

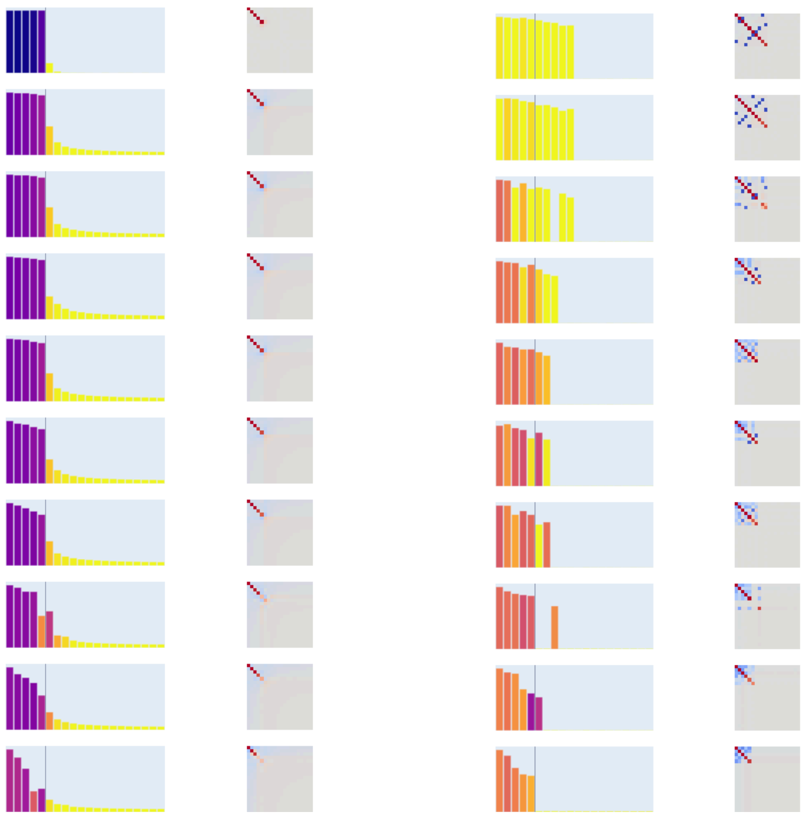
1. Train toy models to predict target functions: with ReLU and GeLU activations
    a. y = |x|
    b. x AND y
    c. x OR y
    d. max(|x|,|y|)
    e. x^2
2. Compare performances, visualize the neuron stacks to see superposition happening and observe differences
3. Extend anthropic's toy model to include dropout, l1 regularization, different activation functions, etc.
    a. Analyze what happens when there is non uniform sparsity
4. Understanding dropout's effects on superposition
    a. Vary levels of feature sparsity and dropout rates
    b. Visualize impact of dropout on the distribution of points on integer fractions

Summary

Understanding dropout's effects on superposition

Dropout has a significant effect on superposition. In low sparsity situations, it both generates interference between features and suppresses less important features. This could be due to the noise induced during training by dropout.

For higher sparsity levels, dropout suppresses the superpositions that the model would normally learn, focusing more on representing the most important features. At extreme dropout levels, solutions resemble non-sparse feature solutions with non-zero negative weights on the diagonal, the role of which is not yet clear.

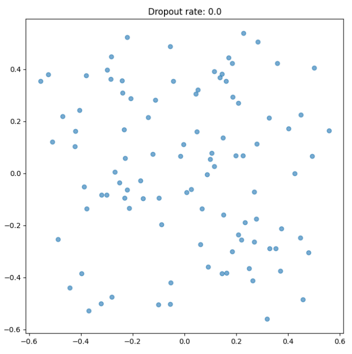


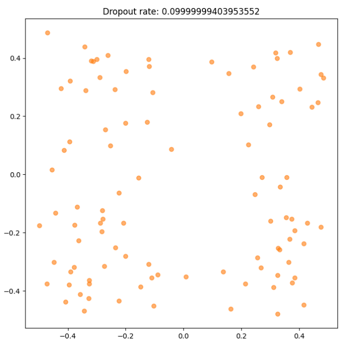
We find that at high dropout levels, dropout in the ReLU model might discourage feature alignment with the coordinate basis.

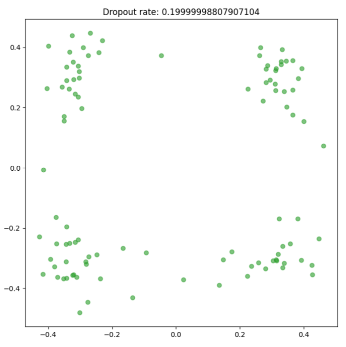
Moreover, dropout may induce a non-random privileged basis that isn't neuron-aligned, possibly because the noise may make the basis preferentially avoid the coordinate-aligned basis. This can be tested by analyzing an arbitrary feature encoding and visualizing its basis weight distribution.

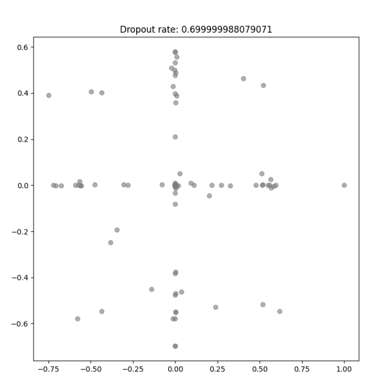
The ReLU output model seems to undergo three phases as dropout increases. At zero dropout, it's rotation invariant. As dropout increases, the model adopts an anti-aligned equilibrium with the coordinate basis. Beyond a 50% dropout rate, coordinate basis-aligned features are encouraged again.

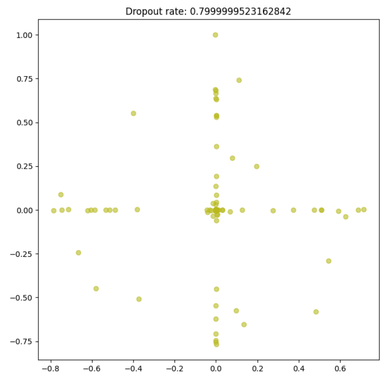
Dropout makes phase transitions smoother in comparison to deterministic models, even with small amounts of dropout.

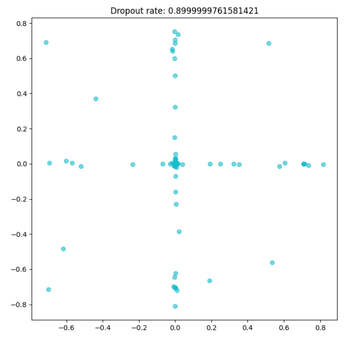




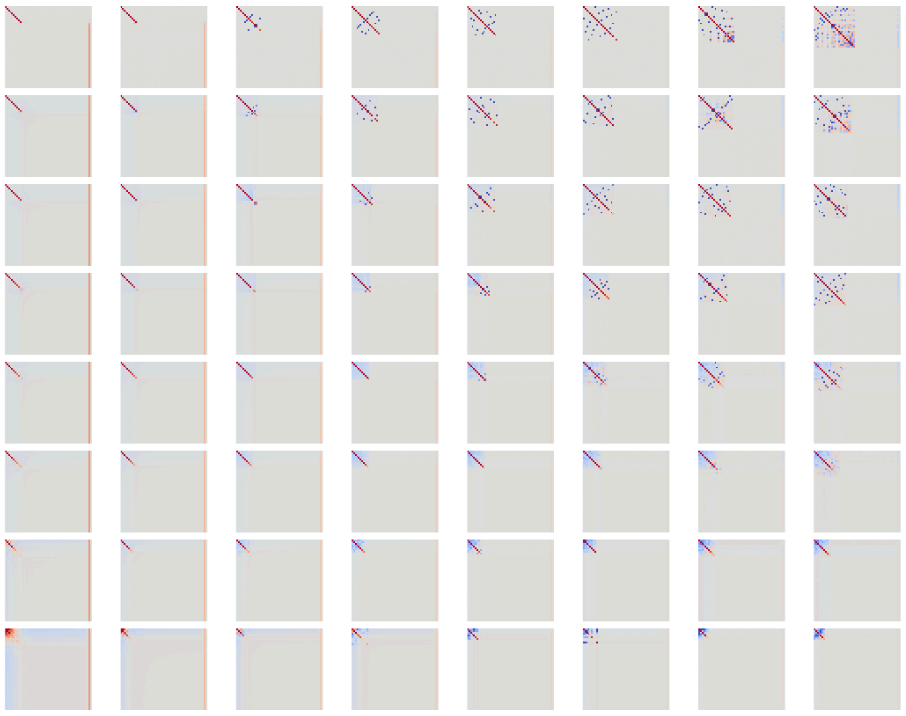








Dropout seems to suppress superposition, particularly the antipodal pairs favored by the deterministic model. Dropout models show negative feature interactions along with a positive diagonal.



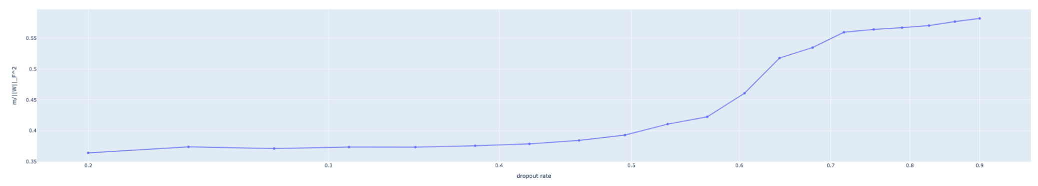
In a low-sparse environment, it's causing some sort of cross-talk between features and dimming down the less important ones, a curious outcome that might be tied to the noise that dropout stirs up during training.

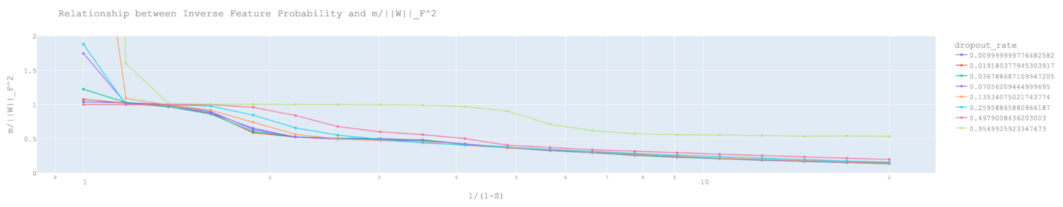
Now, when we crank up that sparsity, dropout flips the script and starts muffling those superpositions that the model would typically learn. Instead, it pushes the model to spotlight the main features. At super-high dropout levels, it starts to look like our plain old non-sparse feature solutions, but with some negative weights on the diagonal

The appearance of negative weights on the diagonal could be a coping mechanism developed by the model to deal with high dropout rates. These negative weights might be helping to balance or offset the impact of the lost information due to dropout. Essentially, they could be performing a kind of "error correction," compensating for the dropout-induced noise in the input data.

They could also be a way to encode inverse or opposing relationships between certain features in the data, allowing the model to robustly learn the complexities of feature space, even under high dropout conditions.

The dimensions per feature seem to increase sharply with increased dropout, showing that the model tries to 'work around' dropout.



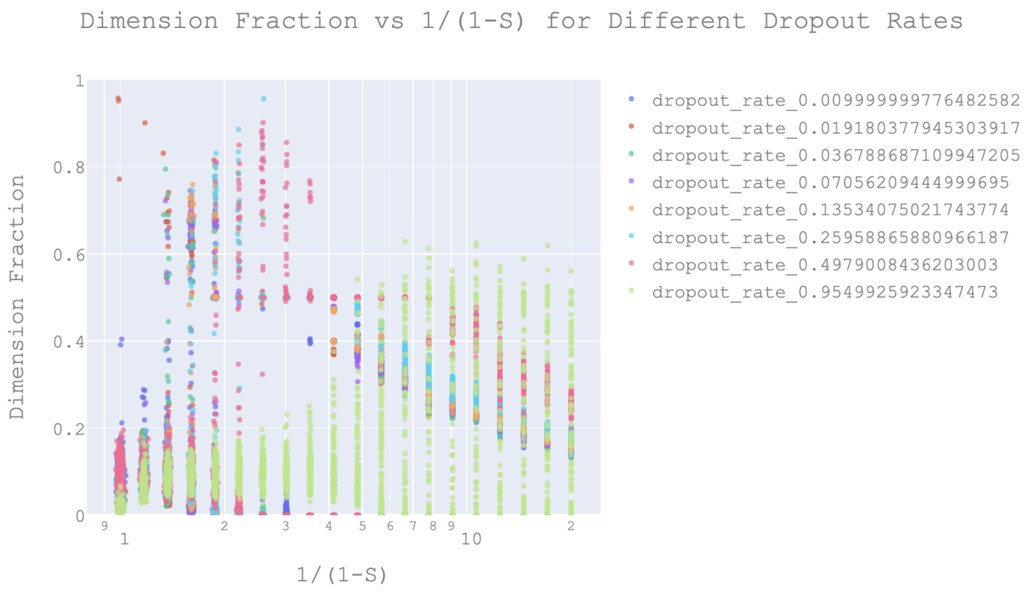


Dropout leads to redundant encoding in dense scenarios and suppresses superposition in sparse conditions, typically reducing the feature-dimension ratio.

In some cases, higher dropout rates result in fewer dimensions per feature. It's unclear whether this makes the model more robust or if dropout noise simply reduces the advantage offered by various polytopes.

Dropout introduces training noise, making feature representation more challenging, which may force the model to use more dimensions, thus reducing its representational power. However, it can also decrease the 'stickiness' of certain superposition solutions, which may be beneficial for learning dynamics.

At extreme levels of dropout, certain 'sticky' solutions still exist, interpretation unclear.



Non-uniform sparsity for toy model

In scenarios of non-uniform sparsity - where we have a mix of less sparse and very sparse features across different instances - the model's strategy changes. For less sparse features, which are more prevalent, the model tends to employ less superposition. The higher prevalence of these features provides ample learning opportunities for the model, resulting in a more straightforward, direct mapping from the features to the output.

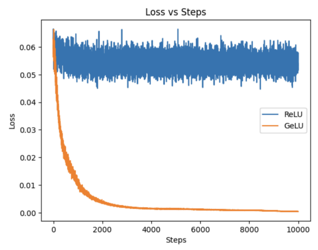
On the other hand, for very sparse features, the ones that aren't seen as often, the model tends to resort to a significant level of superposition. Since these features don't show up as much, the model has fewer instances to learn from, prompting it to use more intricate mappings to ensure the features are represented accurately.

However, the effects of non-uniform sparsity aren't consistent across the board - they vary according to the specific task that the model is learning to perform. If the model is trained with a high priority given to the very sparse features, it tends to learn more complex representations that involve a superposition across both classes of features.

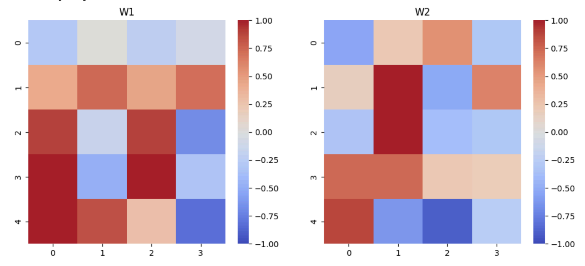
Interestingly, we also found that the choice of activation function could play a role in how the model deals with superposition. When we repeated our experiments with GeLU instead of ReLU, the model seemed to learn a more intricate representation and exhibited higher levels of superposition. The smooth transitioning nature of GeLUs, from being 'off' to 'on', could allow for more nuanced interactions between features in superposition. This opens up the possibility for a feature to partially activate a neuron, leading to more sophisticated patterns of interference, a fascinating direction for future investigation.

Train toy models to predict target functions

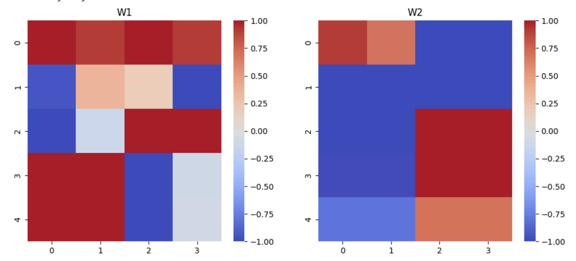
x AND y



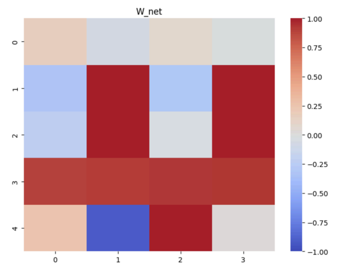
Visualizing weights for model ReLU

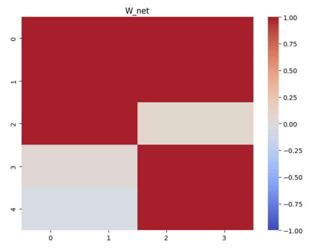


Visualizing weights for model GeLU

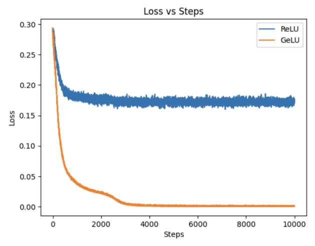


Since W1 and W2 are trained disjoint, it probably makes more sense to look at the combined weight matrix multiplying them

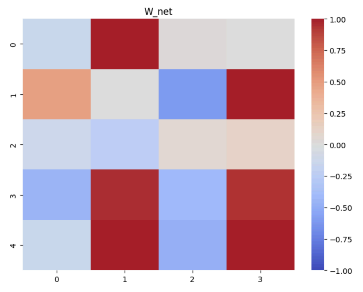




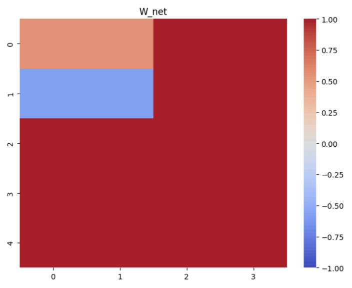
x OR y



ReLU:



GeLU:

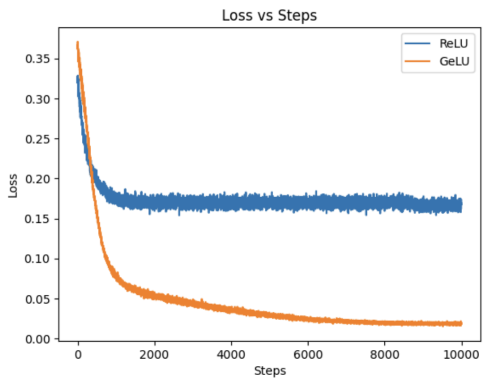


y = x^2

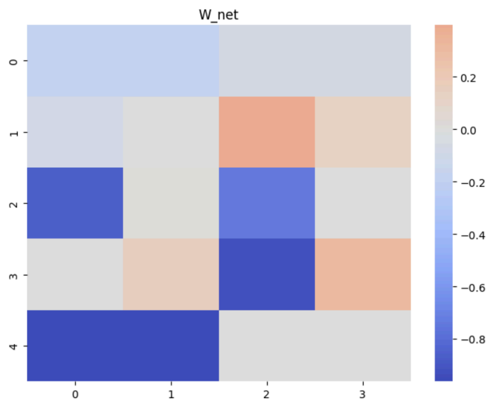
When we use ReLU activation function, the weights learned are relatively balanced between positive and negative values. This indicates that the model is making use of both positive and negative associations between features to perform the task.

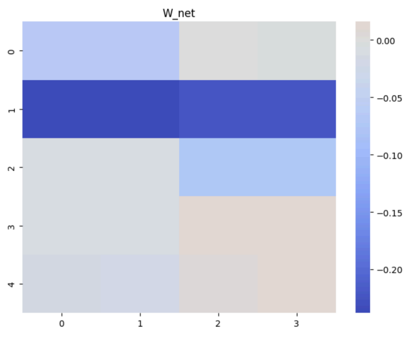
In the GeLU model's weights, we see more extreme weight values compared to the ReLU model. This could potentially indicate that the GeLU model is making use of its ability to learn from both positive and negative inputs, leading to a more nuanced understanding of the feature space.

max(|x|,|y|)



ReLU:





GeLU:

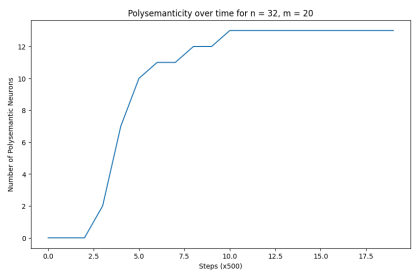
For this function, the model seems to have learned a very different representation than the previous functions, perhaps because it did not have enough neurons, it learns a simpler representation. The ReLU model does not perform as well as the GeLU model, surprisingly.

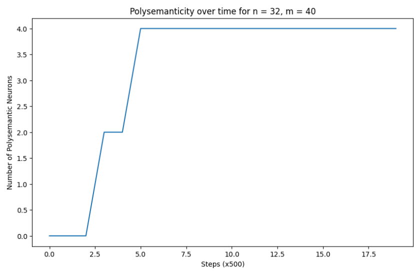
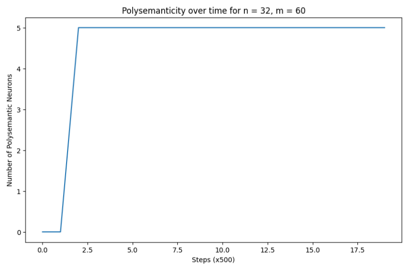
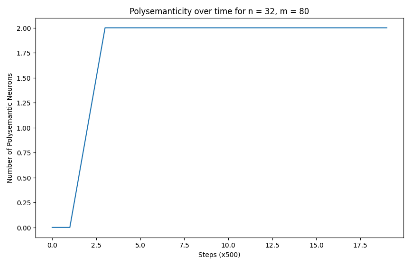
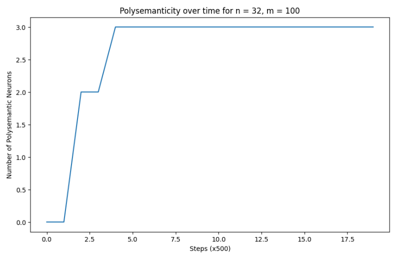
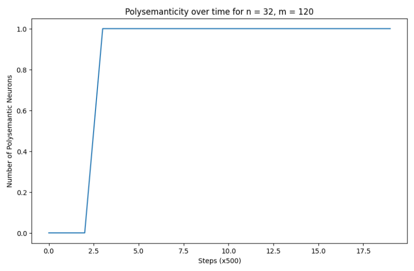
Varying the sizes of hidden layers

We train a 1 layer autoencoder model, very similar setup to the anthropic's toy models of superposition. We tied encoder and decoder weights (Transposed) and put ReLU on the output layer for the activation function.

We train it for learning 32 input features and vary the sizes of the hidden layer as follows: m = [20, 40, 60, 80, 100, 120]

We observe that in all cases, regardless of whether the neurons are more than enough to represent the input features, there is polysemanticity.

Number of input features	Number of neurons in hidden layer	Development of polysemanticity during training
32	20	

32	40	
32	60	
32	80	
32	100	
32	120	

3. Discussion and Conclusion

We made pretty pictures. You can learn a lot about models by staring at them really hard. And hopefully develop intuition for the underlying math. Hopefully.

4. References

Elhage, et al., "Toy Models of Superposition", Transformer Circuits Thread, 2022.

Olah, C., Batson, J., Carter, S. (2023). *Circuits Updates – May 2023* https://transformer-circuits.pub/2023/may-update/index.html#attention-superposition

Eldan, R., Li, Y. (2023). *TinyStories: How Small Can Language Models Be and Still Speak Coherent English?* arXiv:2305.07759v2 https://doi.org/10.48550/arXiv.2305.07759