Compressed Computation: Dense Circuits in a Toy Model of the Universal-AND Problem

Anonymous Author(s)
Affiliation
Address
email

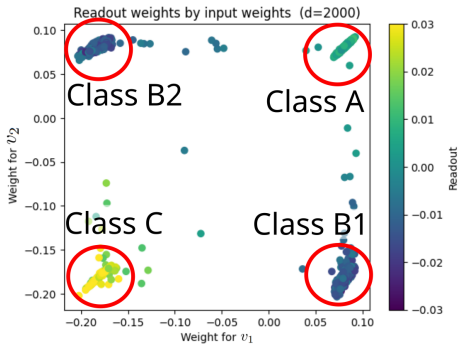
Abstract

Neural networks are capable of superposition - representing more features than there are dimensions. Recent work considers the analogous concept for computation instead of storage, proposing theoretical constructions. But there has been little investigation into whether these circuits can be learned in practice.

In this work, we investigate a toy model for the Universal-AND problem which computes the AND of all $\binom{m}{2}$ pairs of m sparse inputs. The hidden dimension that determines the number of non-linear activations is restricted to pressure the model to find a compute-efficient circuit, called compressed computation.

We find that the training process finds a simple solution that does not correspond to earlier theoretical constructions. It is fully dense - every neuron contributes to every output. The solution circuit naturally scales with dimension, trading off error rates for neuron efficiency. It is similarly robust to changes in sparsity and other key parameters, and extends naturally to other boolean operations and boolean circuits. We explain the found solution in detail and compute why it is more efficient than the theoretical constructions at low sparsity.

Our findings shed light on the types of circuits that models like to form and the flexibility of the superposition representation. This contributes to a broader understanding of network circuitry and interpretability.



Class A	$y_i = \text{ReLU}(uv_1 + uv_2 + X_i + b)$
Class B1	$y_i = \text{ReLU}(uv_1 + lv_2 + X_i + b)$
Class B2	$y_i = \text{ReLU}(lv_1 + uv_2 + X_i + b)$
Class C	$y_i = \text{ReLU}(lv_1 + lv_2 + X_i + b)$

v_1	v_2	A	B1	B2	C	$4(A + C - B1 - B2)$
0	0	0.05	0.05	0.05	0.05	0
0	1	0.15	0	0.15	0	0
1	0	0.15	0.15	0	0	0
1	1	0.25	0	0	0	1

Approximate truth table for each neuron class

Figure 1: Our found circuit: For every pair of inputs (e.g. v_1, v_2) the model neurons separate into 4 classes based on response to those inputs. A linear combination of those classes recreates the AND operator, with some error (not shown).

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

1 Introduction

Models have been found to learn to store a set of sparse features m in a vector of dimension d, where $d \ll m$. They achieve this with a linear representation of nearly orthogonal vectors per feature, called **superposition** [Elhage et al., 2022]. By relying on the sparsity of feature activation, and tolerating a small amount of error due to overlap, an exponential number of features can effectively stored in the vector. An understanding of superposition has been critical for mechanistic interpretability of models: it has led to foundational concepts in interpretability, the development of interpretability tools (e.g. [Cunningham et al., 2023], [Ameisen et al., 2025]) and provided evidence for the Linear Representation Hypothesis [Park et al., 2023].

But storage of features isn't a full descriptor of how models work. Networks must also do efficient, useful computation with features. This presents two difficulties for the model. Firstly, models must be able to efficiently work with features that are already represented in superposition. This is called **computation in superposition** [Hänni et al., 2024]. Secondly, models need to deal with the fact that they are limited to a few parameters and calculation units (non-linear activation of neurons), called **compressed computation** [Braun et al., 2025].

Just as an understanding of superposition is necessary to analyze activation space effectively, an understanding of computation will be necessary to analyze weights and circuits. Without these understandings, activations/circuits can appear inscrutably commingled. A good theory of computation will allow tools and analysis of weights and circuits in a network.

Theoretical models have been proposed that explore computation in superposition ([Hänni et al., 2024], [Bushnaq and Mendel, 2024], [Adler and Shavit, 2024]). These papers find constructions that operate directly on input/output features in superposition, and establish bounds on accuracy or model size. These constructions generally rely on **sparse weights** to ensure that neurons do get too much interference from irrelevant inputs, allowing estimates to be measured. We call a construction sparse if the parameter weights are zero or negligible with probability that approaches one.

Adler and Shavit [2024] in particular notes that "logical operations like pairwise AND can be computed using $O(\sqrt{m'}\log m')$ neurons and $O(m'\log^2 m')$ parameters. There is thus an exponential gap between the complexity of computing in superposition versus merely representing features, which can require as little as $O(\log m')$ neurons".

Thus even with computation in superposition, models are very likely to face bottlenecks in compute, needing compressed computation. Our work focuses specifically on this case. We re-use the same **Universal-AND problem** setting of Hänni et al. [2024], but eliminate the main source of computation in superposition by using monosemantic input and output. A narrow hidden dimension is used to force reuse of neurons for multiple circuits, exploiting the sparsity of the inputs, and controlling the error from unrelated inputs interfering with the calculations.

The Universal-AND problem is the task of efficiently emulating a circuit that takes m sparse boolean inputs and produces $\binom{m}{2}$ outputs that each compute the AND operation of a given pair of inputs, described further in section 3. We train toy models with one layer of ReLU on this problem for various settings of sparsity, s, and hidden dimension size, d.

We find that the model learns a simple binary-weighted dense circuit, i.e. the layer weights only take on two different values. This circuit effectively computes then stores all $\binom{m}{2}$ outputs in superposition with some degree of noise, which can then be linearly read out with an additional linear layer. This circuit is only used when the inputs are sufficiently sparse. The same circuit design is used for almost all values of d, just with higher and higher noise from unrelated inputs.

The circuit design is fairly robust and general. It can be extended to other Boolean circuit operations straightforwardly. We supply a theoretical analysis for how this circuit works and contrast its efficiency to the sparse construction described in Hänni et al. [2024].

This circuit is particularly interesting as every intermediate neuron gives a useful contribution to every output. The model uses the increasing values of d as opportunities to distribute each computation as widely as possible, reducing error. It does not form distinct, sparse, non-overlapping circuits, even for $d = \binom{m}{2}$, where a naive perfect solution assigning each AND operation its own neuron would be possible.

Our contributions include:

- A novel construction for solving the Universal-AND problem with a 1-layer MLP with linear readout (section 5.1). We explain how the construction works and can be extended to other Boolean circuitry in one layer (section 6.1).
- Evidence that this construction can be learned in standard training dynamics (section 5).
- An approximate analysis of the asymptotic error of this circuit, with each input having variance $\mathcal{O}(s^2/d)$. (section 6.2).

2 Related Work

Adler and Shavit [2024, *On the Complexity of Neural Computation in Superposition*] establishes lower/upper parameter and neuron bounds for circuits such as Universal-AND. Like Hänni et al. [2024], it supplies a sparse construction, and computes error bounds asymptotically. It also supplies an information-theoretic lower bound on the bits of parameters. Our model does not approach these theoretical limits, as we generously allocate parameters and only seek to restrict neurons. The paper establishes an exponential gap between the number of features that can be stored in an activation, and the number of computations that can be done in an equally sized network, demonstrating that models are likely to face strong pressures to compress computation to as few neurons as possible.

Elhage et al. [2022] and Scherlis et al. [2022] explore superposition in toy models and investigate how computation is done. The problem setups used in both cases are focused on representation and have largely trivial computation. They rely on a hidden layer smaller than the input size, which means they cannot easily distinguish between the computation in superposition and compressed computation.

Bushnaq and Mendel [2024, *Circuits in Superposition: Compressing many small neural networks*] examines computation in superposition for a different non-trivial problem. They also focus on sparse theoretical constructions and their asymptotic behavior, and do not explore what models actually learn or if a dense circuit could perform better.

3 Background And Setup

Hänni et al. [2024, *Toward A Mathematical Framework for Computation in Superposition*] first posed the question of how computation in superposition works. They introduce the **Universal-AND** problem, which computes the full set of pairwise AND operations on a set of inputs.

Formally, the Universal-AND problem considers a set of Boolean inputs $v_1, \cdots, v_m$ taking values in $\{0, 1\}$. The inputs are s-sparse, i.e. at most s are active at once, $\Sigma v_i \leq s$. The problem is to create a one-layer MLP model that computes a vector of size d, called the neuron activations, such that it's possible to read out every $v_i \wedge v_j$ using an appropriate linear transform.

The problem as originally stated encodes the m inputs in superposition in activation vector of size d_0 but we omit this step and work directly on v_i.

For values of $d \ll \binom{m}{2}$ there is not enough neurons to naively compute every possible AND pair separately. But if we assume $s \ll d$, then the model can take advantage of the sparsity of the inputs to re-use the same model weights for unrelated calculations.

Our model can be described mathematically as

$$\mathbf{y} = \mathrm{ReLU}(W\mathbf{v} + \mathbf{b})$$

$$\mathbf{z} = R\mathbf{y} + \mathbf{c}$$

where $W \in \mathbb{R}^{m \times d}, b \in \mathbb{R}^d, R \in \mathbb{R}^{d \times m^2}, c \in \mathbb{R}^{m^2}$.

In other words, W and $\mathbf{b}$ describe the "compute layer", while R and $\mathbf{c}$ describe the "readout layer". The existence of a readout layer makes the model effectively 2 layers deep. The readout weight matrix is large so the model is not bottlenecked on parameter count (unlike Adler and Shavit [2024]). Rather, the bottleneck is on the non-linear activations, the neurons. The readout layer should be understood as a trainable proxy for some more realistic setting, such as trying to learn a specific linear probe, or further model layers that are interested in a large fraction of possible pairwise AND operations.

For convenience of indexing, the output has dimension m^2 corresponding to ordered pairs. It is symmetric, and the diagonal is unused. This gives a target model output of

$$\hat{z}_{im+j} = \begin{cases} v_i \wedge v_j & i \neq j \\ 0 & i = j \end{cases}$$

4 Method

We trained the above two-layer model on synthetic data where exactly s values of v_i are randomly chosen to be active (i.e. take value 1).

We used RMS loss with different weighting per sample. This is because v_i are sampled uniformly, so test cases of the form $0 \wedge 0$ are much more common than $1 \wedge 0$ and $1 \wedge 1$. So we up-weighted the loss so that the expected contribution from each of those three cases is equal. This encourages the model to focus on the few active results rather than the vast sea of inactive results. We can justify this because the second layer is intended as a "readout" layer. It is a proxy for a larger network that needs a lot of Boolean operations, which would also have unbalanced optimization pressure.

Weight decay of 10^{-6} is used to regularize the network. This matches real-world training runs, and encourages the model to focus on optimal circuits[1]. Each model was trained with 6000 epochs of 10k batches.

We used $m = 100, s = 3$ except where noted differently.

All experiments were trained on a single A40. The full code can be found in the supplemental materials[Anonymous].

5 Results

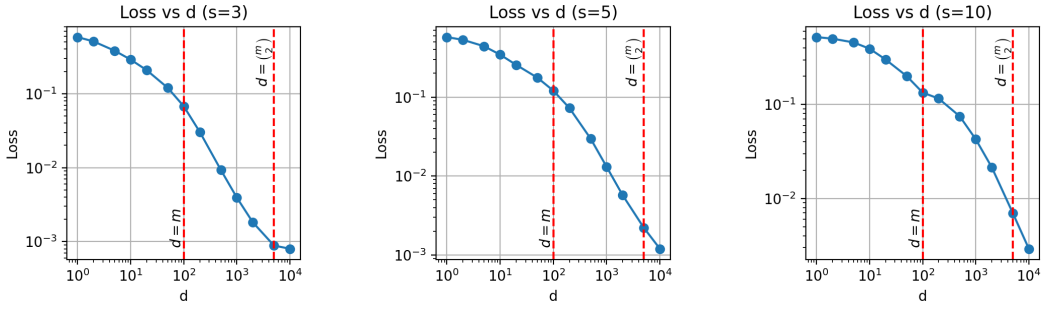


Figure 2: Model loss as d increases.

We find that at low s values, the model does find solutions that are capable of solving the Universal-AND problem, even extending to extremely low values of d. The model weights take on a simple pattern of binary weights described below.

At higher values of s (starting at 10 for $m = 100$) this circuit breaks down. In some cases, the weights of W do not neatly separate into two values, but the readout matrix R is structured similarly (appendix A.1). In other cases, the model prefers to learn an additive approximation $z_{im+j} = 0.4(v_i + v_j)$ (fig. 6).

Figure 9 range of values the Binary Weights Circuit is learnt on.

[1]Runs without weight decay show similar but higher variance results (appendix C.3).

5.1 The Binary Weighted Circuit

The model generally tends towards neuron weights that are binary, i.e. takes on only one of two different values[2]. The specific choices of value seems randomized.

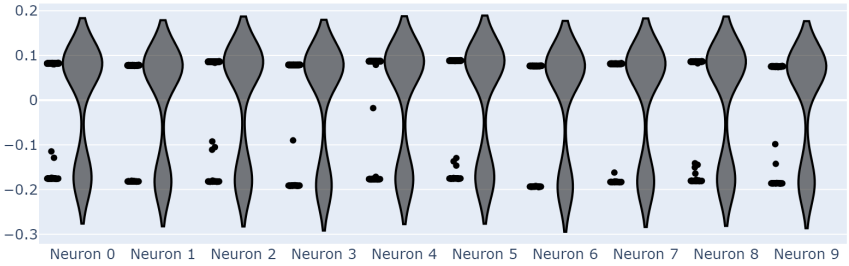


Figure 3: Distribution of W_{ij} values for $1 \leq j \leq 10$.

Expressed mathematically

$$W_{ij} = \begin{cases} u_i & \text{with probability } p_i \\ l_i & \text{with probability } 1 - p_i \end{cases}$$

We discuss why how this circuit works in section 6.1.

Figure 4 charts the specific values of u_i, l_i, p_i in shown in fig. 4. It illustrates that all neuron weights are clustered in a tight region and $u_i p_i + l_i(1 - p_i) \approx 0$.

Figure 4: Distribution of upper/lower weights by neuron ($d = 2000$). Error bars show 90th-percentile deviation from the weight being near to either the upper/lower bound. Charts for more values of d can be found in appendix B.

5.2 Readout Charts

Another way of viewing the model is in terms of how the neurons are read out by matrix R. Pick two arbitrary inputs (say v_1 and v_2), then plot each neuron in a scatter chart based on their weights (i.e. W_{i1} on the x-axis, W_{i2} on the y-axis). Color the neurons based on their readout weight for $v_1 \wedge v_2$ (i.e. $R_{(1m+2),i}$).

The four corners of fig. 5 correspond to classes A (top right), C (bottom left), B1, and B2 as described in section 6.1. Class C has higher weights per-neuron as there are fewer neurons in that class than class A.

[2]More formally, the measured circuits have a bimodal distribution with very low deviation from the modes. In appendix B I give a formula that measures distance from a true binary distribution, which is small.

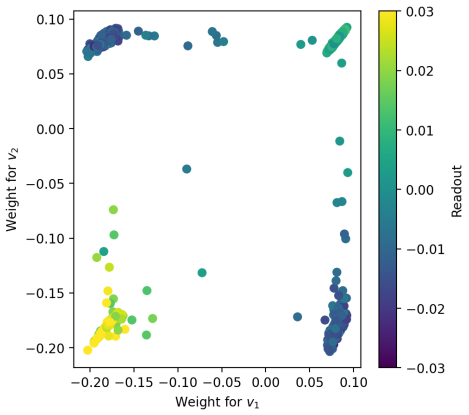


Figure 5: Readout weights by input weights ($d = 2000, s = 3$) showing the Binary Weighted Circuit

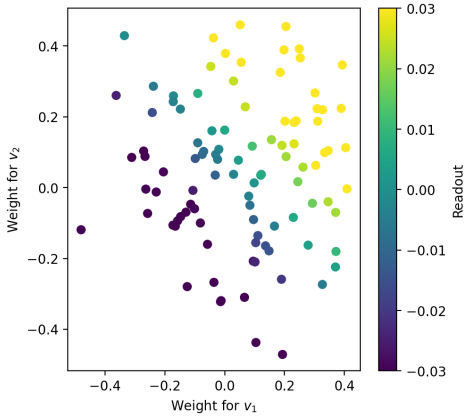


Figure 6: Readout weights by input weights ($d = 100, s = 50$) showing a degenerate circuit.

6 Analysis

6.1 Circuit Analysis

The results show that for a wide range of parameters, the model tends towards binary weights where for any given neuron, the weights it uses only take on two different values, with no discernible pattern. Expressed mathematically

$$W_{ij} = \begin{cases} u_i & \text{with probability } p_i \\ l_i & \text{with probability } 1 - p_i \end{cases}$$

This resembles constructions from Hänni et al. [2024]. Their main result[3] is a sparse construction which followed this exact pattern. We'll call it the **CiS Construction**. It takes on values

$$u_i = 1 \quad l_i = 0 \quad p_i = \log^2 m/\sqrt{d}$$

In the CiS Construction p_i is small, and l_i is zero, meaning the neurons were only sparsely connected. This property was key in proving an upper bound on the loss of the model.

But in the learned Binary Weighted Circuit we see quite different values. For $d = 1000$, we see values similar to the following, which we will use in the subsequent worked example[4].

$$u_i = 0.1 \quad l_i = -0.25 \quad p_i = 0.75$$

Notation: When the values are a constant, we'll omit the i subscript.

Unlike the CiS Construction, this is dense. Every neuron reads a significant value for every possible input.

We now explain how neurons in this architecture can be used to approximate the AND operation with a linear readout. Without loss of generality, we focus on the first two input variables — that is, we aim to read out $v_1 \wedge v_2$ from the activations of d neurons that share the same values for u, l, and p, differing only in their randomly initialized weights.

We can subdivide the neurons into 4 classes based on the weight used for each of the first two inputs, W_{i1} and W_{i2}.

[3] Hänni et al. [2024] also mentions a randomized dense construction which is also relevant. See appendix A.1

[4] As the readout matrix R can supply an arbitrary positive scaling, the only important details of u / l are their signs and ratio. The magnitude tends to be around $1/\sqrt{d}$ as this minimizes regularization loss (weight decay) in a two layer network. Appendix B includes more details on the exact values seen of u and v for different d.

Class A	$y_i = \text{ReLU}(uv_1 + uv_2 + X_i + b)$	occurs with proportion p^2
Class B1	$y_i = \text{ReLU}(uv_1 + lv_2 + X_i + b)$	occurs with proportion $p(1-p)$
Class B2	$y_i = \text{ReLU}(lv_1 + uv_2 + X_i + b)$	occurs with proportion $p(1-p)$
Class C	$y_i = \text{ReLU}(lv_1 + lv_2 + X_i + b)$	occurs with proportion $(1-p)^2$

Where X_i, the contribution from other inputs, is defined as $\sum_{k=3}^{m} W_{ik} v_k$. We call X_i the interference term.

To simplify the explanation, for now we'll ignore X_i. Later in section 6.2 we will model it as random variable of mean zero. We'll also set $b = 0.05$; the exact choice of constants is not relevant to the key argument.

Table 1 shows a truth table for the results of each class of neuron for the 4 possible values of v_1, v_2.

v_1	v_2	A	B1	B2	C	$4(A + C - B1 - B2)$
0	0	0.05	0.05	0.05	0.05	0
0	1	0.15	0	0.15	0	0
1	0	0.15	0.15	0	0	0
1	1	0.25	0	0	0	1

Table 1: Approximate truth tables for each neuron class, and their linear combination.

Taking the right linear combination of the 4 truth tables, we can recreate the AND truth table. This linear combination is a close match for the values seen in section 5.2.

Taking a linear combination will always be possible if the 4 classes are linearly independent, which is true for quite a wide range of choices for the key parameters[5].

As there are many neurons in each class, the readout matrix can average over them, reducing the noise to reasonable levels. This is similar to the proofs on noise bounds in Hänni et al. [2024], discussed further in section 6.2.

Finally, we note that the same argument and weights matrix can be adapted for other tasks. A different choice linear combination in the readout matrix can supply any other truth table instead (appendix A.2). And multiple inputs can be considered: with 3 inputs there are 8 possible neuron classes, to form in linear combination the 8 values of a 3-way truth table[6].

6.2 Circuit Efficiency

Why is this dense Binary Weighted Circuit learned in preference to the CiS Construction described in Hänni et al. [2024]? We present an approximating argument that it produces lower loss values asymptotically.

We do this by computing the variance of model output over randomly sampled sparse input. In other words, we want $\text{Var}(z_j)$. Without loss of generality, we choose $j = 1m + 2$, i.e. the output responsible for computing $v_1 \wedge v_2$.

6.2.1 The variance of a single neuron y_i

As before, that lets us write y_i, the neuron activations, with the contribution from $v_3, \cdots, v_m$ is folded into an interference term X_i:

[5] In particular, if we re-add X_i, it tends to pull the classes' truth tables towards co-linearity. So you can still find a linear combination, but the coefficients grow as X_i gets noisier.

[6] Error grows swiftly with number of inputs, so more complex circuits in models no doubt rely on multiple layers, discussed in Hänni et al. [2024].

Class A	$y_i = \text{ReLU}(uv_1 + uv_2 + X_i + b)$	occurs with proportion p^2
Class B1	$y_i = \text{ReLU}(uv_1 + lv_2 + X_i + b)$	occurs with proportion $p(1-p)$
Class B2	$y_i = \text{ReLU}(lv_1 + uv_2 + X_i + b)$	occurs with proportion $p(1-p)$
Class C	$y_i = \text{ReLU}(lv_1 + lv_2 + X_i + b)$	occurs with proportion $(1-p)^2$

X_i is defined as $\sum_{k=3}^{m} W_{ik}v_k$. As v is exactly s-sparse, we know between s values of v_k will be 1, or rarely $s-2$ or $s-1$ values[7]. Each of these entries will contribute u or l depending on W_{ik}. So, asymptotically, we can model X_i as independent and binomially distributed.

$$X_i \sim (u-l)\,\text{Binom}(s,p) + sl \qquad \text{with} \qquad \text{Var}(X_i) = (u-l)^2 sp(1-p)$$

With some further work, we can justify

$$\text{Var}(y_i) = \text{Var}\left(\text{ReLU}(X_i + \mathcal{O}(1))\right) = \mathcal{O}((u-l)^2 sp(1-p))$$

Now we can use this estimate to compute the variance of the model output for the CiS Construction and the Binary Weighted Circuit.

6.2.2 The variance of output z_j in the CiS construction

In the CiS Construction, $u = 1$, $v = 0$. It sets up the readout matrix to compute z_j as the mean of all y_i in class A. There are approximately dp^2 such neurons.

$$\text{Var}_{\text{CiS}}(z_i) = \text{Var}\left(\frac{\Sigma_{\text{class A}} y_i}{dp^2}\right) = \beta sp(1-p)/dp^2$$

The construction sets $p = \mathcal{O}(log^2 m/\sqrt{d})$, so

$$\text{Var}_{\text{CiS}}(z_i) = \mathcal{O}(s/\sqrt{d}/\log^2 m)$$

6.2.3 The variance of output z_j in the Binary Weighted Circuit

Meanwhile in the Binary Weighted Circuit, we pick readout weights based on the inverse of the 4×4 truth table matrix. It can be shown that the total readout weight for each neuron class class is $\mathcal{O}(\sqrt{s}/(u-l))$. [8].

So z_j is the sum of dp^2 class A neurons, each with readout weight $\mathcal{O}(\sqrt{s}/(u-l)dp^2)$, plus similar for classes B1, B2 and C. Taking $p = O(1)$ and applying the earlier formula for per-neuron variance vives

$$\text{Var}_{\text{Binary}}(z_i) = \mathcal{O}(s^2/d)$$

So the Binary Weighted Circuit has superior efficiency when s grows slower than $\sqrt{d}/\log^2 m$.

This result matches intuition. Increasing p makes the circuitry more dense, i.e. the model is making use more neurons for each calculation. This increases the variance of individual neurons but gives you many more to average over. s determines that per-neuron noise, so determines the trade-off. The CiS Construction merely aimed to minimize the interference term of a single neuron, as this was critical for establishing provable error bounds.

[7] Depending on the value of v_1 and v_2. With more rigor, we could condition on these values, compute variance for each, then combine.

[8] This comes from X_i having variance proportional to s, but the neuron classes only differ from each other by a constant translation of at most $2(u-l)$. As s increases, the classes be ReLU zero-points are at increasingly similar points on the probability distribution.

Aside from accuracy considerations, the existence of weight decay also encourages dense circuits. This is because weight decay penalises a strong weight on a single neuron more than an equivalent collection of weaker weights on several neurons.

7 Limitations

This paper attempts to build on previous theoretical understanding in a more realistic trained setting, but toy models still fall short of real-world models. In particular, our use of monosemantic inputs/outputs and choice of the Universal-AND problem are deliberate simplifications of superposition, and complex circuits found in practice.

The experimental results show that the Binary Weighted Circuit is used at reasonable values of sparsity (fig. 9), but we have not performed a full sweep to fully characterize this. Nor have we explained why the particular values of u, v, p observed are used. Adler and Shavit [2024] gives much tighter and general bounds on what is possible, so we have focused on the mechanics of the circuitry.

Section 6 relies on several approximations that are not rigorously proved. Future theoretical or empirical work would be needed to gain confidence in these claims.

8 Conclusion

We have established the dense Binary Weighted Circuit that solves the Universal-AND problem. We analytically describe the fundamental behaviour of the circuit and approximate its error rate. This represents a useful formulation for understanding compressed computation - previous works either described theoretical sparse circuits that are not learned in practice, or do not give an analytic description of the circuit. Braun et al. [2025] poses the question whether compressed computation and computation in superposition are "subtly distinct phenomena"; this paper answers yes, by finding compressed computation that arises even with monosemantic input/output.

Dense circuits like these challenge the common assumption that circuits can be found by finding a sparse subset of connections inside a larger model, and give an additional explanation why features are rarely monosemantic. If it is better to have many shared noisy calculations than a smaller set of isolated, reliable ones, then a different set of techniques is needed to detect them. It is an important principle to be aware of while conducting interpretability work, or designing new network architectures.

Given the novel structure of this circuit combined with its simplicity, we believe this problem and circuit can act as a good testbed for circuit-based interpretability tooling.

References

Micah Adler and Nir Shavit. On the complexity of neural computation in superposition. *CoRR*, abs/2409.15318, 2024. URL https://doi.org/10.48550/arXiv.2409.15318.

Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, Brian Chen, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/methods.html.

Anonymous. Accompanying code for this paper. URL https://anonymous.4open.science/r/uand-toy-model-80AD/uand.ipynb.

Dan Braun, Lucius Bushnaq, Stefan Heimersheim, Jake Mendel, and Lee Sharkey. Interpretability in parameter space: Minimizing mechanistic description length with attribution-based parameter decomposition, 2025. URL https://arxiv.org/abs/2501.14926.

Lucius Bushnaq and Jake Mendel. Circuits in superposition: Compressing many small neural networks into one. https://www.lesswrong.com/posts/roE7SHjFWEoMcGZKd/circuits-in-superposition-compressing-many-small-neural, 2024. Accessed: 2025-05-09.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

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.

Kaarel Hänni, Jake Mendel, Dmitry Vaintrob, and Lawrence Chan. Mathematical models of computation in superposition. In *ICML 2024 Workshop on Mechanistic Interpretability*, 2024. URL https://openreview.net/forum?id=OcVJP8kClR.

Sam Marks. What's up with llms representing xors of arbitrary features? https://www.lesswrong.com/posts/hjJXCn9GsskysDceS/what-s-up-with-llms-representing-xors-of-arbitrary-features, 2024. Accessed: 2025-05-09.

Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. In *Causal Representation Learning Workshop at NeurIPS 2023*, 2023. URL https://openreview.net/forum?id=T0PoOJg8cK.

Adam Scherlis, Kshitij Sachan, Adam S Jermyn, Joe Benton, and Buck Shlegeris. Polysemanticity and capacity in neural networks. *arXiv preprint arXiv:2210.01892*, 2022.

A Other Considerations

We include brief notes on how our result interacts with related discussions regarding Computation In Superposition.

A.1 Is this Just Feature Superposition?

In a sense, the circuit found here bears a lot of resemblance to simply randomly embedding the m inputs in d dimensional space. In both cases, you get a mix of neurons with different response patterns, and you can approximate the AND operation by taking a dense linear combination of the neurons.

Indeed, training the toy model with the first layer frozen to random values still results in a similar pattern of readouts (fig. 7). Hänni et al. [2024] includes a similar observation in Section 3.3.

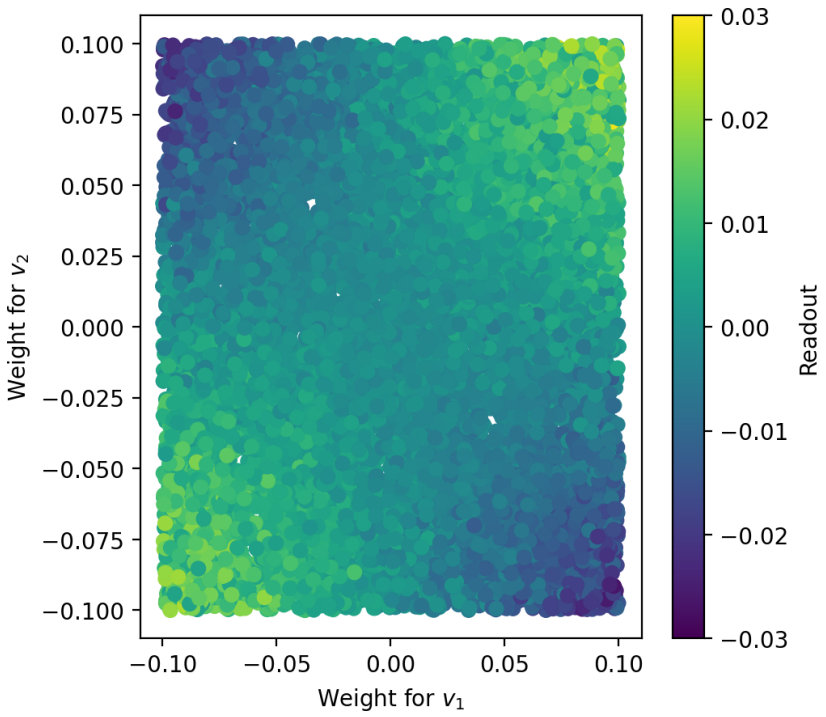


Figure 7: Readout weight by input weights (uniform initialization, $d = 10000$)

The circuit described in this paper uses binary weights, rather than some other random distribution. But we expect such a pure distribution is unlikely to replicate outside of toy models. Using binary weights simplifies analysis and has an improved loss, but only improves loss by a constant factor (fig. 8).

Thus a key takeaway from the paper could be: For a set of sparse binary features stored in superposition with sufficiently random directions, it is possible to linearly readout any Boolean circuit, subject to a certain amount of error. This sort of dense circuitry is easily learnt and is robust to the exact distribution, so we should expect it to be a common mechanism in real world models.

A.2 XOR Circuits

Marks [2024] observes that readout directions corresponding to XOR of input features commonly occur in models. Marks argues that these will cause linear probes to fail to generalize. I.e. if a probe trained to predict v_1 only on data where $v_2 = 0$, then it will be just as likely to identify $v_1 \oplus v_2$ as it is v_1. These give opposite answers when out of distribution $v_2 = 1$, breaking the probe.

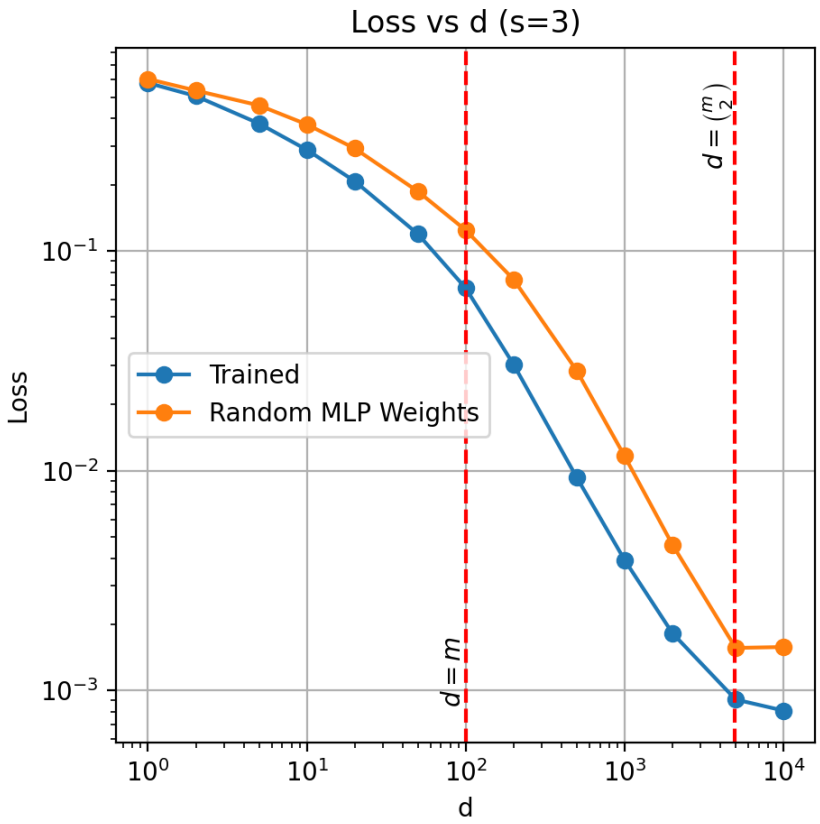


Figure 8

Our result gives an explanation for why XOR circuits may be readaoutable, but probes still have some generalizability. Using the procedure described in section 6.1, for any boolean circuit of v_1, v_2, we can get the readout weights of neuron classes that approximate it. Let's apply that for v_1 and $v_1 \oplus v_2$ under the same numeric values of u, l and b used for table 1.

$$v_1 \sim 4A - 4B1 + \frac{8}{3}B2 - \frac{8}{3}C$$

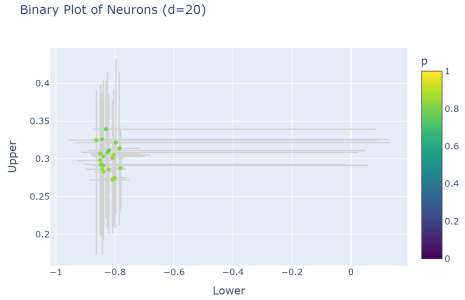
$$v_1 \oplus v_2 \sim \frac{20}{3}B1 + \frac{20}{3}B2 - \frac{40}{3}C$$

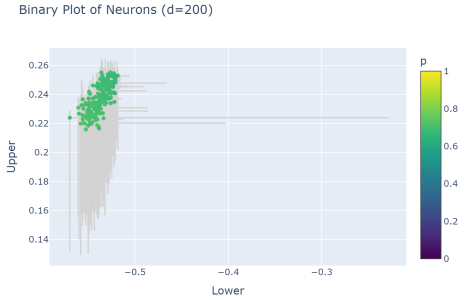
While both target functions are linearly decodable, the XOR direction requires significantly larger weights. As a result, under typical regularization schemes[9] the v_1 direction is likely to be favoured.

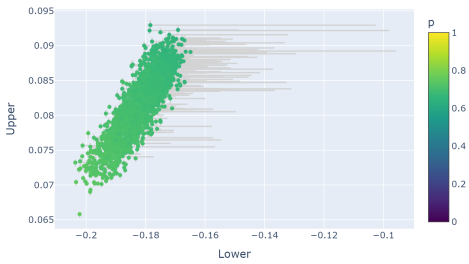
B Binary Weight Charts

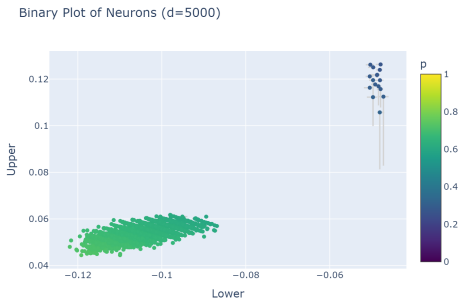
We supply binary weight charts for a range of values of d. Recall that $m = 100, s = 3$. Error bars show 90th-percentile, so indicate the extent to which the individual weights associated with a neuron do not perfectly match u / l.

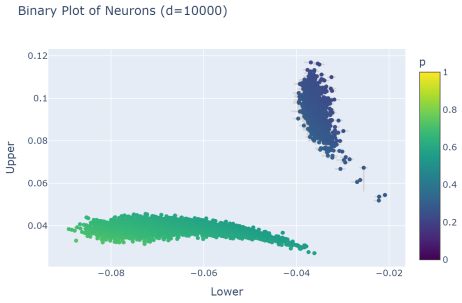
[9]For some choices of p when using L_1 norm relative preference may reverse. Nonetheless, the general principle holds: one of the two linearly accessible directions will be favoured.


Binary Plot of Neurons (d=20)
Upper
Lower
p


Binary Plot of Neurons (d=200)
Upper
Lower
p


Upper
Lower
p


Binary Plot of Neurons (d=5000)
Upper
Lower
p


Binary Plot of Neurons (d=10000)
Upper
Lower
p

In fig. 9 we measure the similarity of the entire weights matrix to a binary distribution, using

$$\text{score} = 1 - \underset{i,j}{\text{mean}} \frac{2\min(|W_{ij} - u_i|, |W_{ij} - l_i|)}{u_i - l_i} \quad \text{for } u_i = \max_j W_{ij}, l_i = \min_j W_{ij}$$

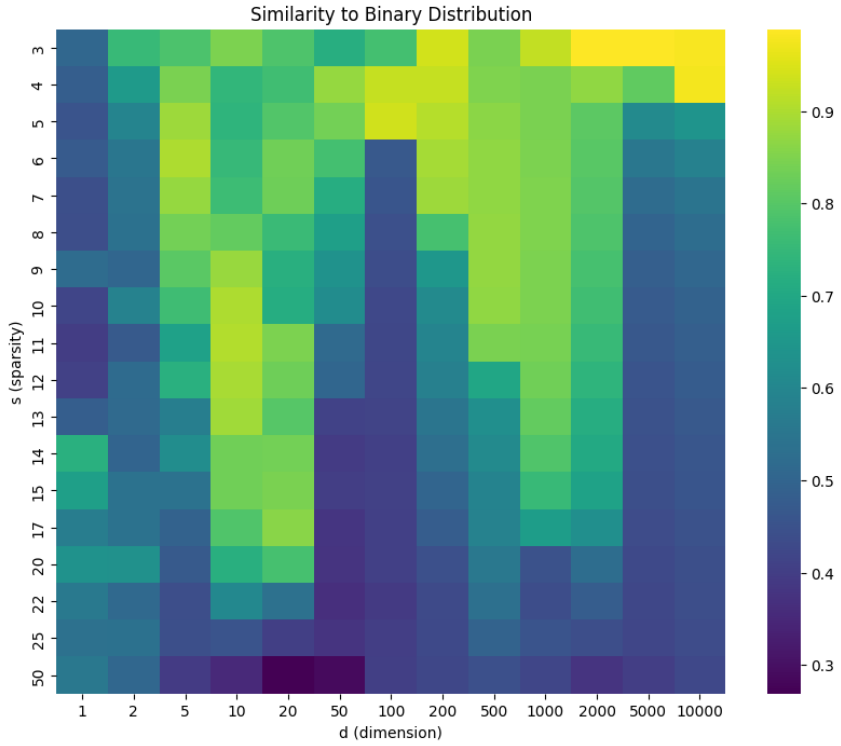


Figure 9: Similarity of learned weight matrix to a binary distribution for various d, s

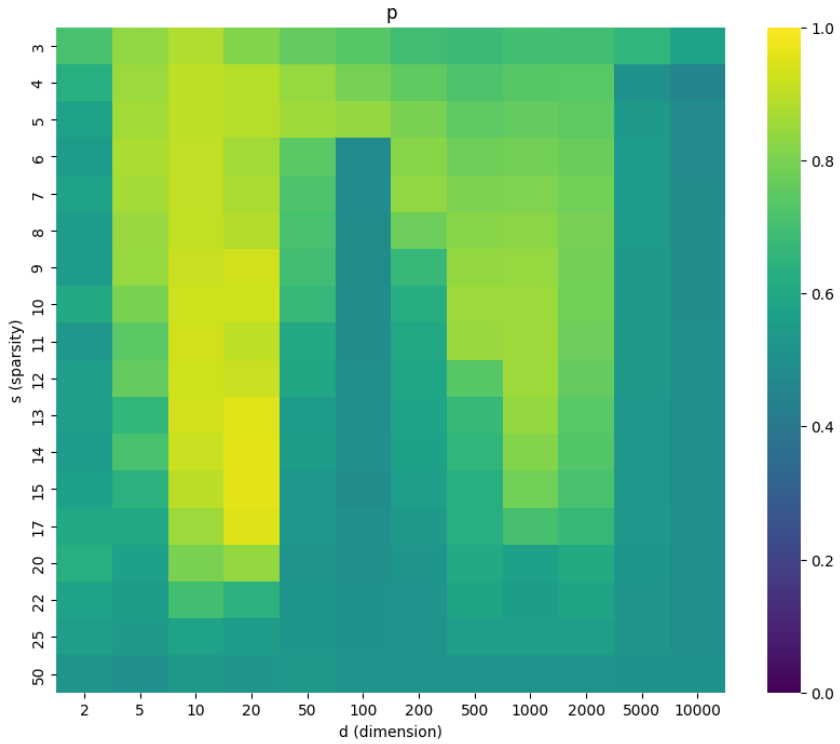
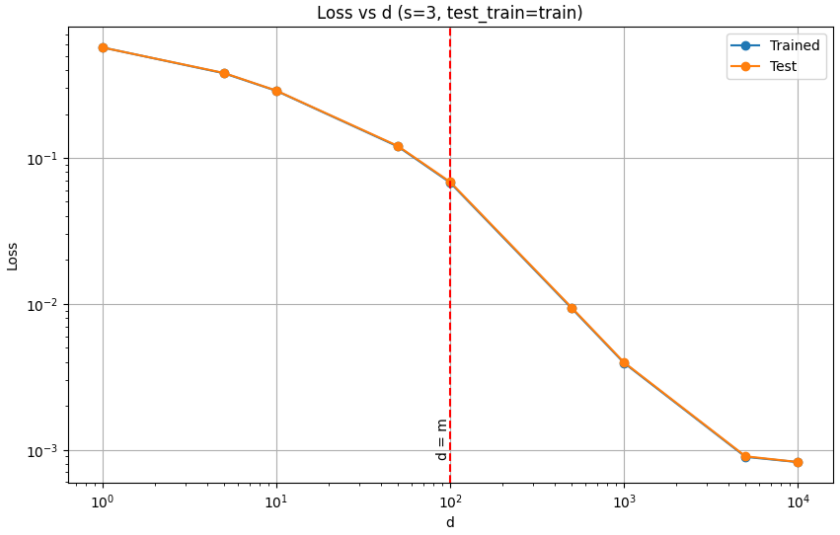


Figure 10: p, or the proportion of W_{ij} entries above $\underset{j}{\text{mean}} W_{ij}$

C Other experiments

C.1 Train/test

We ran some models with 10% of the space of generated data held out to ensure that the learned circuit was capable of generalization. The change in loss was extremely small (appendix C.1) and there were no qualitative changes to note.



C.2 Varying m

As R has m^2 rows, it is not feasible to scale up this toy model to significant values of m. This is a natural consequence of avoiding activation superposition in inputs/outputs. We did vary the value of m to establish that the general patterns replicate at values of m between 50 and 200 (fig. 11, fig. 12).

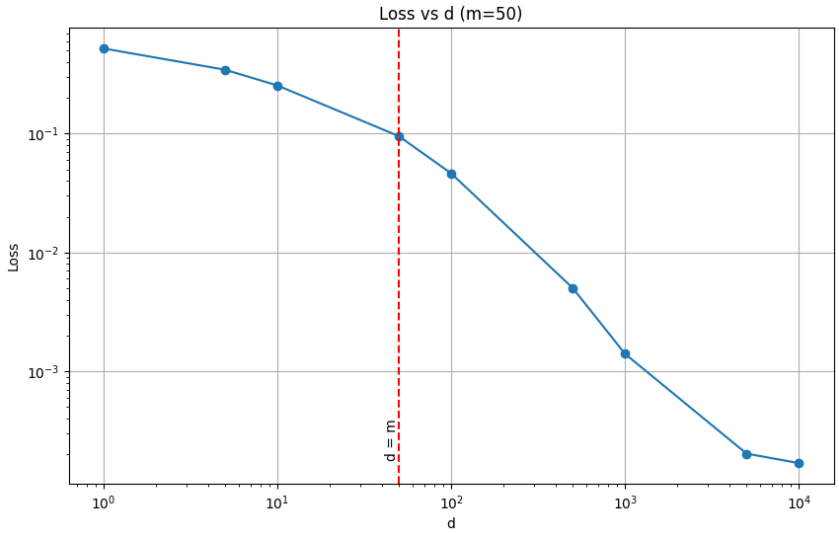


Figure 11

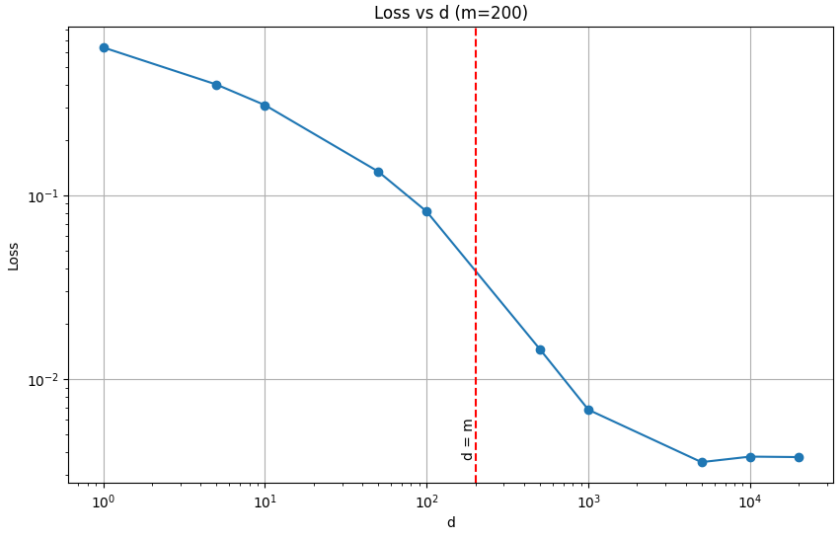


Figure 12

C.3 Weight Decay

Weight decay generally encourages training to use the most efficient circuits. This can be convenient for circuit analysis as it simplifies things without doing long training runs. It also keeps the norm of model parameters in a constrained reasonable range which makes reading graphs easier. Without it, the model could double W and b, halve R and c and have the exact same loss.

I ran the same training without weight decay and got similar, but messier results (fig. 13). I also ran the training for four times longer to check convergence, and found that gave similar clustering to the results with weight decay (fig. 14).

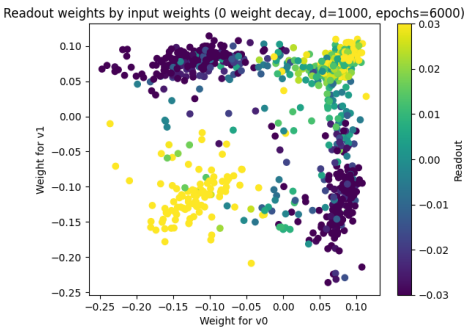


Figure 13

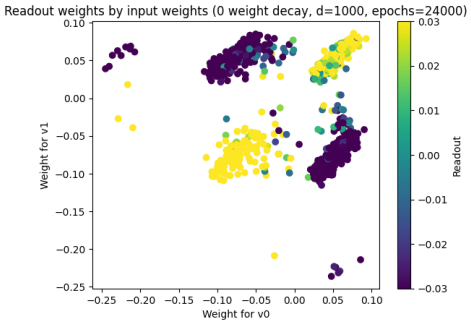


Figure 14