Sparse Representations in Artificial and Biological Neural Networks

A dissertation presented

by

Trenton Bricken

to

The Department of Systems Biology

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in the subject of

Systems, Synthetic, and Quantitative Biology

Harvard University

Cambridge, Massachusetts

March 2025

Dissertation Advisor: Professor Gabriel Kreiman Trenton Bricken

Sparse Representations in Artificial and Biological Neural Networks

Abstract

This thesis explores how sparsity, the idea that only a small fraction of neurons are active at any time, is a common thread connecting biological brains and artificial intelligence. By combining theory, experiments, and real-world applications, we show how sparsity is a key ingredient underlying core cognitive abilities like attention, memory, and learning.

We start by uncovering a surprising link between the "attention" mechanism powering recent artificial intelligence (AI) breakthroughs and a classic theory of human memory called Sparse Distributed Memory (SDM). This suggests that brains and AI may leverage similar computational tricks.

Taking inspiration from the brain's cerebellum, we then use SDM to improve an AI's ability to learn continuously without forgetting previous knowledge. This showcases sparsity's ability to enable more flexible learning.

We also find that simply adding noise during training pushes AI to use sparse representations, causing it to develop more brain-like properties. This provides clues about why sparsity emerges in the brain while offering an easy way to encourage it in AI.

Finally, we use sparsity to peek inside the black box of large language models like ChatGPT and Claude. By pulling apart the tangled web of information these models use to think, we make progress towards more transparent and controllable AI.

Together, these findings paint sparsity as a unifying principle for intelligent systems, be they made of biological neurons or silicon chips. By connecting the dots between neuroscience and AI, this thesis advances our understanding of intelligence while charting a course towards more capable and interpretable AI systems.

Contents

List of Tables

List of Figures

Acknowledgments

There are too many people to thank and acknowledge. It really takes a village. There is that saying "you are the average of the five people you spend the most time with," but I don't think, at least in my case, it should stop at five. It is too easy for me to trace the source of any accomplishments back to the influence of others. I truly have continued to be in the right place at the right time.

Thanks first to Maxwell Kanter, Kayla Colvin, Jason Wang, Brett Mathias, Rowly Graham and Joshua Rollins for reading and sharing feedback on the introduction to this thesis. If it is at all readable, it is thanks to you.

Thanks to Joe Choo-Choy, Miles Turpin, and Max Farrens, you are the most fantastic friends anyone could hope for. Your integrity, thoughtfulness, and kindness have made me a better person. Your curiosity and encouragement have kept me doing research and having fun with it. Joe, I am so lucky to have been in the same boarding house with you for high school and then share another three years at the same university. It takes two to make a tribe and you were my tribe in high school, enabling me to be a weirdo and pursue idiosyncratic interests. Reading Nick Bostrom's Superintelligence with you senior year was particularly formative and first sparked my interest in neuroscience, AI, and the foundations of intelligence. Miles, you immediately joined the tribe in undergrad and have been part of it ever since. You made it cool to wear ten dollar dark orange construction glasses that block blue light before bed and first explained gradient descent to me during a Sunday morning breakfast the first semester of our freshman year. Max, your influences started far earlier when we were five. A motivation for me has always been wanting you to think I'm cool. Your enthusiasm for every one of my pursuits is highly contagious, whether it was making bad artwork for one of your albums or designing a financially ruinous crypto art NFT. Joe, Miles, and Max, have all been there for me during the highest highs and the lowest lows. Thank you for making life so meaningful.

Thanks to Nathan Rollins for being an incredible academic mentor in addition to a friend. Your supervision when I was a bright-eyed intern in Dr. Debora Marks's lab at Harvard

Medical School taught me how exciting science can be. Your meteoric career trajectory has made me more ambitious. Your advice on my PhD applications is the reason I had my choice of graduate schools and was able to find the best fit to thrive.

To Dr. Debora Marks, answering my cold email to intern in your lab my Junior summer was the launchpad for my scientific career. Until that point I was starting to believe that science was for stale, esoteric, boring questions. By the end of that summer, you had given me the opportunity to work: on a DARPA project to modify extremophile proteins to work in humans; an IARPA project to rapidly identify immune system evasion genes in viruses, and a side hustle to grow giant insects! I hope I have learned some fraction of your ability to pursue ambitious research and always ask the right questions.

To Dr. Robert Thompson, who enabled me to first cultivate an interest, and ultimately create my own undergraduate major, in "Biological and Artificial Intelligence". You let a random Sophomore meet with you weekly to discuss papers on human intelligence, brain development, and nutrition as one of my four semesterly courses. Having the time and support to pursue this reasearch was pivotal to me creating my major which you then sponsored and supervised for the remainder of my undergraduate degree. Thank you for giving me the nutrients to grow.

To Dr. David Banks, for teaching me statistics and deep learning. Agreeing to an independent study where we read Ian Goodfellow et al.'s Deep Learning textbook was prescient. I feel lucky to have witnessed your humility, intelligence, and curiosity to learning all there is about the world over many coffee chats across many years. Thank you for continuing to arrange such conversations remotely and remain in touch.

To Julian Robertson, whose scholarship program in undergrad gave me the funding and tokens of prestige I would have otherwise wasted time seeking. This enabled me to instead pursue research and be funded at Harvard Medical School with Dr. Debora Marks. I likely would not have known about, or been qualified for my PhD research, without this experience.

To Dr. Andrew Murray. At the very start of our PhD program, you said, "I don't care

what you do, as long as you do the best beeping research you can." And I knew that this was the program for me. I hope it is okay that I took your advice so literally – I know this PhD has been unconventional – but I believe I have been able to do the best research I can and might not have without your vision.

To Dr. Cengiz Pehlevan, for getting me to dream big and execute on those dreams. I never thought my first paper, which started as a final project in your class my very first semester of graduate school, would ever become a paper at NeurIPS.

To Dr. Bruno Olshausen and Pentti Kanerva, for being such wise and enthusiastic collaborators. You both made it possible for me to be a visiting researcher at Berkeley's Redwood Theoretical Neuroscience Institute which was a highlight of my PhD. It has been an honor to take your research and build upon it in minor ways.

To my committee, Dr. Demba Ba, Dr. Sam Gershman, and Dr. Sean Eddy, I greatly admire your research and it is a privlege to have you evaluate me as a PhD candidate. Your guidance across multiple pre-qualifying exams and other check-ins has made my work more relevant and rigorous.

To Dr. Gabriel Kreiman, without whom this PhD never would have happened – many times over! Publishing "Attention Approximates Sparse Distributed Memory" unlocked multiple exciting research avenues that I wanted to chase down but I didn't have a home for. I was honestly considering dropping out to find this home at another university. Instead, you gave me the home and every piece of encouragement, advice, and support to pursue my research ambitions. During our first meeting you told me to be as ambitious as possible and take lots of risks, to really go for it, and that you would support me. And you really did, every single step of the way. Even when what I wanted to do wasn't what was best for the lab, like turning down the NSF Fellowship so I could pursue a residency at Anthropic. I am really indebted to your relentless support of me.

To my extended family, Akin Blitz, Buck Dominick, John Mathias, TJ Sabo, Michael Todd, Brett Mathias, you have all influenced my life's trajectory in more ways than you know and for the better.

To Grace Bricken, because you have always held over me that you are seven minutes older, you are the only person I now ask to refer to me as "Doctor" in the future! I am lucky to have experienced so many of life's moments with you there.

To Alexander Bricken, you are my best friend and travel companion through so many of life's adventures. Thank you for grounding me, encouraging me, and making life so much fun.

And to my parents. Thank you for providing me with every resource under the sun. Whether it was feeding me and Alexander chicken tenders while we played World of Warcraft non-stop to reach the max level, or driving home and back to school again because I forgot to wear my shoes, I have always had the safety net to be ambitious and take risks. You have both sacrified immensely to give me the opportunities that most could never dream of, opportunities that you yourselves never had, and that you would have made so much of if you were in my shoes. Thank you for everything.

(Taken and adapted from the dedication page of John Steinbeck's East of Eden)

Dear Dan and Kathryn,

You came upon me carving some kind of little figure out of wood and you said, "Why don't you make something for me?" I asked you what you wanted, and you said, "A box."

"What for?"

"To put things in."

"What things?"

"Whatever you have," you said.

Well, here's your box. Nearly everything I have is in it, and it is not full.

Pain and excitement are in it, and feeling good or bad and evil thoughts and good thoughts – the pleasure of design and some despair and the indescribable joy of creation.

And on top of these are all the gratitude and love I have for you.

And still the box is not full.

Introduction

Human intelligence is a key reason you are currently reading this thesis instead of running away from a lion on the African Savannah. Our extraordinary brains have elevated humanity from a precarious existence as hunter-gatherers to the dominant species capable of reshaping the planet.

Yet for all its power, human intelligence remains mysterious. We understand the broad strokes: our brains contain billions of neurons connected by trillions of synapses, forming networks that somehow give rise to cognition. But beyond this general framework, the specifics of intelligence remain elusive. We don't fully understand how memories form, how concepts are represented, or how reasoning unfolds at a mechanistic level. Moreover, it is unclear what level of abstraction, from individual neurons to broader circuits and systems, is key for understanding intelligence. The brain is arguably the most complex system we've ever attempted to reverse engineer, and we're still in the early stages of that journey.

Artificial Intelligence (AI) presents a similar puzzle. Despite being capable of remarkable feats - from defeating the world's best Go players to generating text indistinguishable from human writing - we don't have a comprehensive understanding of how AI works. Even with full access to the AI's architecture, weights, and activations, tracing the path from input to output in a way that meaningfully explains the system's reasoning remains challenging. This is because, while we have created it, state-of-the-art AIs are "grown", not built.

Uncertainty on both fronts contributes to ongoing debates about whether artificial neural networks and biological brains employ fundamentally similar or different mechanisms. Some researchers argue that, just as birds and planes work in very different ways, so too do minds and machines. The argument goes that silicon hardware is very different from neuronal wetware and the evolutionary process that created our brains will look nothing like that which creates AI. Put another way, we are "teaching sand to think" and there is no reason it should think like us.

While these arguments are compelling, this thesis explores the other side of the debate, investigating where today's artificial intelligence already has, or would benefit from, convergence with algorithms used by the brain. Although birds and airplanes may not fly exactly the same way, when you look beyond their raw materials, they have a lot in common: both share similar aerodynamic profiles, use wings and control surfaces like tail rudders, and leverage the same flight mechanics of lift, thrust, and drag.

Similarly, any intelligence will need to perform the same core informational processing such as extracting meaningful signal from vast quantities of noisy input data; remembering the information that will be useful; storing it in a way that it can be easily retrieved when relevant; and using it to best achieve future goals.

Beyond sharing core informational processes, AI is being trained to perform human tasks and is powered by ideas from our own biology and psychology. One of the recent breakthrough algorithms in AI, that this thesis connects to a key circuit in biological brains, is literally called "Attention". "Attention" was inspired by the human ability to pay attention to a small number of things while ignoring distractions, for example, by focusing on a conversation inside a busy restaurant.

Finally, as much as we researchers pride ourselves on the ability to theorize and plan ahead, AI research follows the scientific tradition of trial-and-error breaking the trail of discovery, with theoretical understanding playing catch-up. This makes our search for the recipe behind intelligence look more like the process of evolution than we researchers may otherwise care to admit.

Regardless of philosophically exploring the reasons *why* artificial intelligence may to converge on a similar design to the human brain, this thesis presents four examples of such convergence occurring.

Taken together, these chapters have a unifying theme around sparse representations. "Sparsity", defined as "something thinly dispersed or scattered", refers here to sparse neuron activity, where only a small fraction of the total number of neurons are active at any time. "Representations", refers to how data is encoded. For example, using only 10 neurons out of 10,000 in your brain to encode an image of a lion would be an example of a sparse representation.

We begin in Chapter 1 by showing how a theoretical model of human memory is not only the most common circuit in the brain but also a close approximation to the breakthrough "Attention" algorithm powering state-of-the-art AI models. This memory model, Sparse Distributed Memory (SDM) has a compelling biological mapping to the cerebellum, a brain region which contains $\sim$80% of our neurons and exists in most organisms including fruit flies (Essen *et al.*, 2018). We show in particular that the way SDM chooses to write and read memories closely approximates the approach taken by Attention, suggesting that these seemingly distinct approaches converge on similar computational principles. This link provides a fresh perspective on why the Attention mechanism works so effectively in modern AI systems while simultaneously offering a computational interpretation of cerebellar function.

Chapter 2 shows that we can use additional structures from our own cerebellum to solve memory problems with artificial networks. Many AI models will forget existing knowledge when they are taught something new because they overwrite the previous information. Sparse Distributed Memory, by introducing sparsity, allows the AI to develop on its own specific memory modules that are overwritten less often. These modifications are inspired by inhibitory interneurons in the cerebellum not accounted for in the original SDM model. By systematically testing the contribution of each biologically-inspired component, we show that this approach offers a compelling alternative to existing continual learning techniques, which often rely on hand labelling different types of information.

Chapter 3 shows how simulating a more human-like learning environment with noisy inputs causes artificial neural networks to look more brain-like. Specifically, adding random noise to training data causes artificial neural networks to become sparse. The presence of noise and resulting sparsity cause the artificial network to more closely resemble a biological brain. Furthermore, when trained on image data, the network spontaneously develops receptive fields that look a lot like those found in biological visual systems. We mathematically derive three implicit loss terms introduced by this noisy training to explain these emergent properties.

Chapter 4 uses ideas about how the human brain encodes and decodes information to make AI models more transparent. There is a major open challenge to reverse-engineer the brains of AI language models like ChatGPT and Claude. Why did the model refuse a completely harmless request? Is it telling users only what they want to hear or what it actually believes? Can the AI be trusted? These important questions are difficult to answer because these AI models represent concepts using incredibly dense, intertwined representations. Any component of the model is involved in a hundred different behaviors and cannot be isolated to work out what part it is playing. We call this dense intertwining of representations "superposition" and an analogy is a photograph that has been exposed multiple times: imagine placing multiple transparent sheets on top of each other, each one printed with a different pattern or image. As you stack more layers, the original images visually blend together, making it harder to focus on any individual component. The stack of transparencies ends up holding compressed information from all the original images superimposed together.

To tackle this problem, we introduce "sparse autoencoders" as a method for "undoing" this superposition, pulling apart each of the transparent sheets so they separately display their own image. The end result is a collection of tens of thousands of neatly separated "features", each corresponding to a meaningful concept that the network has learned. This technique is providing novel insights into how AI models organize information and is a stepping stone towards making them more transparent and controllable. There are two connections to biological brains with this work. First, the way we decode superposition builds directly on work around sparse coding from reseach around how the brain represents images. Second, the conditions that make superposition a powerful compression strategy for AI models are likely to also exist in the brain and have been discussed for a long time under the term "population coding". As a result, it is likely that our decoding technique will also work for brain data with new collaborations underway.

The study of sparse representations in neural networks has a rich history spanning neuroscience, machine learning, and theoretical computer science. In the brain, sparsity of neural activity has been a well-observed phenomenon (Purves *et al.*, 2001) and has been proposed as a fundamental principle underlying sensory processing, particularly in the visual cortex (Olshausen and Field, 1997a, 2004). In artificial neural networks, sparsity has been explored through various approaches including sparse coding and its approximations like sparse autoencoders (Olshausen and Field, 1997a; Makhzani and Frey, 2014a). These methods aim to learn compact, interpretable representations that can improve computational efficiency, generalization, and robustness (Ahmad and Scheinkman, 2019; Kurtz *et al.*, 2020; Yang *et al.*, 2020; Aljundi *et al.*, 2019b; Abbasi *et al.*, 2022; Smith *et al.*, 2022). Insights from neuroscience have inspired several advances in artificial intelligence. For example, the development of convolutional neural networks drew inspiration from the hierarchical organization of the visual cortex (LeCun *et al.*, 1989; Krizhevsky, 2009), meanwhile Dropout was inspired by the stochasticity of neural activity (Srivastava *et al.*, 2014).

The bidirectional flow of ideas - from biology to AI and back again - can enrich our understanding of both. Computational models inspired by biology can be tested and refined in ways that would be impractical in biological systems, potentially generating new hypotheses about neural function. Conversely, the surprising emergent properties of artificial systems can suggest new ways of thinking about biological processes.

In the chapters that follow, we will explore this interplay between biological inspiration and artificial implementation, with a particular focus on how sparse representations emerge, function, and can be manipulated in neural systems. Through this exploration, we hope to contribute to our understanding of intelligence in both its natural and artificial forms, and to advance the development of AI systems that can learn continuously, represent information transparently, and ultimately serve human needs more effectively.

Chapter 1

Attention Approximates Sparse Distributed Memory[1]

While Attention has come to be an important mechanism in deep learning, there remains limited intuition for why it works so well. Here, we show that Transformer Attention can be closely related under certain data conditions to Kanerva's Sparse Distributed Memory (SDM), a biologically plausible associative memory model. We confirm that these conditions are satisfied in pre-trained GPT2 Transformer models. We discuss the implications of the Attention-SDM map and provide new computational and biological interpretations of Attention.

This work was published at the Conference on Neural Information Processing Systems (NeurIPS) 2021.

Author Contributions

- Trenton Bricken conceived of the project and theory, conducted the experiments, and wrote the paper with assistance.
- Cengiz Pehlevan provided key commentary and guidance on the research and writing

[1]Co-authored with Dr. Cengiz Pehlevan

of the paper.

1.1 Introduction

Used most notably in the Transformer, Attention has helped deep learning to arguably approach human level performance in various tasks with larger models continuing to boost performance (Vaswani *et al.*, 2017; Bahdanau *et al.*, 2016; Radford *et al.*, 2019; Brown *et al.*, 2020; Chen *et al.*, 2020; Dosovitskiy *et al.*, 2020; Wilson Yan and Srinivas, 2021; Ramesh *et al.*, 2021; Gwern, 2019). However, the heuristic motivations that produced Attention leave open the question of why it performs so well (Vaswani *et al.*, 2017; Rogers *et al.*, 2020). Insights into why Attention is so effective would not only make it more interpretable but also guide future improvements.

Much has been done to try and explain Attention's success, including work showing that Transformer representations map more closely to human brain recordings and inductive biases than other models (Schrimpf *et al.*, 2020; Tuli *et al.*, 2021). Our work takes another step in this direction by showing the potential relationship between Attention and biologically plausible neural processing at the level of neuronal wiring, providing a novel mechanistic perspective behind the Attention operation. This potential relationship is created by showing mathematically that Attention closely approximates Sparse Distributed Memory (SDM).

SDM is an associative memory model developed in 1988 to solve the "Best Match Problem", where we have a set of memories and want to quickly find the "best" match to any given query (Kanerva, 1988; Minsky and Papert, 1969). In the development of its solution, SDM respected fundamental biological constraints, such as Dale's law, that synapses are fixed to be either excitatory or inhibitory and cannot dynamically switch (see Section 1 for an SDM overview and (Kanerva, 1988) or (Kanerva, 1993) for a deeper review). Despite being developed independently of neuroanatomy, SDM's biologically plausible solution maps strikingly well onto the cerebellum (Kanerva, 1988; Kawato *et al.*, 2021).[2]

[2]This cerebellar relationship is additionally compelling by the fact that cerebellum-like neuroanatomy exists in many other organisms including numerous insects (eg. the Drosophila Mushroom Body) and potentially

Abstractly, the relationship between SDM and Attention exists because SDM's read operation uses intersections between high dimensional hyperspheres that approximate the exponential over sum of exponentials that is Attention's softmax function (Section 2). Establishing that Attention approximates SDM mathematically, we then test it in pre-trained GPT2 Transformer models (Radford *et al.*, 2019) (Section 3) and simulations (Appendix of (Bricken and Pehlevan, 2021) section SDM Experiments). We use the Query-Key Normalized Transformer variant (Henry *et al.*, 2020) to directly show that the relationship to SDM holds well. We then use original GPT2 models to help confirm this result and make it more general.

Using the SDM framework, we are able to go beyond Attention and interpret the Transformer architecture as a whole, providing deeper intuition (Section 4). Motivated by this mapping between Attention and SDM, we discuss how Attention can be implemented in the brain by summarizing SDM's relationship to the cerebellum (Section 5). In related work (Section 6), we link SDM to other memory models (Graves *et al.*, 2014; Wu *et al.*, 2018a), including how SDM is a generalization of Hopfield Networks and, in turn, how our results extend work relating Hopfield Networks to Attention (Ramsauer *et al.*, 2020; Krotov and Hopfield, 2016). Finally, we discuss limitations, and future research directions that could leverage our work (Section 7).

1.2 Review of Kanerva's SDM

Here, we present a short overview of SDM. A deeper review on the motivations behind SDM and the features that make it biologically plausible can be found in (Kanerva, 1988, 1993). SDM provides an algorithm for how memories (patterns) are stored in, and retrieved from, neurons in the brain. There are three primitives that all exist in the space of n dimensional binary vectors:

Patterns ($\mathbf{p}$) - have two components: the pattern address, $\mathbf{p}_a^\mu \in \{0,1\}^n$, is the vector

---

cephalopods (Modi *et al.*, 2020; Wolff and Strausfeld, 2016; Litwin-Kumar *et al.*, 2017; Shomrat *et al.*, 2011; Shigeno and Ragsdale, 2015).

representation of a memory; the pattern "pointer", $\mathbf{p}_p^\mu \in \{0,1\}^n$, is bound to the address and points to itself when autoassociative or to a different pattern address when heteroassociative. A heteroassociative example is memorizing the alphabet where the pattern address for the letter a points to pattern address b, b points to c etc. For tractability in analyzing SDM, we assume our pattern addresses and pointers are random. There are m patterns and they are indexed by the superscript $\mu \in \{1, \ldots, m\}$.

Neurons ($\mathbf{x}$) - in showing SDM's relationship to Attention it is sufficient to know there are r neurons with fixed addresses $\mathbf{x}_a^\tau \in \{0,1\}^n$ that store a set of all patterns written to them. Each neuron will sum over its set of patterns to create a superposition. This creates minimal noise interference between patterns because of the high dimensional nature of the vector space and enables all patterns to be stored in an n dimensional storage vector denoted $\mathbf{x}_v^\tau \in \mathbb{Z}_+^n$, constrained to the positive integers. Their biologically plausible features are outlined in (Kanerva, 1988, 1993). When we assume our patterns are random, we also assume our neuron addresses are randomly distributed. Of the 2^n possible vectors in our binary vector space, SDM is "sparse" because it assumes that $r \ll 2^n$ neurons exist in the space.

Query ($\boldsymbol{\xi}$) - is the input to SDM, denoted $\boldsymbol{\xi} \in \{0,1\}^n$. The goal in the Best Match Problem is to return the pattern pointer stored at the closest pattern address to the query. We will often care about the maximum noise corruption that can be applied to our query, while still having it read out the correct pattern. An autoassociative example is wanting to recognize familiar faces in poor lighting. Images of faces we have seen before are patterns stored in memory and our query is a noisy representation of one of the faces. We want SDM to return the noise-free version of the queried face, assuming it is stored in memory.

SDM uses the Hamming distance metric between any two vectors defined: $d(\mathbf{a}, \mathbf{b}) := \mathbb{1}_n^T |\mathbf{a} - \mathbf{b}|$. The all ones vector $\mathbb{1}_n$ is of n dimensions and $|\mathbf{a} - \mathbf{b}|$ takes the absolute value of the element-wise difference between the binary vectors. When it is clear what two vectors the Hamming distance is between, we will sometimes use the shorthand $d_v := d(\mathbf{a}, \mathbf{b})$.

The Hamming distance is crucial for determining how many neurons read and write operations are distributed across. The optimal Hamming distance for the read and write circles denoted d^*, depends upon the number and distribution of patterns in the vector space and what the memories are being used for (e.g. maximizing the number of memories that can be stored versus the memory system's robustness to query noise). We provide three useful reference d^* values, using equations outlined in Appendix section Optimal Hamming Distance (Bricken and Pehlevan, 2021). The Signal-to-Noise Ratio (SNR) optimal d^*_{SNR} maximizes the probability a noise-free query will return its target pattern (Kanerva, 1993). The memory capacity optimal d^*_{Mem} maximizes the number of memories that can be stored with a certain retrieval probability and also assumes a noise-free query. The critical distance d^*_{CD} maximizes, for a given number of patterns, the amount of noise that can be applied to a query such that it will converge to its correct pattern (Kanerva, 1993).

These d^*s are only approximate reference points for later comparisons to Transformer Attention, first and foremost because they assume random patterns to make their derivations tractable. In addition, Transformer Attention will not be optimizing for just one of these objectives, and likely interpolates between these optimal d^*s as it wants to have both a good critical distance to handle noisy queries and a reasonable memory capacity. These optimal d^* are a function of n, r and m. For the Transformer Attention setting (Vaswani *et al.*, 2017), where $n = 64$, $r = 2^n$ and $m \leq 1024$, $d^*_{\text{SNR}} = 11$, $d^*_{\text{Mem}} = 5$, $d^*_{\text{CD}} = 15$, as derived in Appendix section Optimal Hamming Distance of (Bricken and Pehlevan, 2021).

1.2.1 SDM Read Operation

For the connection to Attention we focus on the SDM read operation and briefly summarize the write operation: all patterns write their pointers $\mathbf{p}_p$ in a distributed fashion to all neuron addresses located within Hamming distance d. This means that each neuron will store a superposition of pattern pointers from those pattern addresses within d: $\mathbf{x}^\tau_v = \sum_{\{\mathbf{p}:d(\mathbf{p}^\mu_a, \mathbf{x}^\tau_a) \leq d, \forall \mu\}} \mathbf{p}_p$. Having stored patterns in a distributed fashion across nearby neurons, SDM's read operation retrieves stored pattern pointers from all neurons within distance d of the query and averages them. This average is effectively weighted because the same patterns

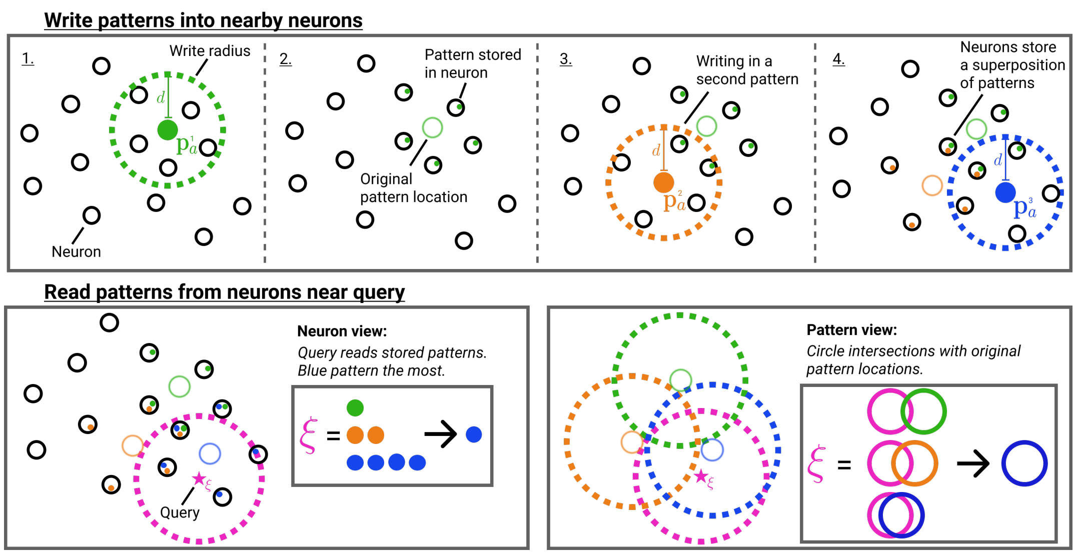


Figure 1.1: *Summarizing the SDM read and write operations.* ***Top Row*** *three patterns being written into nearby neurons. 1. The first write operation; 2. Patterns are stored inside nearby neurons and the original pattern location is shown; 3. Writing a second pattern; 4. Writing a third pattern and neurons storing a superposition of multiple patterns.* ***Bottom Row*** *shows two isomorphic perspectives of the read operation. Neuron view (left) shows the query reading from nearby neurons with the inset showing the number of times each pattern is read. The four blue patterns are a majority which would result in one step convergence. Pattern view (right) is crucial to relating SDM to Attention and defined in Eq. 1.1 below. We abstract away the neurons by assuming they are uniformly distributed through the space. This allows us to consider the circle intersection between the query and the original locations of each pattern where blue has the largest circle intersection.*

have distributed storage across multiple neurons being read from. The pattern weighting will be higher for those patterns with addresses nearer the query because they have written their pointers to more neurons the query reads from. Geometrically, this weighting of each pattern can be interpreted as the intersection of radius d circles[3] that are centered on the query and each pattern address $\mathbf{p}_a^{\mu}$ for all μ. A high level overview of the SDM read and write operations is shown in Fig. 1.1.

The 2^n possible neurons that have both stored this pattern's pointer $\mathbf{p}_p$ *and* been read by $\boldsymbol{\xi}$ is: $|O_n(\mathbf{p}_a, d) \cap O_n(\boldsymbol{\xi}, d)|$, where $|\cdot|$ is the cardinality operator and $O_n(\boldsymbol{\xi}, d) = \{\mathbf{x}_a \in \{0,1\}^n : d(\boldsymbol{\xi}, \mathbf{x}_a) \leq d\}$ is the set of all possible neuronal addresses $\mathbf{x}_a$ within radius d of $\boldsymbol{\xi}$. Mathematically, SDM's read operation sums over each pattern's pointer, weighted by its query circle intersection:

$$\boldsymbol{\xi}^{new} = g\left(\frac{\sum_{\mathbf{p}\in P} |O_n(\mathbf{p}_a, d) \cap O_n(\boldsymbol{\xi}, d)| \mathbf{p}_p}{\sum_{\mathbf{p}\in P} |O_n(\mathbf{p}_a, d) \cap O_n(\boldsymbol{\xi}, d)|}\right), \qquad g(e) = \begin{cases} 1, & \text{if } e > \frac{1}{2} \\ 0, & \text{else} \end{cases}, \tag{1.1}$$

and g acts elementwise on vectors. The denominator normalizes all of the weights so they sum to 1 in the numerator and enables computing if the element-wise majority value is a 0 or 1, using the function $g(\cdot)$. Intuitively, the query will converge to the nearest "best" pattern because it will have the largest circle intersection weighting. The output of the SDM read operation is written as updating the query $\boldsymbol{\xi} \rightarrow \boldsymbol{\xi}^{new}$ so that it can (but is not required to) apply the read operation iteratively if full convergence to its "best match" pattern is desired and was not achieved in one update.

The circle intersection (derived in Appendix section CircleIntersectionCalculationDerivation of (Bricken and Pehlevan, 2021)) is calculated as a function of the Hamming radius for the read and write operations d, the dimensionality n, and the vector distance between the

[3] In this binary space, the Hamming distance around a vector is in fact a hypercube but the vertices of an n dimensional unit cube lie on the surface of an n dimensional sphere with radius $\sqrt{n}/2$ and we refer to this as a circle because of our two dimensional diagrams. We adopt this useful analogy, taken from Kanerva's book on SDM (Kanerva, 1988), throughout the paper.

query and pattern: $d_v = d(\mathbf{p}_a, \boldsymbol{\xi})$, so we use the shorthand $\mathcal{I}(d_v, d, n)$:

$$\mathcal{I}(d_v, d, n) := |O_n(\mathbf{p}_a, d) \cap O_n(\boldsymbol{\xi}, d)| = \sum_{a=n-d-\lfloor \frac{d_v}{2} \rfloor}^{n-d_v} \sum_{c=\max(0,n-d-a)}^{d_v-(n-d-a)} \left(\binom{n-d_v}{a} \cdot \binom{d_v}{c} \right) \tag{1.2}$$

Eq. 1.2 sums over the number of possible binary vectors that can exist at every location inside the circle intersection. Taking inspiration from (Jaeckel, 1989a), this is a new and more interpretable derivation of the circle intersection than that originally developed (Kanerva, 1988). Eq. 1.2 is approximately exponential for the closest, most important patterns where $d(\mathbf{p}_a, \boldsymbol{\xi}) \leq 2d$, which is crucial to how SDM approximates Attention. This is shown for a representative instance of SDM in Fig. 1.2. The details of this approximation are provided in Appendix section Circle Intersection Approximates an Exponential of (Bricken and Pehlevan, 2021), but at a high level the binomial coefficients can be represented as binomial distributions and then approximated by normal distributions that contain exponentials. With the correctly chosen constants, c_1 and c_2, that are independent of the vector distance $d(\mathbf{p}_a, \boldsymbol{\xi})$, we can make the following approximation:

$$\mathcal{I}\big(d(\mathbf{p}_a, \boldsymbol{\xi}), d, n\big) \approx c_1 \exp\big(-c_2 \cdot d(\mathbf{p}_a, \boldsymbol{\xi})\big) \tag{1.3}$$

1.3 Attention Approximates SDM

To be able to handle a large number of patterns, we let the pattern address matrix with each pattern as a column be: $P_a = [\mathbf{p}_a^1, \mathbf{p}_a^2, ..., \mathbf{p}_a^m]$ with pointers $P_p = [\mathbf{p}_p^1, \mathbf{p}_p^2, ..., \mathbf{p}_p^m]$.

The Attention update rule (Vaswani *et al.*, 2017) using its original notation is:

$$\boldsymbol{\xi}^{new} = V\text{softmax}(\beta K^T Q) = (W_v Y)\text{softmax}\big(\beta (W_k Y)^T (W_q \mathbf{q})\big),$$

where K, V, and Q symbolize the "key", "value", and “query” matrices, respectively. $\mathbf{q}$ is a single query vector and Y represents the raw patterns to be stored in memory. The $\text{softmax}(\beta \mathbf{x}) = \exp(\beta \mathbf{x}) / \sum_{i=1}^{n} \exp(\beta x_i)$, where the exponential acts element-wise and At-

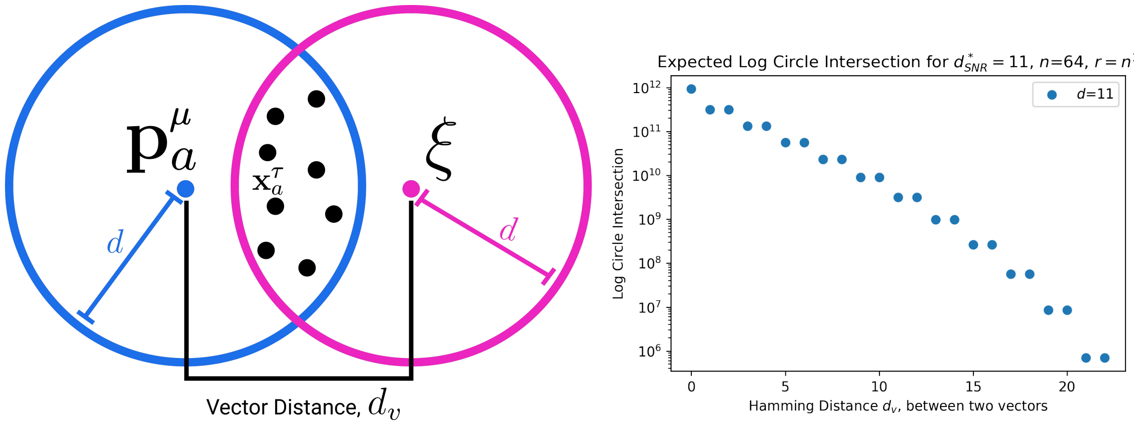


Figure 1.2: ***(Left)*** *SDM's read operation using the Hamming radius d (for reading and writing) and the vector distance $d_v = d(\boldsymbol{\xi}, \mathbf{p}_a^\mu)$. Recall that during the write operation, pattern addresses $\mathbf{p}_a^\mu$ write their pattern pointer $\mathbf{p}_p^\mu$ to neurons located at the addresses $\mathbf{x}_a^\tau$ (denoted here as black dots) within radius d. During the read operation, the query $\boldsymbol{\xi}$ reads from each neuron within radius d, thus creating an intersection.* ***(Right)*** *As d_v between the query and pattern increases (x-axis), the size of their circle intersection falls approximately exponentially (y-axis). We use Eq. 1.2 with $n = 64$ and $d^*_{SNR} = 11$, while varying d_v up to the distance of $d_v = 2d$ beyond which point there is no circle intersection. We plot the y-axis on a log scale to show how, because the curve is approximately linear, the circle intersection is approximately exponential. See Appendix of (Bricken and Pehlevan, 2021) section Circle Intersection Approximate Exponential for a formal analysis of the exponential approximation the circle intersection creates that is robust across parameters n, d, and r.*

tention sets $\beta = 1/\sqrt{n}$. Softmax normalizes a vector of values to sum to 1 and gives the largest values the most weight due to the exponential function, to what extent depending on β. We can re-write this using our notation, including distinguishing continuous vectors in $\mathbb{R}^n$ from binary ones by putting a tilde above them:

$$\tilde{\boldsymbol{\xi}}^{new} = \tilde{P}_p \text{softmax}(\beta \tilde{P}_a{}^T \tilde{\boldsymbol{\xi}}). \tag{1.4}$$

We write $K = W_k Y = \tilde{P}_a$ as the raw input patterns Y are projected by the learnt weight matrix W_k into the SDM vector space to become the addresses $\tilde{P}_a$. Similarly, $V = W_v Y = \tilde{P}_p$ and $Q = W_q \mathbf{q} = \tilde{\boldsymbol{\xi}}$.

Showing the approximation between SDM Eq. 1.1 and Attention Eq. 1.4 requires two steps: (i) Attention must L^2 normalize its vectors. This is a small step because the Transformer already uses LayerNorm (Ba *et al.*, 2016) before and after its Attention operation that we later relate to L^2 normalization; (ii) A β coefficient for the softmax exponential must be chosen such that it closely approximates the almost exponential decay of SDM's circle intersection calculation.

To proceed, we define a map from binary vectors $\mathbf{a}$, $\mathbf{b}$ to L^2 normalized continuous vectors $\hat{\mathbf{a}}$, $\hat{\mathbf{b}}$, $h(\mathbf{a}) = \hat{\mathbf{a}}$, such that for any pair of pattern addresses the following holds:

$$d(\mathbf{a}, \mathbf{b}) = \lfloor \frac{n}{2}\big(1 - \hat{\mathbf{a}}^T \hat{\mathbf{b}}\big) \rfloor, \tag{1.5}$$

where $\lfloor \cdot \rfloor$ is the floor operator. We assume that this map exists, at least approximately. This map allows us to relate the binary SDM circle intersection (Eq. 1.2) to the exponential used in Attention (Eq. 1.4) by plugging it into the exponential approximation of Eq. 1.3:

$$\begin{aligned}\mathcal{I}\big(d(\mathbf{p}_a, \boldsymbol{\xi}), d, n\big) = \mathcal{I}\Big(\lfloor \frac{n}{2}(1 - \hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}}) \rfloor, d, n\Big) &\approx c_1 \exp\Big(-c_2 \lfloor \frac{n}{2}(1 - \hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}}) \rfloor\Big) \\ &\approx c_3 \exp\big(\beta \hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}}\big), \end{aligned} \tag{1.6}$$

where c_3 encompasses the constants outside of the exponential. We replaced the remaining constants in the exponential with β, that is a function of n and d and is an approximation due to the floor operation.

Finally, these results allow us to show the relationship between Attention and SDM:

$$\tilde{\boldsymbol{\xi}}^{new} = \hat{P}_p \text{softmax}(\beta \hat{P}_a^T \hat{\boldsymbol{\xi}}) = \frac{\sum_{\mathbf{p} \in P} \exp(\beta \hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}}) \hat{\mathbf{p}}_p}{\sum_{\mathbf{p} \in P} \exp(\beta \hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}})} \approx \frac{\sum_{\mathbf{p} \in P} \mathcal{I}\Big(\lfloor \frac{n}{2}(1 - \hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}}) \rfloor, d, n\Big) \hat{\mathbf{p}}_p}{\sum_{\mathbf{p} \in P} \mathcal{I}\Big(\lfloor \frac{n}{2}(1 - \hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}}) \rfloor, d, n\Big)}. \tag{1.7}$$

Alternatively, instead of converting cosine similarity to Hamming distance to use the circle intersection Eq. 1.2 in binary vector space, we can extend SDM to operate with L^2 normalized pattern and neuron addresses (Appendix of (Bricken and Pehlevan, 2021) section Continuous SDM).[4] This continuous SDM circle intersection closely matches its binary counterpart in being approximately exponential:

$$\mathcal{I}_c\Big(\hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}}, 1 - \frac{2d}{n}, n\Big) \approx c_4 \exp\big(\beta_c \hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}}\big). \tag{1.8}$$

[4]Pattern pointers can point to a different vector space and thus do not need to be L^2 normalized. However, in canonical SDM they point to pattern addresses in the same space so we write them as also being L^2 normalized in our equations.

We use $\mathcal{I}_c$ to denote this continuous intersection, use Eq. 1.5 to map our Hamming d to cosine similarity, and use coefficients c_4 and β_c to acknowledge their slightly different values. Then, we can also relate Attention as we did in Eq. 1.7 to continuous SDM as follows:

$$\tilde{\boldsymbol{\xi}}^{new} = \hat{P}_p \text{softmax}(\beta \hat{P}_a^T \hat{\boldsymbol{\xi}}) = \frac{\sum_{\mathbf{p}\in P} \exp(\beta \hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}}) \hat{\mathbf{p}}_p}{\sum_{\mathbf{p}\in P} \exp(\beta \hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}})} \approx \frac{\sum_{\mathbf{p}\in P} \mathcal{I}_c \Big(\hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}}, 1 - \frac{2d}{n}, n \Big) \hat{\mathbf{p}}_p}{\sum_{\mathbf{p}\in P} \mathcal{I}_c \Big(\hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}}, 1 - \frac{2d}{n}, n \Big)}. \tag{1.9}$$

We have written Attention with L^2 normalized vectors and expanded out the softmax operation to show that it is approximated when we replace the exponential weights by either the binary or continuous SDM circle intersections (Eqs. 1.7 and 1.9, respectively). The right hand side of Eq. equation 1.7 is identical to Eq. 1.2 aside from using continuous, L^2 normed vectors and dropping the elementwise majority function $g(\cdot)$ that ensured our output was a binary vector. In the Transformer, while the Attention equation does not contain any post-processing function to its query update $\tilde{\boldsymbol{\xi}}^{new}$, it is then post-processed by going through a linear projection and LayerNorm (Vaswani *et al.*, 2017) and can be related to $g(\cdot)$.

To fit β to binary SDM, we convert the Hamming distances into cosine similarity using Eq. 1.5 and use a univariate log linear regression:

$$\log \Big(\mathcal{I}\big(d(\mathbf{p}_a, \boldsymbol{\xi}), d, n\big) \Big) \approx \log(c_3) + \beta(\hat{\mathbf{p}}_a^T \hat{\boldsymbol{\xi}}). \tag{1.10}$$

We expect the exponential behavior to break at some point, if only for the reason that if $d(\mathbf{p}_a, \boldsymbol{\xi}) \geq 2d$ the circle intersection becomes zero. However, closer patterns are those that receive the largest weights and "attention" such that they dominate in the update rule and are the most important.

In Fig. 1.3, we plot the softmax approximation to binary and continuous SDM for our smallest optimal $d^*_{\text{Mem}} = 5$ and largest $d^*_{\text{CD}} = 15$ to show not only the quality of the approximations but also how many orders of magnitude smaller the normalized weights are when $d(\mathbf{p}_a, \boldsymbol{\xi}) > d$. For these plots, we plug into our binary circle intersection equation each possible Hamming distance from 0 to 64 when $n = 64$ and converting Hamming distance to cosine similarity, doing the same for our continuous circle intersection. Here use our binary intersection values to fit β, creating the exponential approximation. To focus our exponential approximation on the most important, closest patterns, we fit our regression to those patterns $d(\mathbf{p}_a, \boldsymbol{\xi}) < d$ and allow it to extrapolate to the remaining values. We then normalize the values and plot them along with an smaller inset plot in log space to better show the exponential relationship. In both plots, looking at the log inset plot first, the point at which the circle intersection in blue ceases to exist or be exponential corresponds to a point in the main normalized plot where the weights are ≈ 0.

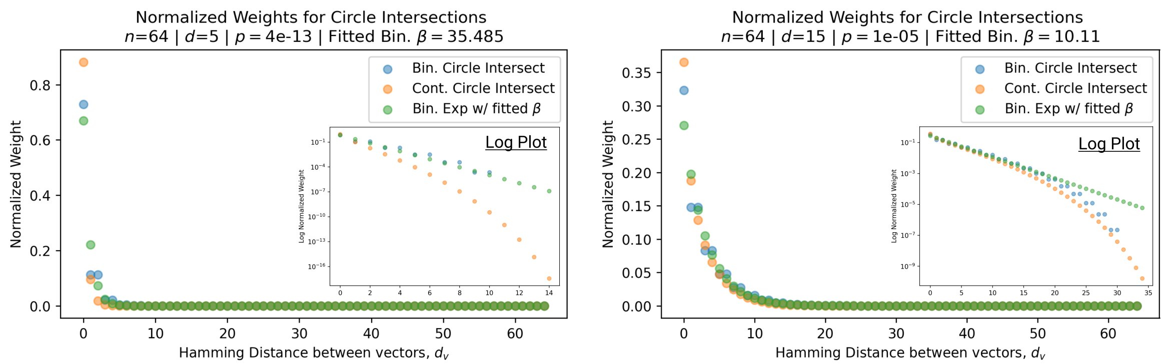


Figure 1.3: *Representative plots showing the relationships in Eqs. 1.7 and 1.9 between the softmax approximation (green) and the normalized binary (blue) and continuous (orange) circle intersections using Transformer Attention parameters. Here the softmax approximation uses the Eq. 1.10 regression to fit its β to binary SDM. We use Eq. 1.5 to relate Hamming distance and cosine similarity for our vector distances on the x-axis. The insets use a log y-axis to show the circle intersections are approximately exponential when $d(\mathbf{p}_a, \boldsymbol{\xi}) \leq 2d$.* ***(Left)*** *Uses $d^*_{Mem} = 5$, corresponding to the Hamming distance reading and writing to $p = 4e - 13$ of the vector space.* ***(Right)*** *Uses $d^*_{CD} = 15$. These results hold well for other values of n and d but we focus on the Attention setting.*

The number of neurons r and how well they cover the pattern manifold are important considerations that will determine SDM's performance and degree of approximation to Attention. Increasing the number of neurons in the circle intersection can be accomplished by increasing the number of neurons in existence, ensuring they cover the pattern manifold, and reducing the dimensionality of the manifold to increase neuron density.[5] In SDM's original formulation, it was assumed that neuronal addresses were randomly distributed and fixed in location, however, extensions to SDM (Keeler, 1988) have proposed biologically plausible

[5]This can be done by learning weight projection matrices like in Attention to make the manifold lower dimensional and also increase separability between patterns.

competitive learning algorithms to learn the manifold (Rumelhart and Zipser, 1985). To ensure the approximations to SDM are tight, we test random and correlated patterns in an autoassociative retrieval task across different numbers of neurons and SDM variants (Appendix of (Bricken and Pehlevan, 2021) section SDMExperiments). These variants include SDM implemented using simulated neurons and the Attention approximation with a fitted β.[6] To summarize, Attention closely approximates the SDM update rule when it uses L^2 normed continuous vectors and a correctly chosen β.

1.4 Trained Attention Converges with SDM

For many instantiations of SDM, there exists a β that can be found via the log linear regression Eq. 1.10 that makes Attention approximate it well. However, depending on the task at hand, there are instantiations of SDM that are better than others as highlighted by the different d^* optimal values. If Attention in the Transformer model is implementing SDM, we should expect for trained Attention to use βs that correspond to reasonable instances of SDM. We use as reference points these optimal d^*s. Attention learns useful pattern representations that are far from random so this SDM β that fits the optimal d^*s are only a weak reference for what β values might be reasonable. However, because these d^* definitions of optimality span from maximizing convergence with query noise, to maximizing memory capacity with noise free queries, we should expect the Transformer dealing with noisy queries and wanting reliable retrieval to interpolate between these d^* values. Here, we provide empirical evidence that this is indeed the case. We analyze the β coefficients learnt by the "Query-Key Normalization" Transformer Attention variant (Henry *et al.*, 2020). Query-Key Normalization makes it straightforward to find β because it is learnt via backpropagation and easy to interpret because it uses cosine similarity between the query and key vectors. To further evaluate the convergence between Attention and SDM β coefficients and make it more general, we also investigate the GPT2 architecture (Radford *et al.*, 2019). However, in

[6]The code for running these experiments, other analyses, and reproducing all figures is available at `https://github.com/trentbrick/attention-approximates-sdm`.

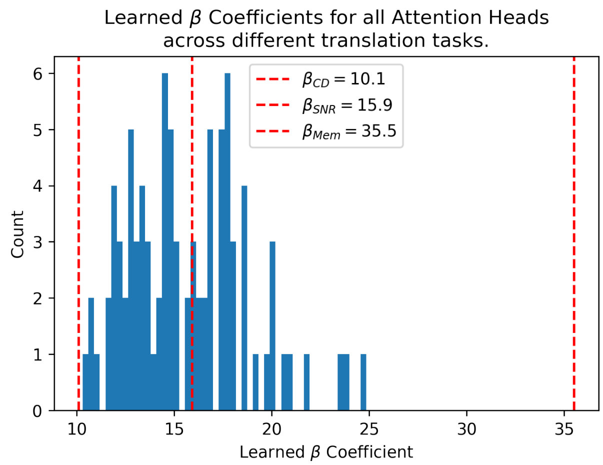


Figure 1.4: *Histogram showing learned β coefficients for all Attention heads across layers for the 5 translation tasks used in (Henry et al., 2020). We plot the β values for Attention that approximate the different d^* definitions showing how the βs are interpolating between them. β_{CD} is optimal for critical distance (maximum noise for each query); β_{SNR} is optimal for Signal-to-Noise ratio. This assumes there is no query noise and SDM wants to minimize noise from other queries. β_{Mem} maximizes memory capacity and also assumes no query noise.*

this case we need to infer "effective" β values from the size of query key dot products in the softmax. This makes these results, more approximate but they remain largely in agreement with the learnt βs of Query-Key Norm (see the Appendix of (Bricken and Pehlevan, 2021) section GPT2 SDM Approximation).

The Query-Key Norm Attention heads learn $\beta \in [10, 25]$ as shown in Fig. 1.4.[7] Note that the whole range of $\beta \in [10, 25]$ interpolates between the d^* values well, in particular with the critical distance optimal that realistically assumes noisy queries and the SNR optimal, where having a high signal to noise ratio is desirable for both memory capacity and critical distance (see Appendix of (Bricken and Pehlevan, 2021) section Optimal Hamming Distance).

In requiring that Attention vectors be L^2 normalized and β fitted, SDM predicted Query-Key Norm. This is interesting because Query-Key Norm has evidence of improving Transformer performance and training speed (Henry *et al.*, 2020; Yang *et al.*, 2021). We say more about its advantages and caveats in Appendix of (Bricken and Pehlevan, 2021) section

[7]These values were graciously provided by the authors of (Henry *et al.*, 2020) in private correspondence for their trained Transformer models.

QueryKeyNorm.

1.5 Transformer Components Interpreted with SDM

We can leverage how Attention approximates SDM to interpret many components of the Transformer architecture.[8]. This exercise demonstrates the explanatory power of SDM and, in relating it to additional unique Transformer components, widens the bridge upon which ideas related to SDM and neuroscience can cross into deep learning and vice versa.

A crucial component of the Transformer is its Feed Forward (FF) layer that is interleaved with the Attention layers and uses approximately 2/3rds of the Transformer's parameter budget (Vaswani *et al.*, 2017). Attention's Transformer implementation and SDM deviate importantly in Attention's use of ephemeral patterns from the current receptive field. In order to model temporal dependencies beyond the receptive field, we want Attention to be able to store persistent memories. Work has compellingly shown that the FF layers store these persistent memories (Sukhbaatar *et al.*, 2019; Geva *et al.*, 2020; Carlini *et al.*, 2020). SDM can be interpreted as this FF layer because in (Sukhbaatar *et al.*, 2019) the FF layer was substituted with additional, persistent key and value vectors that Attention learnt independently rather than projecting from its current inputs. This substitution performed on par with the FF layer which, combined with deeper analysis in (Geva *et al.*, 2020), shows the FF layers are performing Attention with long term memories and thus can be directly interpreted as SDM.

Another crucial component of the Transformer is its use of LayerNorm (Ba *et al.*, 2016; Lu *et al.*, 2021; Vaswani *et al.*, 2017). LayerNorm has a natural interpretation by SDM as implementing a similar constraint to L^2 normalization. However, while it ensures all of the vectors are on the same scale such that their dot products are comparable to each other, it does not constrain these dot products to the interval $[-1, 1]$ like cosine similarity. In addition to providing an interpretation for the importance of LayerNorm, this discrepancy has led to

[8]For a summary of the components that make up the full Transformer architecture see Fig. **??** of Appendix of (Bricken and Pehlevan, 2021) section Transformer Architecture

two insights: First, as previously mentioned, it predicted that Query-Key Normalization could be a useful inductive bias (more details in Appendix of (Bricken and Pehlevan, 2021) section QueryKeyNorm). Second, it has provided caution to interpretations of Attention weights. Fig. **??** of Appendix of (Bricken and Pehlevan, 2021) section ValueL2Norm shows that there are many value vectors that receive very small amounts of attention but have large L^2 norms that dominate the weighted summation of value vectors. This makes it misleading to directly interpret Attention weights without L^2 normalization of the value vectors and this has not been done in work including (Vaswani *et al.*, 2017; Vig, 2019; Tenney *et al.*, 2020; Baan *et al.*, 2019). Beyond helping future interpretations of Attention, we tested L^2 normalized value vectors as a potentially useful inductive bias by training a GPT2 model with it. Our results showed that this did not change performance but L^2 normalization should still be performed in cases where the Attention weights will be interpreted. See Appendix of (Bricken and Pehlevan, 2021) section ValueL2Norm for a full discussion.

Finally, multi-headed Attention is a Transformer component where multiple instantiations of Attention operate at the same hierarchical level and have their outputs combined. Multi-heading allows SDM to model probabilistic outputs, providing an interpretation for why it benefits Attention. For example, if we are learning a sequence that can go "$A \to B$" and "$A \to Z$" with equal probability, we can have one SDM module learn each transition. By combining their predictions we correctly assign equal probability to each. This probabilistic interpretation could explain evidence showing that Attention heads pay attention to different inputs and why some are redundant post training (Michel *et al.*, 2019; Alammar, 2020; Rogers *et al.*, 2020).

An important difference between SDM and the Transformer that remains to be reconciled is in the Transformer's hierarchical stacking of Attention. This is because, unlike in the traditional SDM setting where the pattern addresses (keys) and pattern pointers (values) are known in advance and written into memory, this cannot be done for layers of SDM beyond the first that will need to learn latent representations for its pattern addresses and pointers (keys and values). Transformer Attention solves this problem by learning its higher level keys and values, treating each input token as its own query to generate a new latent representation that is then projected into keys and values (Vaswani *et al.*, 2017). This does not mean SDM would fail to benefit from hierarchy. As a concrete example, the operations of SDM are related to the hierarchical operations of (Chen *et al.*, 2018b). More broadly, we believe thinking about how to learn the latent keys and values for the higher levels of SDM could present new Transformer improvements. A key breakthrough of the recent Performer architecture that highlights the arbitrariness of the original Transformer solution is its use of a reduced set of latent keys and values (Choromanski *et al.*, 2021).

1.6 A Biologically Plausible Implementation of Attention

Here, we provide an overview of SDM's biological plausibility to provide a biologically plausible implementation of Attention. SDM's read and write operations have non trivial connectivity requirements described in (Kanerva, 1988, 1993). Every neuron must: (i) know to fire if it is within d Hamming distance of an input; (ii) uniquely update each element of its storage vector when writing in a new pattern; (iii) output its storage vector during reading using shared output lines so that all neuron outputs can be summed together.

Unique architectural features of the cerebellar cortex can implement all of these requirements, specifically via the three way convergence between granule cells, climbing fibers and Purkinje cells: (i) all granule cells receive inputs from the same mossy fibers to check if they are within d of the incoming query or pattern; (ii) each granule cell has a very long parallel fiber that stores memories in synapses with thousands of Purkinje cells (Hoxha *et al.*, 2016), updated by LTP/LTD (Long Term Potentiation/Depression) from joint firing with climbing fibers; (iii) all granule cells output their stored memories via their synapses to the Purkinje cells that perform the summation operation and use their firing threshold to determine if the majority bit was a 1 or 0, outputting the new query (Kanerva, 1988, 1993). Moreover, the Drosophila mushroom body is highly similar to the cerebellum and the previous cell labels for each function can be replaced with Kenyon cells, dopaminergic neurons, and mushroom body output neurons, respectively (Modi *et al.*, 2020).

While SDM fits the unique features of the cerebellum well, this connection has limitations. Explanations for some of the original model's limitations have been put forward to account for sparse dendritic connections of Granule cells (Jaeckel, 1989b) and the functions of at least two of the three inhibitory interneurons: Golgi, Stellate and Basket cells (Keeler, 1988; Sezener *et al.*, 2021a). However, there are futher challenges that remain, including better explanations of the inputs to the mossy and climbing fibers and outputs from the Purkinje cells; in particular, how the mossy and climbing fiber inputs synchronize for the correct spike time dependent plasticity (Markram *et al.*, 1997). Another phenomenon that SDM does not account for is the ability of Purkinje cells to store the time intervals associated with memories (Gallistel, 2017). Further research is necessary to update the state of SDM's biological plausibility with modern neuroscientific findings.

1.7 Related Work

Previous work showed that the modern Hopfield Network, when made continuous and optimized differently, becomes Attention (Ramsauer *et al.*, 2020; Krotov and Hopfield, 2016). This result was one motivation for this work because Hopfield Networks are another associative memory model. In fact, it has been shown that SDM is a generalization of the original Hopfield Network (Appendix of (Bricken and Pehlevan, 2021)) (Keeler, 1988). While SDM is a generalization of Hopfield Networks, their specific discrepancies provide different perspectives on Attention. Most notably, Hopfield Networks assume symmetric weights that create an energy landscape, which can be powerfully used in convergence proofs, including showing that one step convergence is possible for the modern Hopfield Network, and by proxy, Attention and SDM when it is a close approximation (Ramsauer *et al.*, 2020; Krotov and Hopfield, 2016). However, these symmetric weights come at the cost of biological plausibility that SDM provides in addition to its geometric framework and relation to Vector Symbolic Architectures (Keeler, 1988; Krotov and Hopfield, 2020a).

Other works have tried to reinterpret or remove the softmax operation from Attention because the normalizing constant can be expensive to compute (Katharopoulos *et al.*, 2020;

Schlag *et al.*, 2021). However, while reducing computational cost, these papers show that removing the softmax operation harms performance. Meanwhile, SDM not only shows how Attention can be written as a Feedforward model (Kanerva, 1993) but also reveals that through simple binary read and write operations, (the neuron is either within Hamming/cosine distance or it's not) the softmax function emerges with no additional computational cost.

Since the publication of SDM, there have been a number of advancements not only to SDM specifically, but also through the creation of related associative memory algorithms under the name of "Vector Symbolic Architectures" (Kanerva, 2009). Advancements to SDM include using integer rather than binary vectors (Prager and Fallside, 1989), handling correlated patterns (Keeler, 1988), and hierarchical data storage (Manevitz and Zemach, 1997). Vector Symbolic Architectures, most notably Holographic Reduced Representations, have ideas that can be related back to SDM and the Transformer in ways that may be fruitful (Danihelka *et al.*, 2016; Plate, 1991; Kanerva, 1996; Smolensky, 1990; Gayler, 1998; Hawkins and Ahmad, 2016).

The use of external memory modules in neural networks has been explored most notably with the Neural Turing Machine (NTM) and its followup, the Differentiable Neural Computer (DNC) (Graves *et al.*, 2014, 2016). In order to have differentiable read and write operations to the external memory, they use the softmax function. This, combined with their use of cosine similarity between the query and memory locations, makes both models closely related to SDM. A more recent improvement to the NTM and DNC directly inspired by SDM is the Kanerva Machine (Wu *et al.*, 2018a,b; Gregor *et al.*, 2019; Marblestone *et al.*, 2020). However, the Kanerva Machine remains distinct from SDM and Attention because it does not apply the a Hamming distance threshold on the cosine similarity between its query and neurons. Independent of these discrepancies, we believe relating these alternative external memory modules to SDM presents a number of interesting ideas that will be explored in future work.

1.8 Discussion

The result that Attention approximates SDM should enable more cross pollination of ideas between neuroscience, theoretical models of associative learning, and deep learning. Considering avenues for future deep learning research, SDM's relationship to Vector Symbolic Architectures is particularly compelling because they can apply logical and symbolic operations on memories that make SDM more powerful (Plate, 1991; Eliasmith, 2013; Lake *et al.*, 2016; Piantadosi, 2021; Bengio and Marcus, 2019). SDM and its relation to the brain can inspire new research in not only deep learning but also neuroscience, because of the empirical success of the Transformer and its relation to the cerebellum, via SDM.

Our results serve as a new example for how complex deep learning operations can be approximated by, and mapped onto, the functional attributes and connectivity patterns of neuronal populations. At a time when many new neuroscientific tools are mapping out uncharted neural territories, we hope that more discoveries along the lines of this work connecting deep learning to the brain will be made (Alon *et al.*, 2020; Xu *et al.*, 2020; Scaplen *et al.*, 2020).

Limitations While our work shows a number of convergences between SDM, Attention, and full Transformer models, these relationships remain approximate. The primary approximation is the link between SDM and Attention that exists not only in SDM's circle intersection being approximately exponential but also its use of a binary rather than continuous space. Another approximation is between optimal SDM Hamming radii d^* and Attention β coefficients. This is because we assume patterns are random to derive the d^* values. Additionally, in the GPT2 Transformer models we must infer their effective β values. Finally, there is only an approximate relationship between SDM and the full Transformer architecture, specifically with its Feed Forward and LayerNorm components.

1.9 Conclusion

We have shown that the Attention update rule closely approximates SDM when it L^2 norms its vectors and has an appropriate β coefficient. This result has been shown to hold true in both theory and empirical evaluation of trained Transformer models. SDM predicts that Transformers should normalize their key, query and value vectors, preempting the development of Query-Key Normalization and adding nuance to the interpretation of Attention weights. We map SDM onto the Transformer architecture as a whole, relate it to other external memory implementations, and highlight extensions to SDM. By discussing how SDM can be mapped to specific brain architectures, we provide a potential biological implementation of Transformer Attention. Thus, our work highlights another link between deep learning and neuroscience.

Chapter 2

Sparse Distributed Memory is a Continual Learner[1]

Continual learning is a problem for artificial neural networks that their biological counterparts are adept at solving. Building on work using Sparse Distributed Memory (SDM) to connect a core neural circuit with the powerful Transformer model, we create a modified Multi-Layered Perceptron (MLP) that is a strong continual learner. We find that every component of our MLP variant translated from biology is necessary for continual learning. Our solution is also free from any memory replay or task information, and introduces novel methods to train sparse networks that may be broadly applicable.

This work was published at the International Conference on Learning Representations (ICLR) 2023.

Author Contributions

- Trenton Bricken conceived of the project and theory, conducted all experiments, and wrote the paper with suggestions from co-authors.
- Xander Davies co-discovered Stale Momentum with T.B, conducted early investigations

[1]Co-authored with my advisor

of Dead Neurons, reviewed related work, and participated in discussions.

- Deepak Singh reviewed robustness and Top-K related work and participated in discussions.
- Dmitry Krotov advised on the continual learning experiments, Top-K learning theory, and relations to associative memory models.
- Gabriel Kreiman supervised the project providing guidance throughout on theory and experiments.

2.1 Introduction

Biological networks tend to thrive in continually learning novel tasks, a problem that remains daunting for artificial neural networks. Here, we use Sparse Distributed Memory (SDM) to modify a Multi-Layered Perceptron (MLP) with features from a cerebellum-like neural circuit that are shared across organisms as diverse as humans, fruit flies, and electric fish (Modi *et al.*, 2020; Xie *et al.*, 2022). These modifications result in a new MLP variant (referred to as SDMLP) that uses a Top-K[2] activation function (keeping only the k most excited neurons in a layer on), no bias terms, and enforces both L^2 normalization and non-negativity constraints on its weights and data. All of these SDM-derived components are necessary for our model to avoid catastrophic forgetting.

We encounter challenges when training the SDMLP that we leverage additional neurobiology to solve, resulting in better continual learning performance. Our first problem is with "dead neurons" that are never active for any input and which are caused by the Top-K activation function (Makhzani and Frey, 2014b; Ahmad and Scheinkman, 2019). Having fewer neurons participating in learning results in more of them being overwritten by any new continual learning task, increasing catastrophic forgetting. Our solution imitates the "GABA Switch" phenomenon where inhibitory interneurons that implement Top-K will

[2] Also called "k Winner Takes All" in related literature.

excite rather than *inhibit* early in development (Gozel and Gerstner, 2021).

The second problem is with optimizers that use momentum, which becomes "stale" when training highly sparse networks. This staleness refers to the optimizer continuing to update inactive neurons with an out-of-date moving average, killing neurons and again harming continual learning. To our knowledge, we are the first to formally identify this problem that will in theory affect any sparse model, including recent Mixtures of Experts (Fedus *et al.*, 2021; Shazeer *et al.*, 2017).

Our SDMLP is a strong continual learner, especially when combined with complementary approaches. Using our solution in conjunction with Elastic Weight Consolidation (EWC) (Kirkpatrick *et al.*, 2017), we obtain, to the best of our knowledge, state-of-the-art performance for CIFAR-10 in the class incremental setting when memory replay is not allowed. Another variant of SDM, developed independently of our work, appears to be state-of-the-art for CIFAR-100, MNIST, and FashionMNIST, with our SDMLP as a close second (Shen *et al.*, 2021).

Excitingly, our continual learning success is "organic" in resulting from the underlying model architecture and does not require any task labels, task boundaries, or memory replay. Abstractly, SDM learns its subnetworks responsible for continual learning using two core model components. First, the Top-K activation function causes the k neurons most activated by an input to specialize towards this input, resulting in the formation of specialized subnetworks. Second, the L^2 normalization and absence of a bias term together constrain neurons to the data manifold, ensuring that all neurons democratically participate in learning. When these two components are combined, a new learning task only activates and trains a small subset of neurons, leaving the rest of the network intact to remember previous tasks without being overwritten

As a roadmap of the paper, we first discuss related work (Section 3.2). We then provide a short introduction to SDM (Section 2.3), before translating it into our MLP (Section 2.4). Next we present our results, comparing the organic continual learning capabilities of our SDMLP against relevant benchmarks (Section 2.5). Finally, we conclude with a discussion on the limitations of our work, sparse models more broadly, and how SDM relates MLPs to Transformer Attention (Section 3.4).

2.2 Related Work

Continual Learning - The techniques developed for continual learning can be broadly divided into three categories: architectural (Goodfellow *et al.*, 2014, 2013), regularization (Smith *et al.*, 2022; Kirkpatrick *et al.*, 2017; Zenke *et al.*, 2017; Aljundi *et al.*, 2018), and rehearsal (Lange *et al.*, 2021; Hsu *et al.*, 2018). Many of these approaches have used the formation of sparse subnetworks for continual learning (Abbasi *et al.*, 2022; Aljundi *et al.*, 2019b; Ramasesh *et al.*, 2022; Iyer *et al.*, 2022; Xu and Zhu, 2018; Mallya and Lazebnik, 2018; Schwarz *et al.*, 2021; Smith *et al.*, 2022; Le *et al.*, 2019; Le and Venkatesh, 2022).[3] However, in contrast to our "organic" approach, these methods employ complex algorithms and additional memory consumption to explicitly protect model weights important for previous tasks from overwrites.[4]

Works applying the Top-K activation to continual learning include (Aljundi *et al.*, 2019b; Gozel and Gerstner, 2021; Iyer *et al.*, 2022). (Srivastava *et al.*, 2013) used a local version of Top-K, defining disjoint subsets of neurons in each layer and applying Top-K locally to each. However, this was only used on a simple task-incremental two-split MNIST task and without any of the additional SDM modifications that we found crucial to our strong performance (Table 2).

The Top-K activation function has also been applied more broadly in deep learning (Makhzani and Frey, 2014b; Ahmad and Scheinkman, 2019; Sengupta *et al.*, 2018; Gozel and Gerstner, 2021; Aljundi *et al.*, 2019b). Top-K converges not only with the connectivity of many brain regions that utilize inhibitory interneurons but also with results showing advantages beyond continual learning, including: greater interpretability (Makhzani and

---

[3]Even without sparsity or the Top-K activation function, pretraining models can still lead to the formation of subnetworks, which translates into better continual learning performance as found in (Ramasesh *et al.*, 2022).

[4]Memory replay methods indirectly determine and protect weights by deciding what memories to replay.

Frey, 2014b; Krotov and Hopfield, 2019; Grinberg *et al.*, 2019), robustness to adversarial attacks (Paiton *et al.*, 2020; Krotov and Hopfield, 2018; Iyer *et al.*, 2022), efficient sparse computations (Ahmad and Scheinkman, 2019), tiling of the data manifold (Sengupta *et al.*, 2018), and implementation with local Hebbian learning rules (Gozel and Gerstner, 2021; Sengupta *et al.*, 2018; Krotov and Hopfield, 2019; Ryali *et al.*, 2020; Liang *et al.*, 2020).

FlyModel - The most closely related method to ours is a model of the *Drosophila* Mushroom Body circuitry (Shen *et al.*, 2021). This model, referred to as "FlyModel", unknowingly implements the SDM algorithm, specifically the Hyperplane variant (Jaeckel, 1989b) with a Top-K activation function that we also use and will justify (Keeler, 1988). The FlyModel shows strong continual learning performance, trading off the position of best performer with our SDMLP across tasks and bolstering the title of our paper that SDM is a continual learner.

While both models are derived from SDM, our work both extends the theory of SDM by allowing it to successfully learn data manifolds (App. (Bricken *et al.*, 2023a) section Address Decoding), and reconciles the differences between SDM and MLPs, such that SDM can be trained in the deep learning framework (this includes having no fixed neuron weights and using backpropagation). Both of these contributions are novel and may inspire future work beyond continual learning. For example, learning the data manifold preserves similarity in the data and leads to more specialized, interpretable neurons (e.g., Fig. **??** of App. (Bricken *et al.*, 2023a) section Investigate Continual Learning).

Training in the deep learning framework also allows us to combine SDMLP with other gradient-based methods like Elastic Weight Consolidation (EWC) that the FlyModel is incompatible with (Kirkpatrick *et al.*, 2017). Additionally, demonstrating how MLPs can be implemented as a cerebellar circuit serves as an example for how ideas from neuroscience, like the GABA switch, can be leveraged to the benefit of deep learning.

2.3 Background on Sparse Distributed Memory

Sparse Distributed Memory (SDM) is an associative memory model that tries to solve the problem of how patterns (memories) could be stored in the brain (Kanerva, 1988, 1993) and has close connections to Hopfield networks, the circuitry of the cerebellum, and Transformer Attention (Kanerva, 1988; Bricken and Pehlevan, 2021; Hopfield, 1982, 1984; Krotov and Hopfield, 2016; Tyulmankov *et al.*, 2021; Millidge *et al.*, 2022). We briefly provide background on SDM and notation sufficient to relate SDM to MLPs. For a summary of how SDM relates to the cerebellum, see App. (Bricken *et al.*, 2023a) section SDM Bio Plausibility. We use the continuous version of SDM, where all neurons and patterns exist on the L^2 unit norm hypersphere and cosine similarity is our distance metric (Bricken and Pehlevan, 2021).

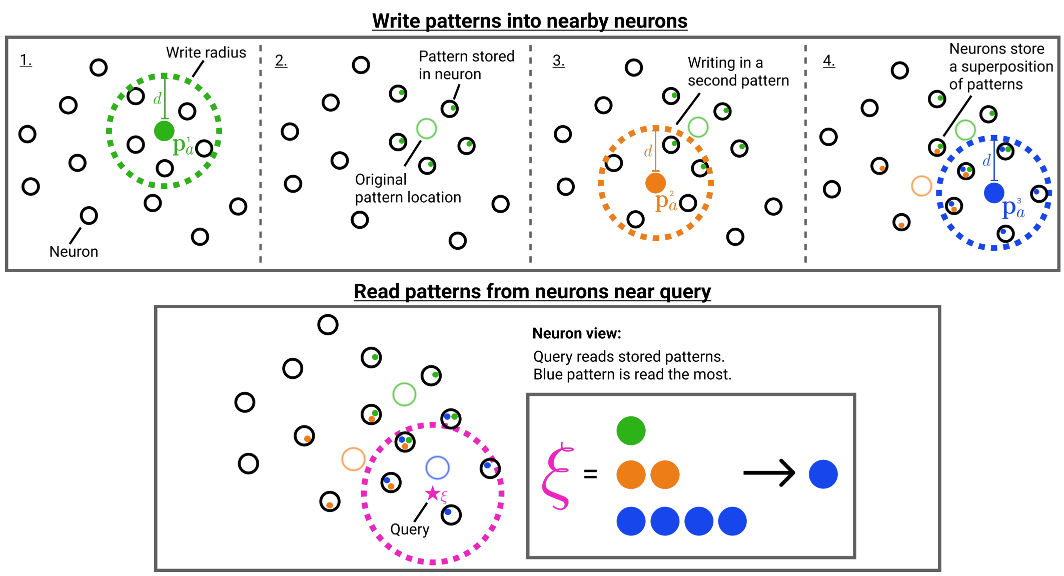


Figure 2.1: ***Graphical Summary of the SDM Write and Read Operations. Top Row -*** *Three patterns being written into nearby neurons. 1. First write operation; 2. Patterns are stored inside neurons and the original pattern address is shown; 3. Writing a second pattern; 4. Writing a third pattern and neurons storing a superposition of multiple patterns.* ***Bottom Row -*** *The SDM read operation. The query reads from nearby neurons with the inset showing the number of times each pattern is read. Blue is read the most as its original pattern location (shown by the hollow circle) is closest to the query. Fig. adapted with permission from (Bricken and Pehlevan, 2021).*

SDM randomly initializes the addresses of r neurons on the L^2 unit hypersphere in an n dimensional space. These neurons have addresses that each occupy a column in our address matrix $X_a \in (L^2)^{n \times r}$, where L^2 is shorthand for all n-dimensional vectors existing on the L^2 unit norm hypersphere. Each neuron also has a storage vector used to store patterns represented in the matrix $X_v \in \mathbb{R}^{o \times r}$, where o is the output dimension. Patterns also have addresses constrained on the n-dimensional L^2 hypersphere determined by their encoding; encodings can be as simple as flattening an image into a vector or as complex as preprocessing with a deep learning model.

Patterns are stored by activating all nearby neurons within a cosine similarity threshold c, and performing an elementwise summation with the activated neurons' storage vector. Depending on the task at hand, patterns write themselves into the storage vector (e.g., during a reconstruction task) or write another pattern, possibly of different dimension (e.g., writing in their one hot label for a classification task). This write operation and the ensuing read operation are summarized in Fig. 2.1.

Because in most cases we have fewer neurons than patterns, the same neuron will be activated by multiple different patterns. This is handled by storing the pattern values in superposition via the aforementioned elementwise summation operation. The fidelity of each pattern stored in this superposition is a function of the vector orthogonality and dimensionality n.

Using m to denote the number of patterns, matrix $P_a \in (L^2)^{n \times m}$ for the pattern addresses, and matrix $P_v \in \mathbb{R}^{o \times m}$ for the values patterns want to write, the SDM write operation is:

$$X_v = P_v b\big(P_a^T X_a\big), \qquad b(e) = \begin{cases} 1, & \text{if } e \geq c \\ 0, & \text{else,} \end{cases} \tag{2.1}$$

where $b(e)$ performs an element-wise binarization of its input to determine which pattern and neuron addresses are within the cosine similarity threshold c of each other.

Having written patterns into our neurons, we read from the system by inputting a query $\boldsymbol{\xi}$, that again activates nearby neurons. Each activated neuron outputs its storage vector and they are all summed elementwise to give a final output $\mathbf{y}$. The output $\mathbf{y}$ can be interpreted as an updated query and optionally L^2 normalized again as a post processing step:

$$\mathbf{y} = X_v b\big(X_a^T \boldsymbol{\xi}\big). \tag{2.2}$$

Intuitively, SDM's query will update towards the values of the patterns with the closest addresses. This is because the patterns with the closest addresses will have written their values into more neurons that the query reads from than any competing patterns. For example, in Fig. 2.1, the blue pattern address is the closest to the query meaning that it appears the most in those nearby neurons the query reads from. SDM is closely related to modern Hopfield networks (Krotov and Hopfield, 2016; Ramsauer *et al.*, 2020; Krotov and Hopfield, 2020b) if they are restricted to a single step update of the recurrent dynamics (Tyulmankov *et al.*, 2021; Bricken and Pehlevan, 2021; Millidge *et al.*, 2022).

2.4 Translating SDM into MLPs for continual learning

A one hidden layer MLP transforms an input $\boldsymbol{\xi}$ to an output $\mathbf{y}$. Using notation compatible with SDM and representing unnormalized continuous vectors with a tilde, we can write the MLP as:

$$\mathbf{y} = \tilde{X}_v f(\tilde{X}_a^T \tilde{\boldsymbol{\xi}} + \tilde{\mathbf{b}}_a) + \tilde{\mathbf{b}}_v, \tag{2.3}$$

where $\tilde{X}_a \in \mathbb{R}^{n \times r}$ and $\tilde{X}_v \in \mathbb{R}^{o \times r}$ are weight matrices corresponding to our SDM neuron addresses and values, respectively. Meanwhile, $\tilde{\mathbf{b}}_a \in \mathbb{R}^r$, $\tilde{\mathbf{b}}_v \in \mathbb{R}^o$ are the bias parameters and $f(\cdot)$ is the activation function used by the MLP such as ReLU (Glorot *et al.*, 2011).

Using this SDM notation for the MLP, it is trivial to see the similarity between the SDM read Eq. 2.2 and Eq. 2.3 for single hidden layer MLPs that was first established in (Kanerva, 1993). However, the fixed random neuron address of SDM makes it unable to effectively model real-world data. Using ideas from (Keeler, 1988), we resolve this inability with a biologically plausible inhibitory interneuron that is approximated by a Top-K activation function. App. (Bricken *et al.*, 2023a) section SDM TopK explains how SDM is modified and the actual Top-K activation function used is presented shortly in Eq. 2.4. This modification makes SDM compatible with an MLP that has no fixed weights. As for the SDM write operation of Eq. 2.1, this is related to an MLP trained with backprop as outlined in App. (Bricken *et al.*, 2023a) section SDM Write Operation MLP.

The Dead Neuron Problem - An issue with the Top-K activation function is that it creates dead neurons (Ahmad and Scheinkman, 2019; Rumelhart and Zipser, 1985; Makhzani and Frey, 2014b; Fedus *et al.*, 2021). Only a subset of the randomly initialized neurons will exist closest to the data manifold and be in the top k most active, leaving all remaining neurons to never be activated. In the continual learning setting, when a new task is introduced, the model has fewer neurons that can learn and must overwrite more that were used for the previous task(s), resulting in catastrophic forgetting.

Rather than pre-initializing our neuron weights using a decomposition of the data manifold (Rumelhart and Zipser, 1985; McInnes and Healy, 2018; Strang, 1993) we ensure that all neurons are active at the start of training so that they update onto the manifold. This approach has been used before but in biologically implausible ways (see App. (Bricken *et al.*, 2023a) section Implausible Dead Neuron Solutions) (Makhzani and Frey, 2014b; Ahmad and Scheinkman, 2019; van den Oord *et al.*, 2017). We instead extend the elegant solution of (Gozel and Gerstner, 2021) by leveraging the inhibitory interneuron we have already introduced. After neurogenesis, when neuronal dendrites are randomly initialized, neurons are excited by the inhibitory GABA neurotransmitter (see App. (Bricken *et al.*, 2023a) section GABA Switch Biology). This means that the same inhibitory interneuron used to enforce competition via Top-K inhibition at first creates cooperation by exciting every neuron, allowing them all to learn. As a result, every neuron first converges onto the data manifold at which point it is inhibited by GABA and Top-K competition begins, forcing each neuron to specialize and tile the data manifold.

We present the full GABA switch implementation in App. (Bricken *et al.*, 2023a) section GABA Switch Considerations that is the most biologically plausible. However, using the positive weight constraint that makes all activations positive, we found a simpler approximation that gives the same performance. This is by linearly annealing our k value from the maximum number of neurons in the layer down to our desired k. A similar approach without the biological justification is used by (Makhzani and Frey, 2014b) but they zero out all values that are not the k most active and keep the remaining k untouched. Motivated by our inhibitory interneuron, we instead subtract the $k+1$-th activation from all neurons that remain on. This change leads to a significant boost in continual learning (see Table 2 and App. (Bricken *et al.*, 2023a) section GABA Switch Considerations). Formally, letting the neuron activations pre inhibition be $\mathbf{a} := X_a^T \boldsymbol{\xi}$ and $\mathbf{a}^*$ post inhibition:

$$\begin{aligned} \mathbf{a}^* &= [\mathbf{a} - I]_+ \\ I &= \text{descending-sort}([\mathbf{a}]_+)_{(k_t+1)} \\ k_t &= \max\left(k_{\text{target}}, \lfloor k_{\max} - E_t(k_{\max} - k_{\text{target}})/s \rfloor\right), \end{aligned} \tag{2.4}$$

where $[\cdot]_+$ is the ReLU operation, $\lfloor \cdot \rfloor$ is the floor operator to ensure k_t is an integer, and descending-sort($\cdot$) sorts the neuron activations in descending order to find the $k+1$-th largest activation. E_t is an integer representing the current training epoch and subscript t denotes that k_t and E_t change over time. Hyperparameter s sets the number of epochs for k to go from its starting value $k_{\max}$, to its target value k_{target}. When $k_t = k_{\text{target}}$ our approximated GABA switch is fully inhibitory for every neuron. An algorithm box in App. (Bricken *et al.*, 2023a) section SDM Train Algorithm summarizes how each SDMLP component functions.

The Stale Momentum Problem - We discovered that even when using the GABA switch, some optimizers continue killing off a large fraction of neurons, to the detriment of continual learning. Investigating why, we have coined the "stale momentum" problem where optimizers that utilize some form of momentum (especially Adam and RMSProp) fail to compute an accurate moving average of previous gradients in the sparse activation setting (Kingma and Ba, 2015; Geoffrey Hinton, 2012). Not only will these momentum optimizers update inactive neurons not in the Top-K, but also explode gradient magnitudes when neurons become activated after a period of quiescence. We explain and investigate stale momenta further in App. (Bricken *et al.*, 2023a) section Stale Momentum and use SGD without momentum as our solution.

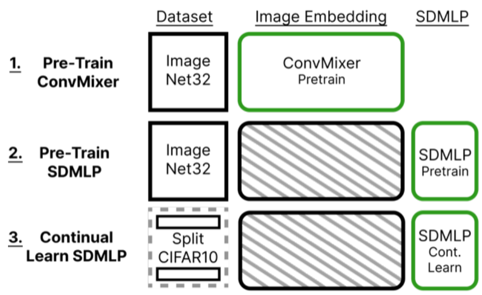
Additional Modifications - There are five smaller discrepancies (expanded upon in App. (Bricken *et al.*, 2023a) section SDM Additional Modifications) between SDM and MLPs: (i) using rate codes to avoid non differentiable binary activations; (ii) using L^2 normalization of inputs and weights as an approximation to contrast encoding and heterosynapticity (Sterling and Laughlin, 2015; Rumelhart and Zipser, 1985; Tononi and Cirelli, 2014); (iii) removing bias terms which assume a tonic baseline firing rate not present in cerebellar granule cells (Powell *et al.*, 2015; Giovannucci *et al.*, 2017); (iv) using only positive (excitatory) weights that respect Dale's Principle (Dale, 1935); (v) using backpropagation as a more efficient (although possibly non-biological (Lillicrap *et al.*, 2020)) implementation of Hebbian learning rules (Krotov and Hopfield, 2019; Sengupta *et al.*, 2018; Gozel and Gerstner, 2021).

2.5 SDM Avoids Catastrophic Forgetting

Here we show that the proposed SDMLP architecture results in strong and organic continual learning. In modifying only the model architecture, our approach is highly compatible with other continual learning strategies such as regularization of important weights (Kirkpatrick *et al.*, 2017; Aljundi *et al.*, 2018; Zenke *et al.*, 2017) and memory replay (Zhang *et al.*, 2021; Hsu *et al.*, 2018), both of which the brain is likely to use in some fashion. We show that combinations of SDMLP with weight regularization are positive sum and do not explore memory replay.

Experimental Setup - Trying to make the continual learning setting as realistic as possible, we use Split CIFAR10 in the class incremental setting with pretraining on ImageNet (Russakovsky *et al.*, 2015). This splits CIFAR10 into disjoint subsets that each contain two of the classes. For example the first data split contains classes 5 and 2 the second split contains classes 7 and 9, etc. CIFAR is more complex than MNIST and captures real-world statistical properties of images. The class incremental setting is more difficult than incremental task learning because predictions are made for every CIFAR class instead of just between the two classes in the current task (Hsu *et al.*, 2018; Farquhar and Gal, 2018). Pretraining on

ImageNet enables learning general image statistics and is when the GABA switch happens, allowing neurons to specialize and spread across the data manifold. In the main text, we present results where our ImageNet32 and CIFAR datasets have been compressed into 256 dimensional latent embeddings taken from the last layer of a frozen ConvMixer that was pre-trained on ImageNet32 (step #1 of our training regime in Fig. 2.2) (Trockman and Kolter, 2022a; Russakovsky *et al.*, 2015).[5]



Method	Neurons	k	Val. Acc.
SDMLP	10K	10	<u>0.71</u>
FlyModel	10K	32	0.82
MAS	10K	NA	0.67
SI	10K	NA	0.44
ReLU	10K	NA	0.21
EWC	10K	NA	0.65
SDMLP+MAS	10K	10	0.84
SDMLP+EWC	10K	10	**0.86**
Oracle	10K	NA	0.93

Figure 2.2: ***Left***, *the three-step training regime. Green outlines denote the module currently being trained. Gray hatches on the ConvMixer denote when it is frozen. Table 1:* ***Right***, *Split CIFAR10 final validation accuracy. We highlight the best performing* <u>*SDMLP*</u>*, baseline, and* ***overall performer*** *in the 10K neuron setting. Oracle was trained on the full CIFAR10 dataset. All results are the average of 5 random task splits.*

Using the image embeddings, we pre-train our models on ImageNet (step #2 of Fig. 2.2). We then reset X_v that learns class labels and switch to continual learning on Split CIFAR10, training every model for 2,000 epochs on each of the five data splits (step #3 of Fig. 2.2). We use such a large number of epochs to encourage forgetting of previous tasks. We test model performance on both the current data split and every data split seen previously to assess forgetting. The CIFAR dataset is split into disjoint sets using five different random seeds to ensure our results are independent of both data split ordering and the classes each split contains.

We use preprocessed image embeddings because single hidden layer MLPs struggle to learn directly from image pixels. This preprocessing is also representative of the visual

[5] This version of ImageNet has been re-scaled to be 32 by 32 dimensions and is trained for 300 epochs (Russakovsky *et al.*, 2015).

processing stream that compresses images used by deeper brain regions (Pisano *et al.*, 2020; Li *et al.*, 2020; Yamins *et al.*, 2014). However, we consider two ablations that shift the validation accuracies but not the rank ordering of the models or conclusions drawn: (i) Removing the ConvMixer embeddings and instead training directly on image pixels (removing step #1 of Fig. 2.2, App. (Bricken *et al.*, 2023a) section CIFAR 10 Raw); (ii) Testing continual learning without any ImageNet32 pre-training (removing step #2 of Fig. 2.2, App. (Bricken *et al.*, 2023a) section No Pretraining). We also run experiments for CIFAR100 (App. (Bricken *et al.*, 2023a) section CIFAR 100), MNIST (App. (Bricken *et al.*, 2023a) section SplitMNIST) and FashionMNIST (App. (Bricken *et al.*, 2023a) section SplitFashionMNIST).

Model Parameters and Baselines - All models are MLPs with 1,000 neurons in a single hidden layer unless otherwise indicated. When using the Top-K activation function, we set $k_{\text{target}} = 10$ and also present $k_{\text{target}} = 1$. We tested additional k values and suggest how to choose the best k_{target} value in App. (Bricken *et al.*, 2023a) section Optimized TopK. Because the k values considered are highly sparse – saving on FLOPs and memory consumption – we also evaluate the 10,000 neuron setting which improves SDM and the FlyModel continual learning abilities in particular.

The simplest baselines we implement are the ReLU activation function, L2 regularization, and Dropout, that are all equally organic in not using any task information (Goodfellow *et al.*, 2014). We also compare to the popular regularization based approaches Elastic Weight Consolidation (EWC), Memory Aware Synapses (MAS), and Synaptic Intelligence (SI) (Kirkpatrick *et al.*, 2017; Aljundi *et al.*, 2018; Zenke *et al.*, 2017; Smith *et al.*, 2022). These methods infer model weights that are important for each task and penalize updating them by using a regularization term in the loss. To track each parameter, this requires at least doubling memory consumption plus giving task boundaries and requiring a coefficient for the loss term.[6]

In the class incremental learning setting, it has been shown that these regularization

[6]We acknowledge that task boundaries can be inferred in an unsupervised fashion but this requires further complexity and we expect the best performance to come from using true task boundaries (Aljundi *et al.*, 2019a).

methods catastrophically forget (Hsu *et al.*, 2018; Gurbuz and Dovrolis, 2022). However, we found these results to be inaccurate and get better performance through careful hyperparameter tuning. This included adding a β coefficient to the softmax outputs of EWC and SI that alleviates the problem of vanishing gradients (see App. (Bricken *et al.*, 2023a) section Beta EWC).We are unaware of this simple modification being used in prior literature but use it to make our baselines more challenging.

We also compare to related work that emphasizes biological plausibility and sparsity. The FlyModel performs very well and also requires no task information (Shen *et al.*, 2021). Specific training parameters used for this model can be found in App. (Bricken *et al.*, 2023a) section Fly Model Parameters. Additionally, we test Active Dendrites (Iyer *et al.*, 2022) and NISPA (Neuro-Inspired Stability-Plasticity Adaptation) (Gurbuz and Dovrolis, 2022). However, these models both use the easier task incremental setting and failed to generalize well to the class incremental setting, especially Active Dendrites, on even the simplest Split MNIST dataset (App. (Bricken *et al.*, 2023a) section SplitMNIST).[7] The failure for task incremental methods to apply in a class incremental setting has been documented before (Farquhar and Gal, 2018). These algorithms are also not organic, needing task labels and have significantly more complex, less biologically plausible implementations. We considered a number of additional baselines but found they were all were lacking existing results for the class incremental setting and often lacked publicly available codebases.[8]

CIFAR10 Results - We present the best continual learning results of each method in Fig. 2.3 and its corresponding Table 1. SDMLP organically outperforms all regularization baselines, where its best validation accuracy is 71% compared to 67% for MAS. In the 1K neuron setting (shown in Table 5 of App. (Bricken *et al.*, 2023a) section CIFAR10 Extras) SDMLP with $k = 1$ ties with the FlyModel at 70%. In the 10K neuron setting, the FlyModel does much better achieving 82% versus 71% for SDMLP. However, the combination of SDMLP

[7]Because we do not test memory replay approaches, we ignore the modified version of NISPA developed for the class incremental setting (Gurbuz and Dovrolis, 2022).

[8]All of our code and training parameters can be found in our publicly available codebase: `https://github.com/TrentBrick/SDMContinualLearner`.

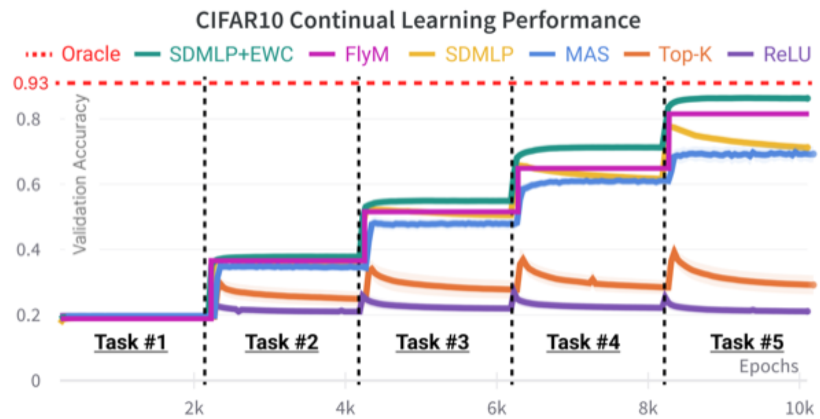


Figure 2.3: ***SDM outperforms regularization methods and their combination is positive sum.*** *We plot validation accuracy across tasks for the best performing methods and baselines on Split CIFAR10. SDMLP+EWC (green line) does the best, then the FlyModel (magenta), then SDMLP (yellow) with MAS (blue) close behind. The Top-K (orange) without SDM modifications and ReLU (purple) baselines do poorly. The FlyModel was only trained for one epoch on each task as per (Shen et al., 2021) but we visually extend the validation accuracy on each task to make method comparison easier. App. (Bricken et al., 2023a) section CIFAR10 Extras visualizes how SDMLP gradually forgets each task compared to catastrophic forgetting of the ReLU baseline. We use the average of 5 random seeds and error bars to show standard error of the mean but the variance is small making them hard to see.*

and EWC leads to the strongest performance of 86%, barely forgetting any information compared to an oracle that gets 93% when trained on all of CIFAR10 simultaneously.[9]

SDMLP and the regularization based methods are complimentary and operate at different levels of abstraction. SDMLP learns what subset of neurons are important and the regularization method learns what subset of weights are important. Because the FlyModel combines fixed weights with Top-K, it can be viewed as a more rigid form of the SDMLP and regularization method combined. The FlyModel also benefits from sparse weights making neurons more orthogonal to respond uniquely to different tasks. We attempted to integrate this weight sparsity into our SDMLP so that our neurons could learn the data manifold but be more constrained to a data subset, hypothetically producing less cross-talk and forgetting between tasks. Interestingly, initial attempts to prune weights either randomly or just using the smallest weights from the ImageNet pretrained models resulted in catastrophic

[9]We found that SGDM for the combined SDMLP+EWC does slightly better than SGD and is used here while SGD worked the best for all other approaches.

forgetting. We leave more in-depth explorations of weight sparsity to future work.

Ablations - We ablate components of our SDMLP to determine which contribute to continual learning in Table 2. Deeper analysis of why each ablation fails is provided in App. (Bricken *et al.*, 2023a) section Investigate Continual Learning but we highlight the two most important here: First, the Top-K activation function prevents more than k neurons from updating for any given input, restricting the number of neurons that can forget information. A small subset of the neurons will naturally specialize to the new data classes and continue updating towards it, protecting the remainder of the network from being overwritten. This theory is supported by results in App. (Bricken *et al.*, 2023a) section Optimized TopK where the smaller k is, the less catastrophic forgetting occurs. We provide further support for the importance of k by showing the number of neurons activated when learning each task in App. (Bricken *et al.*, 2023a) section Investigate Continual Learning.

Second, the L^2 normalization constraint and absence of a hidden layer bias term are crucial to keeping all neurons on the L^2 hypersphere, ensuring they all participate democratically. Without both these constraints, but still using the Top-K activation function, we found that a small subset of the neurons become "greedy" in having a larger weight norm or bias term and always out-compete the other neurons. This results in a small number of neurons updating for every new task and catastrophically forgetting the previous one as shown in the Table 2 ablations. Evidence of manifold tiling is shown in Fig. 6 with a UMAP (McInnes and Healy, 2018) plot fit on the SDM weights (orange) that were pre-trained on ImageNet32 embeddings. We project the embedded CIFAR10 training data (blue) to show that pre-training has given SDM subnetworks that cover the manifold of general image statistics contained in ImageNet32. This manifold tiling is in sharp contrast to the other methods shown in App. (Bricken *et al.*, 2023a) section Investigate Continual Learning and enables SDM to avoid catastrophic forgetting. As further evidence of manifold tiling, we show in the same appendix that SDM weights are often highly interpretable when trained directly on CIFAR10 images.

As an additional ablation to our SDMLP defined by Eq. 2.2, we find that introducing a bias term in the output layer breaks continual learning. This is because the model assigns large bias values to the classes within the current task over any classes in previous tasks, giving the appearance of catastrophic forgetting. Interestingly, this effect only applies to SDM; modifying all of our other baselines by removing the output layer bias term fails to affect their continual learning performance.

[TK fix this table. Probably by importing the subtable package.]

Name	Val. Acc.
SGD	0.63
SGDM	0.54
Adam	0.23
RMSProp	0.20
Linear Subtract	0.63
Linear Mask	0.38
No Linear Anneal	0.35
Negative Weights	0.57
No L^2 Norm	0.20
Hidden Layer Bias Term	0.20
Output Layer Bias Term	0.20

*Table 2: **SDMLP Ablations.** Validation accuracy for ablated versions of SDMLP on CIFAR10 embeddings and $k = 10$ with 1,000 neurons (the same task as Table 1). See main text for details.*

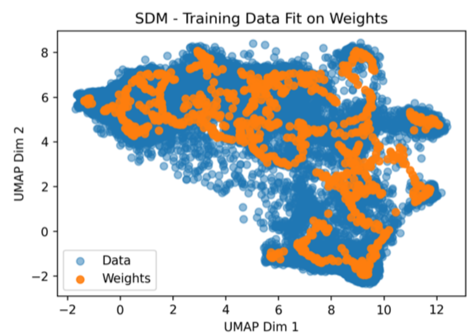


*Fig. 6: **SDM learns the data manifold.** UMAP representation of CIFAR10 embeddings (blue) and SDM weights (orange) pretrained on ImageNet32. The overlap between weights and data is lacking in other methods (see App. (Bricken et al., 2023a) section Investigate Continual Learning).*

2.6 Discussion

Setting out to implement SDM as an MLP, we introduce a number of modifications resulting in a model capable of continual learning. These modifications are biologically inspired, incorporating components of cerebellar neurobiology that are collectively necessary for strong continual learning. Our new solution to the dead neuron problem and identification of the stale momentum problem are key to our results and may further generalize to other sparse neural network architectures.

Being able to write SDM as an MLP with minor modifications is interesting in light of SDM's connection to Transformer Attention (Bricken and Pehlevan, 2021). This link converges with work showing that Transformer MLP layers perform associative memory-like operations that approximate Top-K by showing up to 90% activation sparsity in later layers (Geva *et al.*, 2020; Sukhbaatar *et al.*, 2019; Nelson *et al.*, 2022). Viewing both Attention and MLPs through the lens of SDM presents their tradeoffs: Attention operates on patterns in the model's current receptive field. This increases the fidelity of both the write and read operations because patterns do not need to rely on neurons that poorly approximate their original address location and store values in a noisy superposition. However, in the MLP setting, where patterns are stored in and read from neurons, an increase in noise is traded for being able to store patterns beyond the current receptive field. This can allow for learning statistical regularities in the patterns' superpositions to represent useful prototypes, likely benefiting generalization (Irie *et al.*, 2022).

Limitations - Our biggest limitation remains ensuring that SDM successfully avoids the dead neuron problem and tiles the data manifold in order to continually learn. Setting k_{target} and s decay rate remain difficult and data manifold dependent (App. (Bricken *et al.*, 2023a) section OptimizedTopK). Avoiding dead neurons currently requires training SDM on a static data manifold, whether it is image pixels or a fixed embedding. Initial experiments jointly training SDM modules either interleaved throughout a ConvMixer or placed at the end resulted in many dead neurons and failure to continually learn. We believe this is in large part due to the manifold continuing to change over time.

Constraining neurons to exist on the data manifold and only allowing a highly sparse subset to fire is also suboptimal for maximizing classification accuracy in non continual learning settings. Making k sufficiently large can alleviate this problem but at the cost of continual learning abilities. However, the ability for hardware accelerators to leverage sparsity should allow for networks to become wider (increasing r), rather than decreasing k, and help alleviate this issue (Wang, 2020; Gale *et al.*, 2020). Finally, while our improved version of SDM assigns functionality to five different cell types in the cerebellar circuit, there is more of the circuitry to be mapped such as Basket cells and Deep Cerebellar Nuclei (Sezener *et al.*, 2021b).

Conclusion - We have shown that SDM, when given the ability to model correlated data manifolds in ways that respect cerebellar neurobiology, is a strong continual learner. This result is striking because every component of the SDMLP, including L^2 normalization, the GABA switch, and momentum free optimization are necessary for these continual learning results. Beyond setting a new "organic" baseline for continual learning, we help establish how to train sparse models in the deep learning setting through solutions to the dead neuron and stale momentum problems. More broadly, this work expands the neurobiological mapping of SDM to cerebellar circuitry and relates it to MLPs, deepening links to the brain and deep learning.

Chapter 3

Emergence of Sparse Representations from Noise[1]

A hallmark of biological neural networks, which distinguishes them from their artificial counterparts, is the high degree of sparsity in their activations. This discrepancy raises three questions our work helps to answer: (i) Why are biological networks so sparse? (ii) What are the benefits of this sparsity? (iii) How can these benefits be utilized by deep learning models? Our answers to all of these questions center around training networks to handle random noise. Surprisingly, we discover that noisy training introduces three implicit loss terms that result in sparsely firing neurons specializing to high variance features of the dataset. When trained to reconstruct noisy-CIFAR10, neurons learn biological receptive fields. More broadly, noisy training presents a new approach to potentially increase model interpretability with additional benefits to robustness and computational efficiency.

This work was published at the International Conference on Machine Learning (ICML) 2023.

[1] Co-authored with my advisor

Author Contributions

- Trenton Bricken identified the research direction, implemented and conducted most of the experiments and analyses, mathematically derived the theoretical results and wrote the paper with suggestions from co-authors.
- Rylan Schaeffer helped implement and conduct experiments, helped mathematically develop the theoretical results, and helped write the paper.
- Bruno Olshausen advised on experimental results and theory, particularly concerning sparse coding.
- Gabriel Kreiman supervised the project, providing guidance throughout on theory and experiments.

3.1 Introduction

A striking difference between biological and artificial neural networks is *activation sparsity*. The brain is highly sparse, with an estimated 15% of neurons firing at any given time (Attwell and Laughlin, 2001), whereas deep learning models are often dense. A unifying understanding of this difference is elusive, since there are advantages, disadvantages and unclear implications of sparse representations (Olshausen and Field, 2004).

A popular reason for sparsity in the brain is metabolic efficiency, since action potentials consume $\sim$20% of the brain's energy (Sterling and Laughlin, 2015; Sengupta *et al.*, 2010). This theory is supported by the field of sparse coding which enforces an L_1 penalty on neuron activations and results in neurons learning biological receptive fields when trained on a reconstruction task (Olshausen and Field, 1997a).

In this work, we advance an alternative theory that the need to handle noise is a key driver behind activation sparsity.Removing the L_1 activation penalty and simply adding random isotropic Gaussian noise to the data also produces sparse activations and biological receptive fields.

Previous work found that by taking a first order Taylor approximation, noisy training added a single penalty to the loss function: minimize the Frobenius norm of the model's Jacobian. Formally, $\min_\theta ||\delta f_\theta(\mathbf{x})/\delta\mathbf{x}||_F^2$ where $\mathbf{x}$ is the input data and $f_\theta(\mathbf{x})$ is any non-linear function that maps the input to the output (Bishop, 1995; Camuto *et al.*, 2020). In words, this means the model outputs should be robust with respect to small input perturbations.

However, this result leaves much unexplained. Empirically, training networks with this additional loss term fails to produce sparsity or biological receptive fields. Theoretically, this loss makes no predictions for how the network parameters should change and being a Taylor approximation it only holds in the small noise limit. Moreover, because the Frobenius norm is an L_2 penalty instead of L_1, this adds to the mystery of why noise results in sparse activations.

Motivated by this discrepancy, we avoid the Taylor approximation and disentangle more nuanced effects that noisy training implicitly has on the loss function. To summarize, noisy training adds three additional terms that the loss seeks to *maximize*:

$$\begin{aligned} \text{Noise Terms} = \text{Maximize}\big(&[\text{Neuron Sparsity}] \\ &+ [\text{Neuron Activation Margin}] \\ &+ [\text{Specialized Model Weights}]\big) \end{aligned} \tag{3.1}$$

The theoretical effects of Eq. 3.1 on what each neuron learns are summarized in Fig. 3.1. The ReLU activation function plays a key role. By default, each neuron should have a very negative pre-activation so that it is turned off and unperturbed by any noise. When a neuron must turn on to help in the reconstruction task, it should "jump" from being very negative to very positive in order to maximize the margin around the ReLU activation threshold of 0.

This threshold is where the noise has a non-linear effect on the model output and is avoided by the neuron's receptive field only modelling high variance regions of the distribution. Finally, each neuron should learn receptive fields that de-correlate the effects of noise on the activations, resulting in specialization to non-redundant subsets of the high variance data features.

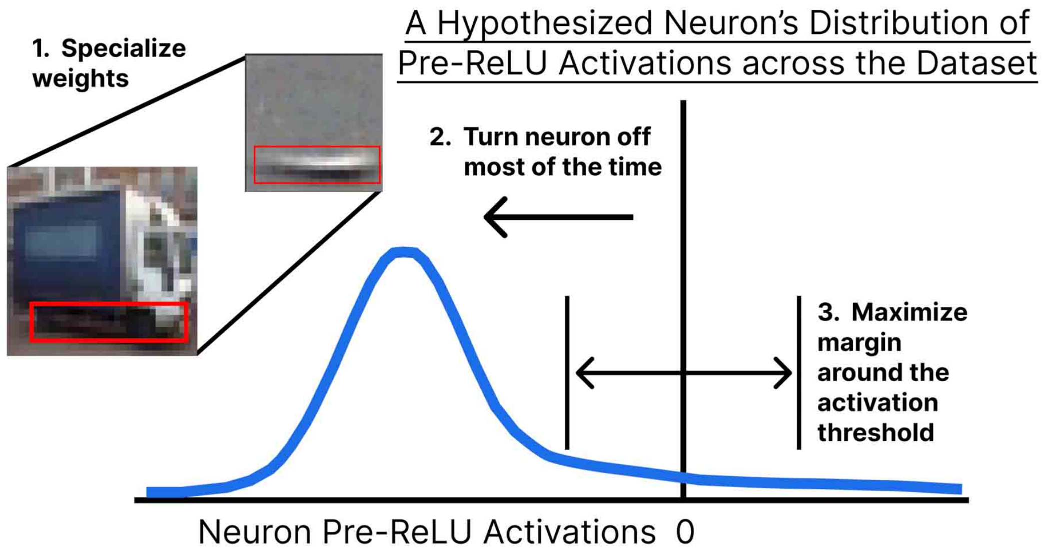


Figure 3.1: ***The three loss objectives introduced by noisy training.*** *We show the distribution of activations across the dataset for an idealized, hypothetical neuron to clarify the loss terms. 1. The neuron should learn specialized weights, in this case we plot a real example of a Gabor filter that will fire strongest for edges like that along the truck bottom. 2. The neuron should be negative such that it spends a majority of its time off (blocking the influence of noise). 3. The neuron when it is activated should "jump" over the ReLU activation threshold to a large positive value, reducing the ability for noise to switch it off. This produces the long right hand tail.*

Empirically, we find that trained networks follow these theoretical predictions (e.g., Fig. 3.2). In proportion to the noise variance up to a cutoff, each neuron learns a negative bias so that it is off by default. Many of the neurons also learn biological receptive fields known to capture the high variance features of natural images and have low activation covariance. Finally, the bias terms and weight norms synchronize such that a sparse $\sim k$ neurons fire for any input datum.

This form of sparsity is distinct from sparsity created via an L_1 activation penalty where many of the neurons in the network are *dead* and never fire for any input. Dead neuron sparsity is misleading and equivalent to having a pruned, smaller network that is densely firing. We distinguish this alternative form of sparsity throughout our work. Note that we consider *activation* sparsity, not *weight* sparsity, which can contribute to activation sparsity but is fundamentally different.

We show that our findings generalize across a number of deep learning tasks, datasets,

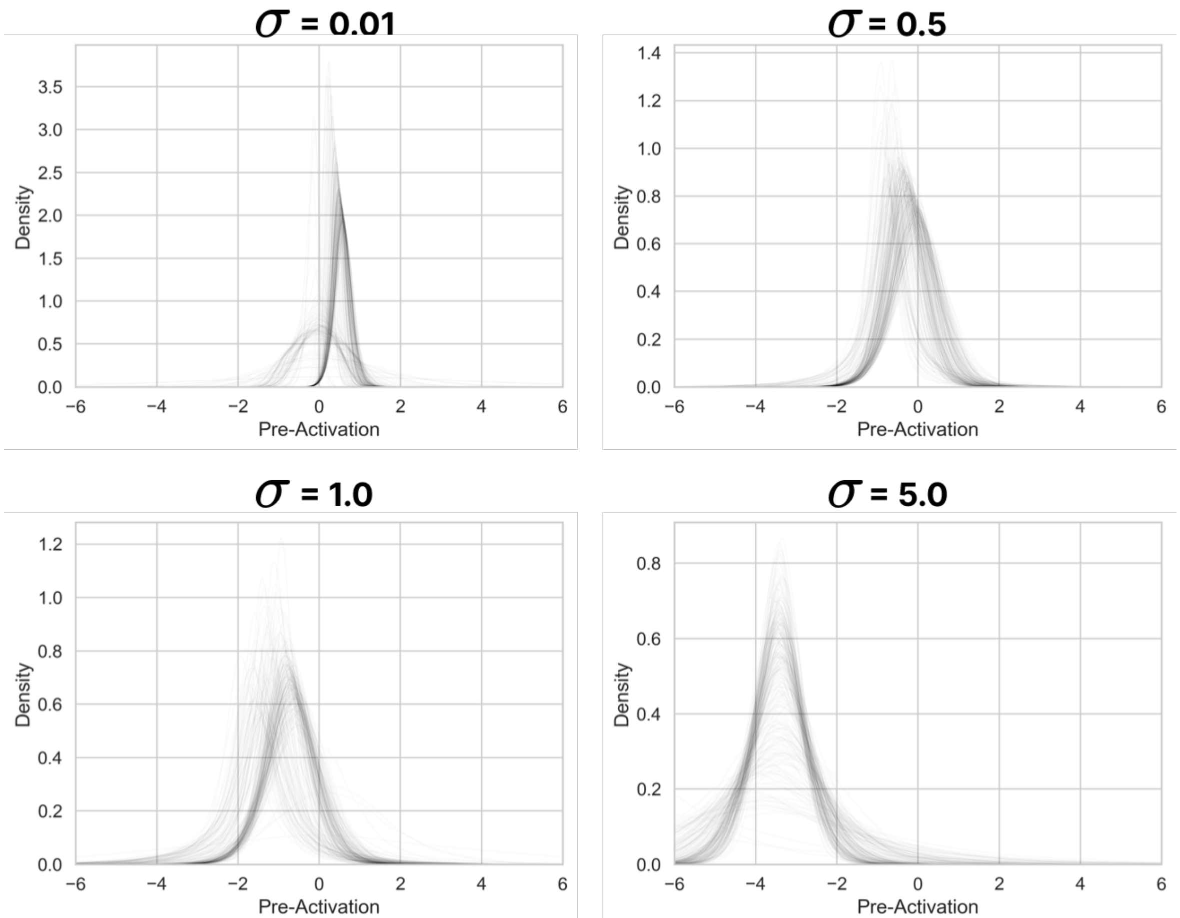


Figure 3.2: ***Pre-Activation Distributions of Shallow Denoising Autoencoders on CIFAR-10 pixels.*** *Distribution of 250 units' pre-activation values, randomly sampled from 10,000 neurons. The bias for each neuron becomes more negative and tails grow longer in proportion to the noise.*

architectures, and can even be used as a pre-training task. Pretraining with noise then removing it still results in sparse activations and, aside from the highest noise variances, has no final performance cost.

Noisy training is appealing because of its simplicity and the inductive biases it introduces. Alternative methods to create activation sparsity are varied but all have in common explicitly penalizing or preventing dense activations and need to account for the issues this introduces (Bricken *et al.*, 2023a; Gregor and LeCun, 2010; Nelson *et al.*, 2022; Srivastava *et al.*, 2014; Martins and Astudillo, 2016). For example, sparse coding with an L_1 penalty needs to be combined with constraints on weight norms to avoid the model "cheating" by shrinking its weights (Olshausen and Field, 1997a).

Meanwhile, methods like Top-k that force exactly the k most active neurons to turn off must pre-determine their k value and need to slowly introduce this constraint (Bricken *et al.*, 2023a). Noisy training endogenously incentivizes sparsity, in addition to weight norm regularization and feature specialization. Noise also results in different receptive fields from sparse coding, resulting in a transition from Gabors to center-surround (Karklin and Simoncelli, 2011). This latter finding suggests that noisy training could potentially be a different approach to removing neuron polysemanticity (Nelson *et al.*, 2022; Elhage *et al.*, 2022).

3.2 Related Work

Injecting noise into training data has a rich history, dating back as early as (Sietsma and Dow, 1991). Noise injection has been interpreted as a form of model regularization to help avoid overfitting (Zur *et al.*, 2009) and improve generalization (Sietsma and Dow, 1991; Matsuoka, 1992). Interest in training with noise rose recently due to de-noising autoencoders playing a critical role in modern diffusion models (Goodfellow *et al.*, 2015; Song and Ermon, 2019).

(Poole *et al.*, 2014) discovered that noise injection results in sparse activations but focused on comparisons to Dropout (Srivastava *et al.*, 2014) and noise injection at different model layers. Our work enriches their contributions by correcting erroneous equations[2], investigating the network sparsity mechanism, deriving more detailed implicit loss terms, exploring the generality of findings across model architectures, and finding the emergence of biological receptive fields (Olshausen and Field, 1997a).

It was previously discovered that de-noising auto encoders, when trained on the MNIST hand written digits dataset (LeCun and Cortes, 2005), learn receptive fields that can be interpreted as Gabor-like (Vincent *et al.*, 2008; Chen *et al.*, 2014). However, because MNIST digits consist of strokes, it is easy to interpret the receptive fields as stroke detectors for this specific dataset, rather than generalized Gabor filters appearing in V1 (Sterling and Laughlin, 2015). To the best of our knowledge, our work is the first to find the emergence of biological receptive fields from MLPs trained on naturalistic whole CIFAR10 images. Regarding the formation of receptive fields found by other models such as AlexNet (Krizhevsky *et al.*, 2012), what is striking about our receptive fields is not how quantitatively similar they look to true biological receptive fields but that they emerge at all just from noisy training.

Another interesting connection exists with the implicit loss terms found in the Kullback-Leibler Divergence (KLD) of variational autoencoders (VAEs). Intuitively, VAEs can be thought of as introducing noise through stochastically sampling from their latent space. This noise is enforced by the prior distribution over the latents, incentivizing the model to parameterize its latent distribution with non-zero variance. (Kumar and Poole, 2020) use the Jacobian approximation to make this relationship explicit while (Chen *et al.*, 2018a) decompose the KLD loss term into three terms which are closely related to maximizing the neuron activation margin and de-correlating neuronal activations. The sparsity term is notably absent but should only appear with rectified non-linearities that are uncommon in VAE latent spaces.

Approaches to enforce activation sparsity include using Top-k activation functions (Ranzato *et al.*, 2007; Makhzani and Frey, 2014a; Ahmad and Scheinkman, 2019), novel regu-

[2]The authors confirmed and kindly helped us to verify the limitations of Eqns. 4 & 7 via private correspondence (Appendix section Poole Derivation has a corrected derivation).

larization terms (Kurtz *et al.*, 2020; Yang *et al.*, 2020) and other approaches (Andriushchenko *et al.*, 2022; Schwarz *et al.*, 2021; Molchanov *et al.*, 2017; Srivastava *et al.*, 2014; Nelson *et al.*, 2022; Martins and Astudillo, 2016).

We state that our model is more robust simply because it has been trained with random noise perturbations (Kireev *et al.*, 2021). However, we do *not* extend this claim to include *adversarial* robustness due to both noise and sparsity causing "gradient masking" that confounds canonical gradient based adversarial attacks. For example, (Xiao *et al.*, 2020) used Top-k activation sparsity to boost adversarial robustness. However, this was overturned by (Tramèr *et al.*, 2020), showing this was the result of gradient masking and alternative attacks remained effective. (Li *et al.*, 2018) claim similar adversarial robustness from additive noise and acknowledge the gradient masking confounder, however, it is unclear if it is adequately accounted for.

Learning reconstructions of training data with sparse activations is the mainstay of sparse coding (for a formal introduction, see App. (Bricken *et al.*, 2023b) section Sparse Coding Summary). Prior work showed that combining the sparse coding L_1 penalty with noisy training produces biological receptive fields (Karklin and Simoncelli, 2011; Doi *et al.*, 2012). (Karklin and Simoncelli, 2011) notably used a non-linear model that not only learned ReLU like activation functions but also negative activation thresholds just like the bias terms our networks learn. The model also shows, in agreement with our findings, that increasing noise causes the transition from Gabors to center-surround receptive fields. Our work brings these findings into the field of deep learning (e.g., using stochastic gradient descent, deeper models, full images) and simplifies the model assumptions by considering only training with noise applied to the input data rather than inside the model, and without additional activation penalties or weight norm constraints.

Finally, sparsely activated networks, when trained with noise, are closely related to the Sparse Distributed Memory (SDM) and Modern Hopfield Network associative memory models that activate a sparse subset of neurons and handle noisy queries (Kanerva, 1988; Krotov and Hopfield, 2016). SDM in particular can be viewed as a de-noising autoencoder and can also be written as an MLP using the Top-k activation function (Bricken *et al.*, 2023a; Keeler, 1988).

3.3 Results

Shallow Denoising Autoencoder - As a starting point, we train a single hidden layer ReLU autoencoder, with additive random noise ϵ sampled from a zero mean, isotropic Gaussian with variance σ^2. The corrupted datum $\tilde{x} = x + \epsilon$ is then fed through the encoder and decoder, and the training objective is to reconstruct the noiseless input. For the single hidden layer case of Eq. 3.2, the encoder and decoder weight matrices and bias terms are: $W_e \in \mathbb{R}^{m \times n}, \mathbf{b}_e \in \mathbb{R}^m, W_d \in \mathbb{R}^{o \times m}, \mathbf{b}_d \in \mathbb{R}^o$ where n is the input dimension, o is output dimension, and m is the number of neurons in the hidden layer. Our loss function uses the mean squared error between the original image and reconstruction across our full dataset, X.

$$\begin{aligned}
&\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim N(0, \sigma^2 I) \\
&\mathbf{z} = W_e \tilde{\mathbf{x}} + \mathbf{b}_e \\
&\mathbf{h} = \varphi(\mathbf{z}) \\
&\tilde{\mathbf{y}} = W_d \, \mathbf{h} + \mathbf{b}_d \\
&\mathcal{L} = \sum_{\mathbf{x} \in X} ||\mathbf{x} - \tilde{\mathbf{y}}||^2.
\end{aligned} \tag{3.2}$$

where φ is the element-wise ReLU nonlinearity. We use Kaiming randomly initialized weights (He *et al.*, 2015) and train until the fraction of active neurons converges.[3] We primarily use the CIFAR10 dataset of 50,000 images with 32x32x3 dimensions, training either on the raw pixels (flattening them into a 3,072 dimensional vector) or latent embeddings of 256 dimensions, produced by a ConvMixer pretrained on ImageNet (Trockman and Kolter, 2022b; Russakovsky *et al.*, 2015). All code and training parameters can be found at: `https://github.com/TrentBrick/SparsityFromNoise`.

[3] This happens after the train and test losses plateau

To visually communicate the relation between noise variance and reconstruction quality, we visualize the reconstructions of shallow autoencoders trained directly on CIFAR10 pixels in Fig. 3.3. Unsurprisingly, the more noise that is applied, the worse reconstruction performance is. This is especially true for noise levels large enough to make a competing image closer to the noisy input than the ground truth input image.

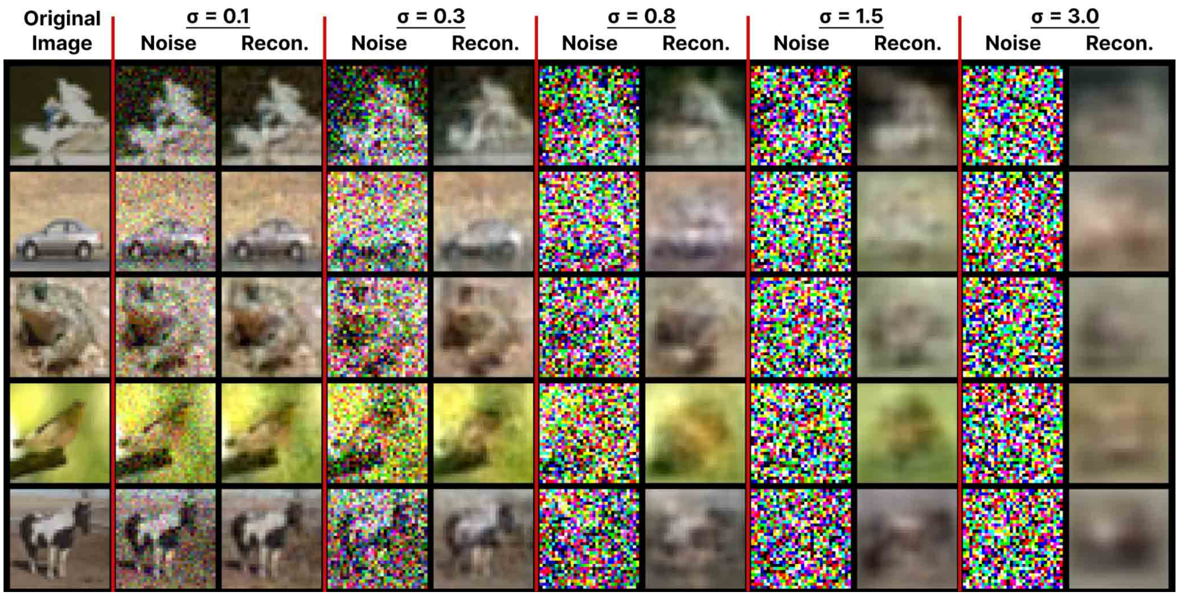


Figure 3.3: ***Example CIFAR10 reconstructions.*** *Example reconstructions obtained at different noise levels for five randomly selected images from the test data. The network at* $\sigma \geq 0.8$ *qualitatively transitions from fuzzy reconstructions to more general image details.*

We summarize our derivation of the three implicit loss terms introduced by noisy training where additional steps can be found in Appendix section Noise Analytical Derivation. We then present empirical results and investigations confirming that noisily trained networks agree with our analysis.

Theory - The key to our result is taking an expectation over the noise (Bishop, 1995).

Working with scalars for clarity:

$$\begin{aligned}\text{Loss} &= \frac{1}{D}\sum_{x\in X}\mathbb{E}_\varepsilon\left[\sum_i^o (x_i-\tilde{y}_i)^2\right]\\ &= \frac{1}{D}\sum_{x\in X}\sum_i^o r_i^2 - 2r_i\mathbb{E}_\varepsilon[\xi_i] + \mathbb{E}_\varepsilon[\xi_i^2] \end{aligned} \tag{3.3}$$

where D is the size of the dataset, $r_i = x_i - \bar{y}_i$, and $\xi_i = \tilde{y}_i - \bar{y}_i$ is the difference between $\tilde{y}_i$, the output produced by the input with noise ε, and the hypothetical output without noise $\bar{y}_i$. Keep in mind that because the input noise is independent of the data, driving the error from noise $\xi_i \to 0$ will maximize the quality of the reconstruction.

Defining $\xi_i = \sum_j^m W_{d_{i,j}}\eta_j$, where $\eta_j = \tilde{h}_j - \bar{h}_j$, we can write $\mathbb{E}_\varepsilon[\xi_i] = \sum_j^m W_{d_{i,j}}\mathbb{E}_\varepsilon[\eta_j]$. The noise penalty ultimately results from the terms $\mathbb{E}_\varepsilon[\eta_j]$, to maximize the activation margin, and $\mathbb{E}_\varepsilon[\eta_j^2]$, to both sparsify activations and de-correlate the encoder weights. We show in Appendix section Noise Analytical Derivation that $\tilde{h}$ is a rectified Gaussian distribution (Salimans, 2016) and $\mathbb{E}_\varepsilon[\eta_j] \geq 0$. Intuitively, this is because, as shown in Fig. 3.4 going from the top row to the second row, the noise creates a Gaussian distribution around the pre-ReLU neuron activation $\bar{z}_j$ which gets truncated where it crosses 0. This destroys symmetry and results in three parts to the Gaussian, the two tails, one of which is now set to 0, and the center which is still symmetric. Taking an expectation, the center cancels with itself but only one of the tails on the positive right hand side is non-zero resulting in the expectation becoming more positive. This positive shift is proportional to how close $\bar{z}_j$ is to the 0 activation threshold and given by the integral:

$$\mathbb{E}_\varepsilon[\eta_j] = \int_{t=|\bar{z}_j|}^{\infty} N(x;0,||\mathbf{w}_{e_j}||\sigma)(x-t)dx \geq 0, \tag{3.4}$$

where $N()$ is the PDF of a normal distribution. Turning to our other term $\mathbb{E}_\varepsilon[\xi_i^2]$, we can expand it as:

$$\mathbb{E}_\varepsilon[(\sum_j^m W_{d_{i,j}}\eta_j)^2] = \mathbb{E}_\varepsilon[\sum_j^m W_{d_{i,j}}^2\eta_j^2 + \sum_{k\neq j}^m W_{d_{i,j}}W_{d_{i,k}}\eta_j\eta_k] \tag{3.5}$$

This results in integrals over the error terms summarized by the bottom row of Fig. 3.4 and given formally in Appendix section Noise Analytical Derivation. There are three

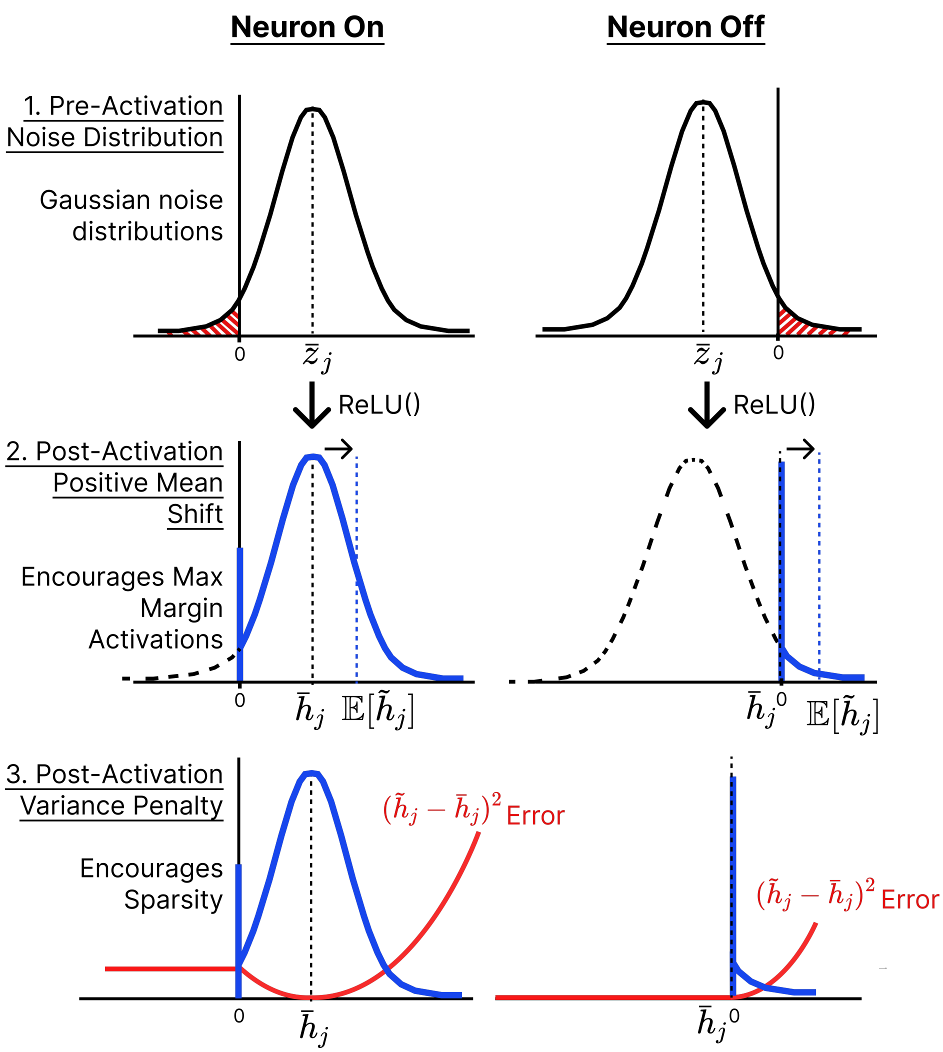


Figure 3.4: ***Intuition for why noise results in max margin activations and sparsity.*** *The columns reference the noise free neuron being on (left, $\bar{z}_j > 0$) or off (right, $\bar{z}_j \leq 0$). Going row-wise: #1. Shows the noise distribution around $\bar{z}_j$. #2. Post-ReLU, the truncated tail results in a positive mean shift $\mathbb{E}_\varepsilon[\tilde{h}_j] \geq \bar{h}_j$ (black vs blue vertical dotted lines). #3. The variance $\mathbb{E}_\varepsilon[\eta_j]$ depends on how much of the distribution is positive.*

observations to be made from these terms: 1. The penalty when the neuron is on ($\bar{h}_j > 0$) is strictly greater than when it is off. 2. When the neuron is off, $\bar{z}_j$ should be as negative as possible to avoid the noise flipping it positive. 3. The cross term $W_{d_{i,j}} W_{d_{i,k}} \mathbb{E}_{\epsilon} \left[\eta_j \eta_k \right]$ incentivizes both minimizing the sum of the decoder weights outer product $||\mathbf{w}_{d_i} \mathbf{w}_{d_i}^T||$ and de-correlating the encoder weights. This is because the term is only present when both neurons are turned on and the probability of this is reduced as $\bar{z}_j \to -\infty$.

Because $\bar{z}_j$ and $\bar{z}_k$ receive the same noise sample they are not independent and their correlation is a function of the cosine similarity between their encoder weights $\mathbf{w}_{e_j}$ and $\mathbf{w}_{e_k}$. Therefore, a way to utilize both neurons in the model (increasing model capacity) but avoid this cross term is by making their encoder weights specialize to unique, localized features of the data.

It is important to note that the encoder weights cannot increase their activation margin by simply increasing their L_2 norm. This is because, increasing their scale will linearly increase noise as much as it does the activation. Therefore, the way to both reduce noise and maximize the activation margin is by having the weights specialize to distinct dataset features, ignoring noise across most of the input while being very activated by unique features. Combining this with the neuron's bias term being strongly negative is a good way to ensure the neuron is not otherwise activated. These desiderata predict that neurons should have activation distributions across the data distribution like in Fig. 3.1 where it has a negative mean and a long right hand tail (shown empirically in Fig. 3.2). Ideally, the tail would in fact be a another mode resulting in a bimodal distribution that straddles the activation threshold, however, this depends upon the distribution of the particular dataset features the neuron specializes to. We now support the theoretical predictions from our derivation with empirical results from training neural networks with noise.

Symmetric Mean Zero Noise Induces Sparse Coding Networks - Our headline Fig. 3.5 shows that a single hidden layer ReLU network with 10,000 neurons learns to become a sparse coding network with a smaller k number of neurons activated by any input as the amount of noise increases. This network was trained on the CIFAR10 embeddings to mimick training within a deeper network.

While Fig. 3.5 only shows the mean number of neurons that are active across all 50,000 CIFAR10 inputs per epoch, Fig. 3.6 shows the fraction of active neurons for every training batch. This reveals that as noise increases, the variance for how many neurons are active shrinks, making the mean an accurate summary statistic. Note that there are no dead neurons. Fig. 3.6 also shows the bias terms for each neuron and the L_2 norm of their encoder weights W_e. It is clear that these values all synchronize to a value decided on by the network and are responsible for the sparse k neurons remaining active after the ReLU nonlinearity is applied.

Figure 3.5 also shows what looks like a delayed phase transition for the higher noise training to sparsify (e.g. $\sigma = 10$, brown line). This delay arises from the network waiting for the L_2 norms of its encoder weights W_e to become sufficiently small and bias terms sufficiently negative to silence more than 50% of the neurons (bottom row of Fig. 3.6). Higher noise levels increase the L_2 norm of the data, which the weights must counteract to keep the neuron activations small enough so that bias term can have an effect (each bias is initialized at 0 and can only become negative so quickly). Appendix section Weight Scales confirms this explanation by initializing weights with smaller L_2 norms that result in faster sparsification.

Shared Bias Terms and Inhibitory Interneurons - In biological neural networks, sparsity is achieved via inhibitory interneurons that suppress all but the most active neurons from firing (Haider *et al.*, 2010). Theoretical and empirical results support their involvement in both silencing noise and separating neural representations. Examples include horizontal interneurons in the retina (Sterling and Laughlin, 2015) and Golgi interneurons in cerebellar-like structures (Fleming *et al.*, 2022; Lin *et al.*, 2014; Xie *et al.*, 2022).

To test the observed transition into a sparse coding network, we modify the encoder bias from $\mathbf{b}_e$ to $b_e\mathbf{1}$, where b_e is a (shared) scalar and $\mathbf{1}$ is the all ones vector. This results in near identical model performance and sparsity.

It is interesting that the bias terms of all the neurons converge to a particular negative

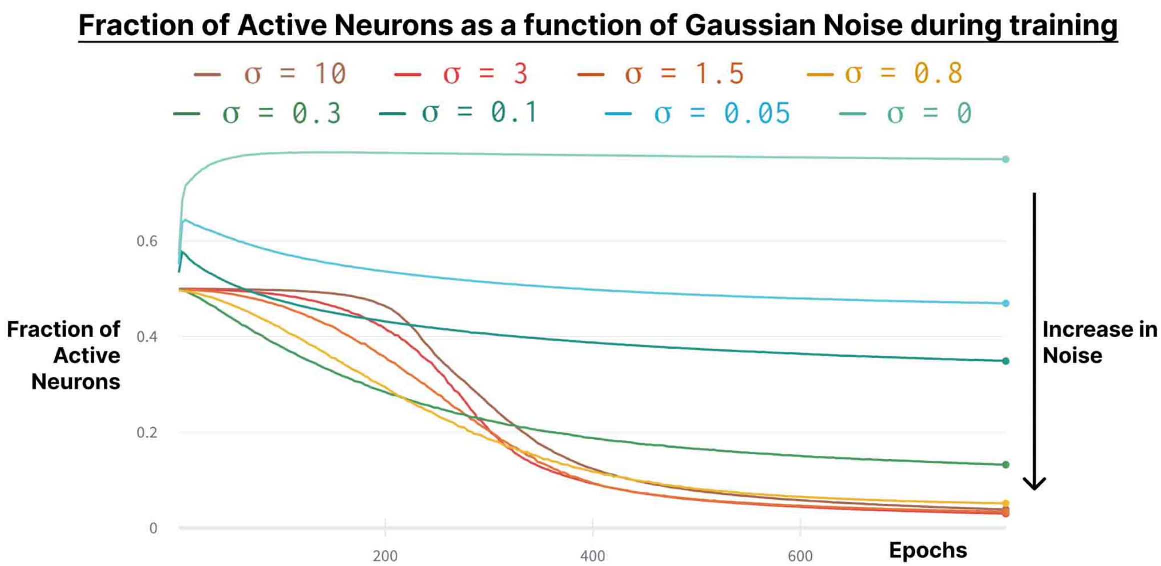


Figure 3.5: ***The positive relationship between activation sparsity and noise.*** *The average fraction of neurons active during each latent CIFAR10 training batch over 800 epochs. Each line corresponds to training randomly initialized networks with different noise levels σ denoted by different colors. We show the average of three different random seeds and their standard error of the mean (not visible as variance is so low). The higher noise levels take longer to sparsify because the noise results in larger magnitude activation values that require more parameter updates to counteract.*

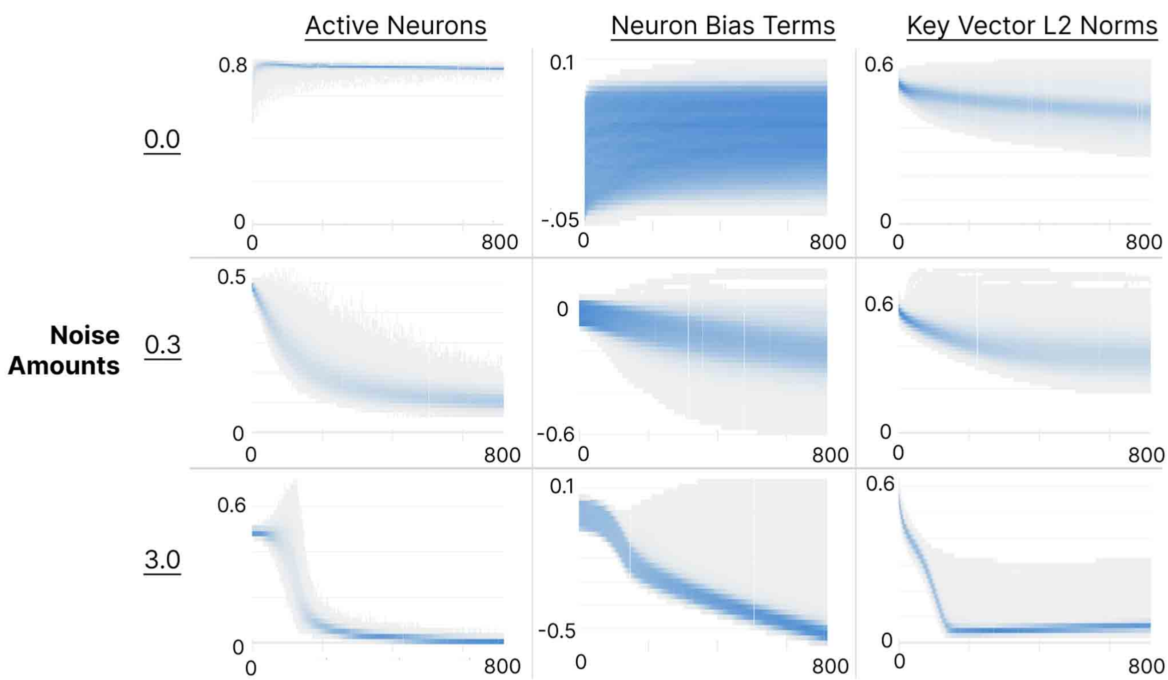


Figure 3.6: ***Noise induces the formation of sparse coding networks.*** *Shown column-wise are the fraction of neurons that are active for each training batch (Left), the values of each of the 10,000 bias terms (Middle) and the L_2 norms of the neuron encoder weights W_e (Right), all as a function of training epochs. Each is shown for three different representative noise levels (rows). These plots show each input to the network and use the density of blue to represent how many times a particular value occurs.*

scalar and L_2 norms of the weights also converge to a different scalar value (Fig. 3.6). This results in any given data point, producing activations that all fall within a particular range of values that is calibrated with the negative bias term such that only a sparse k neurons are active for any given input.

This is analogous to the operation of an inhibitory interneuron implementing a Top-k activation function. This dynamic transition during learning into a Top-k network also avoids complications associated with an explicit Top-k implementation (App. (Bricken *et al.*, 2023b) section L1 vs Noise) (Bricken *et al.*, 2023a; Makhzani and Frey, 2014a; Ahmad and Scheinkman, 2019).

Biological Receptive Fields - Looking closer at the sparse solutions found by our ReLU networks, we discovered that for $\sigma \geq 0.3$, neurons begin to form receptive fields reminiscent of V1 Gabor filters and center-surround retinal ganglion cells (Sterling and Laughlin, 2015; Olshausen and Field, 1997a). Figure 3.7 shows the receptive fields of the 125 neurons most activated (sorted from left-to-right and top-to-bottom) by a noisy image of a car when $\sigma = 0.8$. The longer training progressed, the more the receptive fields resembled Gabor-like and center-surround functions. See Appendix section Receptive Fields for more receptive fields at varying noise levels and over the course of training.

Relations to Sparse Coding - Training with the L_1 penalty and enforcing a unitary L_2 norm on all weights also resulted in biological receptive fields (Fig. **??**). The receptive fields were all Gabors or textures with the latter likely existing due to training on full CIFAR10 images instead of the typical smaller 8x8 patches (Olshausen and Field, 1997a; Karklin and Simoncelli, 2011). The L_2 weight normalization is crucial to obtaining biological receptive fields; otherwise, weights can change to avoid the activation penalty.[4] (Karklin and Simoncelli, 2011) showed, in agreement with our results, that an increase in noise causes a transition from Gabors to center-surround receptive fields. Both Gabors and center-surround fields capture aspects of natural image statistics. However, center-surround covers a smaller

[4]Interestingly, the L_1 penalty is not completely avoided and the network still sparsifies, however, it does not learn biological receptive fields.

portion of the image and we hypothesize that they are thus less sensitive to noise, resulting in their emergence with noisy training.

Another difference between noisy training and L_1 is how sparsity emerges. Noisy training uses the synchronization between its weight norms and negative bias terms without killing off neurons while L_1 uses negative neuron weights to kill off most of its neurons. However, for reasons outlined in Appendix section Dead Neurons Adam, we find that eventually our noisy training will also kill off many neurons when using the Adam optimizer. This means a different trajectory on the optimization landscape is followed but that it is still possible to kill many neurons without affecting performance. Finally, the use of a shared bias term combined with the ReLU activation function can be related to the lambda coefficient that scales the sparse coding L_1 activation penalty (see Appendix (Bricken *et al.*, 2023b) section Lambda Bias Relation).

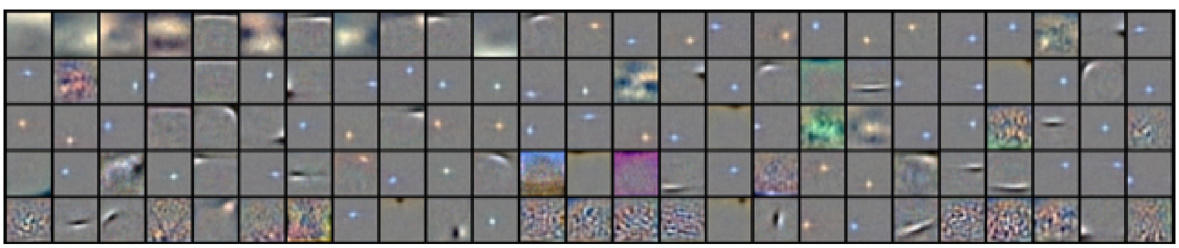


Figure 3.7: ***Biological receptive fields form with noise.*** *Networks trained on CIFAR10 pixels with* $\sigma = 0.8$ *Gaussian noise. The most active 125 neuron receptive fields are reshaped into 32x32x3 and re-scaled so their values span the pixel range* $[0, 255]$*. Neurons are sorted by activity levels (top left is most active, going across rows then down columns) for a randomly chosen car image. Starting at* $\sigma \geq 0.1$ *and particularly for* $\sigma = 0.8$ *(shown here) we observe both Gabor filters and center-surround receptive fields.*

Noisy Pre-Training Retains Model Sparsity - Investigating our noise-trained sparse networks, we find that even after removing noise, the networks remain sparse. We train our models in the same way as in Fig. 3.5 but at epoch 800 linearly anneal the noise to zero over the next 800 epochs. Figure 3.8 shows that the sparsity levels remain largely unperturbed.

For classification (Fig. 3.8, right), it is clear that while the correlation between noise and sparsity holds, it is not quite as strong. This is likely due to the fact that classification only needs to cluster the data into the ten CIFAR10 labels, meaning interference between neural representations is less of an issue and denser representations can be used.

There is also no reduction in reconstruction or classification performance for this single MLP setting.[5]). In fact, for classification both $\sigma \in \{0.1, 0.3\}$ get 94.3% validation accuracy instead of the baseline 93.4%, showing slightly better generalization while also being more sparse (66% for baseline versus 55% for $\sigma = 0.1$ and 45% for $\sigma = 0.3$, see Appendix (Bricken *et al.*, 2023b) section SingleMLPAblations). Note that networks with $\sigma \geq 0.8$ for reconstruction and $\sigma \geq 3.0$ for classification, annealing noise results in dead neurons, presumably because the volume of the data manifold shrinks, leaving some neurons behind. However, these dead neurons neither harm task performance nor entirely explain the sparsity levels. See Appendix (Bricken *et al.*, 2023b) section Dead Neurons Noise Annealing for data on the fractions of dead neurons.

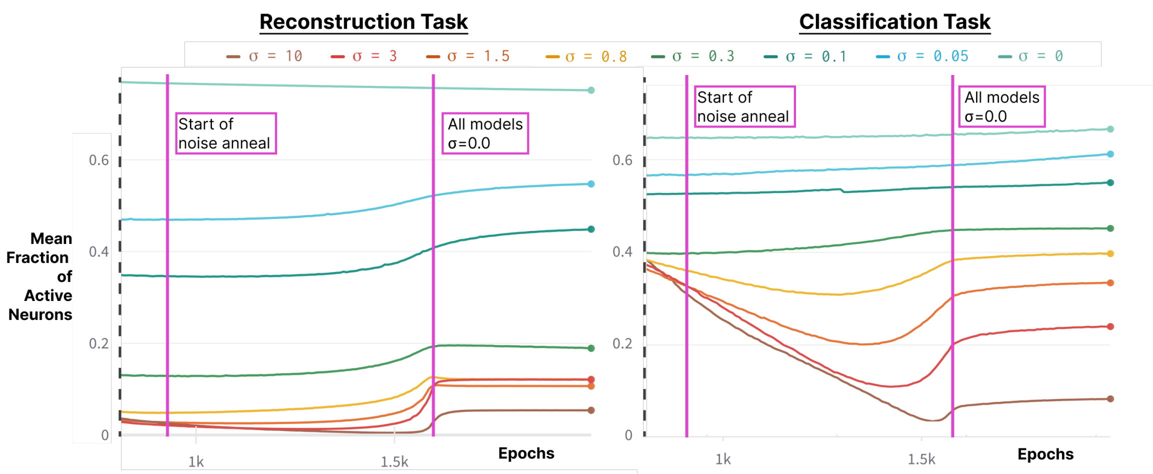


Figure 3.8: ***Noisy pretraining retains network sparsity.*** *The vertical purple lines denote the start and end of noise annealing (epochs 800 and 1,600) where each noise level is linearly decayed to $\sigma = 0$. For both reconstruction (left) and classification (right) the networks remain highly sparse even after noise is removed. Moreover, there is no noticeable performance difference between any of the networks and moderate noise models even perform slightly better than the noise free baseline. We truncate the x-axis to start at epoch 600 (dotted vertical gray line) for the sake of clarity. The classification networks continue to sparsify even when noise annealing has started only because, unlike in the reconstruction task, the sparsity level did not converge within the first 800 epochs.*

Model Ablations and Generalization - We investigate sparsity across experimental

[5]Deeper MLPs or Transformers also don't show performance costs. Only AlexNet does (Appendix (Bricken *et al.*, 2023b) section AlexNet

conditions in the single hidden layer setting and find that they all reproduce our results (Appendix (Bricken *et al.*, 2023b) section Single MLP Ablations): (i) datasets: CIFAR10 pixels, CIFAR10 latent embeddings, MNIST, CIFAR100, and SVHN; (ii) two noise distributions: Gaussian and Laplace; (iii) four numbers of neurons: 100, 1,000, 10,000 and 100,000; (iv) two training tasks: reconstruction and classification.

Additionally, we test multiple nonlinearities: sigmoid, GELU, ELU, and Tanh activation functions (Hendrycks and Gimpel, 2016). We find that the asymmetric ReLU, GELU, and ELU activation functions generally produce sparsity across datasets, albeit to slightly different extents while the symmetric sigmoid and Tanh do not (Appendix (Bricken *et al.*, 2023b) section Single MLP Ablations). This is consistent with our mathematical derivations where noise is minimized by driving pre-activations to near-zero gradient regions of the nonlinearity's domain that correspond to being "off" for the asymmetric ReLU, GELU, and ELU but being either very "on" or very "off" for the symmetric sigmoid or Tanh. On the CIFAR10 pixel dataset, none of the activation functions become as sparse as ReLU but the GELU networks do implement the same sparse coding convergence shown in Fig. 3.6 and form biologically plausible receptive fields like those of Fig. 3.7. It is important to note that ReLU is the only truly sparse activation function with the others requiring arbitrary activation thresholds close to zero that are used to label neurons as "off".

Optimizers and Dead Neurons - Exploring the effects of different optimizers, batch sizes, and learning rates produced a number of interesting findings. First, SGD – the only optimizer without an adaptive learning rate – was found to be capable of learning a non-sparse coding solution on the latent CIFAR10 dataset with $\sim$50% neuron activity and no updating of bias terms. This solution is only found on the easier latent CIFAR10 dataset whereas the CIFAR10 pixels dataset requires all models to sparsify. However, it is nevertheless interesting as it shows how the inductive biases of noise can be counteracted by a combination of a low learning rate and large batch size. We investigate this finding further in Appendix section SGD Learning Rate.

Second, Adam was found to produce dead neurons if we continued training beyond the discovery of the sparse coding solution without any effect on training or validation loss. The final number of dead neurons created corresponded to the complexity of the dataset with many more for the latent CIFAR10 dataset compared to the raw pixels. We hypothesize that neurons that are on less often are gradually killed off due to the "Stale Momentum" phenomenon. (Bricken *et al.*, 2023a) discovered that adaptive optimizers like Adam not only update inactive neurons using an out-of-date moving average but also create large gradient spikes when they are re-activated after periods of quiescence (see Appendix D of (Bricken *et al.*, 2023a) for an investigation of this phenomenon in greater detail).

Crucially, for our single-layer MLP results, the dead neurons appear after the sparse coding solution is found and not beforehand, which would be a misleading source of sparsity. However, this was not the case for some of the deeper models outlined in the next section where dead neurons appear early in training. Changing optimizer and increasing batch size removed dead neurons but still had no effect on training or validation loss (Appendix section Dead Neurons Adam).

Deeper Models - While our primary focus is on single-layer MLPs, we extend the work to Transformers, AlexNet, and three-layer MLPs.

Transformers - We train small GPT2 models (Radford *et al.*, 2019) with three blocks of interleaved Attention and Feedforward MLP layers on the WikiText-103 dataset for next token prediction.[6] In front of each MLP, we inject one of the noise levels $\sigma \in \{0.0, 0.05, 0.1, 0.8, 1.5, 3.0, 8.0\}$ to sparsify each of them. At training step 200k, we remove noise to evaluate the effects of noisy pretraining.

Sparsity increased with noise and is retained even after noise is removed. Furthermore, there are no dead neurons and aside from the highest noise $\sigma = 8$, pre-training has no significant effect on model performance (Appendix section Transformer). The $\sigma = 3$ noisy pretraining results in layers that are $\sim$3x more sparse in the first layer (17% for the baseline vs 6%), $\sim$9x for the second layer (18% vs 2%), and $\sim$3x (30% vs 13%) for the third layer.

[6] These Transformers use the HuggingFace implementation that includes LayerNorm and Residual connections.

AlexNet - Alexnet uses five layers of convolutions followed by three MLP layers. We trained it on raw CIFAR10 image classification for 3.5k epochs with noise $\sigma \in \{0.0, 1.5, 3.0, 5.0, 10.0\}$ applied for the first 1,000 epochs and linearly annealed to 0 over the next 500 epochs, leaving 2,000 epochs of noise-free training. Unlike the Transformer, where we apply noise in front of every MLP layer, we only apply noise to the input pixels.

Interestingly, upon removing the noise, the models recovered perfect training accuracy like the baseline but failed to recover validation accuracy, seeing ~20% reductions (82% baseline vs ~62%). The noisy models remained more sparse but this was misleadingly caused by dead neurons. We first hypothesized that these dead neurons harmed model capacity resulting in poor validation accuracy. However, taking steps to reduce dead neurons section Dead Neurons Adam still failed to improve validation accuracy (Appendix section AlexNet). We suspect that the convolutional layers are responsible and incompatible with noisy training but leave further investigation to future work.

Three-Layer MLP - Using 200 neurons per layer, we train for 2,000 epochs on classifying the latent 256 dimensional CIFAR10 embeddings with noise $\sigma \in \{0.0, 0.1, 0.5, 1.5, 3.0, 5.0, 10.0\}$. Noise annealing starts at epoch 500 and reaches $\sigma = 0$ by epoch 1,000. As in AlexNet, we only apply noise to the input layer. The networks showed sparsity in proportion to noise in the second and third layers but not the first.

Unlike with AlexNet but in line with the single-layer MLP and Transformer, once noise was removed, every noisy model saw equal or slightly improved (up to +1%) validation accuracy compared to the baseline (see Appendix section DeepMLP). However, like with AlexNet, the sparsity was created by dead neurons. Using SGD or SparseAdam as an optimizer solved this problem and allowed for neurons to implement the sparse coding solution.

Additional experiments injected noise only deeper inside the model, just in front of the second or third layers to see how this may affect the sparse coding solution. Interestingly, we found that this allowed the model to "cheat" the noise injection by making the L_2 norm of its weights and bias terms in the layers preceding the noise very large. This produced very large activation magnitudes that reduced the perturbative effect of noise.

3.4 Discussion

Our noise-free baselines show that neural networks do not typically choose to become sparse. So why does noisy training induce our network to become a biologically plausible sparse coding network? Fundamentally, the network should want to use as many neurons as necessary to give the best reconstruction of the input data as possible. However, it must trade-off the increase in accuracy from pooling more neurons with increased noise interference which favours sparsity. An increase in noise will create more erroneous low-level activations in neurons that must be ignored to maximize signal-to-noise ratio. Therefore, during noisy training, the model must learn weight vectors to encode information about the data distribution that can be separated from the noise using only ReLU functions. In natural images, for example, certain projections (e.g., with Gabor filters) have high kurtosis which allows them to be easily distinguished from noise via thresholding or "coring" (Simoncelli and Adelson, 1996). Similarly, the encoding layer in our network is learning such projections, so that thresholding performs noise rejection while also producing sparse activations.

It is interesting that the number of dead neurons does not have an effect on training or validation accuracy and appears only after the model performance and sparsity have fully converged. (Bricken *et al.*, 2023a) found Stale Momentum harmed continual learning but it may be a feature instead of a bug when training within a single data distribution by acting as a way to gradually prune less active, unnecessary neurons.

The positive correlation between noise and sparsity is opposite to Transformer Attention and associative memory models such as SDM and Hopfield Networks (Bricken and Pehlevan, 2021; Krotov and Hopfield, 2016). Because these models don't store patterns in a distributed fashion across neurons, they utilize a form of nearest neighbour lookup. In the low noise regime, where the target pattern is nearby, they use sparse activations to retrieve just the target. With more noise, they resort to averaging over many more neurons and patterns, losing accuracy but giving a solution in the correct neighbourhood (see App. (Bricken *et al.*, 2023b) section TransformersAndHopfieldNets).

Limitations - Regarding the biological plausibility of training with random noise, we acknowledge that some noise in biological systems is dynamic and correlated with neural activity. However, there is also uncorrelated random noise in any biological process. For example, Brownian noise that influences the diffusion of neurotransmitters (Sterling and Laughlin, 2015; Doi *et al.*, 2012). Moreover, for the inputs trained on directly on image pixels, Gaussian and Poisson noise are realistic for modelling photon noise on photoreceptors (Sterling and Laughlin, 2015). We also only provide qualitative comparisons to biological receptive fields rather than formally quantifying these, however, we restate that it is impressive these receptive fields emerge from noisy training alone.

Conclusion - We have shown that simple Gaussian noise results in three implicit loss terms that produce sparse and specialized receptive fields to capture high variance features in the data distribution. Empirical results support our theory and introduce a new approach to sparsify deep neural networks. The fact noise alone results in sparse coding without an explicit sparsity penalty in the loss function suggests that the primary driver behind sparse coding may be the handling of noise with metabolic efficiency as an added benefit, rather than the other way around. Combining noisy training with other approaches to induce sparsity, including taking additional ideas from sparse coding, may result in even higher degrees of sparsity with potential downstream benefits to robustness, continual learning, interpretability, and computational efficiency. More broadly, this work builds a new bridge between artificial and biological neural networks by showing how noise can make them more similar.

Chapter 4

Towards Monosemanticity: Decomposing Language Models With Dictionary Learning[1]

Using a sparse autoencoder, we extract a large number of interpretable features from a one-layer transformer.

This work was published as part of Anthropic's Transformer Circuits Thread and underwent independent review and replication.

Author Contributions

- Infrastructure, Tooling, and Core Algorithmic Work
  - General Infrastructure – The basic framework for our dictionary learning work was built and maintained by Adly Templeton, Trenton Bricken, Tom Henighan, and Tristan Hume. Adly Templeton, in collaboration with Trenton Bricken and Tom Conerly, performed the engineering to scale up our experiments to large

[1]Work done while at Anthropic, who retain all rights to it.

numbers of tokens. Robert Lasenby created tooling that allowed us to train extremely small language models. Yifan Wu imported the "Pile" dataset so we could train models on a standard external dataset. Tom Conerly broadly assisted with our infrastructure and maintaining high code quality. Adly Templeton implemented automatic plots comparing different experiments in a scan.

  - Sparse Autoencoders (Algorithms / ML) – Trenton initially implemented and advocated for sparse autoencoders as an approach to dictionary learning. Trenton Bricken and Adly Templeton then collaborated on the research needed to achieve our results, including scanning hyperparameters, iterating on algorithms, introducing the "neuron resampling" method, and characterizing the importance of dataset scale. (It's hard for the other authors to communicate just how much work Trenton Bricken and Adly Templeton put into iterating on algorithms and hyperparameters.) Josh Batson assisted by analyzing runs and designing metrics with Chris Olah, Adly Templeton and Trenton Bricken to measure success.
  - Analysis Infrastructure – The tooling to collect data for behind the visualizations was created by Trenton Bricken and Adly Templeton. Tom Conerly greatly accelerated this. Adly Templeton implemented the tooling to perform large scale feature ablation analysis, as well as creating the shuffled weights models as baselines.
  - Interface – The interface for visualizing and exploring features was created by Brian Chen, with support from Shan Carter. It replaced a much earlier version by Trenton Bricken.

- Analysis
  - Feature Deep Dives – The dataset examples and ablations came from Brian Chen's interface work and Adly Templeton's ablation infrastructure. Chris Olah developed the activation spectrum plots, broken down by proxy. Josh Batson developed the logit scatter plot visualization and the sensitivity analysis. Tom

Henighan created the pinned sampling visualizations. Nick Turner, Josh Batson and Tom Henighan explored how best to analyze neuron alignment. Many people contributed to a push to systematically analyze many features in detail, including Tom Henighan, Josh Batson, Trenton Bricken, Adly Templeton, Nick Turner, Brian Chen, and Chris Olah.

  - Manual Analysis of Features – Nick Turner and Adam Jermyn created our rubric for evaluating if a feature interval is interpretable. Adam Jermyn manually scored a random set of features.
  - Automated Interpretability – Our automated interpretability pipeline was created by Trenton Bricken, Cem Anil, Carson Denison, and Amanda Askell. Trenton Bricken ran the experiments using it to evaluate our features. This built on earlier explorations of automated interpretability by Shauna Kravec, and was only possible due to infrastructure contributions by Tim Maxwell, Nicholas Schiefer, and Nicholas Joseph.
  - Feature Splitting – Josh Batson, Adam Jermyn, and Chris Olah analyzed feature splitting. Shan Carter produced the UMAPs of features.
  - Universality – Josh Batson analyzed universality, with engineering support from Trenton Bricken and Tom Henighan.
  - "Finite State Automata" – Chris Olah analyzed "Finite State Automata".

- Paper Production
  - Writing – The manuscript was primarily drafted by Chris Olah, Josh Batson, and Adam Jermyn, with extensive feedback and editing from all other authors. The detailed appendix on dictionary learning was drafted by Adly Templeton and Trenton Bricken. Josh Batson and Chris Olah managed the revision process in response to internal and external feedback.
  - Illustration – Diagrams were created by Chris Olah, Josh Batson, Adam Jermyn, and Shan Carter, based on data generated by themselves and others.

- Development of Intuition & Negative Results
  - Early Dictionary Learning Experiments – Many of the authors (including Trenton Bricken, Adly Templeton, Tristan Hume, Tom Henighan, Adam Jermyn, Josh Batson, and Chris Olah) did experiments applying both more traditional dictionary learning methods and sparse autoencoders to a variety of toy problems and small MNIST models. Much of this work focused on finding metrics which we hoped would tell us whether dictionary learning had succeeded in a simple context. These experiments, along with theoretical work, built valuable intuition and helped us discover several algorithmic improvements, and clarified challenges with traditional dictionary learning.
  - Sparse Architectures – Brian Chen ran the experiments and generated the counterexamples which persuaded us that training models for sparse activations was less promising. Crucially, he produced models where neurons typically activated in isolation, and showed they were still polysemantic. He then produced the counterexample described in the text. Robert Lasenby implemented infrastructure that made it easier to experiment with sparse activation architectures.
- Other
  - Support - Alex Tamkin, Karina Nguyen, Brayden McLean, and Josiah E Burke made a variety of contributions through infrastructure, operational support, labeling features, and commenting on the draft.
  - Leadership - Tom Henighan led the dictionary learning project. Tristan Hume led an early version of the project, before handing it over to Tom Henighan. Shan Carter managed the overall team. Chris Olah provided general research guidance.

4.1 Introduction

Mechanistic interpretability seeks to understand neural networks by breaking them into components that are more easily understood than the whole. By understanding the function of each component, and how they interact, we hope to be able to reason about the behavior of the entire network. The first step in that program is to identify the correct components to analyze.

Unfortunately, the most natural computational unit of the neural network – the neuron itself – turns out not to be a natural unit for human understanding. This is because many neurons are polysemantic: they respond to mixtures of seemingly unrelated inputs. In the vision model Inception v1, a single neuron responds to faces of cats and fronts of cars (Olah *et al.*, 2017). In a small language model we discuss in this paper, a single neuron responds to a mixture of academic citations, English dialogue, HTTP requests, and Korean text. Polysemanticity makes it difficult to reason about the behavior of the network in terms of the activity of individual neurons.

One potential cause of polysemanticity is superposition (Arora *et al.*, 2018; Goh, 2016; Olah *et al.*, 2020b; Elhage *et al.*, 2022), a hypothesized phenomenon where a neural network represents more independent "features" of the data than it has neurons by assigning each feature its own linear combination of neurons. If we view each feature as a vector over the neurons, then the set of features form an overcomplete linear basis for the activations of the network neurons. In our previous paper on Toy Models of Superposition (Elhage *et al.*, 2022), we showed that superposition can arise naturally during the course of neural network training if the set of features useful to a model are sparse in the training data. As in compressed sensing, sparsity allows a model to disambiguate which combination of features produced any given activation vector.[2]

In Toy Models of Superposition, we described three strategies to finding a sparse and interpretable set of features if they are indeed hidden by superposition: (1) creating models

[2]For more discussion of this point, see Distributed Representations: Composition and Superposition.

without superposition, perhaps by encouraging activation sparsity; (2) using dictionary learning to find an overcomplete feature basis in a model exhibiting superposition; and (3) hybrid approaches relying on a combination of the two. Since the publication of that work, we've explored all three approaches. We eventually developed counterexamples which persuaded us that the sparse architectural approach (approach 1) was insufficient to prevent polysemanticity, and that standard dictionary learning methods (approach 2) had significant issues with overfitting.

In this paper, we use a weak dictionary learning algorithm called a sparse autoencoder to generate learned features from a trained model that offer a more monosemantic unit of analysis than the model's neurons themselves. Our approach here builds on a significant amount of prior work, especially in using dictionary learning and related methods on neural network activations (e.g. (Faruqui *et al.*, 2015; Arora *et al.*, 2018; Subramanian *et al.*, 2018; Zhang *et al.*, 2019; Panigrahi *et al.*, 2019; Yun *et al.*, 2021)), and a more general allied literature on disentanglement. We also note interim reports (Sharkey *et al.*, 2022; Cunningham and Smith, 2023; Huben, 2023; Smith, 2023b,a; Cunningham, 2023) which independently investigated the sparse autoencoder approach in response to Toy Models, culminating in the recent manuscript of Cunningham *et al.* (2023).

The goal of this paper is to provide a detailed demonstration of a sparse autoencoder compellingly succeeding at the goals of extracting interpretable features from superposition and enabling basic circuit analysis. Concretely, we take a one-layer transformer with a 512-neuron MLP layer, and decompose the MLP activations into relatively interpretable features by training sparse autoencoders on MLP activations from 8 billion data points, with expansion factors ranging from 1× (512 features) to 256× (131,072 features). We focus our detailed interpretability analyses on the 4,096 features learned in one run we call A/1.

This report has four major sections. In Problem Setup, we provide motivation for our approach and describe the transformers and sparse autoencoders we train. In Detailed Investigations of Individual Features, we offer an existence proof – we make the case that several features we find are functionally specific causal units which don't correspond to neurons. In Global Analysis, we argue that the typical feature is interpretable and that they explain a non-trivial portion of the MLP layer. Finally, in Phenomenology we describe several properties of our features, including feature-splitting, universality, and how they can form "finite state automata"-like systems implementing interesting behaviors.

We also provide three comprehensive visualizations of features. First, for all features from 90 learned dictionaries we present activating dataset examples and downstream logit effects. We recommend the reader begin with the visualization of A/1. Second, we provide a data-oriented view, showing all features active on each token of 25 texts. Finally, we coembed all 4,096 features from A/1 and all 512 features from A/0 into the plane using UMAP to allow for interactive exploration of the space of features:

4.2 Summary of Results

- Sparse Autoencoders extract relatively monosemantic features. We provide four different lines of evidence: detailed investigations for a few features firing in specific contexts for which we can construct computational proxies, human analysis for a large random sample of features, automated interpretability analysis of activations for all the features learned by the autoencoder, and finally automated interpretability analysis of logit weights for all the features. Moreover, the last three analyses show that most learned features are interpretable. While we do not claim that our interpretations catch all aspects of features' behaviors, by constructing metrics of interpretability consistently for features and neurons, we quantitatively show their relative interpretability.

- Sparse autoencoders produce interpretable features that are effectively invisible in the neuron basis. We find features (e.g., one firing on Hebrew script) which are not active in any of the top dataset examples for any of the neurons.

- Sparse autoencoder features can be used to intervene on and steer transformer generation. For example, activating the base64 feature we study causes the model to generate base64 text, and activating the Arabic script feature we study produces Arabic text.

(See discussion of pinned feature sampling in Global Analysis.)

- Sparse autoencoders produce relatively universal features. Sparse autoencoders applied to different transformer language models produce mostly similar features, more similar to one another than they are to their own model's neurons. (See Universality)

- Features appear to "split" as we increase autoencoder size. When we gradually increase the width of the autoencoder from 512 (the number of neurons) to over 131,000 (256×), we find features which naturally fit together into families. For example, one base64 feature in a small dictionary splits into three, with more subtle and yet still interpretable roles, in a larger dictionary. The different size autoencoders offer different "resolutions" for understanding the same object. (See Feature Splitting.)

- Just 512 neurons can represent tens of thousands of features. Despite the MLP layer being very small, we continue to find new features as we scale the sparse autoencoder. Features connect in "finite-state automata"-like systems that implement complex behaviors. For example, we find features that work together to generate valid HTML. (See "Finite State Automata".)

4.3 Problem Setup

A key challenge to our agenda of reverse engineering neural networks is the curse of dimensionality: as we study ever-larger models, the volume of the latent space representing the model's internal state that we need to interpret grows exponentially. We do not currently see a way to understand, search or enumerate such a space unless it can be decomposed into independent components, each of which we can understand on its own.

In certain limited cases, it is possible to side step these issues by rewriting neural networks in ways that don't make reference to certain hidden states. For example, in A Mathematical Framework for Transformer Circuits (Elhage *et al.*, 2021), we were able to analyze a one-layer attention-only network without addressing this problem. But this approach becomes impossible if we consider even a simple standard one-layer transformer with an MLP layer with a ReLU activation function. Understanding such a model requires us to have a way to decompose the MLP layer.

In some sense, this is the simplest language model we profoundly don't understand. And so it makes a natural target for our paper. We aim to take its MLP activations – the activations we can't avoid needing to decompose – and decompose them into "features" as shown in Fig. 4.1:

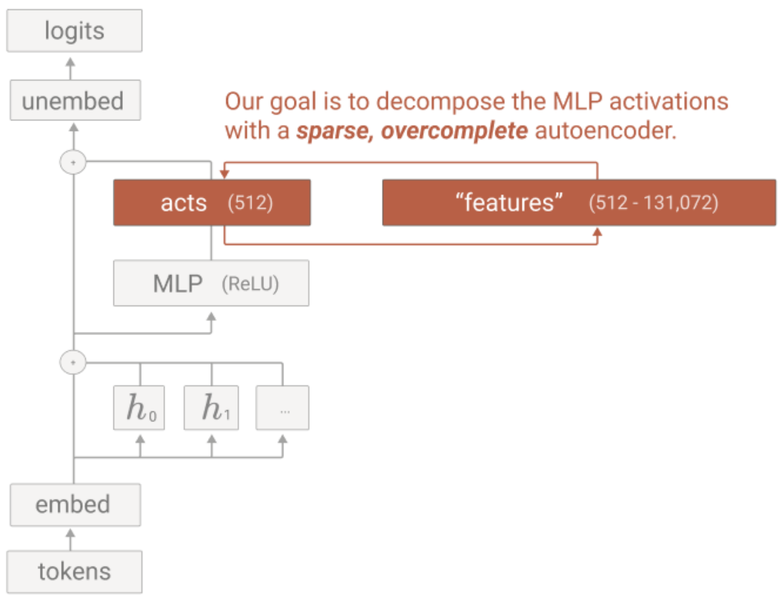


Figure 4.1: ***The basic architecture of the Transformer and where we learn features using our Sparse AutoEncoder (SAE).***

Crucially, we decompose into more features than there are neurons. This is because we believe that the MLP layer likely uses superposition (Arora *et al.*, 2018; Goh, 2016; Olah *et al.*, 2020b; Elhage *et al.*, 2022) to represent more features than it has neurons (and correspondingly do more useful non-linear work!). In fact, in our largest experiments we'll expand to have 256 times more features than neurons, although we'll primarily focus on a more modest 8× expansion.

In the following subsections, we will motivate this setup at more length. Additionally, a more detailed discussion of the architectural details and training of these models can be found in the appendix (Bricken *et al.*, 2023c).

Table 4.1: *Comparison of Transformer and Sparse Autoencoder*

	Transformer	**Sparse Autoencoder**
Layers	1 Attention Block 1 MLP Block (ReLU)	1 ReLU (up) 1 Linear (down)
MLP Size	512	512 (1×) – 131,072 (256×)
Dataset	The Pile Gao *et al.* (2020) (100 billion tokens)	Transformer MLP Activations (8 billion samples)
Loss	Autoregressive Log-Likelihood	L2 reconstruction + L1 on hidden layer activation

4.3.1 Features as a Decomposition

There is significant empirical evidence suggesting that neural networks have interpretable linear directions in activation space. This includes classic work by Mikolov *et al.* (2013) investigating "vector arithmetic" in word embeddings (but see (Levy and Goldberg, 2014)) and similar results in other latent spaces (e.g. (Radford *et al.*, 2015)). There is also a large body of work investigating interpretable neurons, which are just basis dimensions (e.g. in RNNs (Karpathy *et al.*, 2015; Radford *et al.*, 2017); in CNNs (Cammarata *et al.*, 2020; Zhou *et al.*, 2014; Bau *et al.*, 2017); in GANs (Bau *et al.*, 2020); but see (Morcos *et al.*, 2018; Donnelly and Roegiest, 2019)). A longer review and discussion of this work can be found in the Motivation section of Toy Models (Elhage *et al.*, 2022), although it doesn't cover more recent work (e.g. (Nanda *et al.*, 2023; Burns *et al.*, 2022; McGrath *et al.*, 2022)).[3]

If linear directions are interpretable, it's natural to think there's some "basic set" of meaningful directions which more complex directions can be created from. We call these directions features, and they're what we'd like to decompose models into. Sometimes, by happy circumstances, individual neurons appear to be these basic interpretable units (see examples above). But quite often, this isn't the case.

Instead, we decompose the activation vector $\mathbf{x}^j$ as a combination of more general features

[3] We'd particularly highlight an exciting exchange between (Li *et al.*, 2022) and (Nanda *et al.*, 2023): Li et al. found an apparent counterexample where features were not represented as directions, which was the resolved by Nanda finding an alternative interpretation in which features were directions.

which can be any direction:

$$\mathbf{x}^j \approx \mathbf{b} + \sum_i f_i(\mathbf{x}^j)\mathbf{d}_i \tag{4.1}$$

where $\mathbf{x}^j$ is the activation vector of length d_{MLP} for datapoint j, $f_i(\mathbf{x}^j)$ is the activation of feature i, each $\mathbf{d}_i$ is a unit vector in activation space we call the direction of feature i, and $\mathbf{b}$ is a bias.[4] Note that this decomposition is not new: it is just a linear matrix factorization of the kind commonly employed in dictionary learning.

In our sparse autoencoder setup, the feature activations are the output of the encoder

$$f_i(x) = \text{ReLU}(W_e(\mathbf{x} - \mathbf{b}_d) + \mathbf{b}_e)_i,$$

where W_e is the weight matrix of the encoder and $\mathbf{b}_d$, $\mathbf{b}_e$ are a pre-encoder and an encoder bias. The feature directions are the columns of the decoder weight matrix W_d. (See appendix for full notation (Bricken *et al.*, 2023c).)

If such a sparse decomposition exists, it raises an important question: are models in some fundamental sense composed of features or are features just a convenient post-hoc description? In this paper, we take an agnostic position, though our results on feature universality suggest that features have some existence beyond individual models.

Superposition Hypothesis

To see how this decomposition relates to superposition, recall that the superposition hypothesis postulates that neural networks "want to represent more features than they have neurons". We think this happens via a kind of "noisy simulation", where small neural networks exploit feature sparsity and properties of high-dimensional spaces to approximately simulate much larger much sparser neural networks (Fig 4.2) (Elhage *et al.*, 2022).

[4]While not the focus of this paper, we could also imagine decomposing other portions of the model's latent state or activations in this way. For instance, we could apply this decomposition to the residual stream (see e.g. (Yun *et al.*, 2021; Smith, 2023a)), or to the keys and queries of attention heads, or to their outputs.

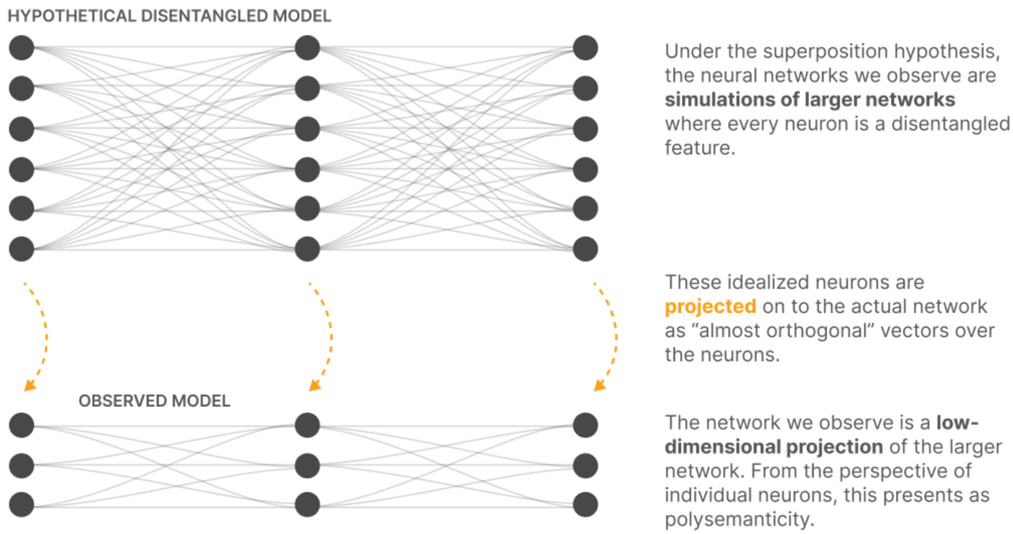


Figure 4.2: ***Superposition.***

A consequence of this is that we should expect the feature directions to form an overcomplete basis. That is, our decomposition should have more directions $\mathbf{d}_i$ than neurons. Moreover, the feature activations should be sparse, because sparsity is what enables this kind of noisy simulation. This is mathematically identical to the classic problem of dictionary learning.

4.3.2 What makes a good decomposition?

Suppose that a dictionary exists such that the MLP activation of each datapoint is in fact well approximated by a sparse weighted sum of features as in equation 1. That decomposition will be useful for interpreting the neural network if: We can interpret the conditions under which each feature is active. That is, we have a description of which datapoints j cause the feature to activate (i.e. for which $f_i(\mathbf{x}^j)$ is large) that makes sense on a diverse dataset (including potentially synthetic or off-distribution examples meant to test the hypothesis).[5] We can interpret the downstream effects of each feature, i.e. the effect of changing the value of f_i on subsequent layers. This should be consistent with the interpretation in (1). The

[5]This property is closely related to the desiderata of Causality, Generality, and Purity discussed in Cammarata *et al.* (2020), and those provide an example of how we might make this property concrete in a specific instance.

features explain a significant portion of the functionality of the MLP layer (as measured by the loss; see How much of the model does our interpretation explain?).

A feature decomposition satisfying these criteria would allow us to:

Determine the contribution of a feature to the layer's output, and the next layer's activations, on a specific example. Monitor the network for the activation of a specific feature (see e.g. speculation about safety-relevant features). Change the behavior of the network in predictable ways by changing the values of some features. In multilayer models this could look like predictably influencing one layer by changing the feature activations in an earlier layer. Demonstrate that the network has learned certain properties of the data. Demonstrate that the network is using a given property of the data in producing its output on a specific example.[6] Design inputs meant to activate a given feature and elicit certain outputs.

Of course, decomposing models into components is just the beginning of the work of mechanistic interpretability! It provides a foothold on the inner workings of models, allowing us to start in earnest on the task of unraveling circuits and building a larger-scale understanding of models.

4.3.3 Why not use architectural approaches?

In Toy Models of Superposition (Elhage *et al.*, 2022), we highlighted several different approaches to solving superposition. One of those approaches was to engineer models to simply not have superposition in the first place.

Initially, we thought that this might be possible but come with a large performance hit (i.e. produce models with greater loss). Even if this performance hit had been too large to use in practice for real models, we felt that success at creating monosemantic models would have been very useful for research, and in a lot of ways this felt like the "cleanest" approach for downstream analysis.

Unfortunately, having spent a significant amount of time investigating this approach, we

[6] This is similar in spirit to the evidence provided by influence functions.

have ultimately concluded that it is more fundamentally non-viable.

In particular, we made several attempts to induce activation sparsity during training to produce models without superposition, even to the point of training models with 1-hot activations. This indeed eliminates superposition, but it fails to result in cleanly-interpretable neurons! Specifically, we found that individual neurons can be polysemantic even in the absence of superposition. This is because in many cases models achieve lower loss by representing multiple features ambiguously (in a polysemantic neuron) than by representing a single feature unambiguously and ignoring the others.

To understand this, consider a toy model with a single neuron trained on a dataset with four mutually-exclusive features (A/B/C/D), each of which makes a distinct (correct) prediction for the next token, labeled in the same fashion. Further suppose that this neuron's output is binary: it either fires or it doesn't. When it fires, it produces an output vector representing the probabilities of the different possible next tokens.

We can calculate the cross-entropy loss achieved by this model in a few cases: Suppose the neuron only fires on feature A, and correctly predicts token A when it does. The model ignores all of the other features, predicting a uniform distribution over tokens B/C/D when feature A is not present. In this case the loss is $\frac{3}{4} \ln 3 \approx 0.8$. Instead suppose that the neuron fires on both features A and B, predicting a uniform distribution over the A and B tokens. When the A and B features are not present, the model predicts a uniform distribution over the C and D tokens. In this case the loss is $\ln 2 \approx 0.7$.

Because the loss is lower in case (2) than in case (1), the model achieves better performance by making its sole neuron polysemantic, even though there is no superposition.

This example might initially seem uninteresting because it only involves one neuron, but it actually points at a general issue with highly sparse networks. If we push activation sparsity to its limit, only a single neuron will activate at a time. We can now consider that single neuron and the cases where it fires. As seen earlier, it can still be advantageous for that neuron to be polysemantic.

Based on this reasoning, and the results of our experiments, we believe that models trained on cross-entropy loss will generally prefer to represent more features polysemantically than to represent fewer "true features" monosemantically, even in cases where sparsity constraints make superposition impossible.

Models trained on other loss functions do not necessarily suffer this problem. For instance, models trained under mean squared error loss (MSE) may achieve the same loss for both polysemantic and monosemantic representations (e.g. (Jermyn *et al.*, 2022)), and for some feature importance curves they may actively prefer the monosemantic solution (Scherlis *et al.*, 2022). But language models are not trained with MSE, and so we do not believe architectural changes can be used to create fully monosemantic language models.

Note, however, that in learning to decompose models post-training we do use an MSE loss (between the activations and their representation in terms of the dictionary), so sparsity can inhibit superposition from forming in the learned dictionary. (Otherwise, we might have superposition "all the way down.")

4.3.4 Using Sparse Autoencoders To Find Good Decompositions

There is a long-standing hypothesis that many natural latent variables in the world are sparse (see (Olshausen and Field, 1997b), section "Why Sparseness?"). Although we have a limited understanding of the features that exist in language models, the examples we do have (e.g. (Nelson *et al.*, 2022)) are suggestive of highly sparse variables. Our work on Toy Models of Superposition (Elhage *et al.*, 2022) shows that a large set of sparse features could be represented in terms of directions in a lower dimensional space.

For this reason, we seek a decomposition which is sparse and overcomplete. This is essentially the problem of sparse dictionary learning(Elad, 2010; Olshausen and Field, 1997b). (Note that, since we're only asking for a sparse decomposition of the activations, we make no use of the downstream effects of $\mathbf{d}_i$ on the model's output. This means we'll be able to use those downstream effects in later sections as a kind of validation on the features found.)

It's important to understand why making the problem overcomplete – which might initially sound like a trivial change – actually makes this setting very different from similar approaches seeking sparse disentanglement in the literature. It's closely connected to why dictionary learning is such a non-trivial operation; in fact, as we'll see, it's actually kind of miraculous that this is possible at all. At the heart of dictionary learning is an inner problem of computing the feature activations $f_i(\mathbf{x})$ for each datapoint $\mathbf{x}$, given the feature directions $\mathbf{d}_i$. On its surface, this problem may seem impossible: we're asking to determine a high-dimensional vector from a low-dimensional projection. Put another way, we're trying to invert a very rectangular matrix. The only thing which makes it possible is that we are looking for a high-dimensional vector that is sparse! This is the famous and well-studied problem of compressed sensing, which is NP-hard in its exact form. It is possible to store high-dimensional sparse structure in lower-dimensional spaces, but recovering it is hard.

Despite its difficulty, there are a host of sophisticated methods for dictionary learning (e.g. (Engan *et al.*, 1999; Aharon *et al.*, 2006)). These methods typically involve optimizing a relaxed problem or doing a greedy search. We tried several of these classic methods, but ultimately decided to focus on a simpler sparse autoencoder approximation of dictionary learning (similar to Sharkey *et al.* (2022)). This was for two reasons. First, a sparse autoencoder can readily scale to very large datasets, which we believe is necessary to characterize the features present in a model trained on a large and diverse corpus.[7] Secondly, we have a concern that iterative dictionary learning methods might be "too strong", in the sense of being able to recover features from the activations which the model itself cannot access. Exact compressed sensing is NP-hard, which the neural network certainly isn't doing. By contrast, a sparse autoencoder is very similar in architecture to the MLP layers in language models, and so should be similarly powerful in its ability to recover features from superposition.

4.3.5 Sparse Autoencoder Setup

We briefly overview the architecture and training of our sparse autoencoder here, and provide further details in Basic Autoencoder Training. Our sparse autoencoder is a model with a bias at the input, a linear layer with bias and ReLU for the encoder, and then another

[7]For the model discussed in this paper, we trained the autoencoder on 8 billion datapoints.

linear layer and bias for the decoder. In toy models we found that the bias terms were quite important to the autoencoder's performance.[8]

We train this autoencoder using the Adam optimizer to reconstruct the MLP activations of our transformer model, with an MSE[9] loss plus an L1 penalty to encourage sparsity.

In training the autoencoder, we found a couple of principles to be quite important. First, scale really matters. We found that training the autoencoder on more data made features subjectively "sharper" and more interpretable. In the end, we decided to use 8 billion training points for the autoencoder (see Autoencoder Dataset).

Second, we found that over the course of training some neurons cease to activate, even across a large number of datapoints. We found that "resampling" these dead neurons during training gave better results by allowing the model to represent more features for a given autoencoder hidden layer dimension. Our resampling procedure is detailed in Neuron Resampling, but in brief we periodically check for neurons which have not fired in a significant number of steps and reset the encoder weights on the dead neurons to match data points that the autoencoder does not currently represent well.

For readers looking to apply this approach, we supply an appendix with Advice for Training Sparse Autoencoders (Bricken *et al.*, 2023c).

How can we tell if the autoencoder is working? Usually in machine learning we can quite easily tell if a method is working by looking at an easily-measured quantity like the test loss. We spent quite some time searching for an equivalent metric to guide our efforts here, and unfortunately have yet to find anything satisfactory.

We began by looking for an information-based metric, so that we could say in some sense that the best factorization is the one that minimizes the total information of the autoencoder and the data. Unfortunately, this total information did not generally correlate with subjective feature interpretability or activation sparsity. (Runs whose feature activations

[8]We tie the biases applied in the input and output, so the result is equivalent to subtracting a fixed bias from all activations and then using an autoencoder whose only bias is before the encoder activation.

[9]Note that using an MSE loss avoids the challenges with polysemanticity that we discussed above in Why Not Architectural Approaches?

had an average L0 norm in the hundreds but low reconstruction error could have lower total information than those with smaller average L0 norm and higher reconstruction error.)

Thus we ended up using a combination of several additional metrics to guide our investigations:

Manual inspection: Do the features seem interpretable? Feature density: we found that the number of “live” features and the percentage of tokens on which they fire to be an extremely useful guide. (See appendix for details (Bricken *et al.*, 2023c).) Reconstruction loss: How well does the autoencoder reconstruct the MLP activations? Our goal is ultimately to explain the function of the MLP layer, so the MSE loss should be low. Toy models: Having toy models where we know the ground truth and so can cleanly evaluate the autoencoder’s performance was crucial to our early progress.

Interpreting or measuring some of these signals can be difficult, though. For instance, at various points we thought we saw features which at first didn’t make any sense, but with deeper inspection we could understand. Likewise, while we have identified some desiderata for the distribution of feature densities, there is much that we still do not understand and which prevents this from providing a clear signal of progress.

We think it would be very helpful if we could identify better metrics for dictionary learning solutions from sparse autoencoders trained on transformers.

4.3.6 The (one-layer) model we’re studying

We chose to study a one-layer transformer model. We view this model as a testbed for dictionary learning, and in that role it brings three key advantages: Because one-layer models are small they likely have fewer "true features" than larger models, meaning features learned by smaller dictionaries might cover all their "true features". Smaller dictionaries are cheaper to train, allowing for fast hyperparameter optimization and experimentation. We can highly overtrain a one-layer transformer quite cheaply. We hypothesize that a very high number of training tokens may allow our model to learn cleaner representations in superposition. We can easily analyze the effects features have on the logit outputs, because these are approximately linear in the feature activations.[10] This provides helpful corroboration that the features we have found are not just telling us about the data distribution, and actually reflect the functionality of the model.

We trained two one-layer transformers with the same hyperparameters and datasets, differing only in the random seed used for initialization. We then learned dictionaries of many different sizes on both transformers, using the same hyperparameters for each matched pair of dictionaries but training on the activations of different tokens for each transformer.

We refer to the main transformer we study in this paper as the "A" transformer. We primarily use the other transformer ("B") to study feature universality, as we can e.g. compare features learned from the "A" and "B" transformers and see how similar they are.

4.3.7 Notation for Features

Throughout this draft, we'll use strings like "A/1/2357" to denote features. The first portion "A" or "B" denote which model the features come from. The second part (e.g. the "1" in "A/1") denotes the dictionary learning run. These vary in the number of learned factors and the L1 coefficient used. A table of all of our runs is available here. Notably, A/0. . . A/5 form a sequence with fixed L1 coefficients and increasing dictionary sizes. The final portion (e.g. the "2357" in "A/1/2357") corresponds to the specific feature in the run.

Sometimes, we want to denote neurons from the transformer rather than features learned by the sparse autoencoder. In this case, we use the notation "A/neurons/32".

[10]Note that this linear structure makes it even more likely that features should be linear. On the one hand, this means that the linear representation hypothesis is more likely to hold for this model. On the other hand, it potentially means that our results are less likely to generalize to multilayer models. Fortunately, others have studied multilayer transformers with sparse autoencoders and found interpretable linear features, which gives us more confidence that what we see in the one-layer model indeed generalizes (Smith, 2023b,a; Cunningham, 2023).

4.3.8 Interface for Exploring Features

We provide an interface for exploring all the features in all our dictionary learning runs. Links to the visualizations for each run can be found here. We suggest beginning with the interface for A/1, which we discuss the most.

These interfaces provide extensive information on each feature. This includes examples of when they activate, what effect they have on the logits when they do, examples of how they affect the probability of tokens if the feature is ablated, and much more as shown in 4.3:

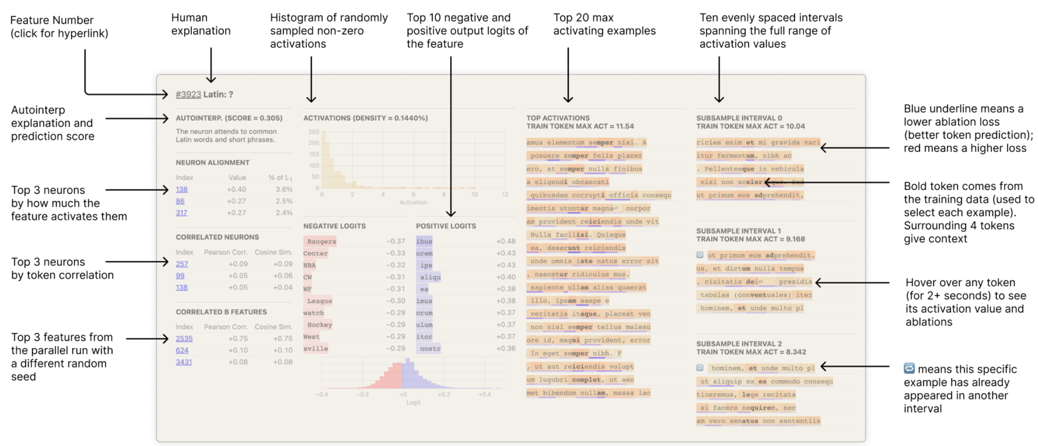


Figure 4.3: *The Feature Visualization Interface.*

Our interface also allows users to search through features (Fig. 4.4):

Additionally, we provide a second interface displaying all features active on a given dataset example in Fig. 4.5. This is available for a set of example texts.

4.4 Detailed Investigations of Individual Features

The most important claim of our paper is that dictionary learning can extract features that are significantly more monosemantic than neurons. In this section, we give a detailed demonstration of this claim for a small number of features which activate in highly specific contexts.

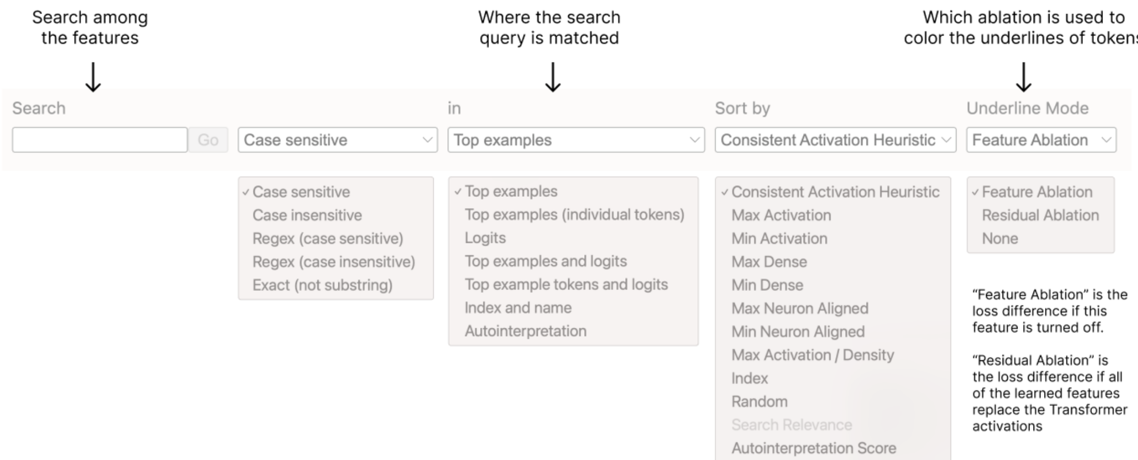


Figure 4.4: *How to Search and Filter for Features.*

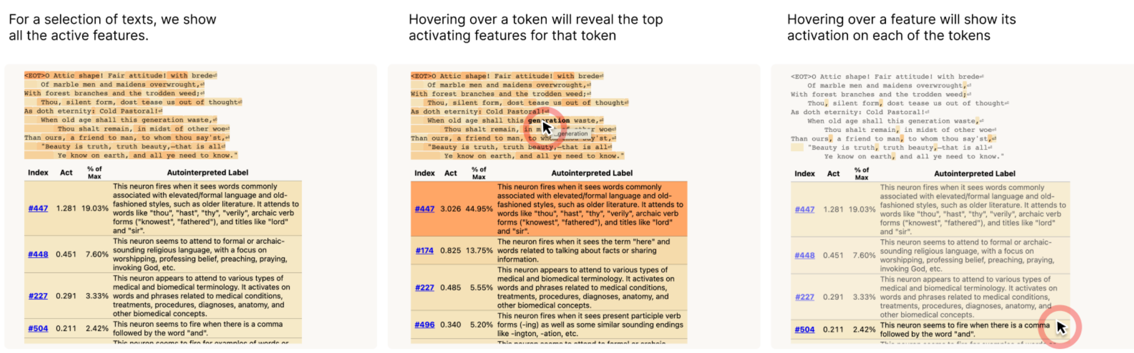


Figure 4.5: *Features active across a text example and per token.*

The features we study respond to:

- Text written in Arabic script
- DNA sequences (like `CCTGGTACTGTACG`)
- base64 strings (like the final characters in `https://www.youtube.com/watch?v=dQw4w9WgXcQ`)
- Text written in Hebrew script

For each learned feature, we attempt to establish the following claims:

1. The learned feature activates with high specificity for the hypothesized context. (When the feature is on the context is usually present.)
2. The learned feature activates with high sensitivity for the hypothesized context. (When the context is present, the feature is usually on.)
3. The learned feature causes appropriate downstream behavior.
4. The learned feature does not correspond to any neuron.
5. The learned feature is universal – a similar feature is found by dictionary learning applied to a different model.

To demonstrate claims 1–3, we devise computational proxies for each context, numerical scores estimating the (log-)likelihood that a string (or token) is from the specific context. The contexts chosen above are easy to model based on the defined sets of unicode characters involved. We model DNA sequences as random strings of characters from `[ATCG]` and we model base64 strings as random sequences of characters from `[a-zA-Z0-9+/]`. For Arabic script and Hebrew features, we exploit the fact that each language is written in a script consisting of well-defined Unicode blocks. Each computational proxy is then an estimate of the log-likelihood ratio of a string under the hypothesis versus under the full empirical distribution of the dataset. The full description of how we estimate $\log(P(s|\text{context})/P(s))$ for each feature hypothesis is given in the appendix section on proxies (Bricken *et al.*, 2023c).

In this section we primarily study the learned feature which is most active in each context. There are typically other features that also model that context, and we find that rare "gaps" in the sensitivity of a main feature are often explained by the activity of another. We discuss this phenomenon in detail in sections on Activation Sensitivity and Feature Splitting.

We take pains to demonstrate the specificity of each feature, as we believe that to be more important for ruling out polysemanticity. Polysemanticity typically involves neurons activating for clearly unrelated concepts. [11] If our proxy shows that a feature only activates in some relatively rare and specific context, that would exclude the typical form of polysemanticity (Fig 4.6).

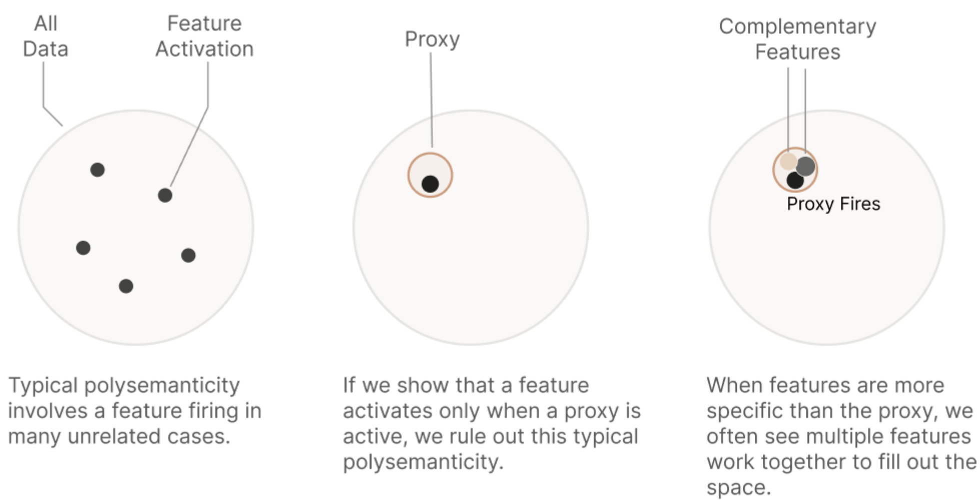


Figure 4.6: ***Feature Activity and its relation to the Proxy Activity.***

We finally note that the features in this section are cherry-picked to be easier to analyze. Defining simple computational proxies for most features we find, such as text concerning fantasy games, would be difficult, and we analyze them in other ways in the following section.

[11]For example, one neuron in our transformer model responds to a mix of academic citations, English dialogue, HTTP requests, and Korean text. In vision models, there is a classic example of a neuron which responds to cat faces and fronts of cars (Olah *et al.*, 2017).

4.4.1 Arabic Script Feature

The first feature we'll consider is an Arabic Script feature, A/1/3450. It activates in response to text in Arabic, Farsi, Urdu (and possibly other languages), which use the Arabic script. This feature is quite specific and relatively sensitive to Arabic script, and effectively invisible if we view the model in terms of individual neurons.

Activation Specificity

Our first step is to show that this feature fires almost exclusively on text in Arabic script. We give each token an "Arabic script" score using an estimated likelihood ratio $\log(P(s|\text{Arabic Script})/P(s))$, and break down the histogram of feature activations by that score (Fig. 4.7). Arabic text is quite rare in our overall data distribution – just 0.13% of training tokens — but it makes up 81% of the tokens on which our feature is active. That percentage varies significantly by feature activity level, from 25% when the feature is barely active to 98% when the feature activation is above 5.

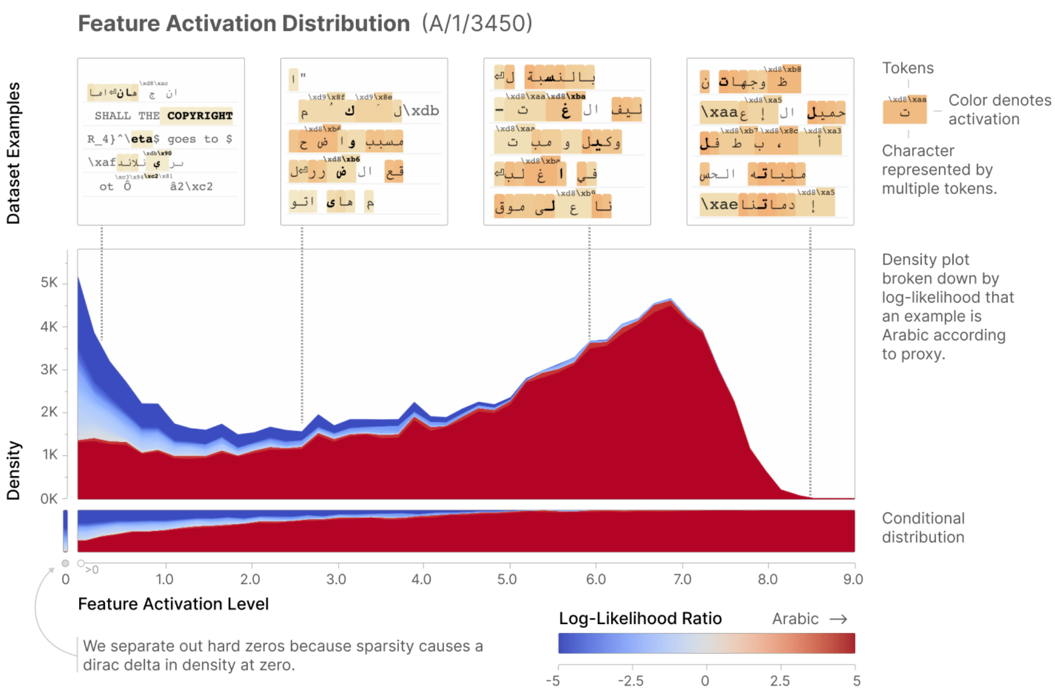


Figure 4.7: *Arabic Feature Activity Histogram.*

We also show dataset examples demonstrating different levels of feature activity. In interpreting them, it's important to note that Arabic Unicode characters are often split into multiple tokens. For example, the arabic character denoted U+062B in unicode is tokenized as `xd8` followed by `xab`.[12] When this happens, A/1/3450 typically only fires on the last token of the Unicode character.

The upper parts of the activation spectrum, above an activity of 5, clearly respond with high specificity to Arabic script. What should we make of the lower portions? We have three hypotheses:

The proxy is imperfect. It has false negatives due to common characters which are in other Unicode blocks (e.g., whitespace, punctuation), and also due to places where tokenization has created strange behavior.[13]

The model may be imperfect (but still calibrated). If features activate proportional to their "confidence" that some property is present, then we should expect them to sometimes be wrong in their weak activations. Although it might seem silly to fail to recognize which language a piece of text is, this is a very weak one layer transformer model, and the fact that characters are often split into multiple tokens adds additional difficulty. The autoencoder may be imperfect: If the width of our autoencoder is less than the number of "true features" being used by the model, then unrecovered features may show up as low activations across many of our learned features.

Regardless, large feature activations have larger impacts on model predictions,[14] so getting their interpretation right matters most. One useful tool for assessing the impact of false positives at low activation levels is the "expected value plot" of Cammarata *et al.* (2020).

[12]This kind of split tokenization is common for Unicode characters outside the Latin script and the most common characters from other Unicode blocks.

[13]For example, in the figure above, there is a newline character that our feature fires on but our proxy assigns a low score to because it is outside the Unicode block.

[14]Why do we believe large activations have larger effects? In the case of a one-layer transformer like the one we consider in this paper, we can make a strong case for this: features have a linear effect on the logits (modulo rescaling by layer norm), and so a larger activation of the feature has a larger effect on the logits. In the case of larger models, this follows from a lot of conjectures and heuristic arguments (e.g. the abundance of linear pathways in the model and the idea of linear features at each layer), and must be true for sufficiently small activations by continuity, but doesn't have a watertight argument.

We plot the distribution of feature activations weighted by activation level in Figure 4.8. Most of the magnitude of activation provided by this feature comes from dataset examples which are in Arabic script.

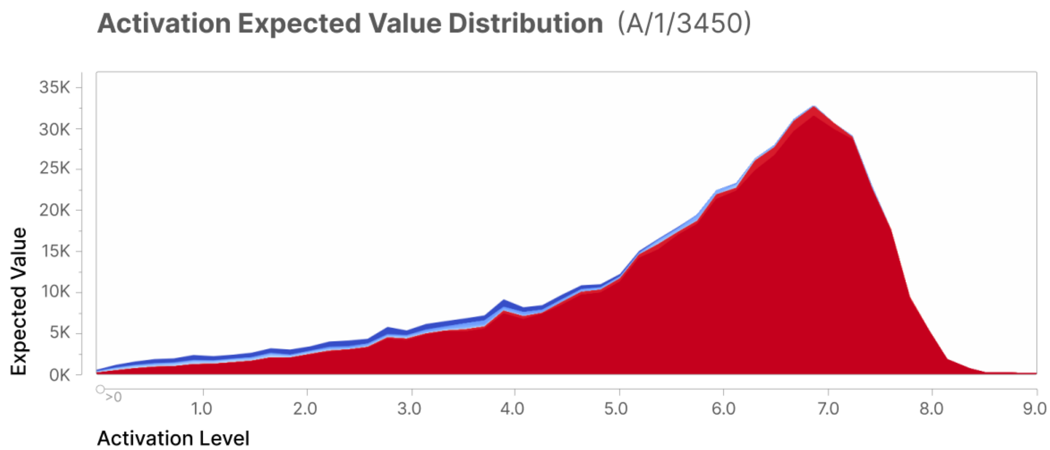


Figure 4.8: ***Multiplying the activity level by the frequency of firing to get an expected value distribution.***

Activation Sensitivity

In the Feature Activation Distribution above, it's clear that A/1/3450 is not sensitive to all tokens in Arabic script. In the random dataset examples, it fails to fire on five examples of the Arabic prefix which is the equivalent of the definite article "the" in English. However, in exactly those places, another feature which is specific to Arabic script, A/1/3134, fires. There are several additional features that fire on Arabic and related scripts (e.g. A/1/1466, A/1/3134, A/1/3399) which contribute to representing Arabic script. Another example deals with Unicode tokenization: when Arabic characters are split into multiple tokens, the feature we analyze here only activates at the final token comprising the character, while A/1/3399 activates on the first token comprising the character. To see how these features collaborate, we provide an alternative visualization showing all the features active on a snippet of Arabic text. We consider such interactions more in the Phenomenology section below.

Nevertheless, we find a Pearson correlation of 0.74 between the activity of our feature and the activity of the Arabic script proxy (thresholded at 0), over a dataset of 40 million tokens. Correlation provides a joint measure of sensitivity and specificity that takes magnitude into account, and 0.74 is a substantial correlation.

4.4.2 Feature Downstream Effects

Because the autoencoder is trained on model activations, the features it learns could in theory represent structure in the training data alone, without any relevance to the network's function. We show instead that the learned features have interpretable causal effects on model outputs which make sense in light of the features' activations. Note that these downstream effects are not inputs to the dictionary learning process, which only sees the activations of the MLP layer. If the resulting features also mediate important downstream behavioral effects then we can be confident that the feature is truly connected to the MLP's functional role in the network and not just a property of the underlying data.

We begin with a linear approximation to the effect of each feature on the model logits. We compute the logit weight following the path expansion approach of (Elhage *et al.*, 2021), [15] multiplying each feature direction by the MLP output weights, an approximation of the layer norm operation combining a projection to remove the mean and a diagonal matrix approximating the scaling terms, and the unembedding matrix ($d_i W_{down} \pi L W_{unembed}$). Because the softmax function is shift-invariant there is no absolute scale for the logit weights; we shift these so that the median logit weight for each feature is zero.

Each feature, when active, makes some output tokens more likely and some output tokens less likely. We plot that distribution of logit weights in Figure 4.9.[16] There is a large primary mode at zero, and a second much smaller mode on the far right, the tokens whose likelihood most increases when our feature is on. This second mode appears to correspond

[15]Also called "logit attribution", see similar work e.g. (Nanda, 2023).

[16]We exclude weights corresponding to extremely rare or never used vocabulary elements. These are perhaps similar to the "anomalous tokens" (e.g., "SolidGoldMagikarp") of Rumbelow and Watkins (2023).

to Arabic characters, and tokens which help represent Arabic script characters (especially `xd8` and

`xd9`, which are often the first half of the UTF-8 encodings of Arabic Unicode characters in the basic Arabic Unicode block).

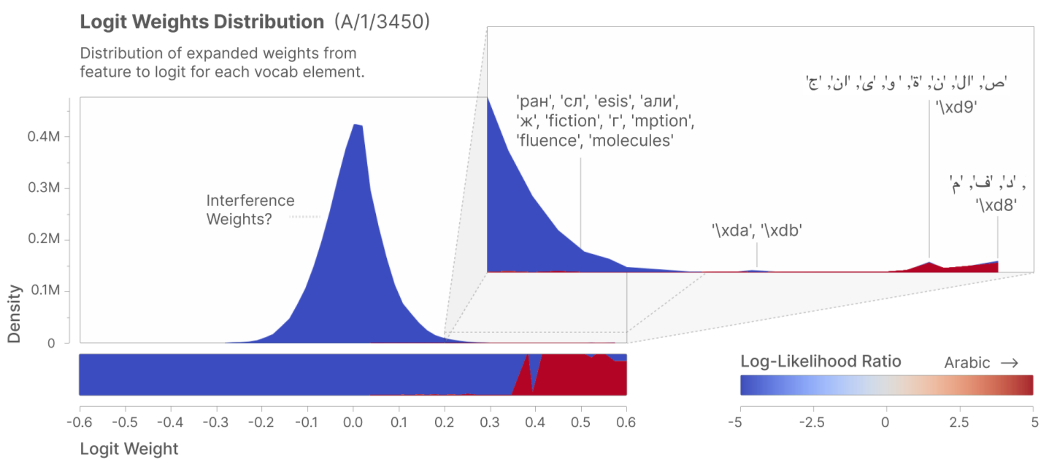


Figure 4.9: ***Arabic Feature's Logit Weights.***

This suggests that activating this feature increases the probability the network predicts Arabic script tokens.[17]

To visualize these effects on actual data, we causally ablate the feature. For a given dataset example, we run the context through the model until the MLP layer, decode the activations into features, then subtract off the activation of A/1/3450, artificially setting it to zero on the whole context, before applying the rest of the model. We visualize the effect of ablating the feature using underlines in the visualization; tokens whose predictions were helped by the feature (ablation decreased likelihood) are underlined in blue and tokens whose predictions were hurt by the feature (ablation increased likelihood) are underlined in red.

[17] There are in theory several ways the logit weights could overestimate the model's actual use of a feature: 1. It could be that these output weights are small enough that, when multiplied by activations, they don't have an appreciable effect on the model's output. 2. The feature might only activate in situations where other features make these tokens extremely unlikely, such the feature in fact has little effect. 3. It is possible that our approximation of linearizing the layer norm (see Framework (Elhage *et al.*, 2021)) is poor. Based on the subsequent analysis, which confirms the logit weight effects, we do not believe these issues arise in practice.

In the example on the right in Figure 4.10 we see that the A/1/3450 was active on every token in a short context (orange background). Ablating it hurt the predictions of all the tokens in Arabic script (purple underlines), but helped the prediction of the period `.` (orange underline). The rest of the figure displays contexts from two different ranges of feature activation levels. (The feature activation on the middle token of examples on the right ("subsample interval 5") is about half that of the middle token of examples on the left ("subsample interval 0")). We see that the feature was causally helping the model predictions on Arabic script through that full range, and the only tokens made less likely by the feature are punctuation shared with other scripts. The magnitudes of the impact are larger when the feature is more active.

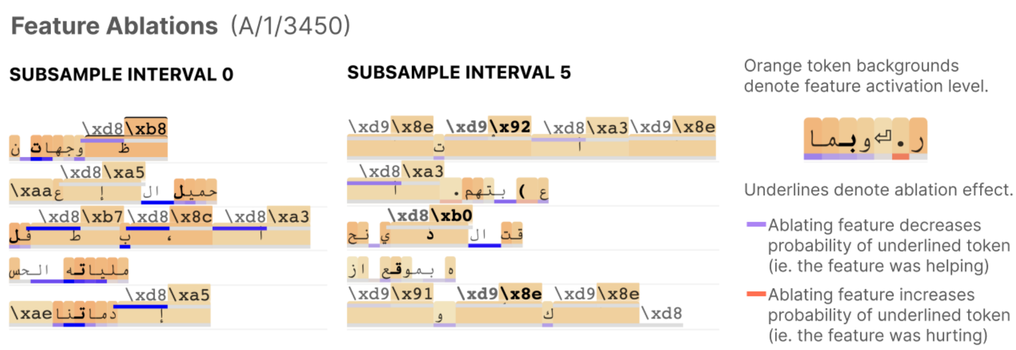


Figure 4.10: *Changes in token probabilities from ablating the Arabic feature.*

We encourage interested readers to view the feature visualization for A/1 to review this and other effects.

We also validate that the feature's downstream effect is in line with our interpretation as an Arabic script feature by sampling from the model with the feature activity "pinned" at a high value. To do this, we start with a prefix `1,2,3,4,5,6,7,8,9,10` where the model has an expected continuation (keep in mind that this is a one layer model that is very weak!). We then instead set A/1/3450 to its maximum observed value and see how that changes the samples in Figure 4.11.

Prefix	Normal Samples	Prefix	Pinned Samples
1,2,3,4,5,6,7,8,9,10	,8,3,3,3,3,3 ,8,8,8,9,7,6 ,8,30,20,8,10,10 ,10,10,10,10,10,10,20 ,10,7,6,8,8,6,8	1,2,3,4,5,6,7,8,9,10	يسوادارية ال أفروكايد� يدولة عم تعفات البعدي يان اسم استخ

Figure 4.11: ***Pinning Feature Activations high result in Arabic text.***

The feature is not a neuron

This feature seems rather monosemantic, but some models have relatively monosemantic neurons, and we want to check that dictionary learning didn't merely hand us a particularly nice neuron.[18] We first note that when we search for neurons with Arabic script in their top dataset 20 examples, only one neuron turns up, with a single example in Arabic script. (Eighteen of the remaining examples are in English, and one is in Cyrillic.)

We then look at the coefficients of the feature in the neuron basis in Figure 4.12, and find that the three largest coefficients by magnitude are all negative (!) and there are a full 27 neurons whose coefficients are at least 0.1 in magnitude.

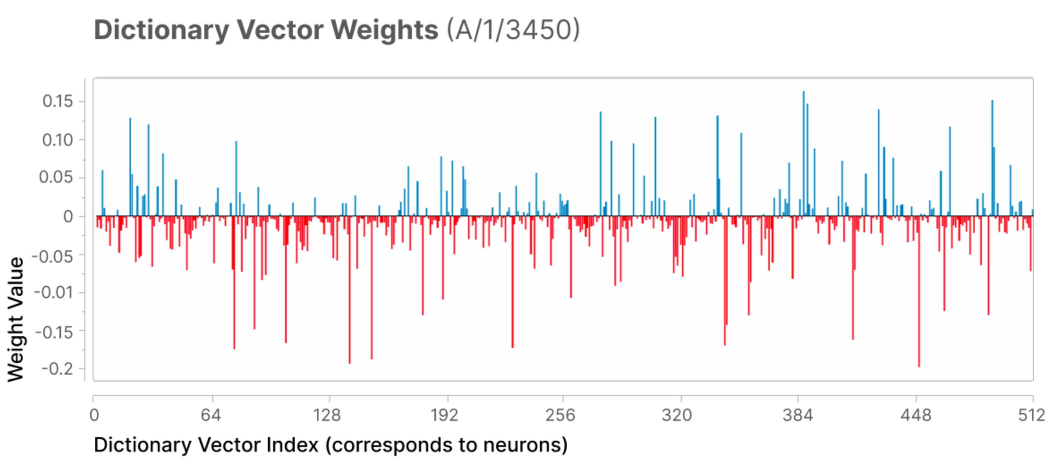


Figure 4.12: ***Feature Decoder Vector Weights.***

[18] Of course, some features might genuinely be neuron aligned. But we'd like to know that at least some of the features dictionary learning discovers were not trivial.

It is of course possible that these neurons engage in a delicate game of cancellation, resulting in one particular neuron's primary activations being sharpened. To check for this, we find the neuron whose activations are most correlated to the feature's activations over a set of 40 million dataset examples. [19] The most correlated neuron (A/neurons/489) responds to a mixture of different non-English languages.[20] We can see this by visualizing in Fig. 4.13 the activation distribution of the neuron – the dataset examples are all other languages, with Arabic script present only as a small sliver.

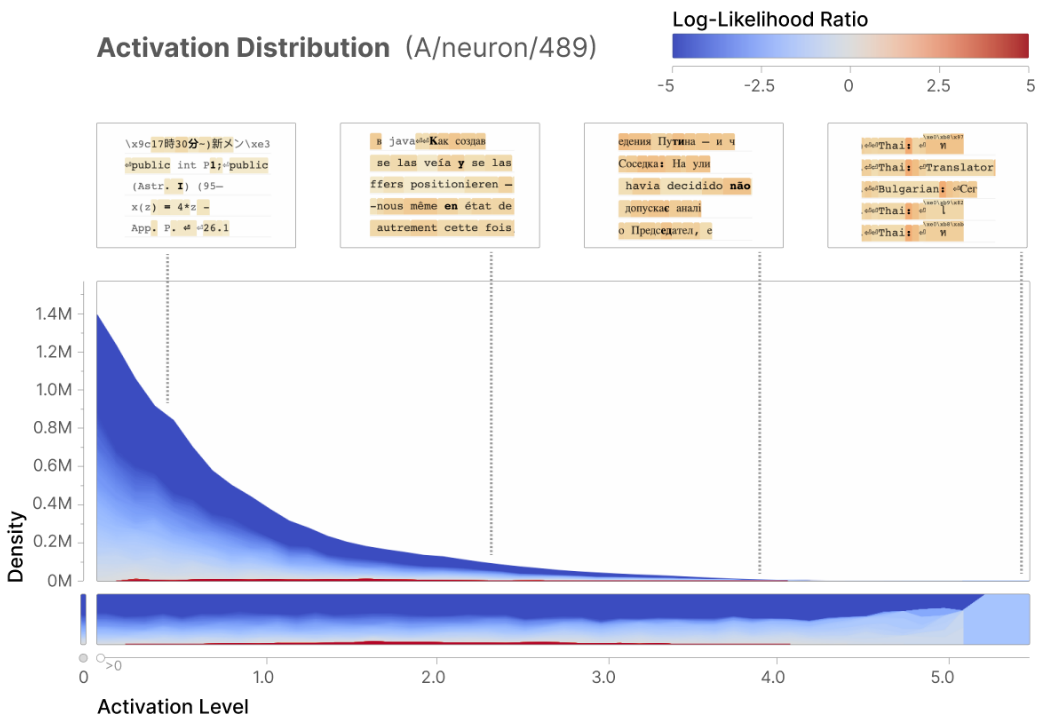


Figure 4.13: ***Closest neuron activations on Arabic text.***

Logit weight analysis is also consistent with this neuron responding to a mixture of languages (Fig. 4.14. For example, in the figure below many of the top logit weights

[19]We also tried looking at the neurons which have the largest contribution to the dictionary vector for the feature. However, we found looking at the most correlated neuron to be more reliable – in earlier experiments, we found rare cases where the correlated method found a seemingly similar neuron, while the dictionary method did not. This may be because neurons can have different scales of activations.

[20]This makes sense as a superposition strategy: since languages are essentially mutually exclusive, they're natural to put in superposition with each other (Olah, 2023).

appear to include Russian and Korean tokens. Careful readers will observe a thin red sliver corresponding to rare Arabic script tokens in the distribution. These Arabic script tokens have weight values that are very slightly positive leaning overall, but some are negative.

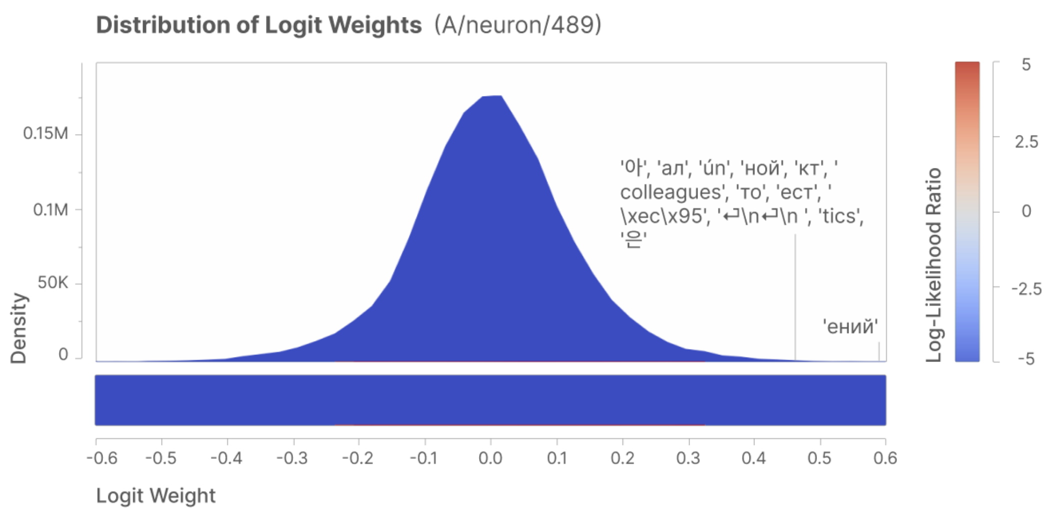


Figure 4.14: ***Closest Neuron Logit Weights.***

Finally, scatter plots and correlations in Fig. 4.15 suggest the similarities between A/1/3450 and the neuron are non-zero, but quite minimal.[21] In particular, note how the y-axis of the logit weights scatter plot (corresponding to the feature) cleanly separates the Arabic script token logits, while the x-axis does not. More generally, note how the y-marginals both clearly exhibit specificity, but the x-marginals do not.

We conclude that the features we study do not trivially correspond to a single neuron. The Arabic script feature would be effectively invisible if we only analyzed the model in terms of neurons.

Universality

We will now ask whether A/1/3450 is a universal feature that forms in other models and can be consistently discovered by dictionary learning. This would indicate we are discovering

[21] By analogy, this also applies to B/1/1334.

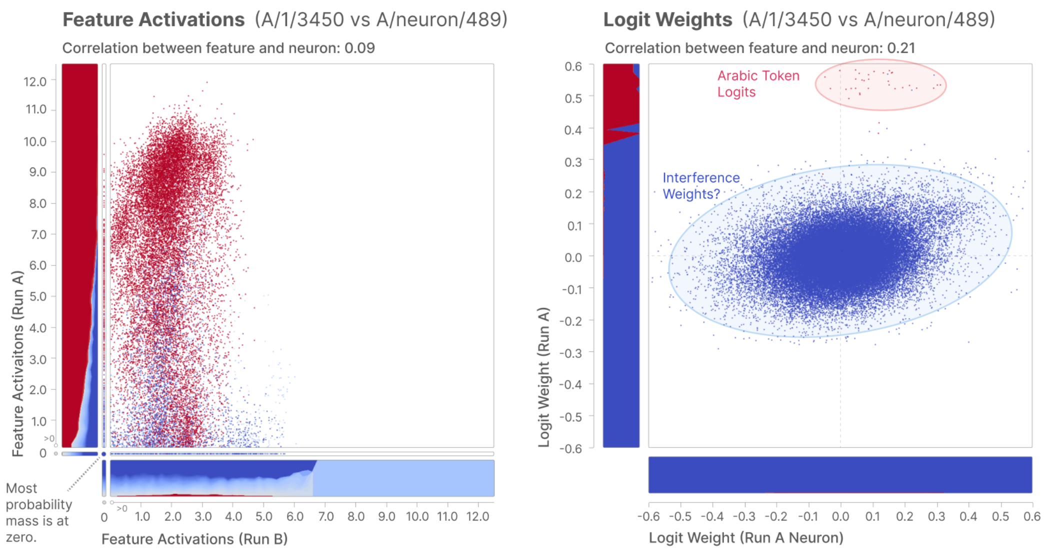


Figure 4.15: *Correlation between the Arabic feature and neuron.*

something more general about how one-layer transformers learn representations of the dataset.

We search for a similar feature in B/1, a dictionary learning run on a transformer trained on the same dataset but with a different random seed. We search for the feature with the highest activation correlation [22] and find B/1/1334 (corr=0.91), which is strikingly similar:

This feature clearly responds to Arabic script as well as shown in Figure 4.16. If anything, it's nicer than our original feature – it's more specific in the 0–1 range. The logit weights tell a similar story in Figure 4.17.

The effects of ablating this feature are also consistent with this (see the visualization for B/1/1334).

To more systematically analyze the similarities between A and B, we look at scatter plots comparing the activations or logit weights in Figure 4.18.

The activations are strongly correlated (Pearson correlation of 0.91), especially in the main Arabic mode.

[22]"Activation correlation" is defined as the feature whose activations across 40,960,000 tokens has the highest Pearson correlation with those of A/1/3450.

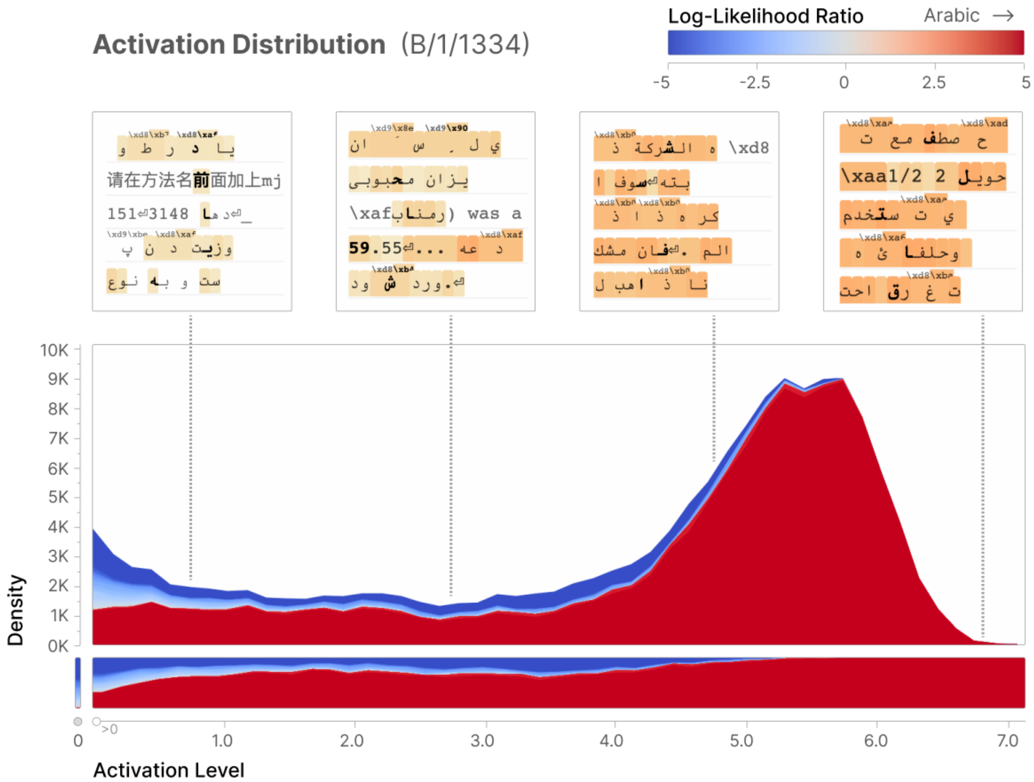


Figure 4.16: *Activation Distribution for the most similar Arabic feature in run B.*

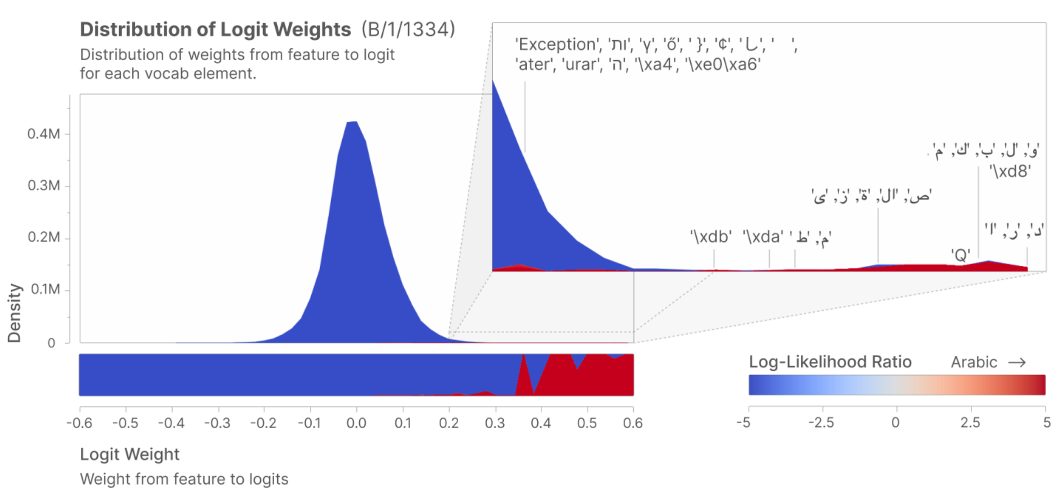


Figure 4.17: *Logit weights for the most similar Arabic feature in run B.*

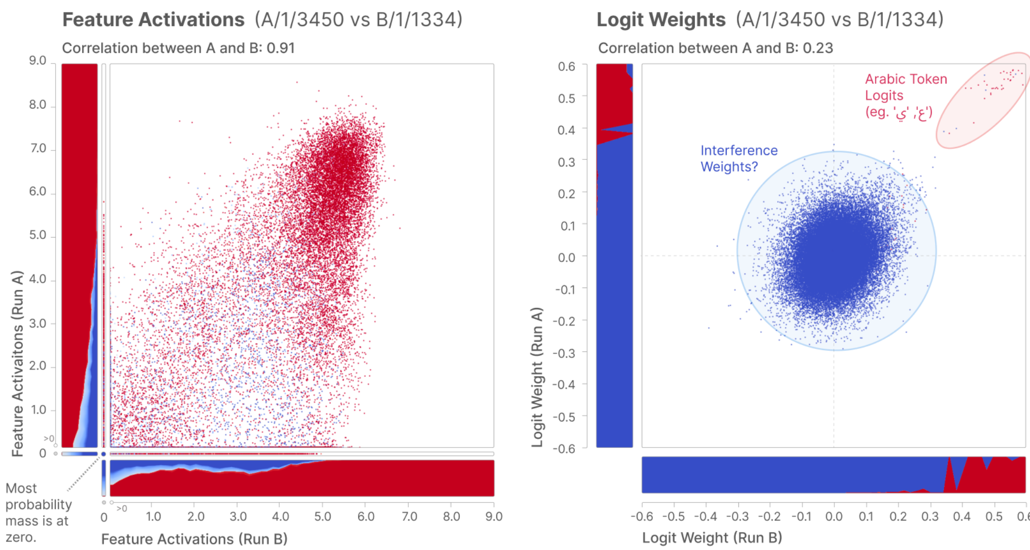


Figure 4.18: ***Correlation Scatter Plots between run A and run B Arabic Feature activations and logit weights.***

The logit weights reveal a two-dimensional version of the bimodality we saw in the histogram for A/1, with logit weights for Arabic tokens clustering at the top right. The correlation is more modest than that of the activations because the distribution is dominated by a relatively uncorrelated mode in the center. We hypothesize this central mode corresponds to "weight interference" and that the shared outlier mode is the important observation – that is, the model may ideally prefer to have all those weights be zero, but due to superposition with other features and their weights, this isn't possible.

4.4.3 DNA Feature

We now consider a DNA feature, A/1/2937. It activates in response to long uppercase strings consisting of `A`, `T`, `C`, and `G`, typically used to represent nucleotide sequences. We closely follow the analysis of the Arabic script feature above to show activation specificity and sensitivity for the feature, sensible downstream effects, a lack of neuron alignment, and universality between models. The main differences will be that (1) A/1/2937 is the only feature devoted to modeling the DNA context, and (2) our proxy is less sensitive to DNA than our feature is, missing strings containing punctuation, spaces, and missing bases.

Activation Specificity and Sensitivity

We begin with the computational proxy for "is a DNA sequence", $\log(P(s|\text{DNA})/P(s))$ used to color the activation distribution in Figure 4.19. Because there are some vocabulary tokens, such as `CAT`, which could occur in DNA but also occur in other contexts, we always look at groups of at least two tokens when evaluating the proxy. The log-probabilities turn out to be quite bimodal, so we binarize the proxy (based on its sign). This binarized proxy then has a Pearson correlation of 0.8 with the feature activations.

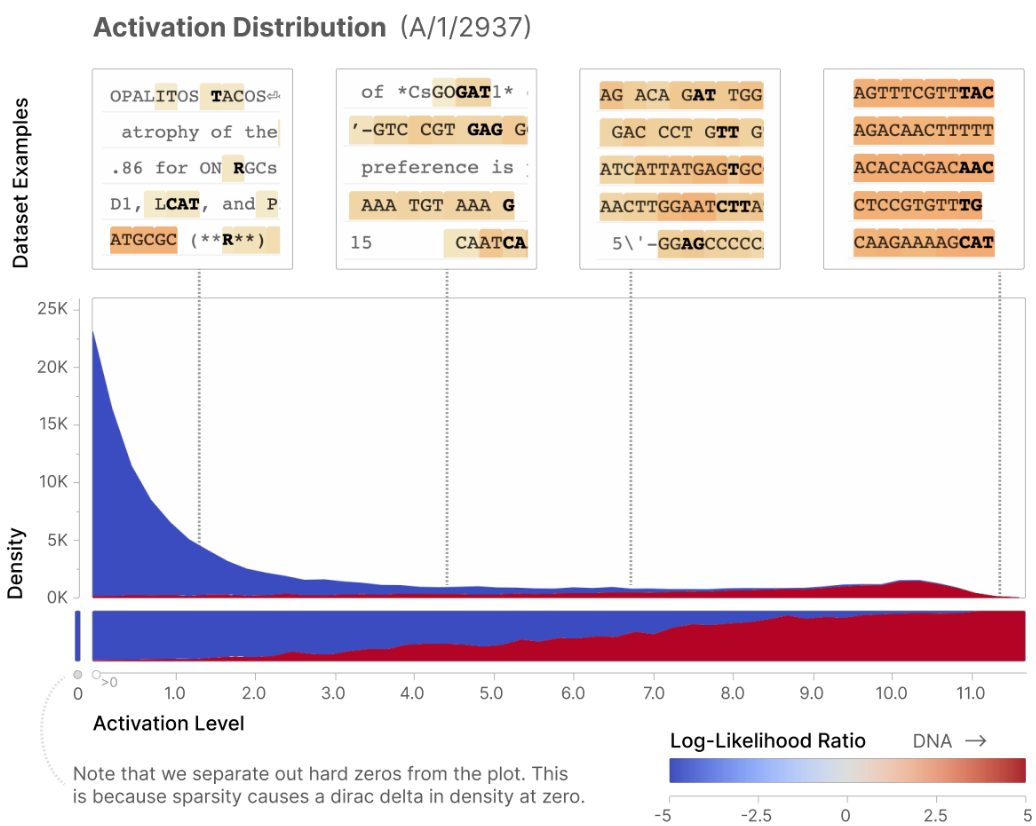


Figure 4.19: *DNA Feature Activation Distribution.*

While the feature appears to be quite monosemantic in the feature's higher registers (all 10 random dataset examples about activation of 6.0 are DNA sequences), there is significant blue indicating the DNA proxy not firing in the lower registers. Below in Fig. 4.20 we show a grid of random examples at four activation levels (including feature off) where the proxy does and doesn't fire.

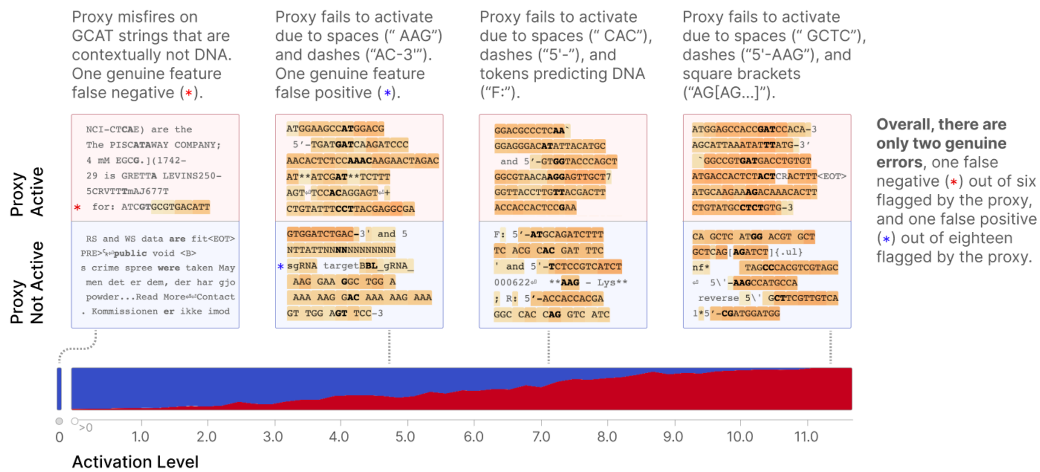


Figure 4.20: *DNA Proxy Examples.*

We note that in all but two cases where the feature and proxy disagree, the feature does indicate a DNA sequence, just one outside our proxy's strict `ATCG` vocabulary. For example, the space present in the triplets `TGG AGT` makes the proxy fail to fire. The feature also fires productively on `'-` in the string `5'-TCT`, because what follows that prefix should be DNA, even though the prefix is not itself DNA. (The causal ablation reveals that turning off the DNA feature hurts the prediction of the strings that follow.) We thus believe that A/1/2937 is quite sensitive and specific for DNA. The case where the proxy fires and the feature does not is indeed a DNA sequence, though the feature begins firing on the very next token. We observe that the DNA feature may not fire on the first few tokens of a DNA sequence, but by the end of a long DNA sequence, it is the only feature active.

Feature Downstream Effect

The downstream effect of the DNA feature being active, as measured by logit weights, make sense, with all the top tokens being combinations of nucleotides like `AGT` and `GCC` (Figure 4.21.

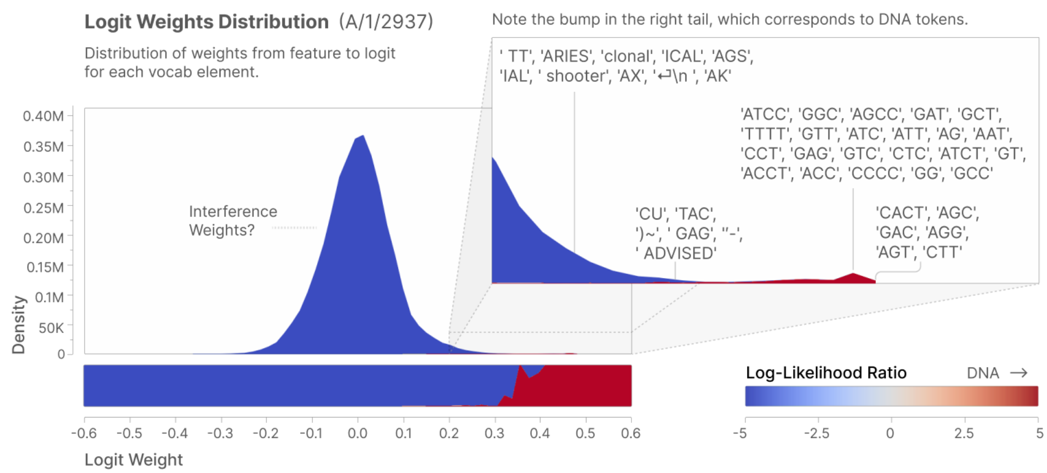


Figure 4.21: ***DNA Feature Logit Weights.***

The Feature Is Not A Neuron

The most similar neuron to A/1/2937, as measured by activation correlation, is A/neurons/67. Figure 4.22 shows that DNA contexts form a tiny sliver of that neuron's activating examples. The neuron whose coefficient is the highest in our feature's vector, A/neurons/227, also has no DNA sequences in its top activating examples.

Universality

A/1/2937 has a correlated feature (corr=0.92) in run B/1, B/1/3680. Their top logit weights agree, and are DNA tokens (e.g. `AGT`) forming a separate mode (circled on the right) than the bulk of the logit weights for both (Fig. 4.23.

4.4.4 Base64 Feature

We now consider a base64 feature, A/1/2357. We're particularly excited by this feature because we discovered a base64 neuron in our SoLU paper (Nelson *et al.*, 2022), suggesting that base64 might be quite universal – even across models trained on somewhat different datasets and with different architectures. We model base64 strings as random sequences of characters from `[a-zA-Z0-9+/]`. The activation distribution in Fig. 4.24 is colored by the

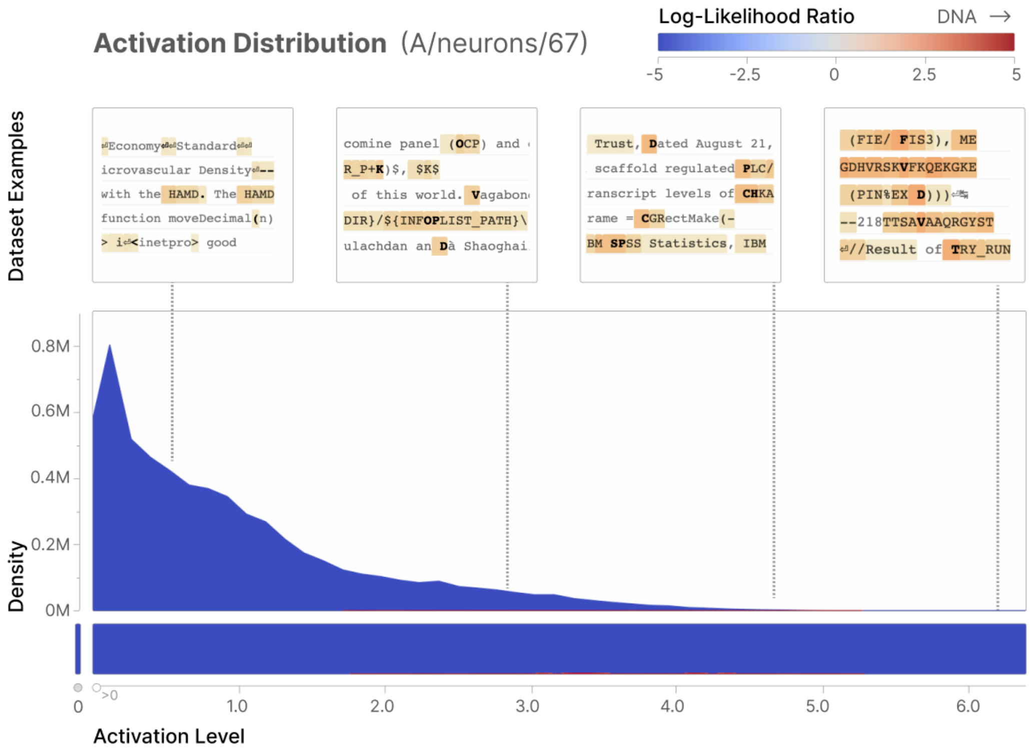


Figure 4.22: *DNA Most Similar Neuron Activation Distribution*

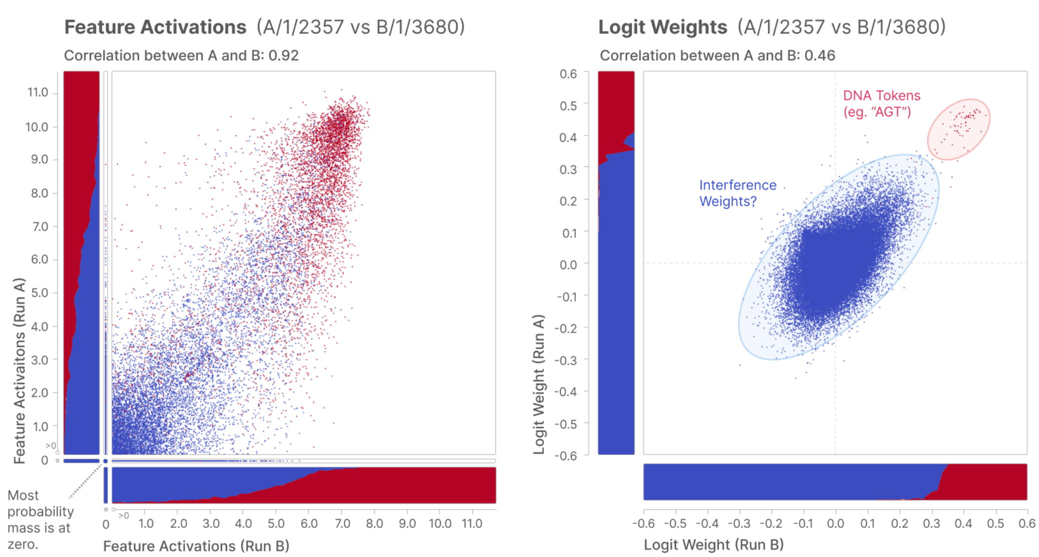


Figure 4.23: *DNA Feature Universality between two runs. Activation and Logit Correlation Scatterplots.*

corresponding computational proxy, together with the random dataset examples from each activation level, shows that this feature is quite specific to base64.

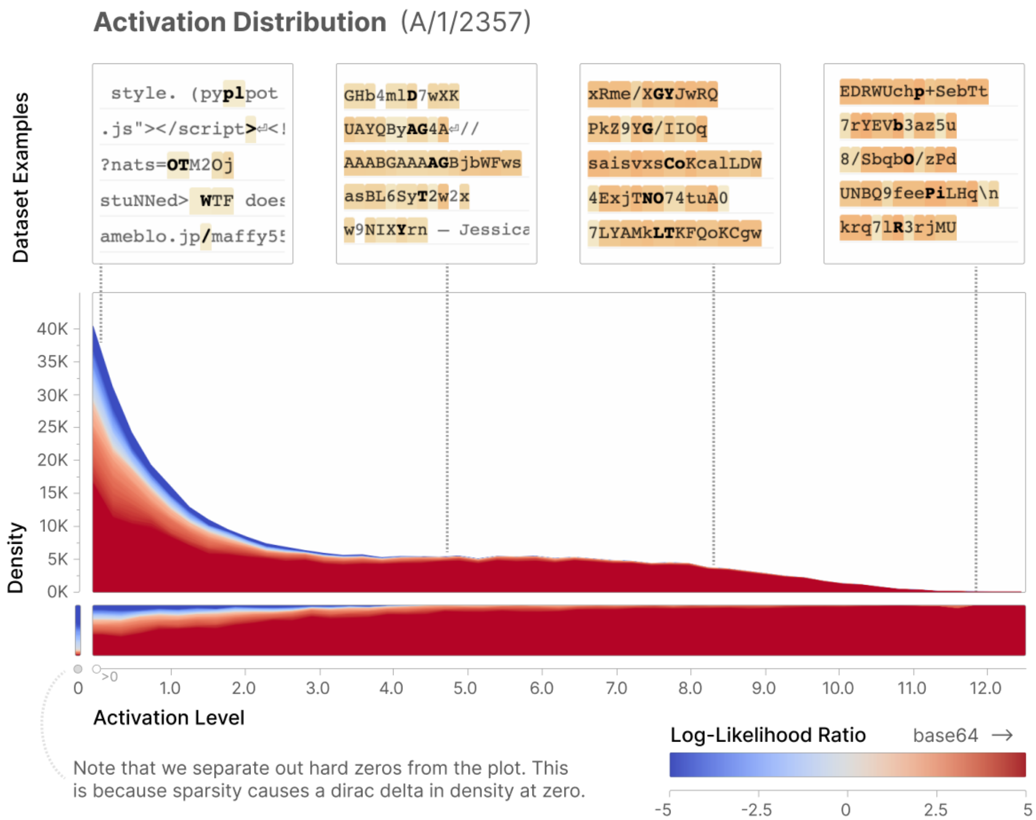


Figure 4.24: *Base64 Feature Activation Distribution.*

This is not the only feature active in base64 contexts, and in a section below we discuss the two others, one of which fires on single digits in base64 contexts (like the 2, 4, 7, and 9 on which A/1/2357 doesn't activate in the figure above), exploiting a property of the BPE tokenizer to make a better prediction.

Turning to the logit weights, they have a second mode consisting of highly base64-specific tokens (Fig 4.25). The main mode seems to primarily be interference, but the right side is skewed towards base64-neutral or slightly base64-leaning tokens. (If we look at the conditional below, we see a more continuous transition to base64-specific tokens.)

There is a more continuous transition between non-base64 and base64 tokens than we

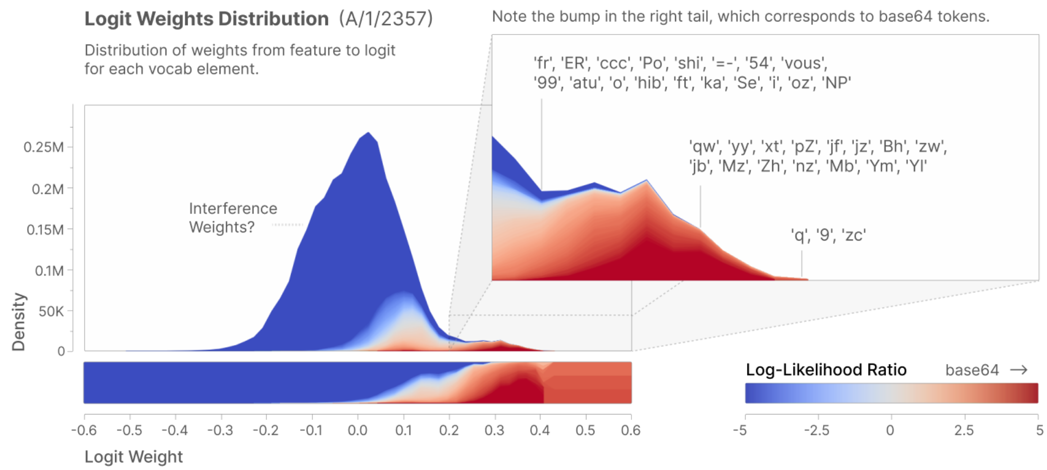


Figure 4.25: ***Base64 Feature Logit Weights.***

saw in the Arabic script example. This difference likely arises because whether a token occurs more in Arabic script than in other text is a relatively binary distinction, whereas whether a token occurs more in base64 or other text varies more continuously. For instance, `fr` is both a common abbreviation for the French language and also a base64 token, so it makes sense for the model to be cautious in up-weighting `fr` because it might already have a higher prior due to use in French. Indeed, any token consisting of letters from the English alphabet will have some nontrivial probability of appearing in base64 strings.

The Pearson correlation between the computational proxy and the activity of A/1/2357 is just 0.38. We believe that is mostly because the proxy is too broad. For example hexadecimal strings (those made of `[0-9A-F]`) activate the proxy, as they are quite different from the overall data distribution, but are actually predicted by a feature of their own, A/1/3817.

Universality

A/1/2357 has a correlated feature (corr=0.85) in run B/1, B/1/2165. It also has high activation specificity for base64 strings (Fig. 4.26):

Like A/1/2357, B/1/2165's logit weights (Fig. 4.27) have a second mode corresponding to base64 token:

Correlations and scatter plots (Fig. 4.28) are also consistent with them being very similar

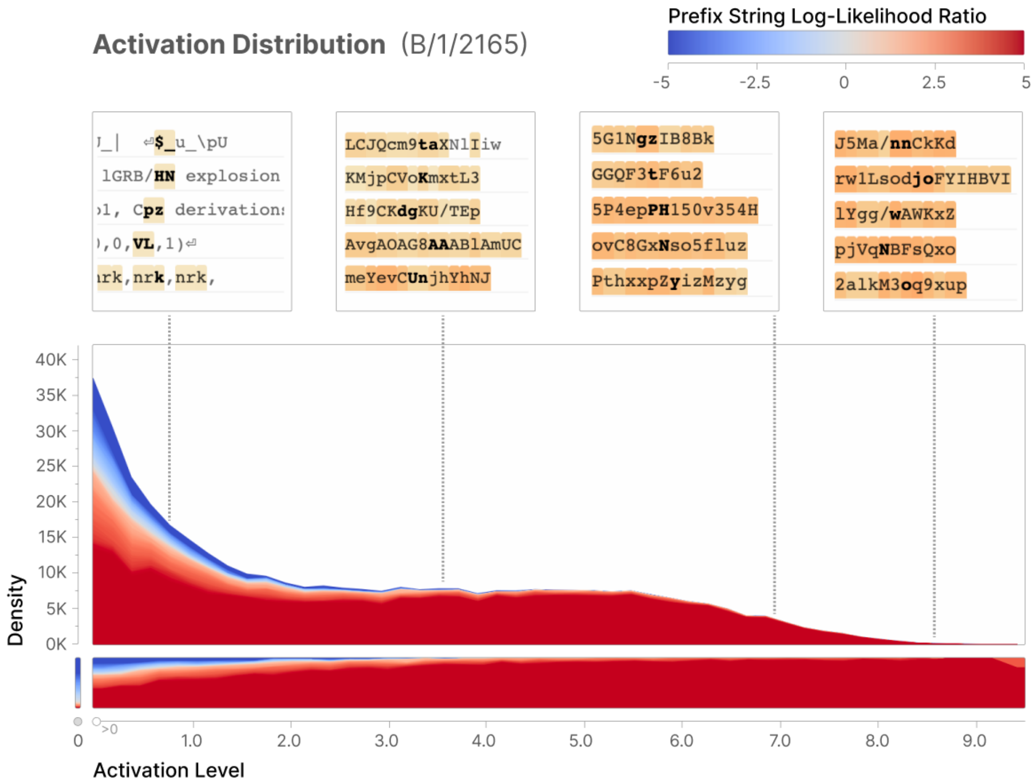


Figure 4.26: *Base64 Most Similar Feature Activations from Run B.*

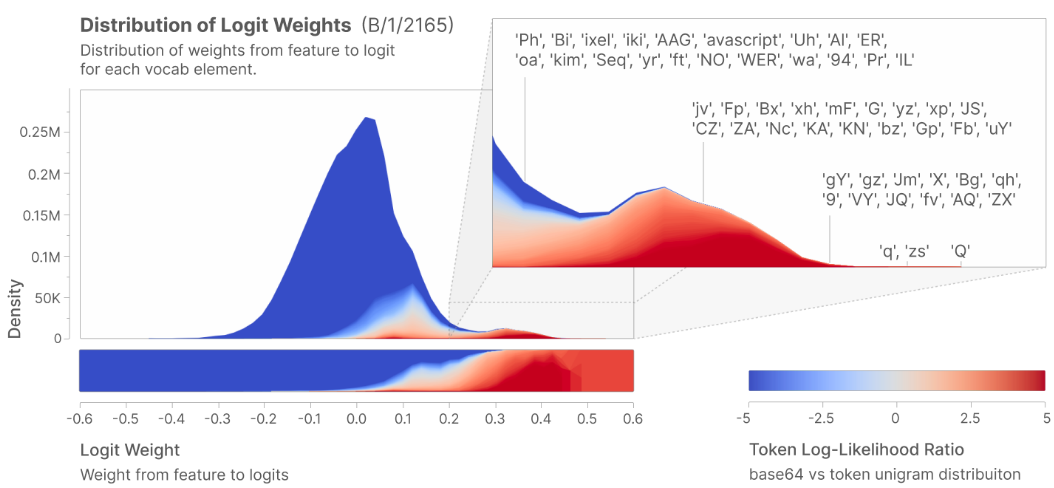


Figure 4.27: *Base64 Most Similar Feature Logit Weights from Run B.*

features:

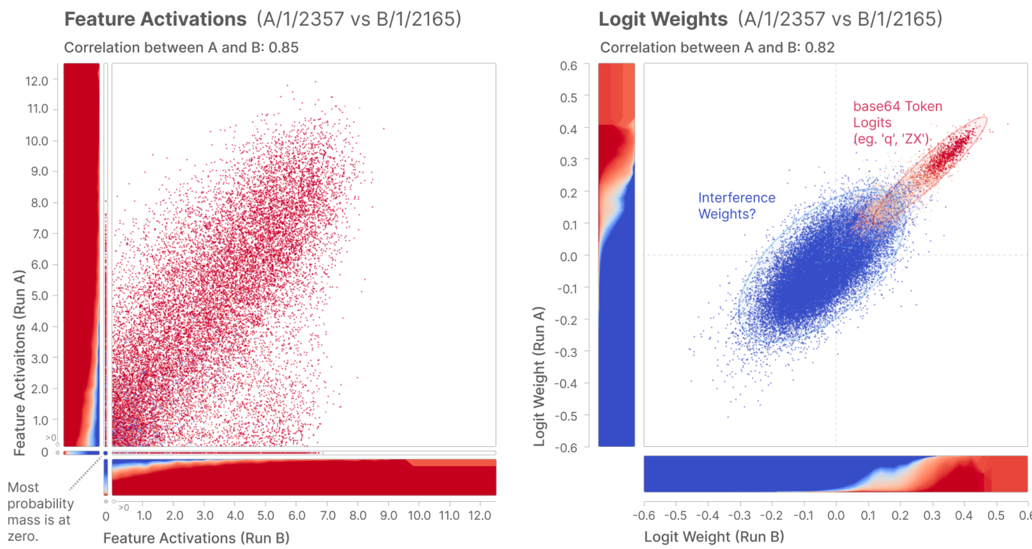


Figure 4.28: ***Base64 Most Similar Feature Correlation Scatter Plots from Run B.***

Note that we expect the overlap between the interference and base64 token logit weights to be from the aforementioned usage of base64 token across many other contexts.

The Feature Is Not A Neuron

Looking at the neuron in model A that most correlates with this feature: A/neurons/470 (corr=0.18), we find that while it does notably respond to base64 strings, it also activates for lots of other things, including code, HTML labels, parts of URLs, etc. (Fig. 4.29):

The logit weights suggest it somewhat increases base64 tokens, but is much more focused on upweighting other tokens, e.g. filename endings (Fig. 4.30).

The activation and logit correlations are consistent with this neuron helping represent the same feature, but largely doing other things (Fig. 4.31).

4.4.5 Hebrew Feature

Another interesting example is the Hebrew feature A/1/416. Like the Arabic feature, it's easy to computationally identify Hebrew text based on Unicode blocks.

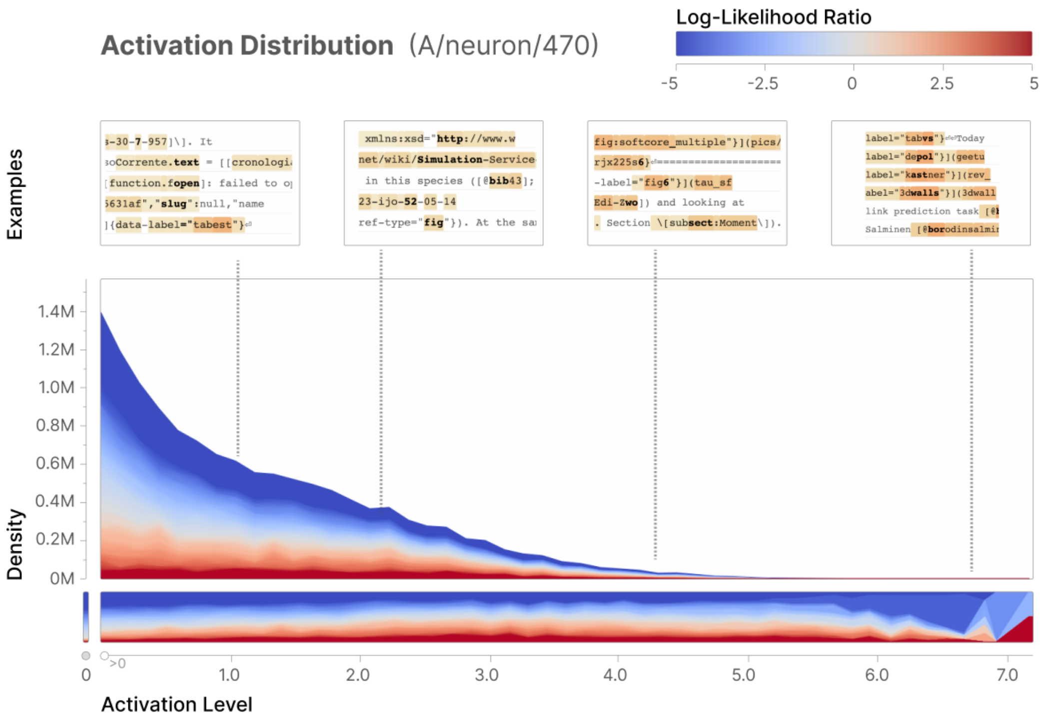


Figure 4.29: *Base64 most similar neuron's Activation Distribution.*

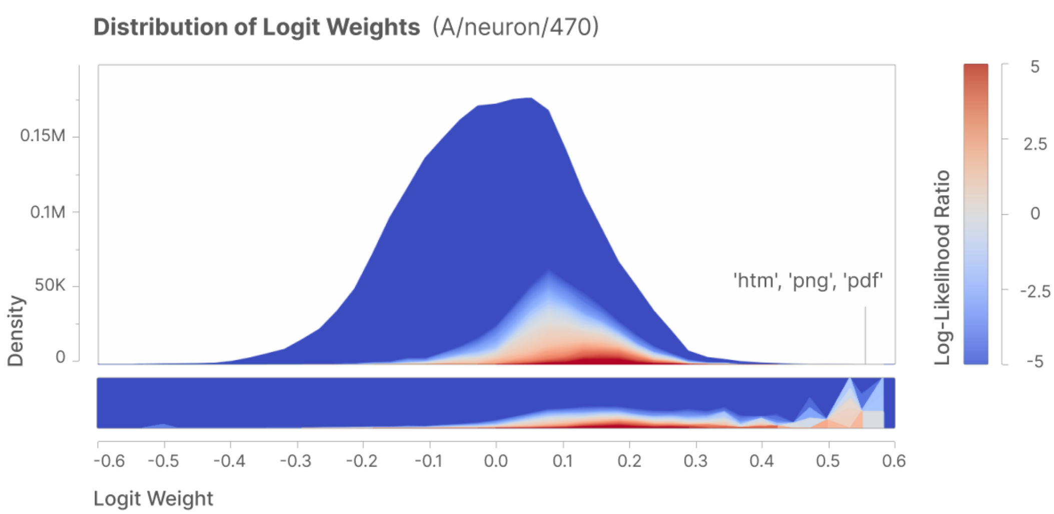


Figure 4.30: *Base64 most similar neuron's Logit Weights.*

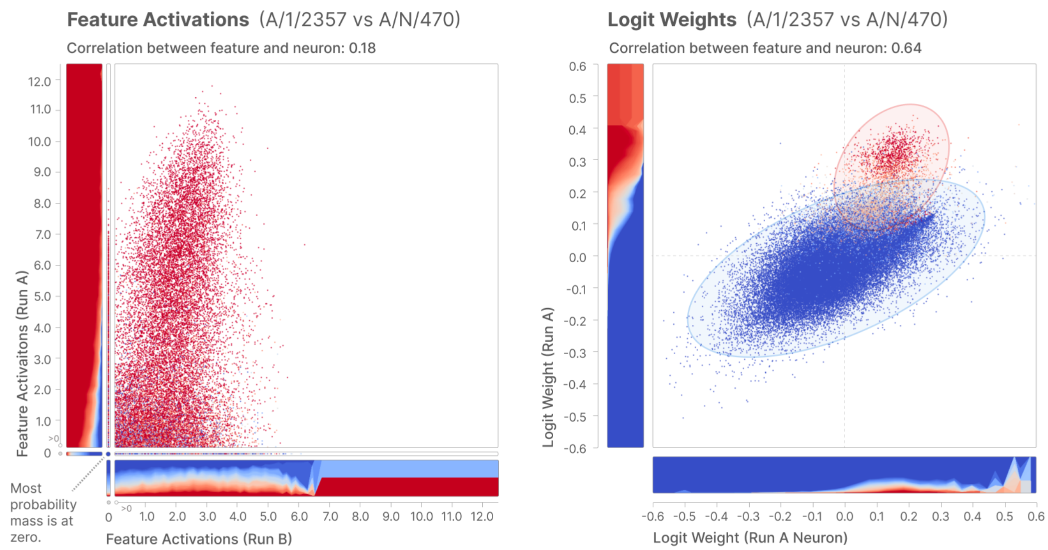


Figure 4.31: ***Base64 most similar neuron's correlation scatter plots.***

Shown in Fig. 4.32, A/1/416 has high activation specificity in the upper spectrum. It does weakly activate for other things (especially other languages with Unicode scripts). There is also some blue in strong activations; this appears to significantly be on "common characters", such as whitespace or punctuation, which are from other unicode blocks (see more discussion of similar issues in the Arabic feature section).

Its logit weights in Fig. 4.33 have a notable second mode, corresponding to Hebrew characters and relevant incomplete Unicode characters. Note that `xd7` is the first token in the UTF-8 encoding of most characters in the basic Hebrew Unicode block.

The Pearson correlation of the Hebrew script proxy with A/1/416 is 0.55. Some of the failure of sensitivity may be due to a complementary feature A/1/1016 that fires on `xd7` and predicts the bytes that complete Hebrew characters' codepoints.

The Feature Is Not A Neuron

There doesn't appear to be a similar neuron. The most correlated neuron in model A is A/neurons/489 (corr=0.1), which has low activation and logit specificity. Consider the

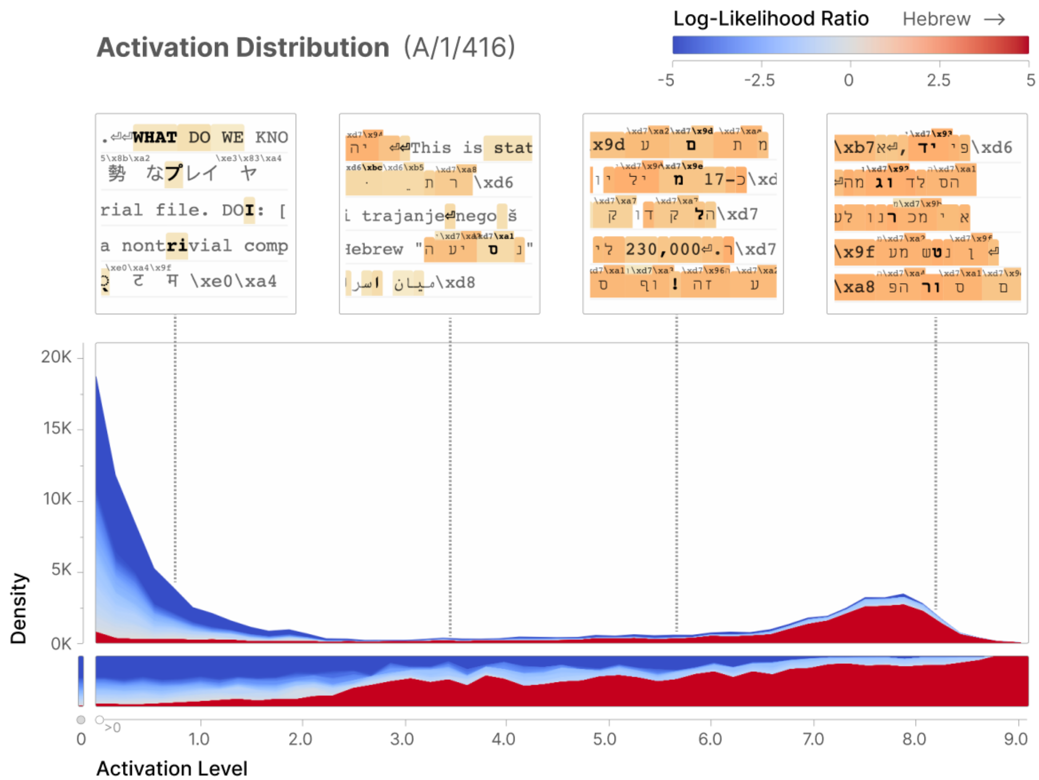


Figure 4.32: *Hebrew Feature Activation Distribution*

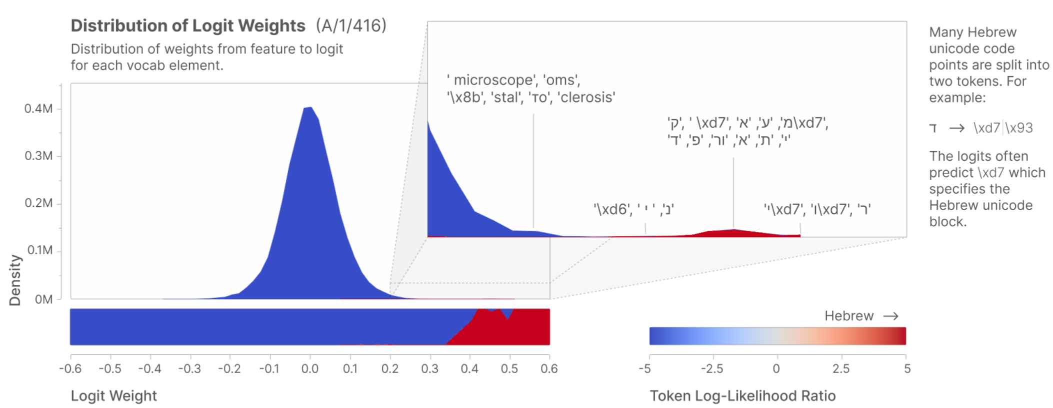


Figure 4.33: *Hebrew Feature Logit Weights*

following activation and logit correlation plots of Figure 4.34:

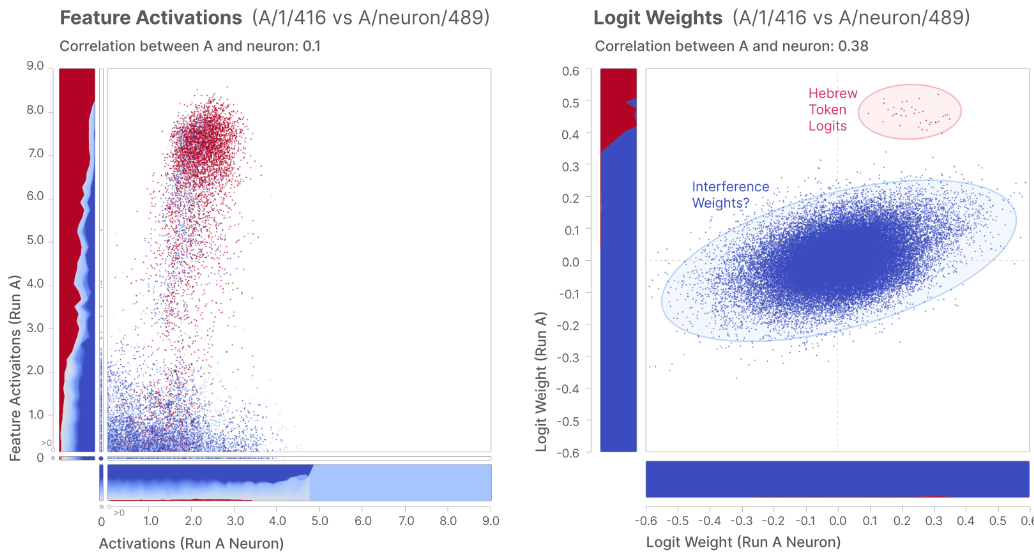


Figure 4.34: ***Hebrew Feature Correlation with most similar Neuron.***

To cross-validate this, we also searched for any neuron where the main Hebrew Unicode block appeared in the top dataset examples. We found none.

Universality

A/1/416 has a correlated feature in the B/1 run, B/1/1901 (corr=0.92) that has significant activation specificity shown in Fig. 4.35.

Logit weights have a second mode, as before in Fig. 4.36.

Activation and logit weight correlations are again consistent (Fig. **??**).

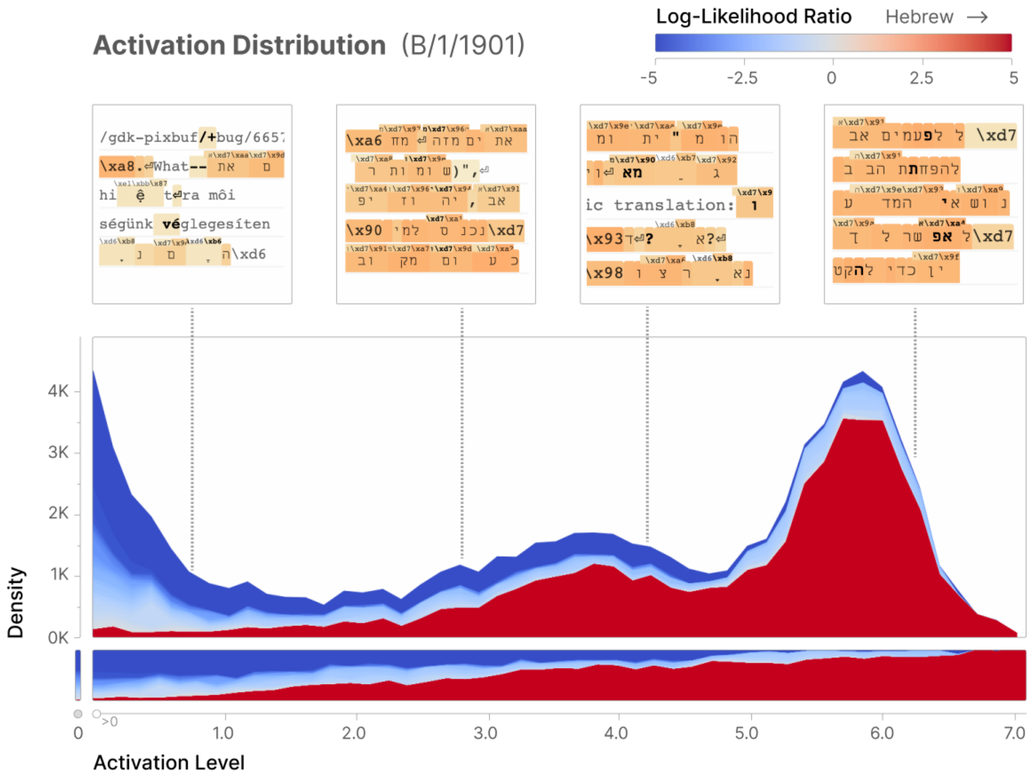


Figure 4.35: *Hebrew Feature from Run B's Activation Distribution.*

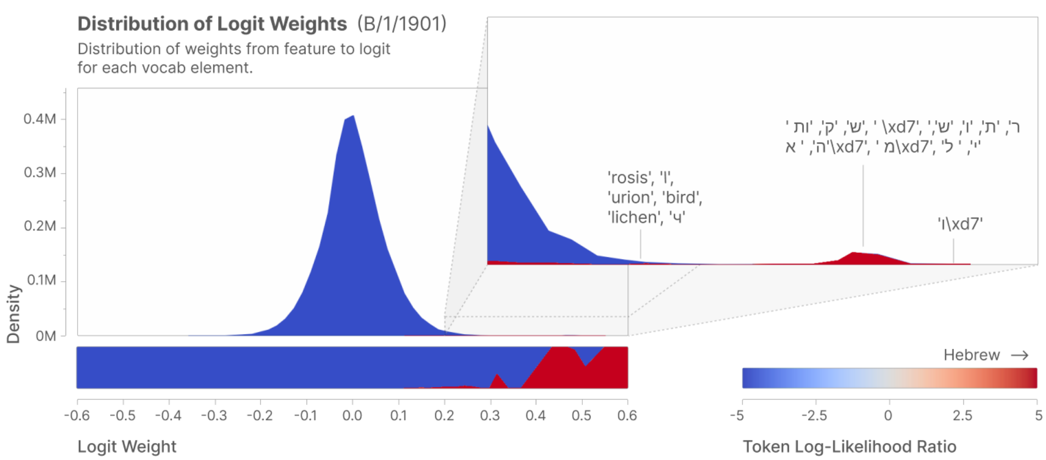


Figure 4.36: *Hebrew Feature from Run B's Logit Weights.*

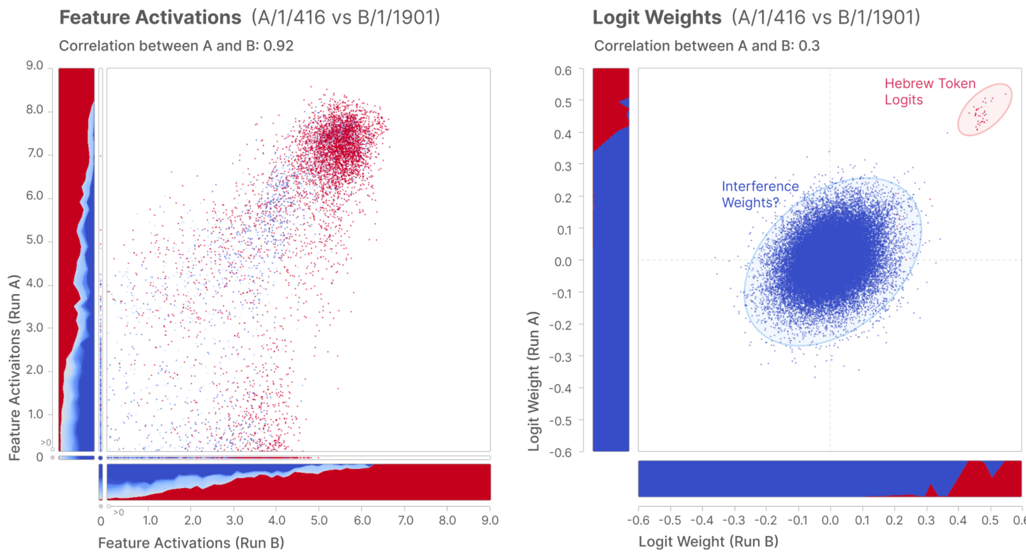


Figure 4.37: ***Hebrew Feature from Run B's Activation and Logit Correlations.***

4.5 Global Analysis

If the previous section has persuaded you that at least some of the features are genuinely interpretable and reflect the underlying model mechanics, it's natural to wonder how broadly this holds outside of those cherry-picked features. The primary focus of this section will be to answer the question, "how interpretable are the rest of the features?" We show that both humans and large language models find our features to be significantly more interpretable than neurons, and quite interpretable in absolute terms.

There are a number of other questions one might also ask. To what extent is our dictionary learning method discovering all the features necessary to understand the MLP layer? Holistically, how much of the MLP layer's mechanics have been made interpretable? We are not yet able to fully answer these questions to our satisfaction, but will provide some preliminary speculation towards the end of this section.

We note that of the 4,096 learned features in the A/1 autoencoder, 168 of them are "dead" (active on none of the 100 million dataset) and 292 of them are "ultralow density", active on less than 1 in a million dataset examples and exhibiting other atypical properties. We exclude both these groups of features from further analyses.

4.5.1 How Interpretable is the Typical Feature?

In this section, we use three different methods to analyze how interpretable the typical feature is, and how that compares to neurons: human analysis, and two forms of automated interpretability. All three approaches find that features are much more interpretable than neurons.

Manual Human Analysis

At present, we do not have any metric we trust more than human judgment of interpretability. Thus, we had a blinded annotator (one of the authors, Adam Jermyn) score features and neurons based on how interpretable they are. The scoring rubric can be found in the appendix (Bricken *et al.*, 2023c) and accounts for confidence in an explanation, consistency of the activations with that explanation, consistency of the logit output weights with that explanation, and specificity.

In doing this evaluation, we wanted to avoid a weakness we perceived in our prior work (e.g., (Nelson *et al.*, 2022)) of focusing evaluation predominantly on maximal dataset examples, and paying less attention to the rest of the activation spectrum. Many polysemantic neurons appear monosemantic if you only look at top dataset examples, but are revealed to be polysemantic if you look at lower parts of the activation spectrum. To avoid this, we draw samples uniformly across the spectrum of feature activations,[23] and score each interval separately in light of the overall hypothesis suggested by the feature.

Unfortunately, this approach is labor intensive and so the number of scored samples is small. In total, 412 feature activation intervals were scored across 162 features and neurons.

We see in Figure 4.38 that features are substantially more interpretable than neurons. Very subjectively, we found features to be quite interpretable if their rubric value was above 8. The median neuron scored 0 on our rubric, indicating that our annotator could not even form a hypothesis of what the neuron could represent! Whereas the median feature interval

[23]In order to sample uniformly across the spectrum of feature activations, we divide the activation spectrum into 11 "activation intervals" evenly spaced between 0 activation and the maximum activation. We sample uniformly from these intervals.

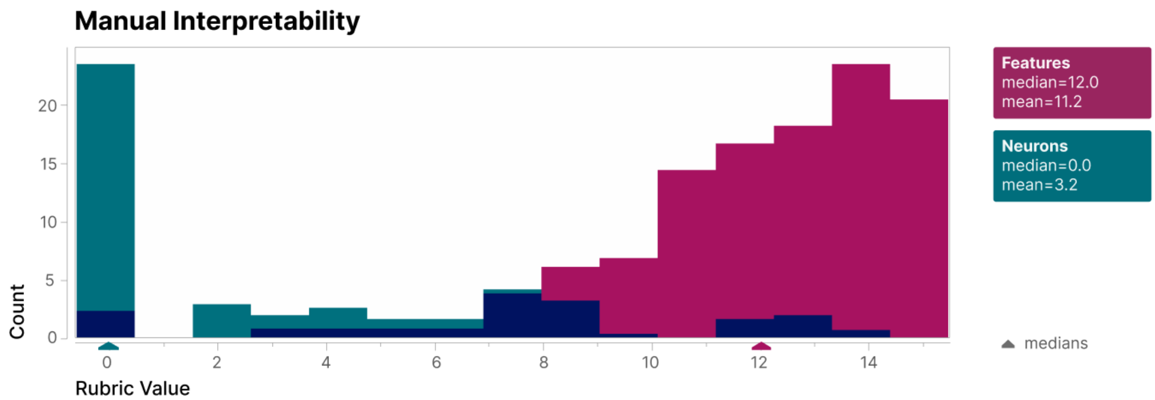


Figure 4.38: ***Human Analysis of Feature and Neuron Interpretability***

scored a 12, indicating that the annotator had a confident, specific, consistent hypothesis that made sense in terms of the logit output weights.

Automated Interpretability – Activations

To analyze features at a larger scale, we turned to automated interpretability (Bills *et al.*, 2023; Hernandez *et al.*, 2021). Following the approach of Bills *et al.* (2023), we have a large language model, Anthropic's Claude, generate explanations of features using examples of tokens where they activate. Next, we have the model use that explanation to predict new activations on previously unseen tokens.[24]

Like with the human analysis, we used samples across the full range of activation intervals to evaluate monosemanticity.[25] Concretely, for each feature, we computed the Spearman correlation coefficient between the predicted activation and the true activations for 60 dataset examples made up of nine tokens each, resulting in 540 predictions per feature. While only using completely random sequences would be the most principled approach to

[24] It's worth explicitly stating that our automated interpretability setup was designed to ensure that there's no leak of information about activation patterns, except for the explanation. For example, when predicting new activations, the model cannot see any true activations of that feature.

[25] This is distinct from the evaluation strategy of Bills et al., who calculated their correlation of predicted and true activations on a mixture of maximal dataset examples and random samples. For sparse features, which don't fire on most random samples, this effectively tests the model's ability to distinguish a feature's large activations from zero.

scoring, half of the examples are from across the feature intervals to get a more accurate correlation. See the appendix for additional information including the use of importance scoring to precisely counter-weight the bias of providing tokens the feature fires for (Bricken *et al.*, 2023c).

In agreement with the human analysis, Claude is able to explain and predict activations for features significantly better than for neurons (Fig. 4.39).[26]

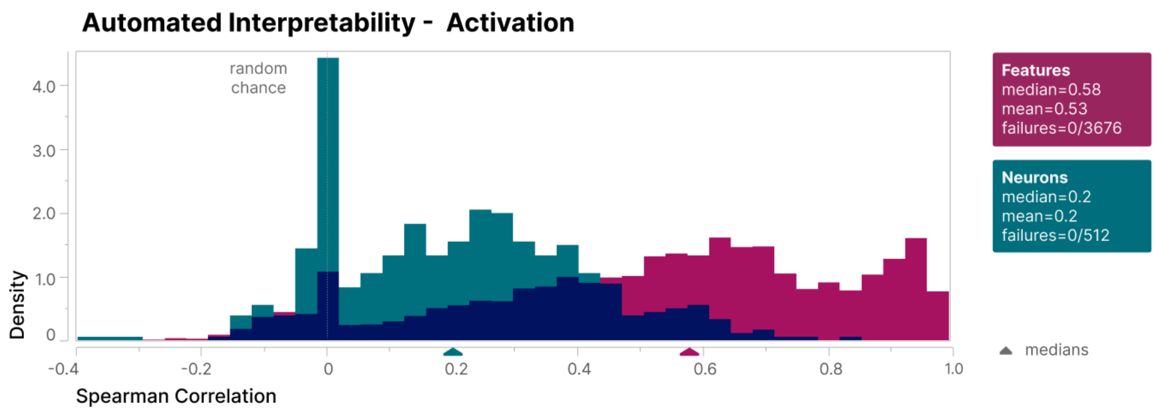


Figure 4.39: ***Automated Analysis of Feature and Neuron Interpretability.***

Automated Interpretability – Logit Weights

In our earlier analysis of individual features, we found that looking at the logits is a powerful tool for cross-validating the interpretability of features. We can take this approach in automated interpretability as well. Using the explanations of features generated in the previous analysis, we ask a language model to predict if a previously unseen logit token is something the feature should predict as likely to come next. This is then scored against a 50/50 mix of top positive logit tokens and random other logit tokens.

Randomly guessing would give a 50% accuracy, but the model instead achieves a 74% average across features, compared to a 58% average across neurons (Fig. 4.40). Failures here refer to instances where Claude failed to reply in the correct format for scoring.

[26]In instances where Claude predicts a constant score, most often all 0s, a correlation can't be computed and

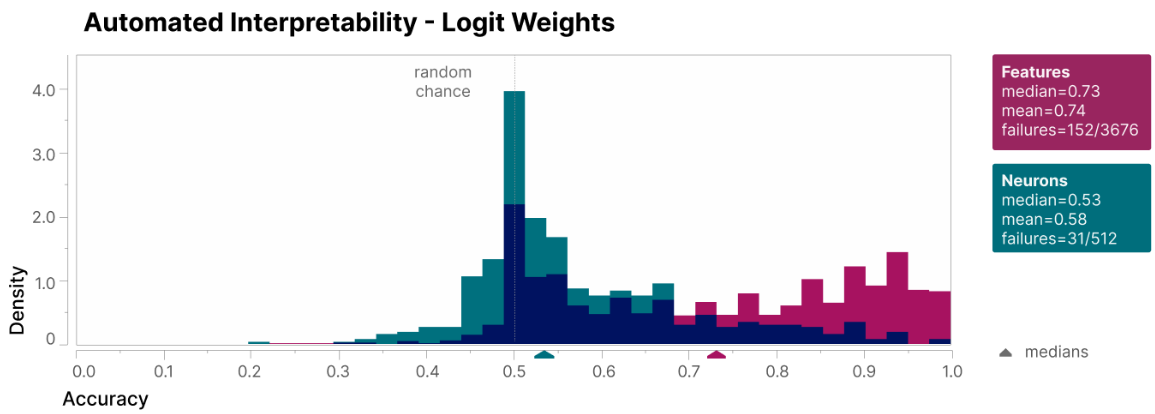


Figure 4.40: ***Automated Logits Analysis of Feature and Neuron Interpretability.***

Activation Interval Analysis

In addition to studying features as a whole, in our manual analysis we can zoom in on portions of the feature activation spectrum using the feature intervals. As before, a feature interval is the set of examples with activations closest to a specific evenly-spaced fraction of the max activation. So, rather than asking if a feature seems interpretable, we ask whether a range of activations is consistent with the overall hypothesis suggested by the full spectrum of the feature's activation. This allows us to ask how interpretability changes with feature activation strength.

Shown in Figure 4.41, higher-activating feature intervals were more consistent with our interpretations than lower-activating ones. In particular: Many features show consistent activations across the entire activation spectrum. Some features show consistent activations across the top 60% of the activation spectrum, and then quickly become less interpretable as we look to smaller and smaller activations.

It is possible that this is a sign that our features are not quite right. For instance, if one of our features is at a slight angle to the feature we'd really like to have learned, that can show up as inconsistent behavior in the lower activation intervals.

---

we assign a score of zero which explains the uptick there.

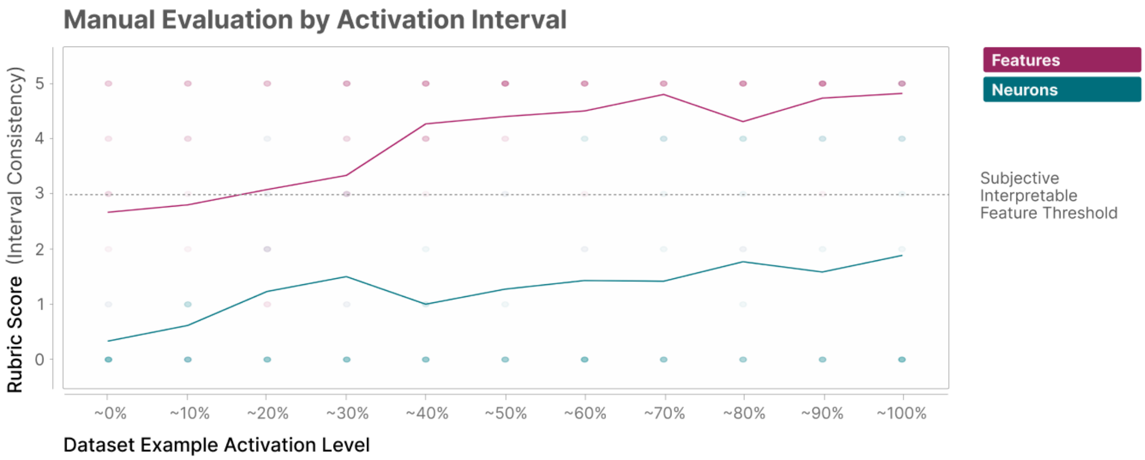


Figure 4.41: *Feature vs Neuron Human Interpretability Scores by Activation Level.*

Caveats

Our manual and automated interpretability experiments have a few caveats:

Feature activations are skewed towards the lower intervals. Most feature activations are quite small, and so fall in the lower (and less-interpretable) feature intervals. That said, as we found in our detailed analysis, most of the effect of a feature[27] is due to its higher activations, which are quite interpretable. The evaluated features are sampled uniformly from among the set of all features, and interpretability might be correlated with importance. One could imagine evaluating the features with the largest magnitude of activations, or the largest effect when ablated. Our sense is that these would be more interpretable than the randomly sampled features we considered. One could imagine a situation where the most important features in some sense were systematically less interpretable, though inspection of the most active features suggests the opposite.

Based on our inspection of many features in the visualization, we believe these caveats do not affect the experimental results. We encourage interested readers to open the visualization for A/1 and the corresponding neurons. You can sort by 'random' to get an unbiased sample and do your own version of the above experiment, or sort features by importance metrics

[27] with respect to the effect it has on the model outputs see e.g. the activation expected value plot in Arabic Feature's Activations Specificity Analysis.

such as max activation and max density to evaluate the final caveat above.

4.5.2 How much of the model does our interpretation explain?

We now turn to the question we're least able to answer – to what extent do these seemingly interpretable features represent the "full story" of the MLP? One could imagine posing this question in a variety of ways. What fraction of the MLP loss contribution have we made interpretable? How much model behavior can we understand? If there really are some discrete set of "true features", what fraction have we discovered?

One way to partly get at this question is to ask how much of the loss is explained by our features. For A/1, the run we've focused most on in this paper, 79% of the log-likelihood loss reduction provided by the MLP layer is recovered by our features. That is, the additional loss incurred by replacing the MLP activations with the autoencoder's output is just 21% of the loss that would be incurred by zero ablating the MLP. This loss penalty can be reduced by using more features, or using a lower L1 coefficient. As an extreme example, A/5 (`n_learned_sparse=131,072`, `l1_coefficient=0.004`) recovers 94.5% of log-likelihood loss.

These numbers should be taken with a significant grain of salt. The biggest issue is that framing this question in terms of fraction of loss may be misleading – we expect there to be a long-tail of features such that as the fraction of loss explained increases, more and more features are needed to explain the residual. Another issue is that we don't believe our features are completely monosemantic (some polysemanticity may be hiding in low activations), nor are all of them necessarily cleanly interpretable. With all of that said, our earlier analyses of individual features (e.g. the Arabic feature, base64 feature, etc.) do show that specific interpretable features are used by the model in interpretable ways – ablating them decreases probabilities in the appropriate way, and artificially activating them causes a corresponding behavior. This seems to confirm that the 79% of loss recovered is measuring something real, despite these caveats.

In principle, one could use automated interpretability to produce a better measure here:

replacing activations with those predicted from explanations. (We believe others in the community have recently been considering this!) The naive versions of this would be quite computationally expensive,[28] although there may be approximations. More generally, there is a much broader space of possibilities here. Perhaps there's a principled way to do this analysis in terms of single features – there are significant conceptual issues, but one might be able to formalize earlier notions of "feature importance" as an independent loss contribution a feature makes, and then analyze how much of that specific feature can be recovered.

Overall, we view the problem of measuring the degree to which a feature-based interpretation explains a model to be an important open question, where significant work is necessary on both defining metrics and finding efficient ways to compute them.

4.5.3 Do features tell us about the model or the data?

A model's activations reflect two things: the distribution of the dataset and the way that distribution is transformed by the model. Dictionary learning on activations thus mixes data and model properties, and intriguing properties of learned features may be attributed to either or both sources. Correlations in the data can persist after application of the first part of the model (up to the MLP), and it is in theory possible that the intriguing features we see are merely artifacts of dataset correlations projected into a different space. However, the use of those features by the second half of the model (MLP downprojection and unembedding) are not an input to dictionary learning, so the interpretability of the downstream effects of those features must be a property of the model.

To assess the effect of dataset correlations on the interpretability of feature activations, we run dictionary learning on a version of our one-layer model with random weights.[29]

[28]Naively, simulating a layer with 100k features would be 100,000 times more expensive than sampling a large language model such as Claude 2 or GPT-4 (we'd need to sample the explaining large model once for every feature at every token). At current prices, this would suggest simulating 100k features on a single 4096 token context would cost $12,500–$25,000, and one would presumably need to evaluate over many contexts.

[29]We generate a model with random weights by randomly shuffling the entries of each weight matrix of the trained transformer used in Run A. This guarantees that the distributions of individual weights match, and differences are due to structure.

The resulting features are here, and contain many single-token features (such as "span", "file", ".", and "nature") and some other features firing on seemingly arbitrary subsets of different broadly recognizable contexts (such as LaTeX or code). However, we are unable to construct interpretations for the non-single-token features that make much sense and invite the reader to examine feature visualizations from the model with randomized weights to confirm this for themselves. We conclude that the learning process for the model creates a richer structure in its activations than the distribution of tokens in the dataset alone.

To assess the interpretability of the downstream feature effects, we again use the three main approaches of the previous section:

1. Logit weight inspection. The logit weights represent the effect of each feature on the logits, and, as demonstrated in our earlier investigations of individual features, they are consistent with the feature activations for the base64, Arabic, and Hebrew features. The reader is invited to inspect logit weights for all features in the visualization.

2. Feature ablation. We set the value of a feature to zero throughout a context, and record how the loss on each token changes. Ablations are available for all features in the visualization.

3. Pinned feature sampling. We artificially pinned the value of a feature to a fixed high number and then sample from the model. We find in Figure 4.42 that the generated text matches the interpretation of the feature.

The empirical consistency of feature activations with their downstream effects across all these metrics provides evidence that the features found are being used by the model.

4.6 Phenomenology

Ultimately, the goal of our work is to understand neural networks. Decomposition of models into features is simply a means to this end, and one might very reasonably wonder if it's genuinely advancing our overall goal. So, in this section, we'll turn attention to the lessons

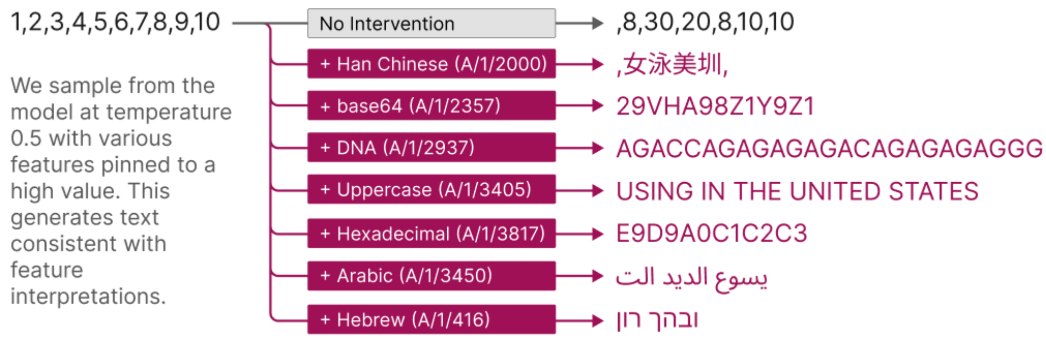


Figure 4.42: ***Steering with features validates their interpretations.***

these features can teach us about neural networks. (We've taken to calling this work of leveraging our theoretical understanding to reason about model properties phenomenology by analogy to phenomenology in physics, and the 2019 ICML workshop on phenomena in deep learning.)

One way to do this would be to give a detailed discussion of the features we've found (similar to (Olah *et al.*, 2020a; Goh *et al.*, 2021)), but we believe the best way to get a sense of the features we discovered is simply to browse the interface for exploring features which we've published alongside this paper. The features we find vary enormously, and no concise summary will capture their breadth. Instead, we will largely focus on more abstract properties and patterns that we notice. These abstract properties will be able to inform – although by no means answer – questions like "Are these the real features?" and "What is actually going on in a one-layer model?"

We begin by discussing some basic motifs and observations about features. We'll then discuss how the features we relate compare to features in other dictionary learning runs and in other models. This will suggest that features are universal and that dictionary learning can be understood as a process of feature splitting that reflects something deep about the geometry of superposition. Finally, we'll explore how features connect together into "finite state automata" as systems that implement more complex behaviors.

4.6.1 Feature Motifs

What kinds of features do we find in our model?

One strong theme is the prevalence of context features (e.g. DNA, base64) and token-in-context features (e.g. `the` in mathematics – A/0/341, < in HTML – A/0/20).[30] These have been observed in prior work (context features e.g. (Nelson *et al.*, 2022; Gurnee *et al.*, 2023; Bills *et al.*, 2023); token-in-context features e.g. (Nelson *et al.*, 2022; Smith, 2023a); preceding observations (Coenen *et al.*, 2019)), but the sheer volume of token-in-context features has been striking to us. For example, in A/4, there are over a hundred features which primarily respond to the token "the" in different contexts.[31] Often these features are connected by feature splitting (discussed in the next section), presenting as pure context features or token features in dictionaries with few learned features, but then splitting into token-in-context features as more features are learned.

Another interesting pattern is the implementation of what seem to be "trigram" features, such as a feature that predicts the `19` in `COVID-19` (A/2/12310). Such features could in principle be implemented with attention alone, but in practice the model uses the MLP layer as well. We also see features which seem to respond to specific, longer sequences of tokens. These are particularly striking because they may implement "memorization" like behavior – we'll discuss this more later.

Finally, it's worth noting that all the features we find in a one-layer model can be interpreted as "action features" in addition to their role as "input features". For example, a base64 feature can be understood both as activating in response to base64 strings, and also as acting to increase the probability of base64 strings. The "action" view can clarify some of the token-in-context features: the feature A/0/341 predicts noun phrases in mathematical

[30]From a purely theoretical lens, attention heads can largely implement "three point functions" (with two inputs, and an output). MLP layers are well positioned to instead implement N-token conjunctions, perhaps the most extreme of which are context features or token-in-context features. Thus, it is perhaps natural that we see many of these.

[31]Token-in-context features may offer a significant opportunity for simplifying analysis of the model – as Nelson *et al.* (2022) note, it may be possible to understand these features as two-dimensional family of features parameterized by a context and a token.

text, upweighting nouns like `denominator` and adjectives like `latter`. Consequently, while it activates most strongly on `the`, it also activates on adjectives like `special` and `this` which are also followed by noun phrases. This dual interpretation of features can be explored by browsing our interface. Several papers have previously explored interpreting neurons as actions (e.g. (Geva *et al.*, 2022)), and one-layer models are particularly suited to this, since it's a particularly principled way to understand the last MLP layer, and the only MLP in a one-layer model is the last layer.

4.6.2 Feature Splitting

One striking thing about the features we've found is that they appear in clusters. For instance, we observed above multiple base64 features, multiple Arabic script features, and so on. We see more of these features as we increase the total number of learned sparse features, a phenomenon we refer to as feature splitting. As we go from 512 features in A/0 to 4,096 features in A/1 and to 16,384 features in A/2, the number of features specific to base64 contexts goes from 1 to 3 to many more.

To understand how the geometry of the dictionary elements correspond to these qualitative clusters, we do a 2-D UMAP on the combined set of feature directions from A/0, A/1, and A/2 (Fig. 4.45).

We see clusters corresponding to the base64 and Arabic script features, together with many other tight clusters from specific contexts and a variety of other interesting geometric structures for other features. This confirms that the qualitative clusters are reflected in the geometry of the dictionary: similar features have small angles between their dictionary vectors.

We conjecture that there is some idealized set of features that dictionary learning would return if we provided it with an unlimited dictionary size. Often, these "true features" are clustered into sets of similar features, which the model puts in very tight superposition. Because the number of features is restricted, dictionary learning instead returns features which cover approximately the same territory as the idealized features, at the cost of being

Figure 4.43: *A UMAP plot showing Feature Splitting across three different sized dictionaries. The larger dictionaries feaures cluster in the same locations as the small dictionary's features.*

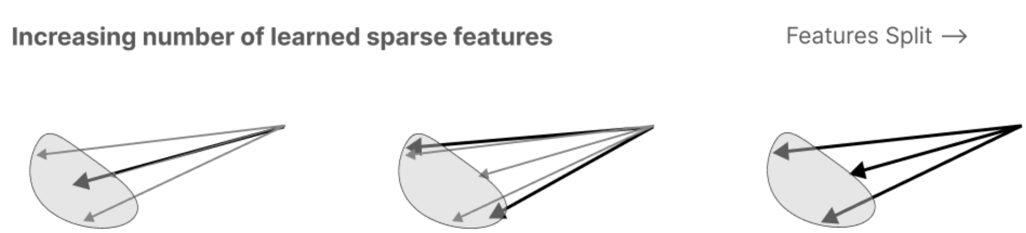


Figure 4.44: *Feature Splitting Theory*

somewhat less specific (Fig. 4.44).

In this picture, the reason the dictionary vectors of conceptually similar features are similar is that they are likely to produce similar behaviors in the model, and so should be responsible for similar effects in the neuron activations. For instance, it would be natural for a feature that fires on periods to predict tokens with a leading space followed by a capital letter. If there are multiple features that fire on periods, perhaps on periods in somewhat different contexts, these might all predict tokens with a leading space, and those predictions might well involve producing similar neuron activations. The combination of features being highly correlated and having similar "output actions", causes real models to have both denser and more structured superposition than what we observed in our previous toy models work (Elhage *et al.*, 2022).[32]

If this picture is true, it would be important for a number of reasons. It suggests that determining the "correct number of features" for dictionary learning is less important than it might initially seem. It also suggests that dictionary learning with fewer features can provide a "summary" of model features, which might be very important in studying large models. Additionally, it would explain some of the stranger features we observe in the process of dictionary learning, suggesting that these are either "collapsed" features which would make sense if split further (see "Bug" 1: Single Token Features), or else highly-specific "split" features which do in fact make sense if analyzed closely (see "Bug" 2: Multiple Features for a Single Context). Finally, it suggests that our basic theory of superposition in toy models is missing an important dimension of the problem by not adequately studying highly correlated and "action sharing" features. Example: Mathematics and Physics Features In this example, our coarsest run (with 512 learned sparse features) has three features describing tokens in different technical settings. Using the masked cosine similarity[33]

[32]Toy Models considered correlated features (see in particular organization of correlated features and collapsing of correlated features), but only features which were correlated in whether they were active (and not their value if active), and had nothing analogous to the similar "output actions" described here. Nonetheless, Toy Models' experiments may be a useful intuition pump, especially in noticing the distinction between similar features in superposition vs features collapsing into a single broader feature.

[33]For each pair of features, we compute the cosine similarity between their activations on the subset of

between feature activations, we are able to identify how these features refine and split in runs with more learned sparse features.

What we see is that the finer runs reveal more fine-grained distinctions between e.g. concepts in technical writing, and distinguish between the articles `the` and `a`, which are followed by slightly different sets of noun phrases (Fig. 4.45). We also see that the structure of this refinement is more complex than a tree: rather, the features we find at one level may both split and merge to form refined features at the next. In general though, we see that runs with more learned sparse features tend to be more specific than those with fewer.

It's worth noting that these more precise features reflect differences in model predictions as well as activations. The general `the` in mathematical prose feature (A/0/341) has highly generic mathematical tokens for its top positive logits (e.g. supporting `the denominator`, `the remainder`, `the theorem`), whereas the more finely split machine learning version (A/2/15021) has much more specific topical predictions (e.g. `the dataset`, `the classifier`). Likewise, our abstract algebra and topology feature (A/2/4878) supports `the quotient` and `the subgroup`, and the gravitation and field theory feature (A/2/2609) supports `the gauge`, `the Lagrangian`, and `the spacetime`.

Features which seemed like Bugs

"Bug" 1: Single-Token Features

When we limit dictionary learning to use very few learned sparse features, the features that emerge sometimes look quite strange. In particular, there are a large number of high-activation magnitude features which each only fire on a single token, and which seem to fire on every instance of that token. Such features are strange because the model could achieve the same effect entirely by learning different bigram statistics, and so should have no reason to devote MLP capacity to these. Similar features were also recently observed in a report by

---

tokens for which one of them fires. We repeat this, restricting to the subset on which the other fires, and take the greater of the two. We draw a connection between the features if this measure exceeds 0.4. This threshold was chosen to balance producing a small enough graph to visualize while also showing some of the richness of feature splitting. We omit the ultralow density features from this analysis.

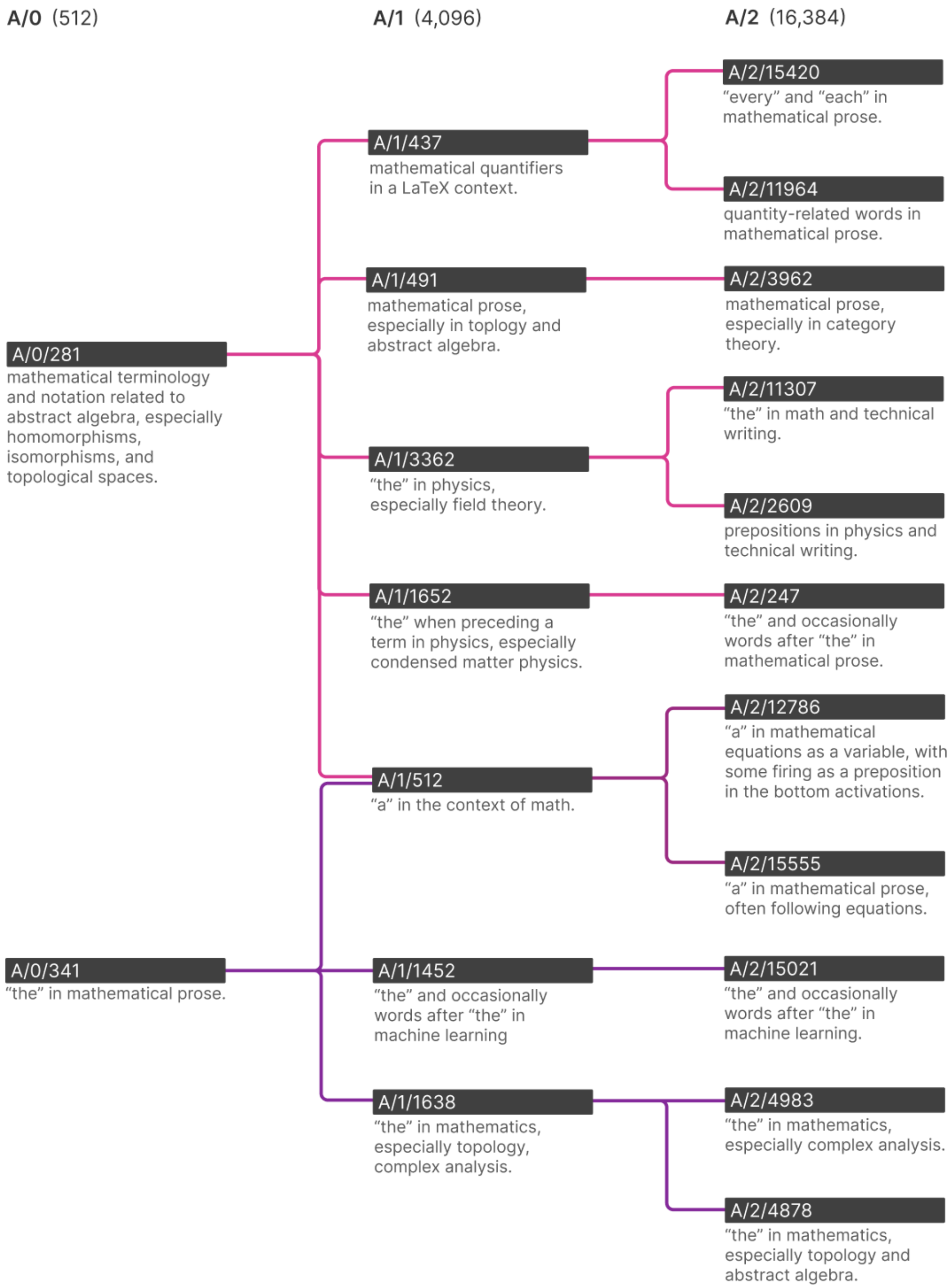


Figure 4.45: *Examples of Feature Splitting between dictionaries of increasing size.*

Smith (Smith, 2023a).

We believe that feature splitting explains this phenomenon: the model hasn't learned a single feature firing on the letter `P`,[34] for instance. Rather, it's learned many features which fire on `P` in different contexts[35], with correspondingly different effects on the neuron activations and output logits (see below). At a sufficiently coarse level dictionary learning cannot tell the difference between these, but when we allow it to use more learned sparse features, the features split and refine into a zoo of different `P` features that fire in different contexts (see Fig. **??**.

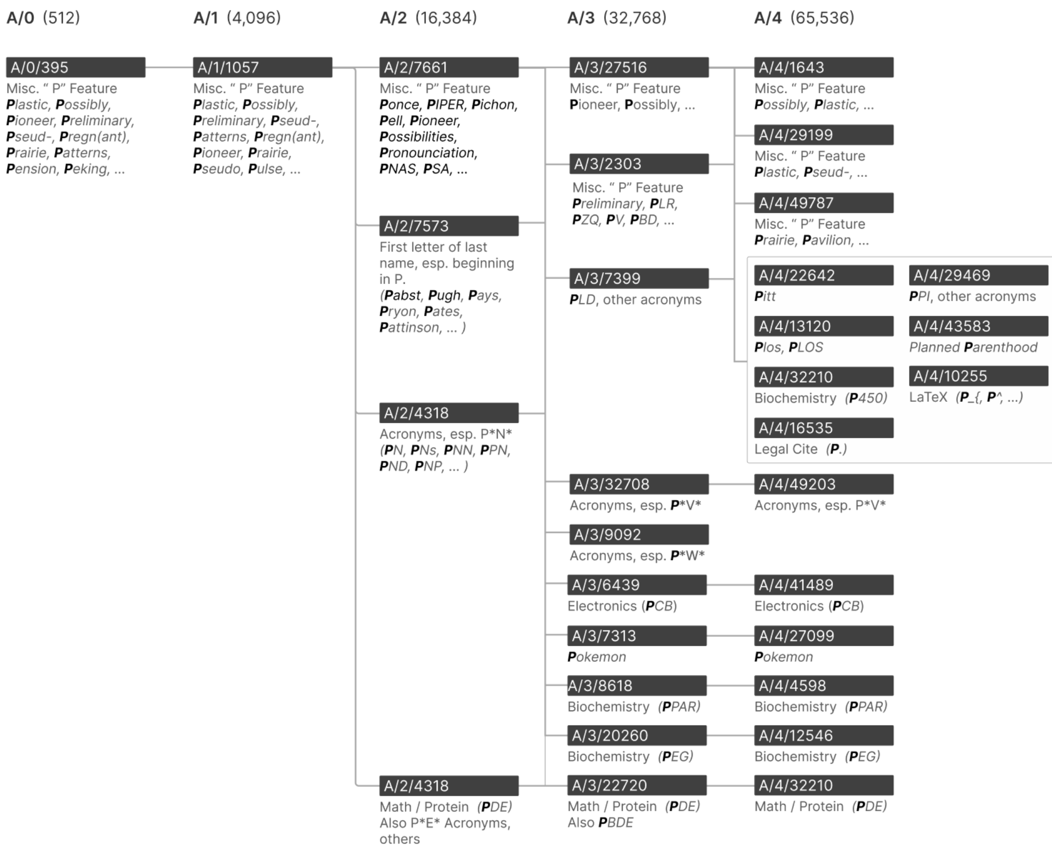


Figure 4.46: ***Examples of words that start with P undergoing multiple rounds of feature splitting.***

"Bug" 2: Multiple Features for a Single Context

[34]The features fire on the token " P" of words where it appears as the first token, such as `[ P][attern]`.

[35]In this instance we found the refined `P` features by manual inspection rather than by cosine similarity.

We also observed the converse, where multiple features seemed to cover roughly the same concept or context. For example, there were three features in A/1 which fired on (subsets of) base64 strings, and predicted plausible base64 tokens like `zf`, `mF`, and `Gp`. One of these features was discussed in detail earlier, where we showed it fired for base64 strings. But we also observed that it didn't fire for all base64 strings – why? And what are the other two features doing? Why are there three?

In A/0 (with 512 features), the story is simple. There is only one base64-related feature, A/0/45, which seems to activate on all tokens of base64-encoded strings. But in A/1, that feature splits into three different features whose activations seem to jointly cover those of A/0/45 as shown in Figure 4.47:

Figure 4.47: ***One base64 feature in the smaller dictionary with 512 features splits into three base64 features with the 4,096 feature dictionary.***

Two of these features seem relatively straightforward. A/1/2357 seems to fire preferentially on letters in base64, while A/1/2364 seems to fire preferentially on digits.

Comparing the logit weights of these features reveals that they predict largely the same sets of tokens, with one significant difference: the feature that firing on digits has much lower logit weights for predicting digits (Fig. 4.48. Put another way, if the present token is made of digits, the model will predict that the next token is a non-digit base64 token.

We believe this is likely an artifact of tokenization! If a single digit were followed by another digit, they would have been tokenized together as a single token; `[Bq][8][9][mp]` would never occur, as it would be tokenized instead as `[Bq][89][mp]`. Thus even in a random base64 string, the fact that the current token is a single digit gives information

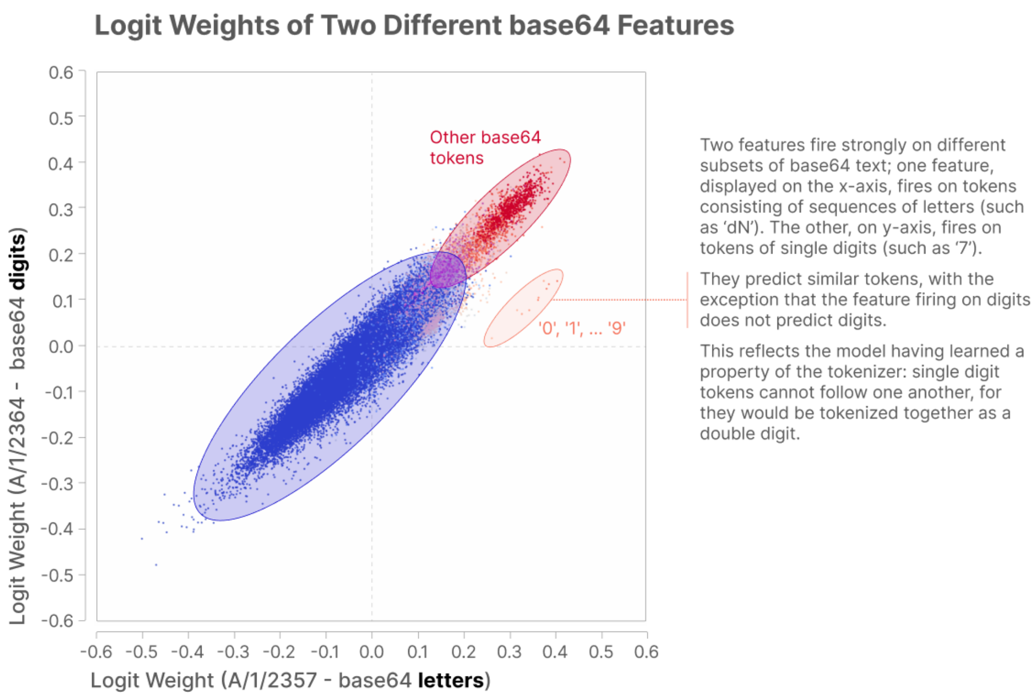


Figure 4.48: ***Two of the split Base64 Features.*** *These features both predict base64 tokens coming next but differ on their predictions for digits.*

about the next token.

But what about the third feature, A/1/1544? At first glance, there isn't an obvious rule for when it fires. But if we look more closely, we notice that it seems to respond to base64 strings which encode ASCII text.[36] If we look at the top dataset examples for each feature (Fig. 4.49), we find that examples for A/1/1544 contain substrings which decode as ASCII, while none of the top activating examples for A/1/2357 or A/1/2364 do:[37]

Figure 4.49: ***Top activating dataset examples for the three base64 features. One of them is specific to the base64 subset that is ASCII decodable.***

This pattern of investigation, where one looks at coarser sets of features to understand categories of model behavior, and then at more refined sets of features to investigate the subtleties of that behavior, may prove well adapted to larger models where the feature set is expected to be quite large.

It's also worth noting how dictionary learning features were able to surprise us here. Many approaches to interpretability are top-down, and look for things we expect. But who would have known that models not only have a base64 feature, but that they distinguish between distinct kinds of base64 strings? This reminds us of cases like high-low frequency detectors (Schubert *et al.*, 2021) or multimodal neurons (Goh *et al.*, 2021) where surprising and unexpected features were discovered in vision models.

[36]Our initial clue that A/1/1544 might fire on base64 strings encoding ASCII text was the token `ICAgICAg` which this feature particularly responds to, and corresponds to six spaces in a row.

[37]Determining when a dataset example encodes ASCII text is somewhat subtle because base64 can only be decoded to ASCII in groups of four characters, since four base64 characters encode triples of ASCII characters. Thus, we select substrings which – when decoded with the python base64 library – contain the maximal number of printable ASCII characters.

4.6.3 Universality

One of the biggest "meta questions" about features is whether they're universal (Li *et al.*, 2015; Olah *et al.*, 2020b)– do the same features form across different models? This question is generally important because it bears on whether the hard-earned lessons from studying one model will generalize to others. But it's especially important in the context of attempting to extract features from superposition because universality could provide significant evidence that the features we're extracting are "real", or at least reproducible.[38]

Earlier, we saw that all the features we performed detailed analyses of (e.g. the Arabic feature, or base64 feature) were universal between two one-layer models. But is this true for typical features in our model? And how broadly is it true – do we only observe the same feature if we train models of the same architectures on the same dataset, or do these features also occur in more divergent models? This section will seek to address these two questions. The first subsection will quantitatively analyze how widespread universality is between the two one-layer models we studied, while the second will compare the features we find to others reported in the literature in search of a stronger form of universality.

We observe substantial universality of both types.[39] At a high-level, this makes sense: if a feature is useful to one model in representing the dataset, it's likely useful to others, and if two models represent the same feature then a good dictionary learning algorithm should find it.

[38]In what sense does universality suggest features are "real"? One basic observation is that they suggest the features we're finding are not just artifacts of the dictionary learning process – or at least that if they come from the dictionary learning process, it's in some consistent way. But there are also several deeper ways in which it's suggestive. It means that, whatever the source of the features, we can talk about features as replicable, reliable, recurring units of analysis. It's also just a surprising observation that one would expect if a strong version of the features in superposition hypothesis was true and models were literally representing some finite, discrete set of features in superposition.

[39]In fact, we found some features so universal that we began to take it for granted as a basic tool in our workflow of evaluating dictionary learning runs. For example, the base64 feature – which we previously observed in SoLU models – was so consistently universal that its presence was a useful debugging heuristic.

Comparing features between two one-layer transformers

To compare features from different models, we need model-independent ways to represent a feature.

One natural approach is to think of a feature as a function assigning values to datapoints; two features would be similar in this sense if they take similar values over a diverse set of data. This general approach has been explored by a number of prior papers (e.g. (Erhan *et al.*, 2010; Li *et al.*, 2015; Olah, 2015; Raghu *et al.*, 2017)). In practice, this can be approximated by representing the feature as a vector, with indices corresponding to a fixed set of data points. We call the correlations between these vectors the activation similarity between features.

A second natural approach is to think of a feature in terms of its downstream effects; two features would be similar in this sense if their activation changes their models' predictions in similar ways. In our one-layer model, a simple approximation to this is the logit weights. This approximation represents each feature as a vector with indices corresponding to vocabulary tokens. We call the correlations between these vectors the logit weight similarity between features.

These two notions of similarity correspond to the correlations of the points in the two scatter plots we used when analyzing individual features earlier. We've reproduced the plots for the Arabic feature below in Figure 4.50:

For each feature in run A/1, we find the closest feature by activation similarity in run B/1, which is a different dictionary learning run trained on different activations from a different transformer with different random seeds but otherwise identical hyperparameters. We find that many features are highly similar between models, with features in A/1 having a median activation correlation of 0.72 with the most similar feature from B/1. (We perform the same analysis finding the closest neurons between the transformers, and find significantly less similarity, with median activation correlation 0.46 as plotted in Fig. 4.51) The features with low activation correlation between models may represent different "feature splittings" in the dictionaries learned or different "true features" learned by the base models.

A natural next question is whether features that fire on the same tokens also have the

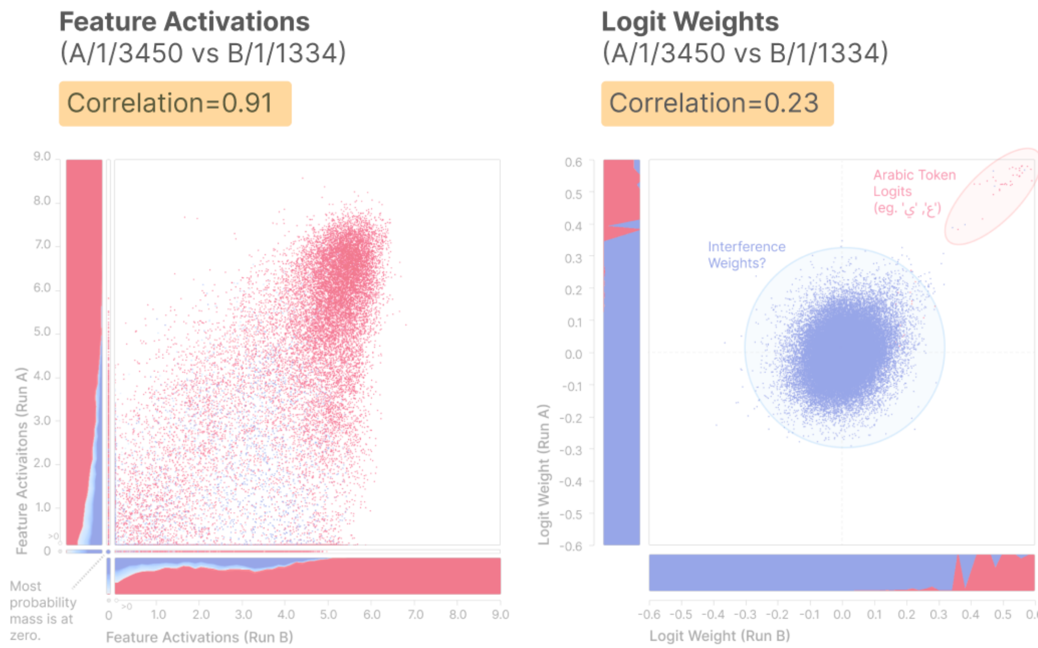


Figure 4.50: *Arabic Feature Universality between runs.*

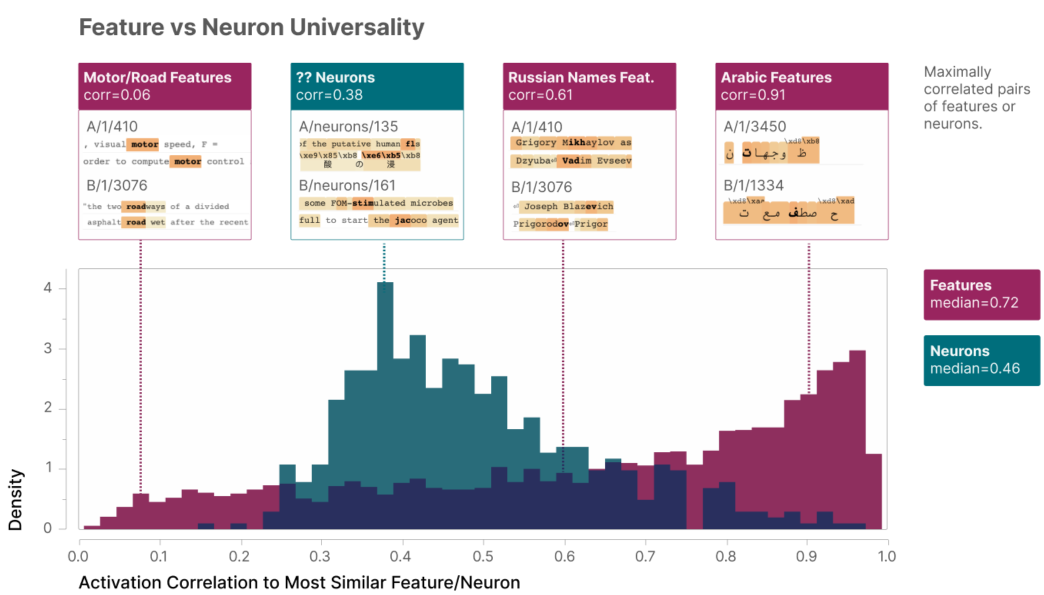


Figure 4.51: *Feature and Neuron Universality. Features are much more universal.*

same logit effects. That is, how well do activation similarity and logit weight similarity agree?

Some gap between the two is visible for the Arabic feature above: the "important tokens" for the features' effects (the ones in Arabic script) are upweighted by features from both models, but there is a large cloud of tokens with smaller effects that appear to almost be isotropic noise, resulting in a logit weight correlation of just 0.23, significantly below the activation correlation of 0.91.

In the scatterplot of Figure 4.52 below, we find that this kind of disagreement is widespread.

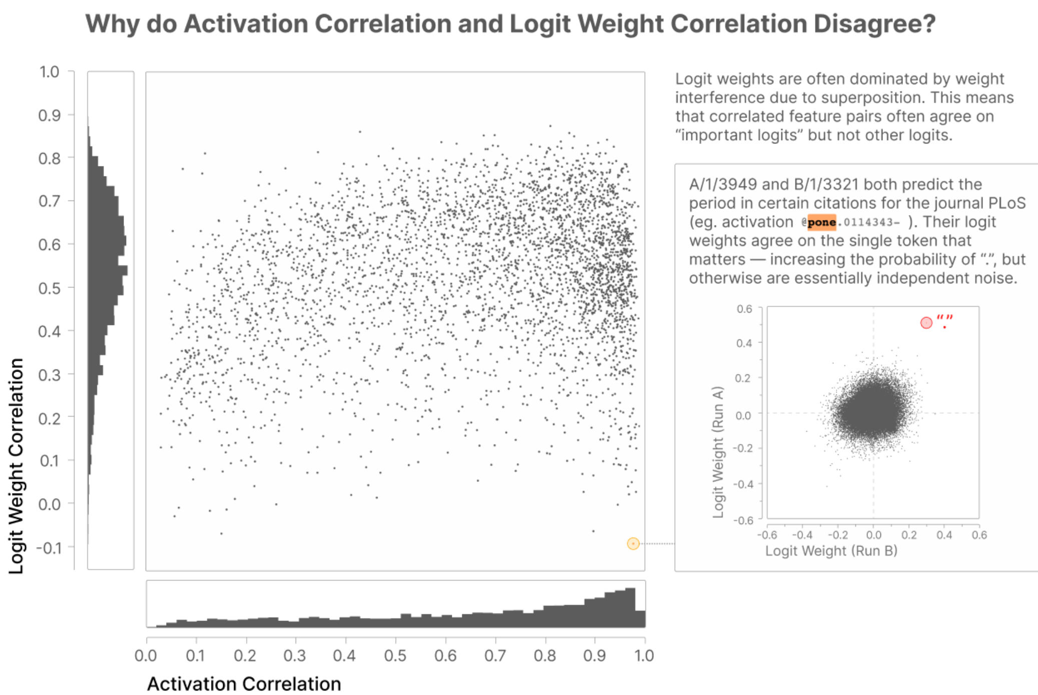


Figure 4.52: ***Similarity Gap.***

The most dramatic example of this disparity is for the features A/1/3949 and B/1/3321, with an activation correlation of 0.98 but a negative logit weight correlation. These features fire on `pone` (and occasionally on `pgen` and `pcbi`) as abbreviations for the journal name PLOSOne in citations, like `@pone.0082392`, and predict the `.` that follows.[40]

[40]The feature is bimodal, monosemantic in the larger of the modes, and fires on 0.02% of tokens. This means

Zooming in on the logit weight scatterplot (inset in the figure above), we see that only the `.` token has high logit weight in both models, and that every other token is in the 'interference' portion of the logit weight distribution. Indeed, the model may simply not care about what the feature does to tokens which were already implausible because they are suppressed by the direct path, attention layer, or other features of the MLP.

We want to measure something more like "the actual effect a feature has on token probabilities." One way to get at this would be to compute a vector of ablation effects for every feature on every data point; pairs of features whose ablations hurt the model's predictions on the same tokens must have been predicting the same thing. Unfortunately, this would be rather expensive computationally. Instead, we scale the activation vector of a feature by the logit weights of the tokens that empirically come next in the dataset to produce an attribution vector.[41] Correlations between those vectors provide an attribution similarity that combines both the activity of the feature with the effect it has on the loss. We find that the attribution similarity correlates quite highly in Fig. 4.53 with the activation similarity, meaning that features that were coactive between models were useful at predicting the same tokens.

In light of this, we feel that the activation correlation used throughout the paper is in fact a good proxy for both notions of universality in the context of our one-layer models.

Comparing features with the literature

So far, we've established that many of our features are universal in a limited sense. Features found in one of our transformers can also be found in an alternative version trained with a different random seed. But this second model has an identical architecture and was trained

that at least 1 in 10,000 tokens in the Pile dataset are abbreviations for PLoS journals in citations! This is an example of how inspecting features can reveal properties of the dataset, in this case the strong bias of the Pile towards scientific content.

[41]Suppose feature f_i has logit weights v_{ik} for $k \in \{1, \ldots, n_{\text{vocab}}\}$. At a given token t_j, we compute the activation of the feature $f_i(t_j)$ and multiply it by the logit weight $v_{it_{j+1}}$ of the token t_{j+1} that comes next to get an attribution score of $f_i(t_j)v_{it_{j+1}}$. The attribution vector is given by stacking the attribution scores for a random sampling of datapoints. This approximates the classic attribution method of multiplying the gradient by the activation, differing in that we ignore the denominators of the softmax and the layer norm.

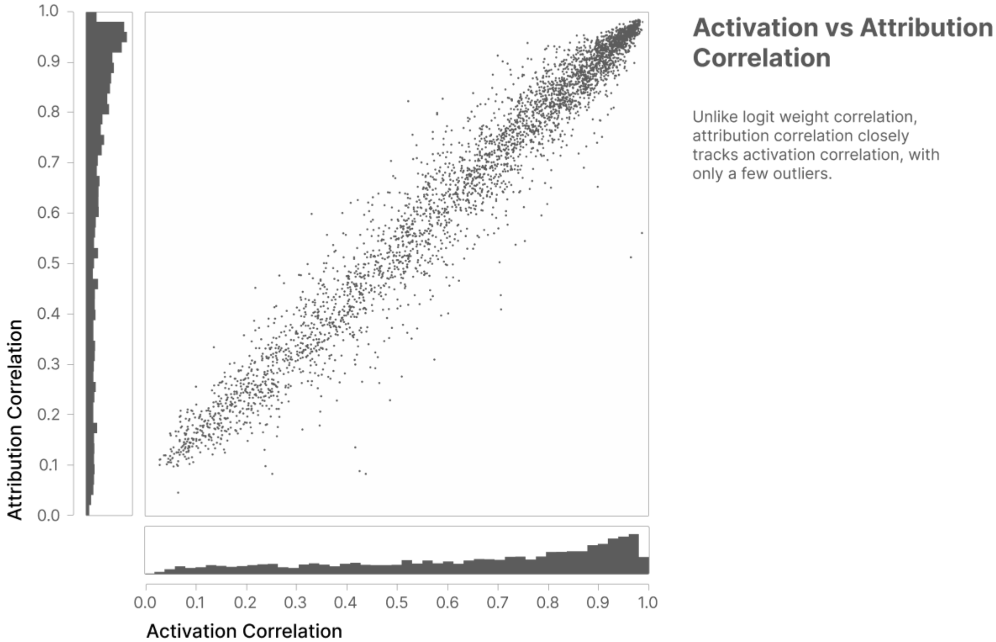


Figure 4.53: ***Activation vs Attribution Correlation.***

on identical data. This is the most minimal version of universality one could hope for. Despite this, we believe that many of the features we've found are universal in a deeper sense, because very similar features have been reported in the literature before.

The first comparison which struck us is that many features seem quite similar to neurons we previously found in one-layer SoLU models, which use an activation function designed to make neurons more monosemantic (Nelson *et al.*, 2022). In particular, we observed a base64 neuron, hexadecimal neuron, and all caps neuron in our SoLU investigations, and base64 (A/0/45), hexademical (A/0/119), and all caps (A/0/317) features here. Discussion of some of these neurons can be found in Section 6.3.1 of the SoLU paper.

We also find many features similar to Smith (2023a), who applies dictionary learning to the residual stream. In addition to us also observing preponderance of single token features they note (see our interpretation of this phenomenon), we find a similar German detector (e.g. A/0/493) and similar title case detectors (e.g. A/0/508). Likewise, we find a number of features similar to Gurnee *et al.* (2023), including a "prime factors" feature (A/4/22414) and a French feature (A/0/14).

At a more abstract level, many features we find seem similar to features reported in multimodal models by Goh *et al.* (2021). For example, we find many similar features including an Australia feature (A/3/16085), Canada feature (A/3/13683), Africa feature (A/3/14490), and Israel-Palestine feature (A/3/739) which predict locations in those regions when grammatically appropriate. This vaguely mirrors "region neurons" reported by Goh et al.'s paper. For other families of features, the parallels are less clear. For example, one of the most striking results of Goh et al. was person detector neurons (similar to famous results in neuroscience). We find some features that are person detectors in very narrow contexts, such as responding to a person's name and predicting appropriate next words, or predicting their name (e.g. A/1/3240 is somewhat similar to Goh et al.'s Trump neuron), but they seem quite narrow. We also don't find features that seem clearly analogous to Goh et al.'s emotion neurons.

4.6.4 "Finite State Automata"

One of the most striking phenomena we've observed in our study of the features in one-layer models is the existence of "finite state automata"-like assemblies of features. These assemblies aren't circuits in the conventional sense – they're formed by one feature increasing the probability of tokens, which in turn cause another feature to fire on the next step, and so on.[42]

The simplest example of this is features which excite themselves on the next token, forming a single node loop (Fig. 4.54). For example, a base64 feature increases the probability of tokens like `Qg` and `zA` – plausible continuations which would continue to activate it.

It's worth noting that these examples are from A/0, a dictionary learning run which is not overcomplete (the dictionary dimensionality is 512, equal to the transformer MLP

[42]The "finite state automata"-esque feature assemblies are also different from circuits in that the model didn't learn them to work together. Rather, in the course of learning to autoregressively model text, it learned features that interact via the token stream because of patterns in the real datasets. In contrast, a language model trained with reinforcement learning might have systems like this – circuits whose feature components interact via generated tokens – which co-evolved and adapted to work together during RL training.

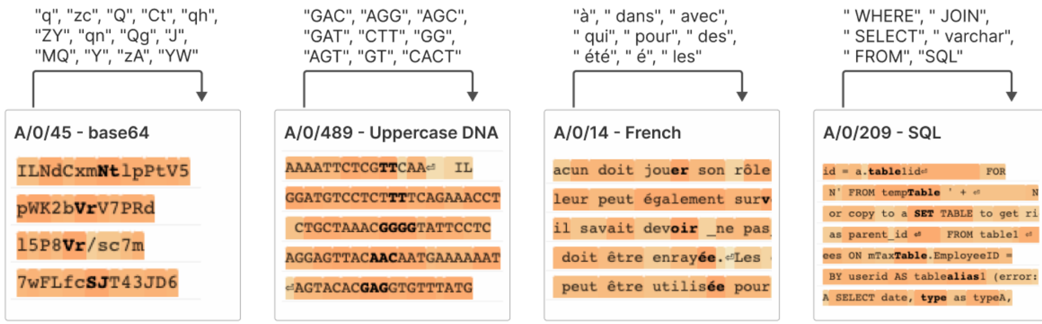


Figure 4.54: *Single state Finite State Machines.*

dimension). As we move to runs with larger numbers of features, the central feature will experience feature splitting, and become a more complex system.

Let's now consider a two-node system for producing variables in "all caps snake case" (e.g. `ARRAY_MAX_VALUE`) in Figure 4.55. One node (A/0/207) activates on the all caps text tokens, the other (A/0/358) on underscores:

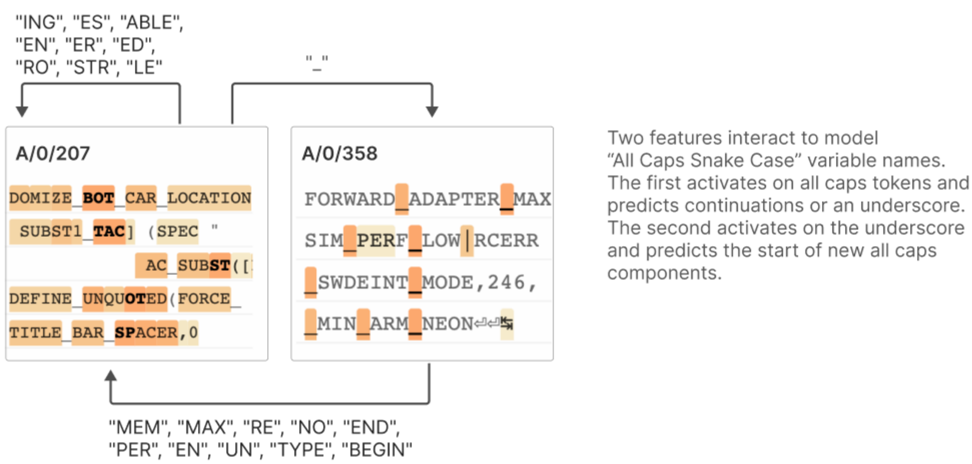


Figure 4.55: *Two state Finite State Machine.*

This type of two-node system is quite common for languages where Unicode characters are sometimes split into two tokens. (Again, with more feature splitting, these would expand into more complex systems.)

For example, Tamil Unicode characters (block U+0B80–U+0BFF) are typically split into two tokens. For example, the character (U+0BA3) is tokenized as `xe0`

`xae` followed by

`xa3`. The first part (

`xe0` or

`xe0`

`xaf`) roughly specifies the Unicode block, while the second component specifies the character within that block. Thus, it's natural for the model to alternate between two features, one for the Unicode prefix token, and one for the suffix token.

A more complex example is Chinese. While many common Chinese characters get dedicated tokens, many others are split. This is further complicated by Chinese characters being spread over many Unicode blocks, and those blocks being large and cutting across many logical blocks specified in terms of bytes. To understand the state machine the model implements to handle this, the key observation is that complete characters are similar to the "suffix" part of a split character: both can be followed by either a new complete character, or a new prefix. Thus, we observe two features, one of which fires on either complete characters or the suffix (predicting either a new complete character, or a prefix), while the other only fires on the prefixes and predicts suffixes (Fig. 4.56).

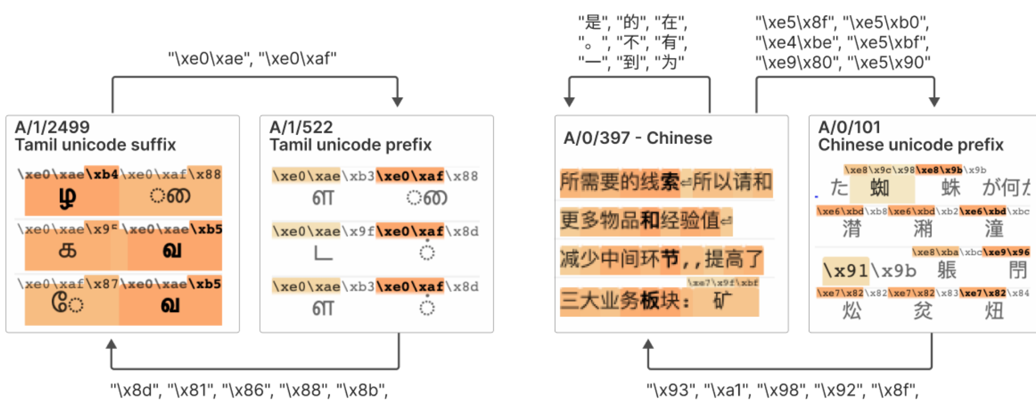


Figure 4.56: *Two state Finite State Machines.*

Let's now consider a very simple four node system which models HTML (Fig. 4.57). The "main path" through it is:

- A/0/20 fires on open tags and predicts tag names
- A/0/0 fires on tag names and predicts tag closes
- A/0/30 fires on tag closes and predicts whitespace
- A/0/494 fires on whitespace and predicts new tag opens.

The full system can be seen below:

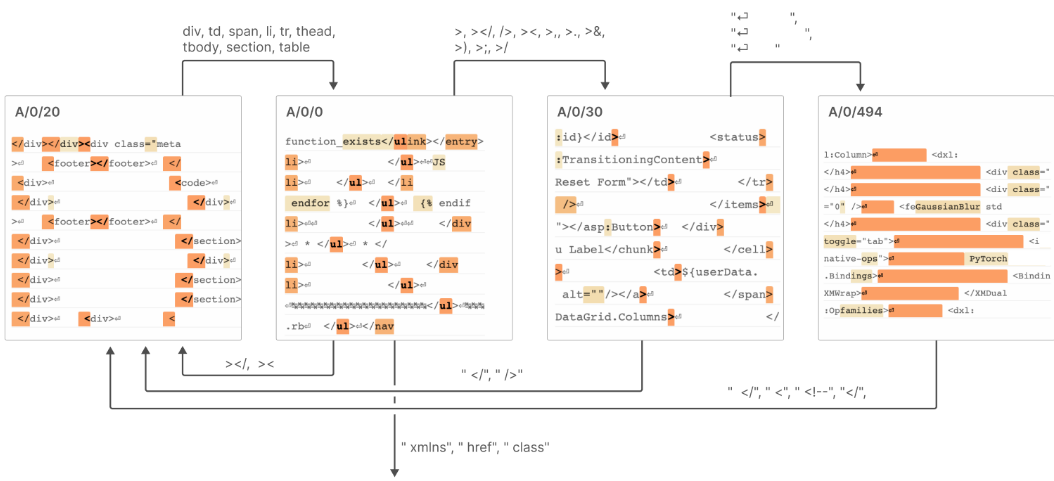


Figure 4.57: *Four state Finite State Machine for HTML formatting.*

Keep in mind that we're focusing on the A/0 features where this is very simple – if we looked at A/1, we'd find something much more complex! One particularly striking shortcoming of the A/0 features is that they don't describe what happens when A/0/0 emits a token like `href`, which leads to a more complex state.

It's important to note that these features can be quite contextual. There are several features related to IRC transcripts which form a totally different finite state automata like system (Fig. 4.58):

A prototypical sample this might generate is something like <nickonia> lol ubuntu ;). Presumably the Pile dataset heavily represents IRC transcripts about linux.

One particularly interesting behavior is the apparent memorization of specific phrases. This can be observed only in runs with relatively large numbers of features (like A/4). In

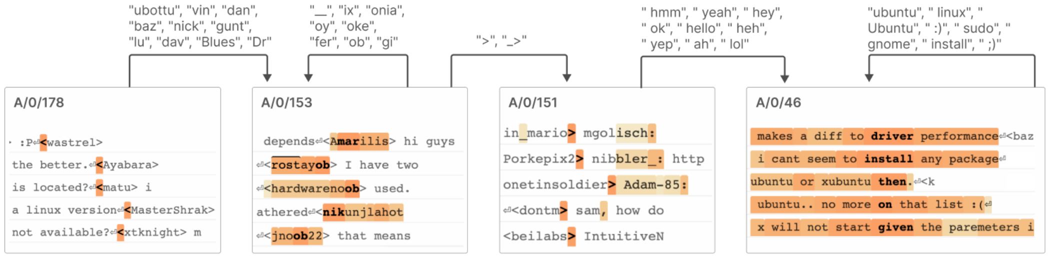


Figure 4.58: *Four state Finite State Machine for an IRC server.*

the following example (Fig. 4.59), a sequence of features seem to functionally memorize the bolded part of the phrase `MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE`. This is a relatively standard legal language, and notably occurs in the file headers for popular open source software licenses, meaning the model likely saw it many times during training.

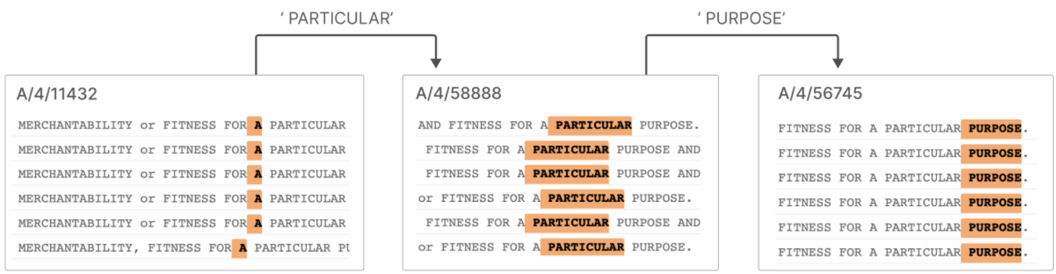


Figure 4.59: *Four state Finite State Machine for Legal Language.*

This seems like an example of the mechanistic theory of memorization we described in Henighan *et al.* (2023) – we observe features which appear to be relatively binary and respond to a very specific situation. This might also be seen as an instance of mechanistic anomaly detection (Christiano, 2022): the model behaves differently in a specific, narrow case. It's somewhat surprising that something so narrow can be found in a model with only 512 neurons; from this perspective it's an interesting example of superpositions' ability to embed many things in few neurons. On the other hand, because these mechanisms are buried deep in superposition, they are likely very noisy.

4.7 Related Work

Superposition and attempts to resolve it have deep connections to many lines of research, including general investigations of interpretable features, linear probing, compressed sensing, dictionary learning and sparse coding, theories of neural coding, distributed representations, mathematical frames, vector symbolic architectures, and much more. Rather than attempt to do justice to all these connections here, we refer readers to the related work section of Toy Models of Superposition (Elhage *et al.*, 2022) where we discuss these topics in depth, and also to our essay Distributed Representations: Composition & Superposition (Olah, 2023). Instead, we'll focus our discussion on work connected to our attempts to solve superposition, and also more recent advancements in our understanding of superposition.

4.7.1 Superposition

Since we published Toy Models of Superposition there has been significant further work attempting to better understand superposition. We briefly summarize below.

Is superposition real? Gurnee *et al.* (2023) demonstrate some compelling examples of features which may be in superposition, using sparse linear probes. Separately, an exchange between Li *et al.* (2022) and Nanda *et al.* (2023) seems like an update on whether the general picture of features as directions is correct; Li et al. seemed to show that it wasn't, putting the hypothesis in jeopardy, which was then resolved by Nanda et al.

When and why does superposition occur? Scherlis *et al.* (2022) provide a mathematical framework for thinking about monosemanticity vs polysemanticity. lsgos (2023) explored the effects of dropout on toy models of superposition.

Memorization – In Henighan *et al.* (2023), we studied the same toy model as in Toy Models of Superposition, but this time trained on many repetitions of finite-sized datasets. We found that small datasets are memorized in superposition, instead of generalizing features in the case of large datasets. Hobbhahn (2023) replicated some of these findings, and further showed extensions to other settings including bottlenecks between layers (analogous to the residual stream between MLP layers). Subsequently, in a monthly update we examined the boundary between the memorization and generalization regimes and found a sharp phase transition, as well as dataset clustering in superposition on the memorization side of the boundary.

4.7.2 Disentanglement and Architectural Approaches

There is also a rich and related literature on disentanglement, which seeks to find representations of data that separate out (disentangle) conceptually-distinct phenomena influencing the data. In contrast to superposition, this work typically seeks to find a number of factors of variation or features which are equal to the dimensionality of the space being represented, whereas superposition seeks to find more.

This is often approached as an architecture/training-time problem. For instance, Kim and Mnih (2018) proposed a method that pushes variational autoencoders to disentangle factors by encouraging independence across dimensions. Similarly, Chen *et al.* (2016) developed a Generative Adversarial Network approach that attempts to disentangle factors by maximizing the mutual information between a small subset of factors and the dataset. And Makhzani and Frey (2013) use a TopK activation to encourage sparsity and hence disentanglement. See Bengio *et al.* (2013) and Räuker *et al.* (2023) for a discussion of other such approaches.

Framed this way, some architectural approaches to superposition may also be understood as attempts at disentanglement. For instance, in Nelson *et al.* (2022), we proposed the SoLU activation function, which increases the number of interpretable neurons in transformers by encouraging features to align to the neuron basis. Unfortunately, it appears that training models with the SoLU activation function may make some neurons more interpretable at the cost of making others even less interpretable than before.

Similarly, Jermyn *et al.* (2022) studied MLP layers trained on a compressed sensing task and found multiple equal-loss minima, with some strongly polysemantic and others strongly monosemantic. This suggested that training interventions could steer models towards more monosemantic minima, though subsequent investigations on more realistic tasks suggested that the equal-loss property was specific to the chosen task.

These two examples of attempts to tackle superposition through architecture, and the challenges they encountered, highlight a key distinction between the problems of disentanglement and that of superposition: disentanglement fundamentally seeks to ensure that the dimensions in the model's latent space are disentangled, whereas superposition hypothesizes that this disentanglement typically hurts performance (since success would require throwing away many features), and that models will typically respond to disentangling interventions by making some features more strongly entangled (as was found by both Mahinpei *et al.* (2021) and Nelson *et al.* (2022) in somewhat different contexts).

4.7.3 Dictionary Learning and Features

Our work builds on a longer tradition of using dictionary learning and sparse autoencoders to decompose neural network activations.

Early work in this space focused on word embeddings and other non-transformer neural networks. Faruqui *et al.* (2015) and Arora *et al.* (2018) both found linear structure in word embeddings using sparse coding approaches. Subramanian *et al.* (2018) similarly found linear factors for word embeddings, in this case using a sparse autoencoder. Zhang *et al.* (2019) solved a similar problem using methods from dictionary learning while Panigrahi *et al.* (2019) approached this with Latent Dirichlet Allocation.

More recently, a number of works have applied dictionary learning methods to transformer models. Yun *et al.* (2021) applied dictionary learning to the residual stream of a 12-layer transformer to find an undercomplete basis of features.

At this point, our work in Toy Models (Elhage *et al.*, 2022) advocated for dictionary learning as a potential approach to superposition. This motivated a parallel investigation by our colleagues Cunningham et al., published as a series of interim reports (Sharkey *et al.*, 2022; Cunningham and Smith, 2023; Huben, 2023; Smith, 2023b,a; Cunningham, 2023) with very similar themes to this paper, culminating in a manuscript (Cunningham *et al.*, 2023). We've been excited to see so many corroborating findings between our work.

In their interim reports, Sharkey *et al.* (2022) used sparse autoencoders to perform dictionary learning on a one-layer transformer, identifying a large (overcomplete) basis of features. (Sharkey et al. deserve credit for focusing on dictionary learning and especially the sparse autoencoder approach, while our investigation was only exploring it as one of several approaches in parallel.) This work was then partially replicated by Cunningham and Smith (2023) and Huben (2023). Next, Smith (2023b) used an autoencoder to find features in one MLP layer of a six-layer model. The resulting features appear interpretable, e.g. detecting 'dollar;' in the context of LaTeX equations. In follow up work, Smith then extended this approach to the residual stream of the same model, identifying a number of interesting features (see earlier discussion). Building on these results, Cunningham (2023) applied autointerpretability techniques from Bills *et al.* (2023) to features in the residual stream and an MLP layer of the same six-layer model, finding that the features discovered by the sparse autoencoder are substantially more interpretable than neurons.

4.8 Discussion

4.8.1 Theories of Superposition

Coming into this work, our understanding of superposition was mostly informed by Toy Models (Elhage *et al.*, 2022). This gave us a picture one might call the isotropic superposition model. Features are discrete, one-dimensional objects which repel from each other due to interference, creating a roughly evenly spaced organization of feature directions.

This work has persuaded us that our previous model was missing something crucial. At a minimum, features seem to clump together in higher density groups of related features (Fig. 4.60). One explanation for this (considered briefly by Toy Models) is that the features may have correlated activations – firing together. Another – which we suspect to be more central – is that the features produce similar actions. The feature which fires on single digits in base64 predicts approximately the same set of tokens as the feature firing on other characters in base64, with the exception of other digits; these similar downstream effects manifest as geometrically close feature directions.

Moreover, it isn't clear that features need to be one-dimensional objects (encoding only some intensity). In principle, it seems possible to have higher-dimensional "feature manifolds" (see earlier discussion here).

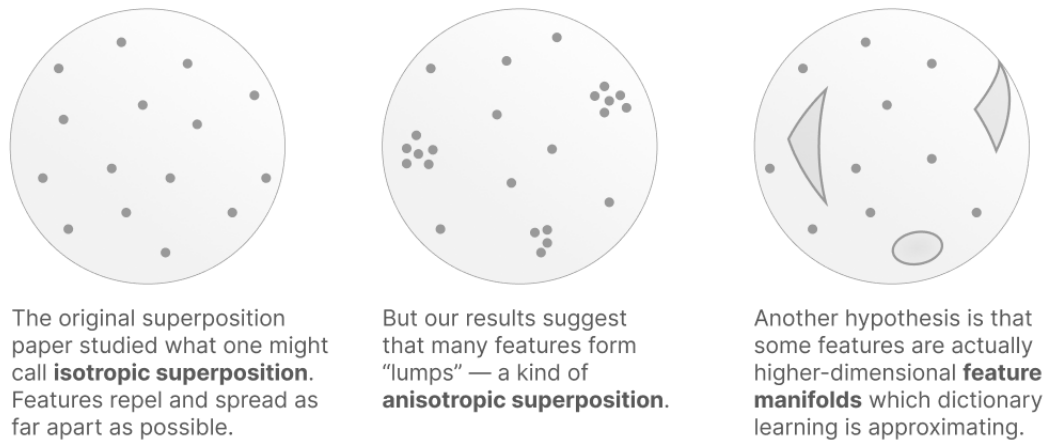


Figure 4.60: *Different superposition geometries.*

These hypotheses are not mutually exclusive. The convex hull of several correlated features might be understood as a feature manifold. On the other hand, some manifolds would not admit a unique description in terms of a finite number of one-dimensional features (Fig. 4.61). (Perhaps this accounts for the continued feature splitting observed above.)

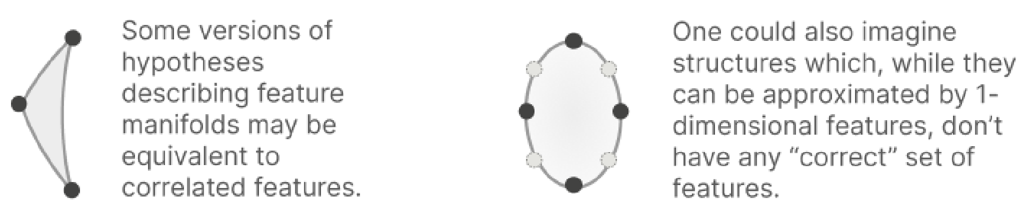


Figure 4.61: *Correlated Features.*

Nevertheless, these experiments have left us more confident that some version of the superposition hypothesis (and the linear representation hypothesis) is true. The number of interpretable features found, the way activation level seems to correspond to "intensity" or "confidence," the fact that logit weights mostly make sense, and the observation of "interference weights": all of these observations are what you would expect from superposition.

Finally, we note that in some of these expanded theories of superposition, finding the "correct number of features" may not be well-posed. In others, there is a true number of features, but getting it exactly right is less essential because we "fail gracefully", observing the "true features" at resolutions of different granularity as we increase the number of learned features in the autoencoder.

4.8.2 Are "Token in Context" Features Real?

One of the most common motifs we found were "token-in-context" features. They also represent many of the features that emerge via feature splitting with increasing dictionary size. Some of these are intuitive – borrowing an example from (Coenen *et al.*, 2019), it makes sense to represent "die" in German (where it's the definite article) as distinct from "die" in English (where it means "death" or "dice").

But why do we see hundreds of different features for "the" (such as "the" in Physics, as distinct from "the" in mathematics)? We also observe this for other common words (e.g. "a", "of"), and for punctuation like periods. These features are not what we expected to find when we set out to investigate one-layer models!

To make the question a bit more precise, it is helpful to borrow the language and examples of local vs compositional representations (Olah, 2023; Thorpe, 1989). Individually representing token-context pairs (such as "the" in Physics) is technically a "local code". The more intuitive way to represent this would instead be a "compositional code" – representing "the" as an independent feature from Physics. So the thing we really want to ask is why we're observing a local code, and whether it's really what's going on. There are two hypotheses:

The underlying transformer uses a compositional code, and a quirk of our dictionary learning scheme produces features using a local code. The underlying transformer is genuinely using a local code (at least in part), and dictionary learning is correctly representing this.

If the former holds, then better dictionary learning schemes may help uncover a more compositional set of features from the same transformer. Local codes are sparser than compositional codes, and our L1 penalty may be pushing the model too far towards sparsity.

However, we believe the second hypothesis is likely to hold to some extent. Let's consider the example of "the" in Physics again, which predicts noun phrases in Physics: if the model represented "the" and Physics context independently, it would be forced to have logits be the sum of "upweight tokens which come after the" and "upweight tokens which occur in Physics". But the model might wish to have "sharper" predictions than this, which is only possible with a local code.

4.8.3 Future Work

Scaling Sparse Autoencoders. Scaling the application of sparse autoencoders to frontier models strikes us as one of the most important questions going forward. We're quite hopeful that these or similar methods will work – Cunningham *et al.* (2023)'s work seems to suggest this approach can work on somewhat larger models, and we have preliminary results that point in the same direction. However, there are significant computational challenges to be overcome. Consider an autoencoder with a 100× expansion factor applied to the activations of a single MLP layer of width 10,000: it would have 20 billion parameters. Additionally, many of these features are likely quite rare, potentially requiring the autoencoder to be trained on a substantial fraction of the large model's training corpus. So it seems plausible that training the autoencoder could become very expensive, potentially even more expensive than the original model. We remain optimistic, however, and there is a silver lining – it increasingly seems like a large chunk of the mechanistic interpretability agenda will now turn on succeeding at a difficult engineering and scaling problem, which frontier AI labs have significant expertise in.

Scaling Laws for Dictionary Learning. It's worth noting that there's enormous uncertainty about the dynamics of scaling dictionary learning and sparse autoencoders discussed above. As we make the subject model bigger, how does the ideal expansion factor change? (Does it stay constant?) How does the necessary amount of data change? The resolution of these questions will determine whether it's possible for this approach, if executed well, to scale up to frontier models. Ideally, we'd like to have scaling laws (Kaplan *et al.*, 2020) which could answer this.

How Can We Recognize Good Features? One of the greatest challenges of this work is that we're "wandering in the dark" to some extent. We don't have a great, systematic way to know if we're successfully extracting high quality features. Automated interpretability (Bills *et al.*, 2023) seems like a strong contender for solving this question. Alternatively, one might hope for some purely abstract definition (e.g. the information-based metric proposal), but we have not yet seen compelling signs of life for this on real data. It would also be helpful to have metrics beyond MMCS, activation similarity, and attribution similarity for comparing sets of features for the purposes of assessing consistency and universality.

Scalability of Analysis. Suppose that sparse autoencoders fully solve superposition. Do we have a home run to fully mechanistically understanding models? It seems clear that there would be at least one other fundamental barrier: scaling analysis of models, so that we can turn microscopic insights into a more macroscopic understanding. Again, one approach here could be automated interpretability. But delegating the understanding of AI to AI may not be fully satisfying, for various reasons. It is possible that there may be other paths based on discovering larger scale structure (see discussion here).

Algorithmic Improvements for Sparse Autoencoders. New algorithms refining the sparse autoencoder approach could be useful. One might explore the use of variational autoencoders (e.g., (Barello *et al.*, 2018)), or sparsity promoting priors regularization techniques beyond a simple L1 penalty on activations (e.g., (Miao *et al.*, 2021; Rentzeperis *et al.*, 2023)), for example encouraging sparsity in the interactions between learned features in different layers. Earlier research has shown that noise injection can also increase neuron interpretability separately from an L1 penalty (Sharkey *et al.*, 2022; Bricken *et al.*, 2023b).

Attentional Superposition? Many of the motivations for the presence of superposition in

MLP layers (Elhage *et al.*, 2022) apply to self-attention layers as well. It seems conceivable that similar methods may extract useful structure from attention layers, although a clear example has not yet been established (e.g., see our May and July Updates). If this is true, addressing this may become a future bottleneck for the mechanistic interpretability agenda.

Theory of Superposition and Features. Many fundamental questions remain for our understanding of superposition, even if the hypothesis is right in some very broad sense. For example, as discussed above, this work suggests extensions of the superposition hypothesis covering clusters of features with similar effects, or continuous families of features. We believe there is important work to be done in exploring the theory of superposition further, perhaps through the use of toy models.

Conclusion

This thesis has explored the role of sparse representations in both biological brains and artificial neural networks. By drawing on insights from neuroscience, machine learning, and information theory, we have demonstrated how sparsity emerges as a computational principle underlying intelligence.

Beginning with the connection between Transformer Attention and Sparse Distributed Memory (SDM), we demonstrated that a biologically plausible associative memory model closely approximates a core mechanism of modern AI. The shared sparse read and write operations that approximate a softmax distribution suggest convergent evolution towards a shared computational strategy. This link not only provides insight into why Attention works so effectively in state-of-the-art AI systems but also offers a computational interpretation of cerebellar function.

Building on this foundation, we showed how the SDM framework could be adapted to improve AI model's continual learning without catastrophic forgetting. By incorporating biologically inspired components such as the GABA switch and inhibitory interneurons, we demonstrated that every element of our design contributes to preserving previously learned information while accommodating new tasks. This "organic" approach to continual learning offers a more flexible, biological alternative to previous continual learning techniques.

Our exploration of sparse representations emerging from noise provided a new perspective on why biological neural networks exhibit activation sparsity. By identifying the three implicit loss terms introduced by noisy training, we showed how this approach naturally leads to the development of specialized receptive fields, neuronal synchronization, and emergent sparsity. These findings suggest that handling noise may be the reason for sparse coding in biological systems, with metabolic efficiency as an added benefit.

Finally, our work on sparse autoencoders demonstrated how sparse representations can be extracted from dense, polysemantic neural activations, making it possible to identify interpretable features in language models. The discovery of features that activate in specific contexts and produce coherent downstream effects provides a path toward more transparent AI systems. The analysis of these features revealed their universality across different models and their ability to form "finite state automata"-like systems that implement complex behaviors.

Collectively, these findings support several overarching conclusions about why sparsity is so ubiquitous in both biological and artificial neural systems:

1. **Efficiency**: In the brain, sparse neural activity translates to metabolic efficiency, consuming less energy, while in artificial networks, it enables computational and memory efficiency.

2. **Robustness**: Sparse representations can provide robustness to noise and perturbations. This is because information is distributed across many elements (neurons), so the loss or corruption of any single element has minimal impact.

3. **Flexibility**: Sparsity enables flexible learning and adaptation. Sparse representations have high effective dimensionality, allowing them to encode a wide variety of patterns and concepts. This flexibility is crucial for continual learning and adapting to new environments.

4. **Interpretability**: Sparse representations are more interpretable because individual elements correspond to meaningful concepts or patterns. Dense, entangled representations are much harder for humans to understand. This interpretability reflects the discrete nature of the world, where at any given moment only a sparse set of objects or concepts are relevant.

While there remains much to unravel, the principles uncovered here suggest that sparsity isn't merely an incidental property of neural systems but a core computational strategy that enables the remarkable capabilities of both biological and artificial intelligence.

This convergence also shows that biological inspiration can lead to practical improvements in artificial systems. From the GABA switch addressing the dead neuron problem, to noise-induced sparsity creating more robust representations, principles derived from neuroscience have enhanced artificial neural networks in tangible ways.

As we continue to advance AI and unravel the mysteries of the brain, this work provides stepping stones towards the development of more capable, adaptable, and transparent intelligent systems. These findings are particularly relevant as the field increasingly recognizes the importance of interpretability and efficiency in large-scale AI models.

Looking ahead, promising research directions include further exploration of the relationship between attention and memory, more biologically plausible learning algorithms, and improved techniques for interpreting and controlling AI systems. Specifically, using sparse autoencoders to find how circuits of features interact together, instead of in isolation, could greatly enhance our ability to interpret complex AI systems. The combination of noisy training with other approaches might also yield even higher degrees of sparsity with downstream benefits for robustness, continual learning, and computational efficiency.

By studying the principles that underlie both biological and artificial minds, we move closer to understanding the nature of intelligence itself. This thesis has aimed to contribute to this grand endeavor by building bridges between neuroscience and artificial intelligence through the unifying lens of sparse representations.

References

ABBASI, A., NOORALINEJAD, P., BRAVERMAN, V., PIRSIAVASH, H. and KOLOURI, S. (2022). Sparsity and heterogeneous dropout for continual learning in the null space of neural activations. *ArXiv*, **abs/2203.06514**.

AHARON, M., ELAD, M. and BRUCKSTEIN, A. (2006). K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, **54** (11), 4311–4322.

AHMAD, S. and SCHEINKMAN, L. (2019). How can we be so dense? the benefits of using highly sparse representations. *ArXiv*, **abs/1903.11257**.

ALAMMAR, J. (2020). Interfaces for explaining transformer language models.

ALJUNDI, R., BABILONI, F., ELHOSEINY, M., ROHRBACH, M. and TUYTELAARS, T. (2018). Memory aware synapses: Learning what (not) to forget. In *ECCV*.

—, KELCHTERMANS, K. and TUYTELAARS, T. (2019a). Task-free continual learning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11246–11255.

—, ROHRBACH, M. and TUYTELAARS, T. (2019b). Selfless sequential learning. *ArXiv*, **abs/1806.05421**.

ALON, S., GOODWIN, D. R., SINHA, A., WASSIE, A., CHEN, F., DAUGHARTHY, E. R., BANDO, Y., KAJITA, A., XUE, A. G., MARRETT, K., PRIOR, R., CUI, Y., PAYNE, A., YAO, C.-C., SUK, H.-J., WANG, R., CHIEH YU, C., TILLBERG, P. W., REGINATO, P., PAK, N., LIU, S., PUNTHAMBAKER, S., IYER, E. P. R., KOHMAN, R. E., MILLER, J., LEIN, E., LAKO, A., CULLEN, N., RODIG, S., HELVIE, K., ABRAVANEL, D. L., WAGLE, N., JOHNSON, B., KLUGHAMMER, J., SLYPER, M., WALDMAN, J., JANÉ-VALBUENA, J., ROZENBLATT-ROSEN, O., REGEV, A., CHURCH, G., MARBLESTONE, A. H. and BOYDEN, E. (2020). Expansion sequencing: Spatially precise in situ transcriptomics in intact biological systems. *bioRxiv*.

ANDRIUSHCHENKO, M., VARRE, A., PILLAUD-VIVIEN, L. and FLAMMARION, N. (2022). Sgd with large step sizes learns sparse features. *ArXiv*, **abs/2210.05337**.

ARORA, S., LI, Y., LIANG, Y., MA, T. and RISTESKI, A. (2018). Linear algebraic structure of word senses, with applications to polysemy. *Transactions of the Association for Computational Linguistics*, **6**, 483–495.

ATTWELL, D. and LAUGHLIN, S. B. (2001). An energy budget for signaling in the grey matter of the brain. *Journal of Cerebral Blood Flow & Metabolism*, **21**, 1133 – 1145.

Ba, J. L., Kiros, J. R. and Hinton, G. E. (2016). Layer normalization.

Baan, J., ter Hoeve, M., Wees, M. V. D., Schuth, A. and Rijke, M. (2019). Do transformer attention heads provide transparency in abstractive summarization? *ArXiv*, **abs/1907.00570**.

Bahdanau, D., Cho, K. and Bengio, Y. (2016). Neural machine translation by jointly learning to align and translate.

Barello, G., Charles, A. S. and Pillow, J. W. (2018). Sparse-coding variational autoencoders. *bioRxiv*.

Bau, D., Zhou, B., Khosla, A., Oliva, A. and Torralba, A. (2017). Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition*.

—, Zhu, J.-Y., Strobelt, H., Lapedriza, A., Zhou, B. and Torralba, A. (2020). Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, **117** (48), 30071–30078.

Bengio, Y., Courville, A. and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, **35** (8), 1798–1828.

— and Marcus, G. (2019). Ai debate : Yoshua bengio | gary marcus.

Bills, S., Cammarata, N., Mossing, D., Tillman, H., Gao, L., Goh, G., Sutskever, I., Leike, J., Wu, J. and Saunders, W. (2023). Language models can explain neurons in language models.

Bishop, C. M. (1995). Training with noise is equivalent to tikhonov regularization. *Neural Computation*, **7**, 108–116.

Bricken, T., Davies, A., Singh, D., Krotov, D. and Kreiman, G. (2023a). Sparse distributed memory is a continual learner. *ICLR*.

— and Pehlevan, C. (2021). Attention approximates sparse distributed memory. *NeurIPS*.

—, Schaeffer, R., Olshausen, B. and Kreiman, G. (2023b). Emergence of sparse representations from noise.

—, Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N., Anil, C., Denison, C., Askell, A., Lasenby, R., Wu, Y., Kravec, S., Schiefer, N., Maxwell, T., Joseph, N., Hatfield-Dodds, Z., Tamkin, A., Nguyen, K., McLean, B., Burke, J. E., Hume, T., Carter, S., Henighan, T. and Olah, C. (2023c). Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. and Amodei, D. (2020). Language models are few-shot learners.

Burns, C., Ye, H., Klein, D. and Steinhardt, J. (2022). Discovering latent knowledge in language models without supervision. *arXiv preprint arXiv:2212.03827*.

Cammarata, N., Goh, G., Carter, S., Schubert, L., Petrov, M. and Olah, C. (2020). Curve detectors. *Distill*.

Camuto, A., Willetts, M., Simsekli, U., Roberts, S. J. and Holmes, C. C. (2020). Explicit regularisation in gaussian noise injections. *ArXiv*, **abs/2007.07368**.

Carlini, N., Tramèr, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, Ú., Oprea, A. and Raffel, C. (2020). Extracting training data from large language models. *ArXiv*, **abs/2012.07805**.

Chen, M., Radford, A., Wu, J., Jun, H., Dhariwal, P., Luan, D. and Sutskever, I. (2020). Generative pretraining from pixels. In *ICML 2020*.

—, Weinberger, K. Q., Sha, F. and Bengio, Y. (2014). Marginalized denoising auto-encoders for nonlinear representations. In *ICML*.

Chen, R. T. Q., Li, X., Grosse, R. B. and Duvenaud, D. K. (2018a). Isolating sources of disentanglement in vaes.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I. and Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, **29**.

Chen, Y., Paiton, D. M. and Olshausen, B. A. (2018b). The sparse manifold transform. In *NeurIPS*.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L. J. and Weller, A. (2021). Rethinking attention with performers. *ArXiv*, **abs/2009.14794**.

Christiano, P. (2022). Mechanistic anomaly detection and elk.

Coenen, A., Reif, E., Yuan, A., Kim, B., Pearce, A., Viégas, F. and Wattenberg, M. (2019). Visualizing and measuring the geometry of bert. *Advances in Neural Information Processing Systems*, **32**.

Cunningham, H. (2023). Autointerpretation finds sparse coding beats alternatives.

—, Ewart, A., Smith, L., Huben, R. and Sharkey, L. (2023). Sparse autoencoders find highly interpretable model directions. *arXiv preprint arXiv:2309.08600*.

— and Smith, L. (2023). [replication] conjecture's sparse coding in toy models.

Dale, H. H. (1935). Pharmacology and nerve-endings (walter ernest dixon memorial lecture): (section of therapeutics and pharmacology). *Proceedings of the Royal Society of Medicine*, **28 3**, 319–32.

Danihelka, I., Wayne, G., Uria, B., Kalchbrenner, N. and Graves, A. (2016). Associative long short-term memory. In *ICML*.

Doi, E., Gauthier, J. L., Field, G. D., Shlens, J., Sher, A., Greschner, M., Machado, T. A., Jepson, L. H., Mathieson, K., Gunning, D. E., Litke, A. M., Paninski, L., Chichilnisky, E. J. and Simoncelli, E. P. (2012). Efficient coding of spatial information in the primate retina. *The Journal of Neuroscience*, **32**, 16256 – 16264.

Donnelly, J. and Roegiest, A. (2019). On interpretability and feature representations: An analysis of the sentiment neuron. In *European Conference on Information Retrieval*, Springer, pp. 795–802.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. and Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale.

Elad, M. (2010). *Sparse and redundant representations: from theory to applications in signal and image processing*, vol. 2. Springer.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., Grosse, R., McCandlish, S., Kaplan, J., Amodei, D., Wattenberg, M. and Olah, C. (2022). Toy models of superposition. *Transformer Circuits Thread*.

—, Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S. and Olah, C. (2021). A mathematical framework for transformer circuits. *Transformer Circuits Thread*.

Eliasmith, C. (2013). How to build a brain: A neural architecture for biological cognition.

Engan, K., Aase, S. O. and Husoy, J. H. (1999). Method of optimal directions for frame design. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, IEEE, vol. 5, pp. 2443–2446.

Erhan, D., Courville, A., Bengio, Y. and Vincent, P. (2010). Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, pp. 201–208.

Essen, D. V. V., Donahue, C. and Glasser, M. (2018). Development and evolution of cerebral and cerebellar cortex. *Brain, Behavior and Evolution*, **91**, 158 – 169.

Farquhar, S. and Gal, Y. (2018). Towards robust evaluations of continual learning. *ArXiv*, **abs/1805.09733**.

Faruqui, M., Tsvetkov, Y., Yogatama, D., Dyer, C. and Smith, N. (2015). Sparse overcomplete word vector representations. *arXiv preprint arXiv:1506.02004*.

Fedus, W., Zoph, B. and Shazeer, N. (2021). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *ArXiv*, **abs/2101.03961**.

Fleming, E., Tadross, M. R. and Hull, C. (2022). Local synaptic inhibition mediates cerebellar pattern separation necessary for learned sensorimotor associations. *bioRxiv*.

Gale, T., Zaharia, M. A., Young, C. and Elsen, E. (2020). Sparse gpu kernels for deep learning. In *SC*.

Gallistel, C. R. (2017). The coding question. *Trends in Cognitive Sciences*, **21**, 498–508.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S. and Leahy, C. (2020). The pile: An 800gb dataset of diverse text for language modeling.

Gayler, R. (1998). Multiplicative binding, representation operators and analogy.

Geoffrey Hinton, K. S., Nitish Srivastava (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent, coursera Deep Learning Lecture Slides.

Geva, M., Caciularu, A., Wang, K. R. and Goldberg, Y. (2022). Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. *arXiv preprint arXiv:2203.14680*.

—, Schuster, R., Berant, J. and Levy, O. (2020). Transformer feed-forward layers are key-value memories. *ArXiv*, **abs/2012.14913**.

Giovannucci, A., Badura, A., Deverett, B., Najafi, F., Pereira, T. D., Gao, Z., Ozden, I., Kloth, A. D., Pnevmatikakis, E. A., Paninski, L., Zeeuw, C. I. D., Medina, J. F. and Wang, S. S.-H. (2017). Cerebellar granule cells acquire a widespread predictive feedback signal during motor learning. *Nature Neuroscience*, **20**, 727–734.

Glorot, X., Bordes, A. and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *AISTATS*.

Goh, G. (2016). Decoding the thought vector.

—, Cammarata, N., Voss, C., Carter, S., Petrov, M., Schubert, L., Radford, A. and Olah, C. (2021). Multimodal neurons in artificial neural networks. *Distill*, https://distill.pub/2021/multimodal-neurons.

Goodfellow, I. J., Bengio, Y. and Courville, A. C. (2015). Deep learning. *Nature*, **521**, 436–444.

—, Mirza, M., Da, X., Courville, A. C. and Bengio, Y. (2014). An empirical investigation of catastrophic forgeting in gradient-based neural networks. *CoRR*, **abs/1312.6211**.

—, Warde-Farley, D., Mirza, M., Courville, A. C. and Bengio, Y. (2013). Maxout networks. In *ICML*.

Gozel, O. and Gerstner, W. (2021). A functional model of adult dentate gyrus neurogenesis. *eLife*, **10**.

Graves, A., Wayne, G. and Danihelka, I. (2014). Neural turing machines. *ArXiv*, **abs/1410.5401**.

—, —, Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwinska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K. and Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, **538**, 471–476.

Gregor, K. and LeCun, Y. (2010). Learning fast approximations of sparse coding. In *ICML*.

—, Rezende, D. J., Besse, F., Wu, Y., Merzic, H. and van den Oord, A. (2019). Shaping belief states with generative environment models for rl. *ArXiv*, **abs/1906.09237**.

Grinberg, L., Hopfield, J. J. and Krotov, D. (2019). Local unsupervised learning for image analysis. *ArXiv*, **abs/1908.08993**.

Gurbuz, M. B. and Dovrolis, C. (2022). Nispa: Neuro-inspired stability-plasticity adaptation for continual learning in sparse networks. *ArXiv*, **abs/2206.09117**.

Gurnee, W., Nanda, N., Pauly, M., Harvey, K., Troitskii, D. and Bertsimas, D. (2023). Finding neurons in a haystack: Case studies with sparse probing. *arXiv preprint arXiv:2305.01610*.

Gwern (2019). May 2020 news and on gpt-3.

Haider, B., Krause, M. R., Duque, A., Yu, Y., Touryan, J., Mazer, J. A. and McCormick, D. A. (2010). Synaptic and network mechanisms of sparse and reliable visual cortical activity during nonclassical receptive field stimulation. *Neuron*, **65** (1), 107–121.

Hawkins, J. and Ahmad, S. (2016). Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in Neural Circuits*, **10**, 23.

He, K., Zhang, X., Ren, S. and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034.

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv: Learning*.

Henighan, T., Carter, S., Hume, T., Elhage, N., Lasenby, R., Fort, S., Schiefer, N. and Olah, C. (2023). Superposition, memorization, and double descent. *Transformer Circuits Thread*.

Henry, A., Dachapally, P. R., Pawar, S. and Chen, Y. (2020). Query-key normalization for transformers. In *EMNLP*.

Hernandez, E., Schwettmann, S., Bau, D., Bagashvili, T., Torralba, A. and Andreas, J. (2021). Natural language descriptions of deep visual features. In *International Conference on Learning Representations*.

Hobbhahn, M. (2023). More findings on memorization and double descent.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, **79** (8), 2554–2558.

— (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, **81** (10), 3088–3092.

Hoxha, E., Tempia, F., Lippiello, P. and Miniaci, M. C. (2016). Modulation, plasticity and pathophysiology of the parallel fiber-purkinje cell synapse. *Frontiers in Synaptic Neuroscience*, **8**.

Hsu, Y.-C., Liu, Y.-C. and Kira, Z. (2018). Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *ArXiv*, **abs/1810.12488**.

Huben, R. (2023). [research update] sparse autoencoder features are bimodal.

Irie, K., Csord'as, R. and Schmidhuber, J. (2022). The dual form of neural networks revisited: Connecting test time predictions to training patterns via spotlights of attention. In *ICML*.

Iyer, A., Grewal, K., Velu, A., Souza, L. O., Forest, J. and Ahmad, S. (2022). Avoiding catastrophe: Active dendrites enable multi-task learning in dynamic environments. In *Frontiers in Neurorobotics*.

Jaeckel, L. A. (1989a). An alternative design for a sparse distributed memory.

— (1989b). A class of designs for a sparse distributed memory.

Jermyn, A. S., Schiefer, N. and Hubinger, E. (2022). Engineering monosemanticity in toy models. *arXiv preprint arXiv:2211.09169*.

Kanerva, P. (1988). *Sparse distributed memory*. MIT Pr.

— (1993). Sparse distributed memory and related models.

— (1996). Binary spatter-coding of ordered k-tuples. In *ICANN*.

— (2009). Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, **1**, 139–159.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J. and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Karklin, Y. and Simoncelli, E. P. (2011). Efficient coding of natural images with a population of noisy linear-nonlinear neurons. *Advances in neural information processing systems*, **24**, 999–1007.

Karpathy, A., Johnson, J. and Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

Katharopoulos, A., Vyas, A., Pappas, N. and Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. *ArXiv*, **abs/2006.16236**.

Kawato, M., Ohmae, S. and Sanger, T. (2021). 50 years since the marr, ito, and albus models of the cerebellum. *Neuroscience*, **462**, 151–174.

Keeler, J. D. (1988). Comparison between kanerva's sdm and hopfield-type neural networks. *Cognitive Science*, **12** (3), 299 – 329.

Kim, H. and Mnih, A. (2018). Disentangling by factorising. In *International Conference on Machine Learning*, PMLR, pp. 2649–2658.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *CoRR*, **abs/1412.6980**.

Kireev, K., Andriushchenko, M. and Flammarion, N. (2021). On the effectiveness of adversarial training against common corruptions. In *Conference on Uncertainty in Artificial Intelligence*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N. C., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D. and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, **114**, 3521 – 3526.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

—, Sutskever, I. and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, **60**, 84 – 90.

Krotov, D. and Hopfield, J. (2016). Dense associative memory for pattern recognition. In *NIPS*.

— and — (2020a). Large associative memory problem in neurobiology and machine learning. *ArXiv*, **abs/2008.06996**.

— and — (2020b). Large associative memory problem in neurobiology and machine learning. *ArXiv*, **abs/2008.06996**.

— and Hopfield, J. J. (2018). Dense associative memory is robust to adversarial inputs. *Neural Computation*, **30**, 3151–3167.

— and — (2019). Unsupervised learning by competing hidden units. *Proceedings of the National Academy of Sciences of the United States of America*, **116**, 7723 – 7731.

Kumar, A. and Poole, B. (2020). On implicit regularization in β-vaes. In *International Conference on Machine Learning*.

Kurtz, M., Kopinsky, J., Gelashvili, R., Matveev, A., Carr, J., Goin, M., Leiserson, W. M., Nell, B., Shavit, N. and Alistarh, D. (2020). Inducing and exploiting activation sparsity for fast neural network inference.

Lake, B., Ullman, T. D., Tenenbaum, J. and Gershman, S. (2016). Building machines that learn and think like people. *Behavioral and Brain Sciences*, **40**.

Lange, M. D., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G. G. and Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, **PP**.

Le, H., Tran, T. and Venkatesh, S. (2019). Neural stored-program memory. *ArXiv*, **abs/1906.08862**.

— and Venkatesh, S. (2022). Neurocoder: General-purpose computation using stored neural programs. In *International Conference on Machine Learning*.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E. and Jackel, L. D. (1989). Handwritten digit recognition with a back-propagation network. In *Neural Information Processing Systems*.

— and Cortes, C. (2005). The mnist database of handwritten digits.

Levy, O. and Goldberg, Y. (2014). Linguistic regularities in sparse and explicit word representations. In *Proceedings of the eighteenth conference on computational natural language learning*, pp. 171–180.

Li, B., Chen, C., Wang, W. and Carin, L. (2018). Certified adversarial robustness with additive noise. In *Neural Information Processing Systems*.

Li, J., Mahoney, B., Jacob, M. S. and Caron, S. J. C. (2020). Visual input into the drosophila melanogaster mushroom body. *bioRxiv*.

Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H. and Wattenberg, M. (2022). Emergent world representations: Exploring a sequence model trained on a synthetic task. *arXiv preprint arXiv:2210.13382*.

Li, Y., Yosinski, J., Clune, J., Lipson, H., Hopcroft, J. E. *et al.* (2015). Convergent learning: Do different neural networks learn the same representations? In *FE@ NIPS*, pp. 196–212.

Liang, Y., Ryali, C., Hoover, B., Grinberg, L., Navlakha, S., Zaki, M. J. and Krotov, D. (2020). Can a fruit fly learn word embeddings? In *International Conference on Learning Representations*.

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. and Hinton, G. (2020). Backpropagation and the brain. *Nature Reviews Neuroscience*, **21**, 335–346.

Lin, A. C., Bygrave, A. M., de Calignon, A., Lee, T. and Miesenbock, G. (2014). Sparse, decorrelated odor coding in the mushroom body enhances learned odor discrimination. *Nature neuroscience*, **17**, 559 – 568.

Litwin-Kumar, A., Harris, K. D., Axel, R., Sompolinsky, H. and Abbott, L. (2017). Optimal degrees of synaptic connectivity. *Neuron*, **93**, 1153–1164.e7.

lsgos (2023). Dropout can create a privileged basis in the relu output model.

Lu, K., Grover, A., Abbeel, P. and Mordatch, I. (2021). Pretrained transformers as universal computation engines. *ArXiv*, **abs/2103.05247**.

Mahinpei, A., Clark, J., Lage, I., Doshi-Velez, F. and Pan, W. (2021). Promises and pitfalls of black-box concept learning models. *arXiv preprint arXiv:2106.13314*.

Makhzani, A. and Frey, B. J. (2013). k-sparse autoencoders. *CoRR*, **abs/1312.5663**.

— and — (2014a). k-sparse autoencoders. *CoRR*, **abs/1312.5663**.

— and — (2014b). k-sparse autoencoders. *CoRR*, **abs/1312.5663**.

Mallya, A. and Lazebnik, S. (2018). Packnet: Adding multiple tasks to a single network by iterative pruning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7765–7773.

Manevitz, L. and Zemach, Y. (1997). Assigning meaning to data: Using sparse distributed memory for multilevel cognitive tasks. *Neurocomputing*, **14**, 15–39.

Marblestone, A., Wu, Y. and Wayne, G. (2020). Product kanerva machines: Factorized bayesian memory.

Markram, H., Lubke, J., Frotscher, M. and Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, **275**, 213 – 215.

Martins, A. F. T. and Astudillo, R. F. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International Conference on Machine Learning*.

Matsuoka, K. (1992). Noise injection into inputs in back-propagation learning. *IEEE Trans. Syst. Man Cybern.*, **22**, 436–440.

McGrath, T., Kapishnikov, A., Tomašev, N., Pearce, A., Wattenberg, M., Hassabis, D., Kim, B., Paquet, U. and Kramnik, V. (2022). Acquisition of chess knowledge in alphazero. *Proceedings of the National Academy of Sciences*, **119** (47), e2206625119.

McInnes, L. and Healy, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *ArXiv*, **abs/1802.03426**.

Miao, N., Mathieu, E., Siddharth, N., Teh, Y. W. and Rainforth, T. (2021). On incorporating inductive biases into vaes. *arXiv preprint arXiv:2106.13746*.

Michel, P., Levy, O. and Neubig, G. (2019). Are sixteen heads really better than one? *ArXiv*, **abs/1905.10650**.

Mikolov, T., Yih, W.-t. and Zweig, G. (2013). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pp. 746–751.

Millidge, B., Salvatori, T., Song, Y., Lukasiewicz, T. and Bogacz, R. (2022). Universal hopfield networks: A general framework for single-shot associative memory models. *ArXiv*, **abs/2202.04557**.

Minsky, M. and Papert, S. (1969). *Time vs. memory for best matching - an open problem*, p. 222 – 225.

Modi, M., Shuai, Y. and Turner, G. (2020). The drosophila mushroom body: From architecture to algorithm in a learning circuit. *Annual review of neuroscience*.

Molchanov, D., Ashukha, A. and Vetrov, D. P. (2017). Variational dropout sparsifies deep neural networks. In *ICML*.

Morcos, A. S., Barrett, D. G., Rabinowitz, N. C. and Botvinick, M. (2018). On the importance of single directions for generalization. *arXiv preprint arXiv:1803.06959*.

Nanda, N. (2023). Attribution patching: Activation patching at industrial scale.

—, Lee, A. and Wattenberg, M. (2023). Emergent linear representations in world models of self-supervised sequence models. *arXiv preprint arXiv:2309.00941*.

Nelson, E., Tristan, H., Catherine, O., Neel, N., Tom, H., Scott, J., Sheer, E., Nicholas, J., Nova, D., Ben, M., Danny, H., Amanda, A., Kamal, N., Jones, , Dawn, D., Anna, C., Yuntao, B., Deep, G., Liane, L., Zac, H.-D., Jackson, K., Tom, C., Shauna, K., Stanislav, F., Saurav, K., Josh, J., Eli, T.-J., Jared, K., Jack, C., Tom, B., Sam, M., Dario, A. and Christopher, O. (2022). Softmax linear units. *Transformer Circuits Thread*, https://transformer-circuits.pub/2022/solu/index.html.

Olah, C. (2015). Visualizing representations: Deep learning and human beings.

— (2023). Distributed representations: Composition superposition.

—, Cammarata, N., Schubert, L., Goh, G., Petrov, M. and Carter, S. (2020a). An overview of early vision in inceptionv1. *Distill*, https://distill.pub/2020/circuits/early-vision.

—, —, —, —, — and — (2020b). Zoom in: An introduction to circuits. *Distill*, https://distill.pub/2020/circuits/zoom-in.

—, Mordvintsev, A. and Schubert, L. (2017). Feature visualization. *Distill*.

Olshausen, B. A. and Field, D. J. (1997a). Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, **37**, 3311–3325.

— and — (1997b). Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, **37** (23), 3311–3325.

— and — (2004). Sparse coding of sensory inputs. *Current opinion in neurobiology*, **14** (4), 481–487.

Paiton, D. M., Frye, C. G., Lundquist, S. Y., Bowen, J., Zarcone, R. and Olshausen, B. A. (2020). Selectivity and robustness of sparse coding networks. *Journal of Vision*, **20**.

Panigrahi, A., Simhadri, H. V. and Bhattacharyya, C. (2019). Word2sense: sparse interpretable word embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5692–5705.

Piantadosi, S. (2021). The computational origin of representation. *Minds and Machines*, **31**, 1–58.

Pisano, T. J., Dhanerawala, Z. M., Kislin, M., Bakshinskaya, D., Engel, E. A., Hansen, E. J., Hoag, A. T., Lee, J., de Oude, N. L., Venkataraju, K. U., Verpeut, J. L., Hoebeek, F. E., Richardson, B. D., Boele, H.-J. and Wang, S. S.-H. (2020). Homologous organization of cerebellar pathways to sensory, motor, and associative forebrain. *bioRxiv*.

Plate, T. (1991). Holographic reduced representations: Convolution algebra for compositional distributed representations. In *IJCAI*.

Poole, B., Sohl-Dickstein, J. N. and Ganguli, S. (2014). Analyzing noise in autoencoders and deep networks. *ArXiv*, **abs/1406.1831**.

Powell, K., Mathy, A., Duguid, I. and Häusser, M. (2015). Synaptic representation of locomotion in single cerebellar granule cells. *eLife*, **4**.

Prager, R. and Fallside, F. (1989). The modified kanerva model for automatic speech recognition. *Computer Speech and Language*, **3**, 61–81.

Purves, D., Augustine, G. and Fitzpatrick, D. (2001). *Neurons Often Release More Than One Transmitter*, Sinauer Associates. 2nd edn.

Radford, A., Jozefowicz, R. and Sutskever, I. (2017). Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*.

—, Metz, L. and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

—, Wu, J., Amodei, D., Amodei, D., Clark, J., Brundage, M. and Sutskever, I. (2019). Better language models and their implications. *OpenAI Blog https://openai. com/blog/better-language-models*.

Raghu, M., Gilmer, J., Yosinski, J. and Sohl-Dickstein, J. (2017). Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., pp. 6078–6087.

Ramasesh, V. V., Lewkowycz, A. and Dyer, E. (2022). Effect of model and pretraining scale on catastrophic forgetting in neural networks.

Ramesh, A., Pavlov, M., Goh, G. and Gray, S. (2021). Dall·e: Creating images from text.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., Greiff, V., Kreil, D., Kopp, M., Klambauer, G., Brandstetter, J. and Hochreiter, S. (2020). Hopfield networks is all you need.

Ranzato, M., Boureau, Y.-L., Chopra, S. and LeCun, Y. (2007). A unified energy-based framework for unsupervised learning. In *AISTATS*.

Rentzeperis, I., Calatroni, L., Perrinet, L. and Prandi, D. (2023). Beyond sparse coding in v1. *arXiv preprint arXiv:2301.10002*.

Rogers, A., Kovaleva, O. and Rumshisky, A. (2020). A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics*, **8**, 842–866.

Rumbelow, J. and Watkins, M. (2023). Solidgoldmagikarp (plus, prompt generation).

Rumelhart, D. E. and Zipser, D. (1985). Feature discovery by competitive learning.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C. and Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, **115**, 211–252.

Ryali, C., Hopfield, J., Grinberg, L. and Krotov, D. (2020). Bio-inspired hashing for unsupervised similarity search. In *International Conference on Machine Learning*, PMLR, pp. 8295–8306.

Räuker, T., Ho, A., Casper, S. and Hadfield-Menell, D. (2023). Toward transparent ai: A survey on interpreting the inner structures of deep neural networks. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, IEEE, pp. 464–483.

Salimans, T. (2016). A structured variational auto-encoder for learning deep hierarchies of sparse features. *ArXiv*, **abs/1602.08734**.

Scaplen, K. M., Talay, M., Fisher, J. D., Cohn, R., Sorkaç, A., Aso, Y., Barnea, G. and Kaun, K. (2020). Transsynaptic mapping of drosophila mushroom body output neurons. *bioRxiv*.

Scherlis, A., Sachan, K., Jermyn, A. S., Benton, J. and Shlegeris, B. (2022). Polysemanticity and capacity in neural networks. *arXiv preprint arXiv:2210.01892*.

Schlag, I., Irie, K. and Schmidhuber, J. (2021). Linear transformers are secretly fast weight programmers. In *ICML*.

Schrimpf, M., Blank, I., Tuckute, G., Kauf, C., Hosseini, E. A., Kanwisher, N., Tenenbaum, J. and Fedorenko, E. (2020). The neural architecture of language: Integrative reverse-engineering converges on a model for predictive processing. *bioRxiv*.

Schubert, L., Voss, C., Cammarata, N., Goh, G. and Olah, C. (2021). High-low frequency detectors. *Distill*, https://distill.pub/2020/circuits/frequency-edges.

Schwarz, J., Jayakumar, S. M., Pascanu, R., Latham, P. E. and Teh, Y. W. (2021). Powerpropagation: A sparsity inducing weight reparameterisation. In *NeurIPS*.

Sengupta, A. M., Tepper, M., Pehlevan, C., Genkin, A. and Chklovskii, D. (2018). Manifold-tiling localized receptive fields are optimal in similarity-preserving neural networks. *bioRxiv*.

Sengupta, B., Stemmler, M. B., Laughlin, S. B. and Niven, J. E. (2010). Action potential energy efficiency varies among neuron types in vertebrates and invertebrates. *PLoS Computational Biology*, **6**.

Sezener, E., Grabska-Barwinska, A., Kostadinov, D., Beau, M., Krishnagopal, S., Budden, D., Hutter, M., Veness, J., Botvinick, M. M., Clopath, C., Häusser, M. and Latham, P. E. (2021a). A rapid and efficient learning rule for biological neural circuits. *bioRxiv*.

—, —, —, —, —, —, —, —, —, —, — and Latham, P. E. (2021b). A rapid and efficient learning rule for biological neural circuits. *bioRxiv*.

Sharkey, L., Braun, D. and Millidge, B. (2022). [interim research report] taking features out of superposition with sparse autoencoders.

Shazeer, N. M., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E. and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *ArXiv*, **abs/1701.06538**.

Shen, Y., Dasgupta, S. and Navlakha, S. (2021). Algorithmic insights on continual learning from fruit flies. *ArXiv*, **abs/2107.07617**.

Shigeno, S. and Ragsdale, C. W. (2015). The gyri of the octopus vertical lobe have distinct neurochemical identities. *Journal of Comparative Neurology*, **523**.

Shomrat, T., Graindorge, N., Bellanger, C., Fiorito, G., Loewenstein, Y. and Hochner, B. (2011). Alternative sites of synaptic plasticity in two homologous "fan-out fan-in" learning and memory networks. *Current Biology*, **21**, 1773–1782.

Sietsma, J. and Dow, R. J. F. (1991). Creating artificial neural networks that generalize. *Neural Networks*, **4**, 67–79.

Simoncelli, E. P. and Adelson, E. H. (1996). Noise removal via bayesian wavelet coring. In *Proceedings of 3rd IEEE International Conference on Image Processing*, IEEE, vol. 1, pp. 379–382.

Smith, J., Tian, J., Hsu, Y.-C. and Kira, Z. (2022). A closer look at rehearsal-free continual learning. *ArXiv*, **abs/2203.17269**.

Smith, L. (2023a). Really strong features found in residual stream.

— (2023b). (tentatively) found 600+ monosemantic features in a small lm using sparse autoencoders.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artif. Intell.*, **46**, 159–216.

Song, Y. and Ermon, S. (2019). Generative modeling by estimating gradients of the data distribution. *ArXiv*, **abs/1907.05600**.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, **15**, 1929–1958.

Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F. J. and Schmidhuber, J. (2013). Compete to compute. In *NIPS*.

Sterling, P. and Laughlin, S. (2015). Principles of neural design.

Strang, G. (1993). Introduction to linear algebra.

Subramanian, A., Pruthi, D., Jhamtani, H., Berg-Kirkpatrick, T. and Hovy, E. (2018). Spine: Sparse interpretable neural embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32.

Sukhbaatar, S., Grave, E., Lample, G., Jégou, H. and Joulin, A. (2019). Augmenting self-attention with persistent memory. *ArXiv*, **abs/1907.01470**.

Tenney, I., Wexler, J., Bastings, J., Bolukbasi, T., Coenen, A., Gehrmann, S., Jiang, E., Pushkarna, M., Radebaugh, C., Reif, E. and Yuan, A. (2020). The language interpretability tool: Extensible, interactive visualizations and analysis for nlp models. In *EMNLP*.

Thorpe, S. J. (1989). Local vs. distributed coding. *Intellectica*, **8**, 3–40.

Tononi, G. and Cirelli, C. (2014). Sleep and the price of plasticity: From synaptic and cellular homeostasis to memory consolidation and integration. *Neuron*, **81**, 12–34.

Tramèr, F., Carlini, N., Brendel, W. and Madry, A. (2020). On adaptive attacks to adversarial example defenses. *ArXiv*, **abs/2002.08347**.

Trockman, A. and Kolter, J. Z. (2022a). Patches are all you need? *ArXiv*, **abs/2201.09792**.

— and — (2022b). Patches are all you need? *ArXiv*, **abs/2201.09792**.

Tuli, S., Dasgupta, I., Grant, E. and Griffiths, T. (2021). Are convolutional neural networks or transformers more like human vision? *ArXiv*, **abs/2105.07197**.

Tyulmankov, D., Fang, C., Vadaparty, A. and Yang, G. R. (2021). Biological key-value memory networks. *Advances in Neural Information Processing Systems*, **34**.

van den Oord, A., Vinyals, O. and Kavukcuoglu, K. (2017). Neural discrete representation learning. In *NIPS*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I. (2017). Attention is all you need.

Vig, J. (2019). Visualizing attention in transformer-based language representation models. *ArXiv*, **abs/1904.02679**.

Vincent, P., Larochelle, H., Bengio, Y. and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *ICML '08*.

WANG, Z. (2020). Sparsert: Accelerating unstructured sparsity on gpus for deep learning inference. *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*.

WILSON YAN, P. A., YUNZHI ZHANG and SRINIVAS, A. (2021). Videogen: Generative modeling of videos using {vq}-{vae} and transformers. In *Submitted to International Conference on Learning Representations*, under review.

WOLFF, G. and STRAUSFELD, N. (2016). Genealogical correspondence of a forebrain centre implies an executive brain in the protostome–deuterostome bilaterian ancestor. *Philosophical Transactions of the Royal Society: Biological Sciences*, **371**.

WU, Y., WAYNE, G., GRAVES, A. and LILLICRAP, T. (2018a). The kanerva machine: A generative distributed memory.

—, —, GREGOR, K. and LILLICRAP, T. (2018b). Learning attractor dynamics for generative memory. In *NeurIPS*.

XIAO, C., ZHONG, P. and ZHENG, C. (2020). Enhancing adversarial defense by k-winners-take-all. *arXiv: Learning*.

XIE, M., MUSCINELLI, S. P., HARRIS, K. D. and LITWIN-KUMAR, A. (2022). Task-dependent optimal representations for cerebellar learning. *bioRxiv*.

XU, C., JANUSZEWSKI, M., LU, Z., TAKEMURA, S., HAYWORTH, K., HUANG, G., SHINOMIYA, K., MAITIN-SHEPARD, J. B., ACKERMAN, D., BERG, S. E., BLAKELY, T., BOGOVIC, J., CLEMENTS, J., DOLAFI, T., HUBBARD, P. M., KAINMUELLER, D., KATZ, W., KAWASE, T., KHAIRY, K., LEAVITT, L., LI, P. H., LINDSEY, L., NEUBARTH, N. L., OLBRIS, D. J., OTSUNA, H., TROUTMAN, E. T., UMAYAM, L., ZHAO, T., ITO, M., GOLDAMMER, J., WOLFF, T., SVIRSKAS, R., SCHLEGEL, P., NEACE, E., KNECHT, C. J., ALVARADO, C. X., BAILEY, D. A., BALLINGER, S., BORYCZ, J., CANINO, B. S., CHEATHAM, N., COOK, M., DREHER, M., DUCLOS, O., EUBANKS, B., FAIRBANKS, K., FINLEY, S., FORKNALL, N., FRANCIS, A., HOPKINS, G., JOYCE, E. M., KIM, S., KIRK, N. A., KOVALYAK, J., LAUCHIE, S., LOHFF, A., MALDONADO, C., MANLEY, E. A., MCLIN, S., MOONEY, C., NDAMA, M., OGUNDEYI, O., OKEOMA, N., ORDISH, C., PADILLA, N. L., PATRICK, C., PATERSON, T., PHILLIPS, E. E., PHILLIPS, E. M., RAMPALLY, N., RIBEIRO, C., ROBERTSON, M. K., RYMER, J., RYAN, S., SAMMONS, M., SCOTT, A. K., SCOTT, A. L., SHINOMIYA, A., SMITH, C., SMITH, K., SMITH, N. L., SOBESKI, M. A., SULEIMAN, A., SWIFT, J., TAKEMURA, S., TALEBI, I., TARNOGORSKA, D., TENSHAW, E., TOKHI, T., WALSH, J., YANG, T., HORNE, J., LI, F., PAREKH, R., RIVLIN, P., JAYARAMAN, V., ITO, K., SAALFELD, S., GEORGE, R., MEINERTZHAGEN, I., RUBIN, G., HESS, H., SCHEFFER, L. K., JAIN, V. and PLAZA, S. M. (2020). A connectome of the adult drosophila central brain. *bioRxiv*.

XU, J. and ZHU, Z. (2018). Reinforced continual learning. In *NeurIPS*.

YAMINS, D., HONG, H., CADIEU, C. F., SOLOMON, E. A., SEIBERT, D. and DICARLO, J. J. (2014). Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, **111**, 8619 – 8624.

YANG, G., CHEN, X., LIU, K. and YU, C. (2021). Deeppseudo: Deep pseudo-code generation via transformer and code feature extraction. *ArXiv*, **abs/2102.06360**.

Yang, H., Wen, W. and Li, H. H. (2020). Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. *ArXiv*, **abs/1908.09979**.

Yun, Z., Chen, Y., Olshausen, B. A. and LeCun, Y. (2021). Transformer visualization via dictionary learning: contextualized embedding as a linear superposition of transformer factors. *arXiv preprint arXiv:2103.15949*.

Zenke, F., Poole, B. and Ganguli, S. (2017). Continual learning through synaptic intelligence. *Proceedings of machine learning research*, **70**, 3987–3995.

Zhang, J., Chen, Y., Cheung, B. and Olshausen, B. A. (2019). Word embedding visualization via dictionary learning. *arXiv preprint arXiv:1910.03833*.

Zhang, M., Badkundri, R., Talbot, M. B. and Kreiman, G. (2021). Hypothesis-driven stream learning with augmented memory. *ArXiv*, **abs/2104.02206**.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A. and Torralba, A. (2014). Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*.

Zur, R. M., Jiang, Y., Pesce, L. L. and Drukker, K. (2009). Noise injection for training artificial neural networks: a comparison with weight decay and early stopping. *Medical physics*, **36 10**, 4810–8.