Emergence of Sparse Representations from Noise

Anonymous Authors\textsuperscript{1}

Abstract

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 orthogonal, high variance features of the dataset. When trained to reconstruct 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.

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 & 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 & 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 & 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 & Field, 1997).

\textsuperscript{1}Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

In this work, we advance an alternative theory that the need to handle noise is a key driver behind activation sparsity (Olshausen & Field, 2004). 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, \( \min_{\theta} \| \delta f_{\theta}(x) / \delta x \|_F^2 \), where \( x \) is the input data and \( f_{\theta}(x) \) 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 the Taylor approximation 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 the loss seeks to maximize:

\[
\text{Noise Terms} = \text{Maximize}([\text{Neuron Sparsity}] + [\text{Neuron Activation Margin}] + [\text{Orthogonal Model Weights}])
\]

The theoretical effects of Eq. 1 on what each neuron learns are summarized in Fig. 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 can be accomplished by the neuron’s receptive field only modelling high variance regions of the distribution. Finally, each neuron should learn orthogonal receptive fields...
to de-correlate the effects of noise on the neural activations, resulting in specialization to a non-redundant subset of the high variance data features.

![A Hypothesized Neuron's Distribution of Pre-ReLU Activations across the Dataset](image)

Figure 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 term effects. 1. The neuron should learn orthogonal weights, in this case we plot a Gabor filter learnt by the model and how it may hypothetically be strongly activated by the edge along the bottom of the truck but not be activated for anything else. 2. The mean activation of the neuron (dotted blue line) should be negative such that the neuron 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. 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 to be orthogonal to one another. Finally, the bias terms and weight norms synchronize such that a sparse \( k \) neurons fire for any input datum. These operations, and their resulting sparsity, approximate that of a well-known circuit motif: inhibitory interneurons.

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. Throughout this work we indicate and investigate this alternative dead neuron form of sparsity. Note that in this work we consider activation sparsity, not weight sparsity, which can contribute to activation sparsity but is fundamentally different and beyond the domain of sparse coding.

We show that our findings generalize across a number of deep learning tasks, datasets, architectures, and can even be used as a pre-training task. Pretraining with noise then removing it still results in a sparse model and (excluding 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., 2023; Gregor & LeCun, 2010; Elhage et al., 2022a; Srivastava et al., 2014; Martins & 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 & Field, 1997). 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., 2023). Noisy training endogenously incentivizes sparsity, as well as weight norm regularization and feature orthogonality. Noise also results in different receptive fields from sparse coding, resulting in a transition from Gabor to center-surround (Karklin & Simoncelli, 2011). This latter finding suggests that noisy training may be better at removing neuron polysemantics (Elhage et al., 2022a; b).

2. Related Work

Injecting noise into training data has a rich history, dating back as early as Sietsma & 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 & Dow, 1991; Matsuoka, 1992). Interested in training with noise recently rose due to de-noising autoencoders that play a critical role in modern diffusion models (Goodfellow et al., 2015; Song & 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\(^1\), investigating the network sparsity mechanism, deriving our more detailed implicit loss terms, exploring the generality of findings across model architectures, and finding the emergence of biological receptive fields (Olshausen & Field, 1997).

It was previously discovered that de-noising auto encoders, when trained on the MNIST handwritten digits dataset (LeCun & 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 & Laughlin, 2015).

To the best of our knowledge, our work is the first to find

\(^1\)Eqns. 4 & 7 are incorrect, as confirmed by the authors via private correspondence (Appendix G has a corrected derivation).
Emergence of Sparse Representations from Noise

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 from noisy training alone.

Approaches to enforce activation sparsity include using Top-k activation functions (Ranzato et al., 2007; Makhzani & Frey, 2014; Ahmad & Scheinkman, 2019), novel regularization 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; Elhage et al., 2022a; Martins & Astudillo, 2016).

Learning reconstructions of training data with sparse activations is the mainstay of sparse coding (for a formal introduction, see App. A (Olshausen & Field, 1997)). Prior work showed that combining the sparse coding $L_1$ penalty with noisy training produces biological receptive fields (Karklin & Simoncelli, 2011; Doi et al., 2012). Karklin & Simoncelli (2011) notably used a non-linear model that not only learned ReLU like activation functions but also activation thresholds are negative 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, without additional activation penalties and enforced weight norms.

Finally, sparse $k$ 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 & 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., 2023; Keeler, 1988).

3. Results

Shallow Denoising Autoencoder - As a starting point, we train a single hidden layer ReLU autoencoder with additive random noise $\epsilon$ sampled from a zero mean, isotropic Gaussian with variance $\sigma^2$. 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. 4, the encoder and decoder weight matrices and bias terms are: $W_e \in \mathbb{R}^{m \times n}, b_e \in \mathbb{R}^m, W_d \in \mathbb{R}^{n \times m}, b_d \in \mathbb{R}^n$ where $n$ is the input dimension, $o$ is output dimension, and $m$ is the number of neurons in the hidden layer. We will use the term “key vector” to refer to rows of $W_e$ and “value vector” to refer to the rows of $W_d$ that correspond to each neuron. Our loss function uses the mean squared error between the original image and reconstruction across our full dataset, $X$.

$$ \tilde{x} = x + \epsilon, \quad \epsilon \sim N(0, \sigma^2 I) $$

$$ z = W_e \tilde{x} + b_e $$

$$ h = \varphi(z) $$

$$ \tilde{y} = W_d h + b_d $$

$$ \mathcal{L} = \sum_{x \in X} ||x - \tilde{y}||^2. $$

where $\varphi$ 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. 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 & Kolter, 2022; Russakovsky et al., 2015). All code and training parameters can be found at: https://github.com/anon8371/AnonPaper2.

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. 2. 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 2: 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.](https://github.com/anon8371/AnonPaper2)
results and investigations confirming that noisily trained networks agree with our analysis. The key to our result is taking an expectation over the noise (Bishop, 1995).

\[
\text{Loss} = \frac{1}{D} \sum_{x \in X} E_x \left[ \sum_{i}^a (x_i - \bar{y}_i)^2 \right] = \frac{1}{D} \sum_{x \in X} \sum_{i}^a r_i^2 - 2 r_i E_x[\xi_i] + E_x[\xi_i^2] \tag{5}
\]

where \(D\) is the size of the dataset, \(r_i = x_i - \bar{y}_i\), and \(\xi_i = \bar{y}_i - \bar{y}_i\) is the difference between \(\bar{y}_i\), the output produced by the input with noise \(\varepsilon\), and the 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.

The noise penalty ultimately results from the terms \(E_x[\xi_j]\), to maximize the activation margin, and \(E_x[\xi_j^2]\), to both sparsify activations and de-correlate the encoder weights. Defining \(\xi_j = \sum_{j} W_{d_{i,j}} \eta_j\), where \(\eta_j = \bar{h}_j - h_j\), we can write \(E_x[\xi_i] = \sum_{j}^m W_{d_{i,j}} E_x[\eta_j]\). We show in Appendix F that \(\bar{h}\) is a truncated Gaussian distribution and \(E_x[\eta_j] \geq 0\). Intuitively, this is because, as shown in Fig. 3 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:

\[
E_x[\eta_j] = |\|W_{e_j}\|| \frac{\sigma}{\sqrt{2\pi}} \exp \left( -\frac{1}{2} \left( \frac{\bar{h}_j}{\|W_{e_j}\|\sigma} \right)^2 \right) \geq 0 \tag{6}
\]

Turning to our other term \(E_x[\eta_j^2]\), we can expand it as:

\[
E_x[\sum_{j} W_{d_{i,j}} \eta_j^2] = E_x[\sum_{j} W_{d_{i,j}}^2 \eta_j^2 + \sum_{k \neq j} W_{d_{i,j}} W_{d_{i,k}} \eta_j \eta_k] \tag{7}
\]

This results in integrals over the error terms summarized by the bottom row of Figure 3 and given formally in Appendix F. There are three 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 \(W_{d_{i,j}} W_{d_{i,k}} E_x[\eta_j \eta_k]\) incentivizes both minimizing the sum of the decoder weights outer product \(|\|W_{d_{i}} W_{d_{i}}^T||\|\) 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 \(w_{e_j}\) and \(w_{e_k}\). Therefore, a way to utilize both neurons in the model (increasing model capacity) but avoid this cross term is by having their encoder weights be orthogonal.

It is important to note that the encoder weights should capture high variance regions of the data distribution to maximize the activation margin instead of simply increasing the scale of their weights. This is because, increasing their scale will increase noise as much as it does the activation because it has a linear effect on both: \(w_{e_j} x / w_{e_j} \varepsilon\). Therefore, the way to both reduce noise and also maximize the activation threshold margin is by having the weights specialize to distinct dataset features. This will maximize the signal to noise ratio by ignoring noise across most of the input while being very activated by unique data features, therefore not firing all the time and firing strongly when it does. 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 fire across the data distribution like in Fig. 1 where it has a negative mean and a long right hand tail (see Appendix C for empirical results). Ideally, the tail would in fact be another mode resulting in a bimodal distribution that straddles the activation threshold, however, this depends upon the distribution of

Figure 3: Intuition for why noise results in max margin and sparsity. The left column applies to \(\bar{z}_j \geq 0\) and right is \(\bar{z}_j \leq 0\). Going row-wise: #1. Shows the noise distribution around the noise free \(\bar{z}_j\). #2. Post-ReLU, the truncated tail results in a positive mean shift \(E_x[\bar{h}_j] \geq \bar{h}_j\). #3. The variance \(E_x[\eta_j]\) depends on how much of the distribution is positive.
Emergence of Sparse Representations from Noise

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. 4 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. 4 only shows the mean number of neurons that
are active across all 50,000 CIFAR10 inputs per epoch, Fig.
5 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, so that the mean
is an accurate summary statistic. Note that there are no
dead neurons. Fig. 5 also shows the bias terms for each
neuron and the \( L_2 \) norm of their key vectors. 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 4 also shows what looks like a delayed phase transi-
tion 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 key vectors to become sufficiently small
and bias terms sufficiently negative to silence more than
50% of the neurons (bottom row of Fig. 5). 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
B confirms this explanation by initializing weights with
smaller \( L_2 \) norms that result in faster sparsification.

Shared Bias Terms and Inhibitory Interneurons - In bio-
logical 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 & Laughlin,
2015) and Golgi interneurons in cerebellar-like structures
(Fleming et al., 2022; Lin et al., 2014; Xie et al., 2022).
Biologically, these inhibitory interneurons implement a neg-
ative feedback loop whereby the more active the excitatory
neurons are, the more active the interneuron becomes, and
the more it inhibits the excitatory neurons.

To test the observed transition into a sparse coding network,
we modify the encoder bias from \( b_e \) to \( b_e \cdot I \), where \( b_e \)
is a (shared) scalar, then train new autoencoders. This results in
near identical model performance and sparsity. Using this
shared bias is interesting because, in conjunction with the synchro-
nization of the \( L_2 \) weight norms, it is analogous to the
network dynamically learning a \( k \) value as if using Top-
\( k \), but avoiding the complications associated with an explicit
Top-\( k \) implementation (Bricken et al., 2023; Makhzani & Frey, 2014; Ahmad & 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 reminis-
cent of V1 Gabor filters and center-surround retinal ganglion
cells (Sterling & Laughlin, 2015; Olshausen & Field, 1997).
Figure 6 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 pro-
gressed, the more the receptive fields resembled Gabor-like
and center-surround functions. See Appendix D for more
receptive fields at varying noise levels across epochs.

Relations to Sparse Coding - Training with the \( L_1 \) penalty
plus enforcing a unitary \( L_2 \) norm on all weights also resulted in
biological receptive fields. The receptive fields were all Gabor or textures with the latter likely existing due
to training on full CIFAR10 images instead of the typical
smaller 8x8 patches (Olshausen & Field, 1997; Karklin & Simoncelli, 2011). The \( L_2 \) weight normalization is crucial
to obtaining biological receptive fields; otherwise, weights
can shrink to get around the activation penalty.2 Karklin & Simoncelli (2011) showed, in agreement with our results,

\[ \text{Fraction of Active Neurons as a function of Gaussian Noise during training} \]

\[ \begin{align*}
\sigma = 10 & \quad \sigma = 3 & \quad \sigma = 1.5 & \quad \sigma = 0.8 \\
\sigma = 0.3 & \quad \sigma = 0.1 & \quad \sigma = 0.05 & \quad \sigma = 0 \\
\end{align*} \]

Figure 4: The positive relationship between activation spar-
sity and noise. The average fraction of neurons active
during each latent CIFAR10 training batch over 800 epochs.
Each line corresponds to training randomly initialized net-
works with different noise levels \( \sigma \) 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 val-
ues that require more parameter updates to counteract.

2Interestingly, the \( L_1 \) penalty is not completely avoided and
the network does still become more sparse, however, it does not
learn biological receptive fields.
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 cover a smaller portion of the image and we hypothesize that they are thus less sensitive to noise, resulting in their appearance 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 J.2, 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 off many of the 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 H).

**Noisy Pre-Training Retains Model Sparsity** - Investigating our noise-trained sparse networks, we find that even after the noise is removed, the networks remain sparse. We train our models in the same way as in Fig. 4 but at epoch 800 linearly anneal each noise level down to 0 over the next 800 epochs. Figure 7 shows that the sparsity levels remain largely unperturbed.

For classification (Fig. 4, 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. 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 I.1). Note that for the networks with $\sigma \geq 0.8$ for reconstruction and $\sigma \geq 3.0$ for classification, annealing the noise results in dead neurons, presumably because as noise is annealed 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 J.3 for data on the fractions of dead neurons.

**Model Ablations and Generalization** - We investigate sparsity across experimental conditions in the single hidden layer setting and find that they all reproduce our results (Appendix I.1): (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 (Hendrycks & Gimpel, 2016), ELU, and Tanh activation functions. We find that the asymmetric ReLU, GELU, ELU activation functions generally produce sparsity across datasets, albeit to slightly different extents while the symmetric sigmoid and Tanh do not (Appendix I.1). 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 corresponds to be-

---

Figure 5: **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 key vectors (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.

Figure 6: **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 re-shaped 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.
The vertical purple lines denote the start of noise annealing 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.

**Figure 7**: Noisy pretraining retains network sparsity.

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 removes dead neurons but had no effect on training or validation loss (Appendix J.2).

### Deeper Models -

While our analyses focus 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. 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 K.1). 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.

**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 $\sim 20\%$ reductions (82% baseline vs $\sim 62\%$). 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, while

---

4 These Transformers use the HuggingFace implementation that includes LayerNorm and Residual connections.
using the approaches in Appendix J.2 reduced dead neurons, they failed to improve validation accuracy (Appendix K.3). We suspect that the convolutional layers are responsible and incompatible with noisy training but leave further investigation to future work.

Three-Layer MLP - We used 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 K.2). 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 using synchronized bias terms and $L_2$ norms. However, the number of dead neurons still had no effect on the final training or validation accuracies.

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 meant the noise had less of a perturbation effect.

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 & 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. (2023) found Stale Momentum harmed continual learning but it may be a feature instead of a bug, when training within task 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 & Pehlevan, 2021; Krotov & 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. L).

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 & 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 & Laughlin, 2015).

Conclusion - We have shown that simple Gaussian noise results in three implicit loss terms that produce sparse and orthogonal receptive fields that 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 that noise alone results in sparse coding rather than needing 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 they can become more similar.
References


Kanerva, P. Sparse Distributed Memory. MIT Pr., 1988.


Emergence of Sparse Representations from Noise


Sterling, P. and Laughlin, S. Principles of neural design. 2015.


Appendix

A. Sparse Coding Overview

Sparse coding uses an overcomplete basis with an $L_1$ activation penalty. The image reconstruction can be written as

$$\hat{x} = \sum_{i=1}^{N} a_i w_i,$$

where $w_i$ is a column of weight matrix $W$, corresponding to the output weights of a neuron and $a_i$ is the neuron’s activity. The objective function aims to minimize the reconstruction error while having sparse neuron activity:

$$\arg\min_{W,a} (x - \hat{x})^2 + \lambda \sum_{i=1}^{N} |a_i|.$$  

This sparse coding task is an example of a statistical model converging with neuroscience because, in addition to producing sparse representations, the neuron weights learn to become Gabor filters (Olshausen & Field, 1997).

B. Varying Weight Initialization Scale

Here we confirm the hypothesis that the reason higher noise models take longer to become sparse is because they need to significantly shrink the weight norms of their key vectors. We test this by initializing network key vectors with weights of varying scales and using $\sigma = 3.0$ noise. Figure 8 shows how networks with larger initialized weights take more time to become sparse. In addition, their bias terms become larger negative values and $L_2$ norms fall until they reach values where the bias terms can effectively implement sparsity.
Figure 8: **Varying Weight Initialization Scale.** We use $s$ to denote the amount we multiply our initialized weights by before training. $s = 1.0$ (yellow) is the default Kaiming initialization (He et al., 2015).
C. Pre-Activation Distributions

Figure 9: Pre-Activation Distributions of Shallow Denoising Autoencoders on CIFAR-10. Distribution of 250 units’ pre-activation values, randomly sampled from 10000 units. Left-to-right: $\sigma = 0.01, 0.5, 1.0, 5.0$. The bias for each neuron becomes more negative and tails grow longer.

We confirm that as the noise variance $\sigma^2$ increases, the distribution of preactivation shifts negatively. We visualize the distribution of preactivation values from 250 randomly selected units in 10k unit denoising autoencoders trained on CIFAR-10 (Fig. 9).
D. Receptive Fields

Here we show additional images of the receptive fields for networks trained with different amounts of noise (Fig. 10), the queries used to activate these neurons (Fig. 11) and how the receptive fields look more biological as the network is trained for longer (Fig. 12).
Figure 10: **Receptive fields across noise levels.** We show across all noise levels the $\leq 625$ most active neurons for the truck query (middle of Fig. 11). For $0.8 \geq \sigma$ which activates fewer neurons, we use the extra space to also show the cat query. Biological receptive fields start to appear for new neurons when $\sigma = 0.1$ and can be seen most clearly when $\sigma \in \{0.3, 0.8, 1.5\}$. 
Figure 11: **Queries used to activate the receptive fields.** These queries were randomly selected. We use the cat (left) and truck images (middle) with the relevant noise levels to activate the neurons shown in Fig. 10. We use the car image (right) for the main text Fig. 6 and receptive field changes across epochs in Fig. 12.
Emergence of Sparse Representations from Noise

Figure 12: **Receptive fields become more biological across epochs.** We show how the most active receptive fields for the $\sigma = 0.3$ network evolve over the course of training to become more biologically similar. The specific number of epochs used (3,000, 9,000 and 19,000) were chosen simply because of when the models were checkpointed and re-loaded for additional training. We use the car image as our noisy query here (rightmost in Fig. 11).
E. \( L_1 \) compared to Noise

To test the performance, sparsity, and receptive fields of \( L_1 \) against noisy training, we trained 10,000 neuron models using Adam for 2,000 epochs to reconstruct CIFAR10 pixels. With the noisy models, we anneal noise from epochs 1000 to 1500 to give a more accurate final performance comparison. This resulted in dead neuron like those shown in Appendix J.3 but did not significantly change the number of dead neurons like for the reconstruction task in Fig. 7. We test noise levels \( \sigma \in \{0.05, 0.1, 0.3, 0.8, 1.5, 3.0, 10.0\} \) and \( L_1 \in \{1e-05, 1e-06, 1e-07, 1e-08\} \) where \( \leq 1e-04 \) resulted in all dead neurons.

Across methods there is also clear trend lines between dead neurons, sparsity, and model performance shown in Fig. 13. This suggests that even though noise and \( L_1 \) use different approaches and coefficients, they result in similar effects.

![Figure 13: Validation loss versus sparsity and dead neurons. The points within an approach that have more sparsity/more dead neurons correspond to higher noise or larger \( L_1 \) coefficient penalties.](image-url)
F. Noise Analytical Derivation

Some proofs have used first order Taylor approximations to show that noisy training minimizes the Frobenius Norm of the model's Jacobian. Here, we make no such approximation and are able to disentangle many more nuanced effects that noisy training implicitly has on the loss function.

\[
\text{Loss} = \frac{1}{D} \sum_{x \in X} \mathbb{E}_\varepsilon \left[ \sum_{i} (x_i - \tilde{y}_i)^2 \right] 
\]

(8)

\[
(x_i - \tilde{y}_i)^2 = (x_i - (\tilde{y}_i + \bar{y}_i - \bar{y}_i)) = (x_i - \bar{y}_i - \xi_i)^2 
\]

(9)

\[
= (r_i - \xi_i)^2 
\]

(10)

\[
\text{Loss} = \frac{1}{D} \sum_{x \in X} \sum_{i} r_i^2 = 2r_i \mathbb{E}_\varepsilon [\xi_i] + \mathbb{E}_\varepsilon [\xi_i^2] 
\]

(11)

(12)

where \( r_i = x_i - \bar{y}_i \) and \( \xi_i = \tilde{y}_i - \bar{y}_i \) is the difference between \( \tilde{x}_i \), the output produced by the input with noise \( \varepsilon \), and the output without noise \( \bar{y}_i \). Keep in mind that because the input noise is independent of the data, having \( \xi_i = 0 \) will maximize the quality of the reconstruction.

Figure 14: Graphical depiction of how each neuron in the hidden layer is affected by noise. Through our derivation we focus on one output unit and can trace the effects of noise back through the network.
To understand the loss, we expand fully the operations performed for a single $\xi_i$:

$$\xi_i = \sum_j W_{d_{i,j}} \eta_j$$  

$$= \sum_j W_{d_{i,j}} (\tilde{h}_j - \bar{h}_j)$$  

$$= \sum_j W_{d_{i,j}} \left([w_{e_j}(x + \varepsilon) + b]_+ - [w_{e_j}x + b]_+ \right)$$  

$$= \sum_j W_{d_{i,j}} \left(\sum_k W_{e_{j,k}}(x_k + \varepsilon_k) + b_j \right) - \left[\sum_k W_{e_{j,k}} x_k + b_j \right]$$  

The noise penalty will ultimately come down to the terms $E_x[\eta_j]$ and $E_x[\eta_j^2]$ shown next. The former will depend on how far an activation is away from the ReLU threshold at 0 and the latter how often the neuron is active. When combined, these terms explain why our network learns to become sparse and obtain biological receptive fields. The loss also has a de-correlation term that tries to learn orthogonal weights for the encoder neurons.

Looking at each of the noise terms in the loss we have:

$$-2r_i E_x[\xi_i] + E_x[\xi_i^2] = -2r_i \left(\sum_j W_{d_{i,j}} E_x[\eta_j]\right) + E_x \left[\sum_j W_{d_{i,j}} \eta_j \right]^2$$  

$$E_x \left[\sum_j W_{d_{i,j}} \eta_j \right]^2 = E_x \left[\sum_j W_{d_{i,j}} \eta_j^2 + \sum_{k \neq j} W_{d_{i,j}} W_{d_{i,k}} \eta_j \eta_k \right]$$  

We now show $\tilde{h}$ is a truncated Gaussian distribution and that $E_x[\eta_j] \geq 0$ because $E_x[\tilde{h}_j] \geq \bar{h}_j$.

Before applying the ReLU:

$$\tilde{\varepsilon}_j = \sum_k W_{e_{j,k}}(x_k + \varepsilon_k) + b_j$$  

$$= \sum_k W_{e_{j,k}} x_k + W_{e_{j,k}} N(0, \sigma^2) + b_j$$  

$$\sim N(\tilde{\varepsilon}, \sigma^2||w_{e_j}|^2)$$

Giving us a Gaussian centered on the noise free point. With $CDF(\frac{-\tilde{\varepsilon}}{\sigma ||w_{e_j}||})$ the noise is sufficient to flip a neuron from being on to off or vice versa. The mean of the truncated Gaussian will be $\bar{\varepsilon} + \int_{-\infty}^{\tilde{\varepsilon}} p(\varepsilon) \varepsilon d\varepsilon$ where this later portion is the region of the Gaussian where symmetry is not applied (these positive values are not cancelled out by the opposite values). Solving this integral gives us:

$$E_x[\tilde{h}_j] = \bar{h}_j + ||w_{e_j}|| \frac{\sigma}{\sqrt{2\pi}} \exp \left(\frac{1}{2} \left(\frac{\bar{h}_j}{||w_{e_j}|| \sigma} \right)^2 \right)$$

See Fig. 15 for intuition on why this is the case. Intuitively, the way to reduce the expected deviation is by: 1. shrinking the weight norm for the neuron $||w_{e_j}||$; 2. setting $\tilde{h}_j$ to be as far away from the ReLU 0 activation threshold as possible in either the negative or positive direction. While this penalty is independent of if the noise free activation is on or off, the next terms seek to have the neuron be off:

$$E_x[\eta_j^2] = E_x \left[(\bar{h}_j - \tilde{h}_j)^2 \right]$$
Emergence of Sparse Representations from Noise

Figure 15: **ReLU truncates the Gaussian causing a positive shift in the mean.** We provide intuition for why only the positive right hand side of the tail remains resulting in the mean of the post activation noisy neuron being higher than it would otherwise be. This increase in the mean is a function of how much noise causes the neuron to cross the activation threshold.

This is the deviation between the noisy and noise free activation, squared. The deviation depends upon if $\bar{z}_j > 0$:

When $\bar{z}_j > 0$:

$$E_\epsilon [(\bar{h}_j - \tilde{h}_j)^2] = \bar{z}_j^2 \int_{-\infty}^{-|\bar{z}_j|} p(\epsilon) d\epsilon + \int_{|\bar{z}_j|}^{\infty} p(\epsilon) \epsilon^2 d\epsilon$$

$$= \bar{z}_j^2 \text{CDF}(\frac{-\bar{z}_j}{\sigma^2|w_j|^2}) + \int_{|\bar{z}_j|}^{\infty} p(\epsilon) \epsilon^2 d\epsilon$$

(23)

When $\bar{z}_j \leq 0$:

$$E_\epsilon [(\bar{h}_j - \tilde{h}_j)^2] = 0 \cdot \int_{-\infty}^{-|\bar{z}_j|} p(\epsilon) d\epsilon + \int_{|\bar{z}_j|}^{\infty} p(\epsilon) \epsilon^2 d\epsilon$$

$$= \int_{|\bar{z}_j|}^{\infty} p(\epsilon) \epsilon^2 d\epsilon$$

(24)

See Fig. 16 for intuition on these integrals. Which can be integrated but is unnecessary to show a few things: 1. The penalty when the neuron is on ($\bar{h}_j > 0$) is strictly greater than when it is off. Not only is the integral over a larger interval $2|\bar{z}_j|$
Emergence of Sparse Representations from Noise

Figure 16: The variance of the noise and its contribution to the error. This term incentivizes neurons to be as negative as possible to avoid noise turning the neuron on.

longer, spanning \([-|\bar{z}_j|, \infty]\) instead of \([|\bar{z}_j|, \infty]\) but also there is the additional penalty term; 2. When the neuron is off \(\bar{z}_j\) should be as negative as possible. 3. When the neuron is on, there may be an equilibrium point where the two penalties equal each other, this will be a function of the neurons weights and bias term.

Finally, we look at the cross term which incentivizes de-correlating the neuron weights: \(E_{\epsilon}[\eta_j\eta_k]\).

This term is only present when both neurons are turned on and the probability of this occurring is reduced when \(\bar{h} < 0\) and becomes increasingly negative. Note also that \(\bar{h}_j\) and \(\bar{h}_k\) are not independent because they receive the same noise input. This means their correlation is a function of the cosine similarity between their weights. Therefore, a way to both utilize both neurons in the model (increasing model capacity) but avoid this cross term is by having their weights be orthogonal.

To summarize thus far, Eqs. 24 and 26 want to turn off neurons as much as possible (the reconstruction term will force some to be on); Eq. 22 wants neuron activations to be maximally far away from the ReLU 0 activation threshold. This equation also wants to minimize the amount of noise which is a function of the noise variance and the weight norm; Finally, \(E_{\epsilon}[\eta_j\eta_k]\) wants the neuron weights to be orthogonal. you can just as well decorrelate the neurons paid attention to by the decoder.

As a final analysis, we look at how each neuron should modify its weights and bias term to accomplish the above. The bias term is easy to start with, the model should set it to be as negative as possible while still making it possible for some neurons to become active, this will be a function of the features in the dataset, the norms of these features and the norms of weights and noise. What should the weights learn?

For any data point, the norm will scale noise as much as it scales the activation with the data because it has a linear effect on both: \(w_{xj}x/w_{ej} \epsilon = c\). Therefore, the way to both reduce noise and also maximize the activation threshold margin is by having the weights specialize to distinct dataset features. This will maximize the signal to noise ratio by ignoring noise across most of the input while being very activated by unique data features, therefore not firing all the time and firing strongly when it does. In other words, the weights should focus on capturing high variance regions of the data distribution and also should be orthogonal to other neurons.

Relating this loss to the \(L_1\) activation penalty of sparse coding we want to minimize \(E_{\epsilon}[\xi^2]\) which we have shown is always \(\geq 0\). Therefore, we can add an absolute value and seek to minimize:

\[
\sum_{j}^{m} W_{d_{i,j}}^2 |E_{\epsilon}[\eta_j^2]|
\]

Where this penalty is a linear sum of the number of neurons that are on as has the same flavour as an \(L_1\) penalty.
Bringing everything together, our final loss is:

\[
\text{Loss} = \frac{1}{D} \sum_{x \in X} \sum_{i} \epsilon^2 - 2r_i \sum_{j} W_{d,i,j} \mathbb{E}_{\epsilon}[\eta_j] + \sum_{j} W_{d,i,j}^2 \mathbb{E}_{\epsilon}[\eta_j^2] + \sum_{j} \sum_{k \neq j} W_{d,i,j} W_{d,i,k} \mathbb{E}_{\epsilon}[\eta_j \eta_k]
\]

(27)

A possible critique of this derivation is that there is a negative term that could reduce the loss by becoming large. Looking at the overall loss from which these terms are produced, we know that we want to minimize the noise in order to give the best reconstruction term. However, we still address this concern more deeply below.

\[
-2r_i \mathbb{E}_{\epsilon}[\xi_i] = -2(x_i - \bar{y}_i)(\sum_{j} W_{d,i,j} \mathbb{E}_{\epsilon}[\eta_j])
\]

(28)

\[
= 2 \sum_{j} W_{d,i,j} \mathbb{E}_{\epsilon}[\eta_j](\sum_{j} W_{d,i,j} \bar{h}_j - x_i)
\]

(29)

\[
= 2 \sum_{j} W_{d,i,j} \mathbb{E}_{\epsilon}[\eta_j](W_{d,i,j} \bar{h}_j + \sum_{k \neq j} W_{d,i,k} \bar{h}_k - x_i)
\]

(30)

\[
(31)
\]

If \(x_i\) is mean centered the \(-x_i\) term becomes 0 in expectation over the dataset.\(^5\) This leaves only the positive terms that we seek to minimize by: (i) shrinking our weights (the double weight summation can be written as the sum of the weight outer product matrix \([W_d W_d^T]\)); (ii) reducing \(\mathbb{E}_{\epsilon}[\eta_j]\) by increasing our maximum margin; (iii) turning off as many neurons with \(\bar{h}_j = 0\) as possible.

If \(x_i\) is not mean centered (it is not mean centered for the CIFAR10 pixels where \(0 \leq x_i \leq 1\); \(\forall i\)) then the loss can hypothetically be exploited by maximizing the reconstruction error \(x_i - \bar{y}_i\) and having the weights be the same sign. However, this is capped by the squared reconstruction error term in the loss function. Moreover, the way to maximize this term is by either: (i) making the decoder weights \(w_{d,i}\) large. This is up to the limit where \(x_i \geq \sum_{j} W_{d,i,j} \bar{h}_j\); (ii) making \(\bar{z}_j = 0\) so that \(\bar{h}_j = 0\) but this is capped by the variance terms which are minimized by \(\bar{z}_j\) being as negative as possible.

Therefore, while this is an interesting special case, there are many constraints on this solution that mean the optimal solution is still likely to be minimizing \(r\) and the noise terms.

G. Taylor Series Approximation Derivation

Here we trace the reasoning of (Poole et al., 2014), correcting incorrect results and furthering the analysis, to show how a similar conclusion can be reached via a Taylor series approximation.

Let \(\hat{x}(\epsilon)\) denote the reconstructed input with additive input noise \(\epsilon\), and let \(\hat{x}\) denote the reconstructed input without noise.

The loss is:

\[
\mathcal{L} = \sum_{x \in X} ||x - \hat{x}(\epsilon)||_2^2
\]

(32)

For small noise \(\epsilon\) with mean 0 and variance \(\sigma^2\), the first-order Taylor series expansion is \(\hat{x}(\epsilon) \approx \hat{x} + J(x)\epsilon\), where \(J(x) \equiv \nabla_x \hat{x}\) is the input-output Jacobian at \(x\). The expected loss (averaging over the noise) can then be written as (Bishop, 1995; Rifai et al., 2011):

\[
\mathbb{E}_{\epsilon}[\mathcal{L}] \approx \sum_{x \in X} ||x - \hat{x}||_2^2 + \sigma^2||J||_F^2
\]

(33)

The noise regularizes the Jacobian of the network, proportional to the noise’s variance. In our toy autoencoder model, the Jacobian is:

\[
J(x) = W_c^T \text{diag}(\phi'(z)) W_d^T
\]

(34)

\(^5\)Setting \(\mathbb{E}_{\epsilon}[\eta_j]\) to any positive constant and bringing in the expectation over the dataset gives \(\mathbb{E}_{x \in X}[x] = 0\).
where \( z \overset{\text{def}}{=} W_e x + b_e \) is the pre-activation. For commonly used nonlinearities, the nonlinearity domain can be split into two regions with non-zero gradients, and regions with (near) 0 gradients. We can split the Jacobian’s squared Frobenius norm into two terms based on these gradient regions:

\[
||J(x)||^2_F = \sum_{ij} |J(x)|^2_{ij}
\]

\[
= \sum_{ij; \phi'_h(z)\neq 0} \left( [W^T_e]_{ih} [\text{diag}(\phi'(z))]_{hh} [W^T_d]_{hj} \right)^2
\]

\[+
\sum_{ijh; \phi'_h(z)=0} \left( [W^T_e]_{ih} [\text{diag}(\phi'(z))]_{hh} [W^T_d]_{hj} \right)^2
\]

Minimizing the Jacobian’s squared Frobenius norm can be accomplished in one of two non-mutually-exclusive ways: (1) minimise the (gradient-scaled) squared dot product of the encoder and decoder weights, and/or (2) drive the pre-activations \( z \) to regions where the nonlinearity’s gradient is 0. Because the first term is a squared error, it is likely to contribute at least some error, but the second term contributes exactly zero error, meaning pushing all pre-activations \( z \) into the zero-gradient region is highly advantageous. However, the reconstruction loss creates an opposing pressure by incentivizing the network to faithfully recapitulate the uncorrupted data. To balance these competing demands, a compromise is struck: keep enough units “on” to reconstruct the data, then for the weights in use, regularize them, and for the weights not in use, drive the corresponding activations to the regions where the nonlinearity gradient is 0.

We note that while the math here describes a DAE with a single hidden layer, the same reasoning holds generally for feedforward networks on generic losses: the loss forces the network to propagate useful information in the data, while the noise implicitly regularizes the network’s Jacobian’s squared Frobenius norm, and since the Jacobian will always have the form of the pre-activation times derivative of nonlinearity times gradient of network output with respect to post-activation, moving the pre-activation to regions where the nonlinearity’s derivative is zero will help minimize the loss. Because most nonlinearities’ zero-gradient regions coincide with where the nonlinearity is “inactive” (e.g., \(< 0 \) for ReLU), the network’s activations become sparse.

This understanding yields two predictions: First, preactivations of different nonlinearities should increasingly concentrate in zero-gradient regions of the nonlinearity as noise variance \( \sigma^2 \) increases, for both standard nonlinearities (e.g., ReLU, GELU) and non-standard nonlinearities (e.g. horizontally reflected ReLU, \( \text{sat}(x) \overset{\text{def}}{=} \max(\min(x, 0.5), -0.5) \)). Second, the same noise-inducing-sparsity result should hold qualitatively in nontrivial deep feedforward neural networks.

We test each prediction in turn. Specifically, we test the first using the single-layer DAE by checking whether increasing noise drives preactivation values towards zero-gradient regions of different nonlinearities (Fig. 17), then test the second using nontrivial deep feedforward, convolutional and transformer architectures on standard benchmark tasks by checking whether increasing noise drives increasing sparsity (main text).

Figure 17: Noise drives nonlinearities to low-gradient regions of the nonlinearity’s domain
H. Relation between $\lambda$ coefficient and noise bias terms

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 H), a result from sparse coding. Assuming that the weights that sparse coding learns are orthogonal, this results in the optimal solution for neuron activity $a_i$ constrained to be non-negative is $\text{ReLU}(w_i^T x - \lambda)$, where $w_i^T x$ is the value obtained by projecting the data onto the neuron output weights $w_i$ (Ba, 2020; Gregor & LeCun, 2010). Intuitively, if the values of our target image $x$ reconstructed by $a_i w_i^T$ are too small, it is better for $a_i = 0$ than to predict $x$ with the threshold for this tradeoff given by $\lambda$. This $-\lambda$ term behaves as an activation threshold in an analogous fashion to the bias terms of our noisily trained network converging to a single shared negative value.

I. Ablations

I.1. Single MLP Layer Ablations

![Figure 18: Training ablations. The positive correlation between noise and sparsity is robust to (a) neuron counts, (b) datasets, (c) noise distributions, and (d) training tasks.](image)

For CIFAR10, we train each model for 3,000 epochs and present the mean number of active neurons for GELU (Fig. 19) and sigmoid (Fig. 20). We don’t show the latent CIFAR10 results as sigmoid and GELU are fully dense with no sparsity when the 0.0001 activation threshold is used. Neither of these networks are as sparse as the ReLU network. Also neither network creates sparsity through dead neurons, but given that all neurons have non-zero absolute activity, having dead neurons is impossible.

While we just show the mean active neurons here, both solutions look like those of Fig. 5 with the neurons for all inputs converging to a Top-$k$ solution where the bias terms become negative and synchronize along with the key vector $L_2$ norms.
Figure 19: The GELU network sparsifies with noise when trained on CIFAR10. The GELU network closely resembles the ReLU network by sparsifying (but not to the same degree). It also forms biological receptive fields. We use the arbitrary 0.0001 absolute activation threshold to label if a neuron is on or off.

Figure 20: The sigmoid network also sparsifies with noise for CIFAR10. While sigmoid sparsifies in proportion to noise, it does not form biological receptive fields and is less sparse than GELU or ReLU.
Figure 21: A subset of commonly used nonlinearities increase the sparsity of representations. On CIFAR-10 pixels, ReLU, GELU, ELU and Sigmoid nonlinearities cause the shallow denoising autoencoder representations to sparsify; sigmoid has a lesser effect, while tanh has no effect. This is explained by our mathematical analysis.
I.2. SGD Learning Rate and Batch Size

We discover that SGD is able to avoid the sparsifying effects of noise when either its batch size is large enough or its learning rate is low enough as shown in Figure 22. We hypothesize that both of these observations are explained by the random noise being cancelled out by the central limit theorem. Formally, let $g^*$ be the true gradient update on the full dataset and $\hat{g}$ be the minibatch gradient that approximates $g^*$ with some error $\epsilon_b$ caused by our minibatch: $g^* = \hat{g} + \epsilon_b$. In the presence of random noise, we can decompose our minibatch gradient further into $\hat{g} = \hat{g}_b + \hat{g}_n$ to represent the gradient from the data and the gradient from the noise. This gives us:

$$g^* = \hat{g}_b + \hat{g}_n + \epsilon_b.$$

Because $\hat{g}_n$ come from independent random variables with mean 0 and finite variance, by the Lyapunov Central Limit Theory, as our batch size $b \to \infty$ the variance of $\hat{g}_n$ will shrink by $\sigma / \sqrt{b}$ to zero. This means that as our batch size increases, the contribution of noise to the gradient converges to zero by cancelling itself out; the same reasoning can also be applied to the noise $\epsilon_b$ from the batch size.

Reducing the learning rate approximates increasing the batch size by allowing for the noise terms of the gradient step to cancel each other out across multiple steps. This can only occur if each gradient step is sufficiently small and the optimization landscape sufficiently smooth such that each intermediate gradient step does not cause the model to deviate from its noise free optimization trajectory.

As an additional ablation we tried giving our bias terms a separate, higher learning rate, hypothesizing that without a parameter specific adaptive optimizer the learning rate may be too small for the bias terms that have larger values than the weights. However, even with a large parameter sweep we found that having a different bias term learning rate did not have an effect.

When we test SGD without any noise, we find that neither modifying the batch size nor learning rate have any sparsity effect. This supports our hypothesis that increasing batch size or reducing learning rate helps to reduce the random noise injection, rather than having a smaller batch size introducing a new source of noise. However, there is evidence that with a very high learning rate and a sustained plateau in training loss, the minibatch can be a source of label noise that induces sparsity as analyzed in (Andriushchenko et al., 2022).

Therefore, using SGD with a low enough learning rate or large enough batch size removes the effects of noising training. This removes the implicit loss term of Eq. ?? and results in a model that is not sparse.\(^6\)

The fact a non-sparse solution to the de-noising problem exists is on the surface problematic for our statements about sparse coding being more optimal. However, we found that this result did not generalize to the more challenging raw CIFAR10 pixels dataset. Even using the full batch of 50,000 images and a low learning rate resulted in a sparse coding solution that was strongly correlated with train and validation loss.

More broadly, the fact that SGD with low learning rate and large batch size can have different inductive biases may generalize to other problems and findings. For example, we are aware of SGD with a low learning rate failing to produce networks with weights that could be interpolated between using Git Rebasin (Ainsworth et al., 2022).

\(^6\)One caveat is that the models retain their initial 50% sparsity rather than becoming fully dense as they do when there is no noise present.
Figure 22: **SGD Sparsity as a function of Batch Size and Learning Rate.** We train 1,000 neurons in a single hidden layer on the reconstruction task for latent CIFAR10 with three noise amounts (one for each plot). We allow the lower learning rates to have more gradient steps where the [learning rate, training steps] pairs are: [0.001, 800k], [0.01, 600k], [0.1, 400k], [1.0, 200k] and we confirm that all runs have converged to the same training loss and there are no dead neurons. To make the sweeps more efficient, we randomly sample 5,000 datapoints out of the 50,000 from the CIFAR10 256 dimensional latent embeddings and use this same subset for all experiments. The only anomaly in our results is in subfigure (c) on the bottom row where batch size is 128 and learning rate is 1.0 and the model is denser than other batch sizes. This anomaly is robust to random seeds and deserves further investigation.
J. Dead Neurons

J.1. Stale Momentum

In order to avoid Stale Momentum SGD must be used but this removes the very advantages that adaptive optimizers were developed for – namely faster convergence that is more robust to hyperparameter choices. As a result, we introduce SparseAdam which was originally developed to make backpropagation more computationally efficient for highly sparse models in the “Sparse Layers” library (PyTorch, 2022). We implemented SparseAdam to work with dense layers and found that it reduces the number of dead neurons somewhat but not entirely as shown in Fig. 23 for the three layer MLP. We believe this is because, while it prevents inactive neurons from being updated by an out of date moving average, it fails to prevent gradient spikes when neurons are re-activated after a long period of quiescence.

The AlexNet (Sec. K.3) and Deep MLP (Sec. K.2) appendices both have further analysis of the different optimizers effect on dead neurons and sparsity. Meanwhile, the next section J.2 provides further analysis on when and why Adam results in dead neurons.

![Figure 23: Adam leads to more dead neurons than Sparse Adam or SGD in the later layers of Deep MLPs.](image-url)

We compare the final fraction of dead neurons for Adam and SGD (top row) and Adam and Sparse Adam (bottom row) on the full CIFAR10 latent embedding dataset on a classification task. We train each model for 2,000 epochs and anneal all initial noise levels down to zero between epochs 500 and 1,000. We leave out Layer 0 because there are no dead neurons for any of optimizers or noise levels.
J.2. Dead Neurons from Adam

We investigated more deeply when and why Adam produces dead neurons resulting in a few conclusions:

- Adam kills fewer neurons when it has a larger batch size – this is because more neurons are activated by at least one batch element resulting in a non-stale gradient update (Fig. 24).
- Adam kills neurons only after it has minimized its training and validation losses and converged to its sparse coding solution (Fig. 25).
- The total number of dead neurons is a function of dataset complexity. This includes the size of the dataset and the intrinsic dimensionality of the datapoints (Fig. 26).
- Neurons die more as a function of epochs through the whole dataset than the number of gradient steps (Fig. 27).

As shown in Fig. 24, there are fewer dead neurons with a larger batch size or smaller learning rate. This is because a large enough batch size results in more neurons being activated at least once and receiving a gradient update that is not stale.

Figure 24: Larger batches reduce dead neurons with Adam. Across noise levels, the larger the batch size the fewer the dead neurons. All models achieve the same validation accuracy. The models are sparser in direct proportion to the number of dead neurons. We don’t show the first layer (Layer 0) because it does not have any dead neurons. For these experiments the learning rate is always 0.0001.

Figure 25 shows how Adam starts to kill neurons only after both training loss and the number of dead neurons have converged. Figure 26 shows how dataset size and complexity determines the number of dead neurons Adam eventually creates. These two plots show different datasets (latent CIFAR10 (top) and raw CIFAR10 pixels (bottom)) and we vary the total size of the dataset $D$ where the full dataset size is $D = 50,000$. The two main observations are: (i) the less complex latent CIFAR10 dataset results in many more dead neurons, almost independently of $D$; (ii) the larger $D$ is the more slowly neurons are killed off and the lower the total number is.

Finally, neurons die as a function of epochs rather than gradient steps. It is more than just the size of the dataset and noise introduced by the batch size but also the number of passes through the full dataset. In Fig. 27 we show how the smallest batch size of 2 kills neurons very quickly as a function of epochs but very slowly as a function of gradient steps because it takes much longer to iterate through the dataset of 5,000 latent CIFAR10 images.
Figure 25: **AlexNet Killing Neurons Long After Convergence of both Training Loss and Active Neurons**. We use the same settings as Fig. 4 with $\sigma = 3.0$ on the full CIFAR10 latent dataset but 1,000 neurons instead of 10,000 and for a much larger number of epochs. We label the 800th epoch which shows the point where both training loss (top left) and mean number of active neurons (top right) have converged. Meanwhile, the fraction of dead neurons is only just about to go from $\sim 0$ to 92% (bottom right). Validation loss looks the same as train loss.
Figure 26: Adam Dead Neurons as a Function of Dataset Complexity and Size. We train Adam keeping all parameters constant and $\sigma = 3.0$ aside from the size of the dataset denoted by $D$ for latent and raw CIFAR10. The runs with smaller dataset sizes did not complete as many gradient steps due to a bug with model checkpointing but the results are sufficient for a few observations: First, the 256 dimensional latent CIFAR dataset allows for much higher numbers of dead neurons than the 3072 dimensional raw CIFAR dataset. Second, the smaller the dataset, the faster dead neurons are created. Finally, the latent CIFAR10 dataset is sufficiently simple that $\sim 92\%$ of neurons eventually die almost independently of dataset size. We truncate the latent CIFAR10 (top) at 2 million gradient steps to make the gaps between lines more visible.
Figure 27: **Neurons die as a function of epochs through the full dataset.** Using the random 5,000 latent CIFAR10 images and $\sigma = 3.0$ with Adam we vary the batch size from full batch $B = 5,000$ (green) to $B = 2$ (dark blue). The top row shows epochs (truncated for clarity to 60k) and the bottom shows gradient steps. The rate at which neurons are killed is reversed between the plots. This shows that $B = 2$ kills neurons at a slower rate per gradient step but much faster as a function of total passes over the dataset.
J.3. Dead Neurons from Noise Annealing

Here we show the number of dead neurons that are present when we anneal our noise value to 0 for reconstruction (Fig. 28) and classification (Fig. 29). This source of dead neurons is different from that which emerges from using the Adam optimizer.

Figure 28: **Dead Neurons during the reconstruction task.** During noise annealing, the higher noise levels $\sigma \geq 0.8$ see a number of dead neurons. However, this does not fully explain their sparsity levels and on this trivial task it does not hurt their reconstruction performance.
Figure 29: **Dead Neurons during the classification Task.** Here there are fewer dead neurons, they only appear for $\sigma \geq 3.0$ and they only appear close to when the noise is entirely turned off. Again these results do not fully explain the sparsity amounts found.
K. Deeper Models

K.1. Transformer

Here we provide the training and validation performance for the Transformer as well as sparsity values. This is after 1.7 million training steps when the models look to have converged. Table 1 shows all of the data and Fig. 30 provides a visual depiction of how sparsity levels change for each layer as a function of noise. Only the largest noise level $\sigma = 8$ leads to a slight reduction in model performance.

<table>
<thead>
<tr>
<th>Noise $\sigma$</th>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
<th>Train Loss</th>
<th>Val. Loss</th>
<th>Val. Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.171</td>
<td>0.184</td>
<td>0.296</td>
<td>2.790</td>
<td>3.048</td>
<td>0.437</td>
</tr>
<tr>
<td>0.05</td>
<td>0.173</td>
<td>0.178</td>
<td>0.291</td>
<td>2.782</td>
<td>3.038</td>
<td>0.437</td>
</tr>
<tr>
<td>0.1</td>
<td>0.164</td>
<td>0.165</td>
<td>0.287</td>
<td>2.793</td>
<td>3.029</td>
<td>0.438</td>
</tr>
<tr>
<td>0.3</td>
<td>0.172</td>
<td>0.117</td>
<td>0.252</td>
<td>2.769</td>
<td>3.025</td>
<td>0.439</td>
</tr>
<tr>
<td>0.8</td>
<td>0.155</td>
<td>0.056</td>
<td>0.186</td>
<td>2.759</td>
<td>3.002</td>
<td>0.442</td>
</tr>
<tr>
<td>1.5</td>
<td>0.133</td>
<td>0.046</td>
<td>0.148</td>
<td>2.778</td>
<td>3.009</td>
<td>0.441</td>
</tr>
<tr>
<td>3.0</td>
<td>0.058</td>
<td>0.022</td>
<td>0.137</td>
<td>2.830</td>
<td>3.011</td>
<td>0.440</td>
</tr>
<tr>
<td>8.0</td>
<td>0.013</td>
<td>0.001</td>
<td>0.055</td>
<td>3.073</td>
<td>3.134</td>
<td>0.420</td>
</tr>
</tbody>
</table>

Table 1: Transformer Sparsity and Performance Levels - We show the mean number of active neurons for each batch across all three layers in addition to the noise levels and training loss obtained at the end of 1.7 million training steps on the WikiText-103 dataset. After 200k training steps the noise levels are annealed down to zero. Only the highest noise level $\sigma = 8$ has a noticeable hit to performance while $\sigma = 3$ has a minor hit to performance while being significantly sparser.
Figure 30: **Transformer Layer Sparsity.** Only the most sparse model $\sigma = 8.0$ has a real hit to train and validation loss.
K.2. Deep MLP

Here we provide the validation accuracies for the Deep MLPs in Table 2 and use Fig. 31 to show how sparsity levels and dead neurons change for each layer as a function of noise, comparing the different optimizers.

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>Noise</th>
<th>Train Acc</th>
<th>Val. Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam</td>
<td>0.0</td>
<td>1.0</td>
<td>0.915</td>
</tr>
<tr>
<td>SGD</td>
<td>0.0</td>
<td>1.0</td>
<td>0.920</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>0.0</td>
<td>1.0</td>
<td>0.915</td>
</tr>
<tr>
<td>Adam</td>
<td>0.1</td>
<td>1.0</td>
<td>0.925</td>
</tr>
<tr>
<td>SGD</td>
<td>0.1</td>
<td>1.0</td>
<td>0.923</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>0.1</td>
<td>1.0</td>
<td>0.924</td>
</tr>
<tr>
<td>Adam</td>
<td>0.5</td>
<td>1.0</td>
<td>0.926</td>
</tr>
<tr>
<td>SGD</td>
<td>0.5</td>
<td>1.0</td>
<td>0.924</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>0.5</td>
<td>1.0</td>
<td>0.927</td>
</tr>
<tr>
<td>Adam</td>
<td>1.5</td>
<td>1.0</td>
<td>0.924</td>
</tr>
<tr>
<td>SGD</td>
<td>1.5</td>
<td>1.0</td>
<td>0.928</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>1.5</td>
<td>1.0</td>
<td>0.923</td>
</tr>
<tr>
<td>Adam</td>
<td>3.0</td>
<td>1.0</td>
<td>0.917</td>
</tr>
<tr>
<td>SGD</td>
<td>3.0</td>
<td>1.0</td>
<td>0.923</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>3.0</td>
<td>1.0</td>
<td>0.921</td>
</tr>
<tr>
<td>Adam</td>
<td>5.0</td>
<td>1.0</td>
<td>0.916</td>
</tr>
<tr>
<td>SGD</td>
<td>5.0</td>
<td>1.0</td>
<td>0.919</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>5.0</td>
<td>1.0</td>
<td>0.917</td>
</tr>
<tr>
<td>Adam</td>
<td>10.0</td>
<td>1.0</td>
<td>0.915</td>
</tr>
<tr>
<td>SGD</td>
<td>10.0</td>
<td>1.0</td>
<td>0.923</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>10.0</td>
<td>1.0</td>
<td>0.915</td>
</tr>
</tbody>
</table>

Table 2: Deep MLP Validation Accuracy is better for small amounts of noisy pretraining but independent of dead neurons - We show the final validation accuracies for the Deep MLP models that have been noise annealed for the CIFAR10 latent embeddings classification task. Models with small amounts of noise do slightly better than the baseline without noise. There is no significant difference in performance between noise levels or optimizers, even though they kill off different numbers of neurons (Fig. 31).

The SGD optimized model only has dead neurons in its final layer and is almost as sparse as Adam which has the most dead neurons. This is accomplished by the SGD model using negative bias terms in its first layer and then negative weights in its later layers to ensure fewer neurons are activated. This is in contrast to Adam and SparseAdam which only create sparsity through dead neurons. It is worth pointing out that SparseAdam does use negative bias terms to create sparsity and never has dead neurons before noise is annealed, but when it is the neurons then die and become the source of sparsity while the bias terms then become positive.
Figure 31: **Deep MLP Layer Sparsity and Dead Neurons across Optimizers.** Adam (top row) produces the most sparse models but this is almost entirely a consequence of dead neurons. Meanwhile, SGD (bottom row) is the only model to avoid dead neurons in its second layer (orange line) and implement a sparse coding solution analogous to the results from our single layer MLP. Note the different ranges for the y-axes.
K.3. AlexNet

Here we provide the validation accuracies for AlexNet in Table 3 and use Fig. 32 to show how sparsity levels change for each layer as a function of noise. Interestingly, the noisy pretraining significantly hurts validation accuracy even for the smallest noise amounts but not training accuracy (the model always overfits in every case to get 100% train accuracy). The noise was found to produce dead neurons like in the DeepMLP setting. Believing these dead neurons may explain the decrease in validation accuracy, we trained the network using SGD and SparseAdam. While this solved the dead neuron problem (see Fig. 33), Table 3 shows no increase in validation accuracy. Testing AlexNet with Average instead of Max Pooling also did not alleviate the noise performance gap.

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>Noise</th>
<th>Train Acc</th>
<th>Val. Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam</td>
<td>0.0</td>
<td>1.0</td>
<td>0.815</td>
</tr>
<tr>
<td>SGD</td>
<td>0.0</td>
<td>1.0</td>
<td>0.710</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>0.0</td>
<td>1.0</td>
<td>0.820</td>
</tr>
<tr>
<td>Adam</td>
<td>0.5</td>
<td>1.0</td>
<td>0.614</td>
</tr>
<tr>
<td>SGD</td>
<td>0.5</td>
<td>1.0</td>
<td>0.575</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>0.5</td>
<td>1.0</td>
<td>0.618</td>
</tr>
<tr>
<td>Adam</td>
<td>1.5</td>
<td>1.0</td>
<td>0.540</td>
</tr>
<tr>
<td>SGD</td>
<td>1.5</td>
<td>1.0</td>
<td>0.559</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>1.5</td>
<td>1.0</td>
<td>0.551</td>
</tr>
<tr>
<td>Adam</td>
<td>3.0</td>
<td>1.0</td>
<td>0.597</td>
</tr>
<tr>
<td>SGD</td>
<td>3.0</td>
<td>1.0</td>
<td>0.568</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>3.0</td>
<td>1.0</td>
<td>0.602</td>
</tr>
<tr>
<td>Adam</td>
<td>5.0</td>
<td>1.0</td>
<td>0.613</td>
</tr>
<tr>
<td>SGD</td>
<td>5.0</td>
<td>1.0</td>
<td>0.593</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>5.0</td>
<td>1.0</td>
<td>0.618</td>
</tr>
<tr>
<td>Adam</td>
<td>10.0</td>
<td>1.0</td>
<td>0.648</td>
</tr>
<tr>
<td>SGD</td>
<td>10.0</td>
<td>1.0</td>
<td>0.625</td>
</tr>
<tr>
<td>SparseAdam</td>
<td>10.0</td>
<td>1.0</td>
<td>0.647</td>
</tr>
</tbody>
</table>

Table 3: AlexNet Validation Accuracy Damage From Any Noisy Pretraining - We show the final train and validation accuracies for the AlexNet models across optimizers. The SGD and Sparse Adam reductions in dead neurons do not translate into validation accuracy improvements.

In this light, it is interesting that the Transformer is immune to dead neurons and that noise does not affect final validation accuracy. This is particularly surprising given that noise is injected at every Transformer MLP layer instead of just at the first layer. We hypothesize this may be due to the Transformer’s residual connections and LayerNormalization (Ba et al., 2016).

We also experimented with injecting noise after the five convolutional layers, just before the MLP layers. However, this also harmed validation accuracy (albeit to a lesser extent). The noisy pretraining results in validation accuracies from 76% (for $\sigma = 0.5$) down to 70% (for $\sigma = 10$) compared to the baseline 82%. Like with injecting noise at the deeper layers of the MLP, doing so caused the convolutional layers to become more active than they would otherwise be and this may have contributed to the reduced performance.
Figure 32: **AlexNet Sparsity and Dead Neurons By Layer.** We see the same effect upon noise annealing here as with the DeepMLP where the first layer starts sparse but then becomes significantly more dense than baseline while the later layers all remain more sparse. Note the different y-axes scales for each plot.
Figure 33: **AlexNet Dead Neurons By Optimizer.** SGD (left plot) results in no dead neurons while SparseAdam (right) creates fewer dead neurons in all but one case (all points aside from one are below the dotted line $y = x$). Dots correspond to the layer and we plot all noise levels on the same plot for parsimony.
L. Why Transformer Attention, Hopfield Nets, and Sparse Distributed Memory show the inverse relationship between sparsity and noise

For Transformer Attention, the difference is that it has direct access to every input within its receptive field but no further. This provides for high accuracy when dealing with these inputs (keys) but an inability to store long term memories. As a result, in the low noise regime, it is optimal for Attention to implement a form of nearest neighbour lookup, activating few keys. When there is more noise, especially noise that moves the query so far from its target key that it is no longer the nearest, it is optimal to activate more keys and average over them, losing accuracy but giving a solution in the correct neighbourhood.

By contrast, SDM can store long term memories. However, it has the same negative correlation between noise and sparsity as Attention because it attempts to store full patterns with maximum fidelity within each neuron instead of distributing the features of each pattern across neurons (Kanerva, 1988). This results in SDM attempting to also perform a nearest neighbour lookup, activating only as many neurons around the noisy query as is necessary to activate those storing the target pattern.

Our sparse coding networks in this work take the opposite approach where they learn subcomponents that generalize across images instead of all the features of specific images. As a result, they always want to use as many neurons as possible, but must reduce the number used in the presence of noise to avoid interference.