Sparse Coding Dream Machines

Henry Allen, Saarang Panchavati, Abhinav Pottabathula, Ashwath Radhachandran

VS265 - December 2020

1 Introduction and Background

In 1983, Crick and Mitchison proposed a theory of REM sleep, where the purpose of dreams is hypothesized to be to "unlearn," or to remove parasitic connections between neurons in the brain. One such mathematical and computational formalism of this is the wake-sleep algorithm [2], which aims to recreate this biological idea through an dual phase unsupervised learning algorithm. In the wake phase we aim to strengthen recognition connections between layers — we try to generate a suitable representation of the data. In the sleep phase, we aim to strengthen generative connections between increasingly abstract layers. That is, from the layer of most abstraction, we aim to generate weights that can generate the representation of the next most abstract layer. The output of this sleep phase can be viewed as a "dream." We in visualized these dreams to see how a wake sleep model may create a representation of natural images. To make this more biologically relevant, instead of directly training a wake sleep model on natural images, we trained it on a sparse coding representation of natural images. This additional abstraction and sparsity has the potential to give us very fascinating insights into how we learn, and how we dream. In general, most successful vision models are not biologically relevant or meaningful. Here we have made an effort to augment traditional neural networks with more biologically relevant formalisms.

2 Methods

2.1 Sparse Coding

For this experiment, we built a vanilla sparse coding model and trained it on a variety of image sets. The model was constructed in Python using numpy. There are two main algorithmic components of the model; a Locally Competitive Algorithm (calcLCA) that uses a gradient descent procedure to learn activations for an input, and a second gradient descent loop (calcPhi) that learns a feature set from a batch of images. See figure 1 for details. We trained this model on 4 different image sets, shown in figure 2. From left to right, these are randomly generated bars, mnist numbers, natural images, and faces.

The randomly generated bars were created in python with numpy, and were designed to mimic the random bars found in Foldiak's [2] paper on anti-hebbian learning. These images were 16x16, with 32 possible horizontal or vertical bars. We expect that the model will learn each of the individual bars, i.e. the components that make up the bar images. The face dataset came from sklearn's Olivetti faces dataset [5]. We centered each of these 64x64 images around the mean. This dataset was ultimately not included in our final results, as the images were too time-consuming to train on and the dataset was too small, but we did perform initial analyses on these faces, some of which can be seen in our demo colab [7] and in the results section. The numbers were taken from the classic MNIST dataset, which contains 28x28 handwritten digits, labelled from 0 to 9. We ended up not including the sparse coding features from MNIST in our final dream machine, but we did test on these images, and we trained our final dream machine on MNIST inputs. We trained our final sparse coding feature set on 28x28 whitened natural image patches. These patches were cut out of larger natural images, and randomly shuffled for use in our sparse coding model.

For training, we used a batch size of 100 for all inputs. For the random bars, fewer training steps were required due to the low complexity of the components (3000 iterations). The natural images required a much longer training period, 6000 iterations, as there were many more activations (64 vs. 400), and the inputs were a larger size (256 vs. 784 pixels). We needed this larger 28x28 input size for our features because we would eventually want to input 28x28 MNIST images into the wake phase of our Sparse Coding Helmholtz machine, which would require image features that were also 28x28.

```
calcLCA(Phi, X, alpha, lambda, thresh) {
    G = Phi.T @ Phi
    G = G - I
    a = zeros([num_activations, num_images])
    u = zeros(shape(a))
    do{
        prev_u = u
        u = (1 - lambda) * u + alpha * (Phi.T @ X - G @ a)
        a = sign(u) * max(abs(u) - lambda, 0)
    } while (||prev_u - u|| > thresh)
    return a
}
```

```
calcPhi(X, n, eta, batch_size) {
    Phi = randomArray([num_activations, image_size])
    Phi = Phi @ diag(1 / ||Phi[i]||)
    for (i = 0; i < n; i++) {
        batch = sample(X, batch_size)
        a = calcLCA(Phi, batch, 0.01, 0.1, 0.05)
        Phi = Phi + eta * (batch - Phi @ a) @ a.T
        Phi = Phi @ diag(1 / ||Phi[j]||)
    }
    return Phi
}
```

Figure 1: Code

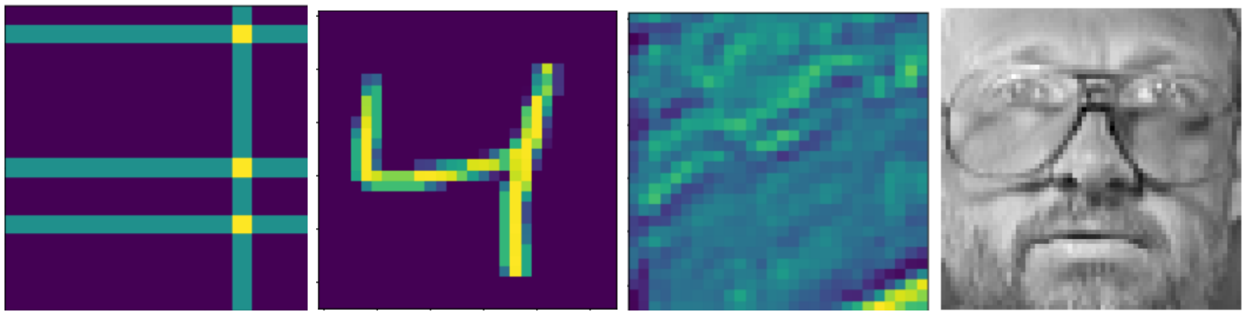

Figure 2: Example sparse coding inputs

2.2 Helmholtz Machine

Inspired by Hinton's 1995 paper [4] on the wake-sleep algorithm, we chose to implement this algorithm through a three-layer Helmholtz machine. The wake-sleep algorithm is an unsupervised learning algorithm that is able to learn the underlying representation of input vectors during the recognition phase, and then reconstruct an approximation of the inputs during the generative phase. As outlined in Hinton's paper, a three-layer Helmholtz machine was successful in generating "fantasies" of handwritten digits. Hinton and his colleagues chose to train a separate network for each digit, and this model was used to generate learned representations of the digit. Our goal was to replicate this experiment by generating similar "fantasies" of handwritten digits. After successfully doing this, we looked to train the Helmholtz machine using the activations from our sparse coding model and then test the model's ability to generate interpretable images.

The Helmholtz machine module was written in python using numpy. We adapted pseudocode written by Kevin Kirby, a professor from Northern Kentucky University, so that our Helmholtz implementation was able to take in user inputs for number of layers and number of nodes per layer. This provides the user with more flexibility when determining the model's architecture. Our implementation of the Helmholtz machine's wake and sleep phase are outlined in the psuedocode below. See figure 3 for implementation details. During the wake phase, the neurons are driven by the recognition weights, and the generative weights are adapted to increase the probability of reconstructing the vector passed in from the layer below. On the other hand, during the sleep phase, neurons are driven by the generative weights, and the recognition weights are then adapted to increase the probability of producing the vector from the layer above. A simple visualization of a three-layer Helmholtz machine is show in figure 4.

```
wake_phase(X) {
    outputs = [X]
    for layer in layers:
        sig = sigmoid(outputs[-1] @ layer.R)
        outputs.append(sample(sig))

    zeta = sigmoid(bias)
    bias = bias + epsilon * (outputs[-1] - zeta)

    for i, layer in enumerate(layers):
        delta = sigmoid(outputs[i+1] @ layer.G.T)
        layerG_upd =
            outputs[i + 1].T @ (outputs[i] - delta)
        layerG_upd = epsilon * layerG_upd
        layer.G = layer.G + layerG_upd.T
}
```

```
sleep_phase(X) {
    p = sigmoid(bias)

    outputs = [sample(p)]
    for layer in layers[::-1]:
        p = sigmoid(layer.G @ outputs[-1])
        outputs.append(sample(p))

    dreams.append(outputs[-1])
    for i,layer in enumerate(layers[::-1]):
        psi = sigmoid(outputs[i + 1].T @ layer.R)
        layerR_upd =
            outputs[i+1] @ (outputs[i].T - psi)
        layerR_upd = epsilon * layerR_upd
        layer.R = layer.R + layerR_upd
}
```

Figure 3: Code

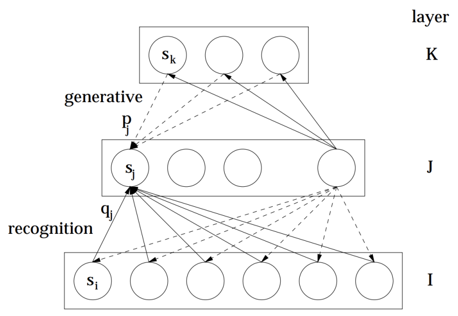

Figure 4: Visualization of Helmholtz Machine

2.3 Model Integration

The integration of our model can be summarized by three main steps: sparse coding activations, followed by Helmholtz, followed by sparse coding image generation. The goal behind this effort was to create a sparse feature representation prior to feeding the features into the Helmholtz wake-sleep model. The idea is that this would then improve the efficiency of the Helmholtz model, and would provide a more sparse representation of the images. We first began by writing the algorithms for sparse coding and Helmholtz separately and tested them on datasets of bars, mnist and faces. Then we replaced what would normally be raw images inputted into the Helmholtz model, and replaced them with the activations that were generated as a result of the sparse coding model. After doing this, the regenerated images seemed to be completely blank images which was surprising. It was then realized that the images needed to be processed in order for the pixel values to have enough contrast to get properly sparsely encoded. When we had used the bar images for sparse coding this was not a problem, but when we first used images of faces it became clear that the facial features did not have a strong enough contrast to the surroundings of the picture. In order to remediate this issue, we experimented with changing the images' contrast, blur, and sharpen. Contrast was implemented by squaring each pixel value. Blur and sharpen were implemented by adding a gaussian filter over the image. Changing the contrast served to be the most effective tool, as it allowed the sparse coding model to better identify important parts of the image. The idea of increasing the contrast of the image. After adding this intermediary step of stratifying the pixel values, the face image activations were properly generated by the sparse coding and were able to successfully be processed by the Helmholtz machine.

2.4 Experiments

After integrating the model, a series of experiments were run in an effort to quantify the impact of using sparse coding as a setup for helmholtz. The following are a list of experiments which are further elaborated in the Results section.

- Standalone Helmholtz Model
- Standalone Sparse Coding Model
- Performance Comparison of Naive and Sparse Coding Helmholtz Models
- Investigation of Network Self-Organization

We have made these experiments openly available in the form of Google Collab notebooks [6] [7] [8].

3 Results

3.1 Helmholtz Model

We trained a 3 layer Helmholtz Model on subsets of the MNIST dataset. We found that even with 1000 iterations, the simple model generated a reasonable representation of the data, and was able to generally have fairly accurate "dreams" after several iterations.

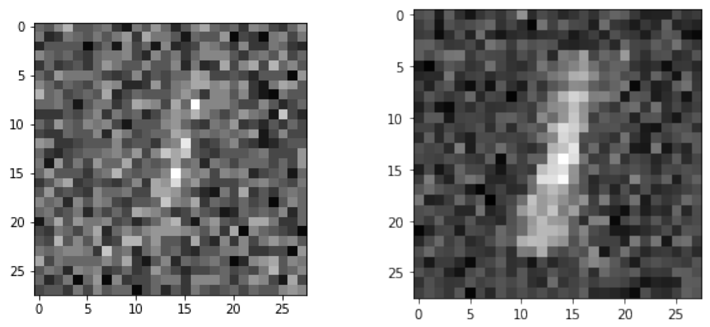

Figure 5: Dreams after 0 and 1000 iterations of the wake sleep algorithm on a 2 layer Helmholtz model.

The classical Helmholtz model uses the binomial distribution on a learned parameter p in its representation and generation of new images. However, this only works for simple *binary* images, and not more complex greyscale images. In order to generate dreams on more complex images, we used a beta distribution with one parameter. This introduced some additional noise in the dreaming, as the parameter of the beta distribution can never be 0. However, this allowed us to generate more robust dreams of complex and richer images, rather than simpler binarized images.

3.2 Sparse Coding Model

Initially, we only tested our sparse coding model with 3 different image input types: random bars, mnist, and faces. The random bar test was a sanity check to make sure our model was working correctly. A correct model should learn individuals bars from the lattice of overlaid bars, and this is exactly what we found. This is seen in figure 6, which shows the learned features, the image input, and then a reconstructed version of the image that uses the learned activations and features. We then proceeded optimistically to testing on face datasets and image datasets, as we wanted to generate these types of inputs in our Helmholtz machine, but quickly ran into a problem. The features we generated just looked like different versions of the images that we input into the model, not the sparse features we expected, see figure 7. We realized that there was a flaw in this methodology, and there wouldn't be any way to learn the individual components from inputs that were too similar to each other.

We then pivoted towards learning sparse features from natural image datasets, which we could then use to reconstruct numbers or faces. This gave us the features we expected, as seen in fig 8, and allowed us to continue with our investigation using the Helmholtz machine. The natural image features resemble spatial frequency patches, similar to the oriented simple cells that we would find in V1. Our learned features code for a wide variety of frequency and orientation, similar to what we could find in a V1 hypercolumn. This similarity in structure gives us the motivation for our experimentation with the Helmholtz Machine; we wanted to treat the features we learned through the sparse coding model as neurons that might get activated in V1, and we wanted to examine how an unsupervised model might organize itself. This might give us some insight into how mid-level visual cortex areas organize themselves given inputs from V1 cells.

Ultimately, our sparse coding model worked correctly, but we did not initially understand how to use it. After some missteps with our input datasets, we switched over to training on natural image patches, and we found that we were able to reconstruct most images that we passed into the model, even numbers from the mnist set. This was a much better alternative to individually training a model for each feature set we would want to learn, e.g. for faces, numbers, flowers, etc., which did not actually work as intended, and would take too long to train.

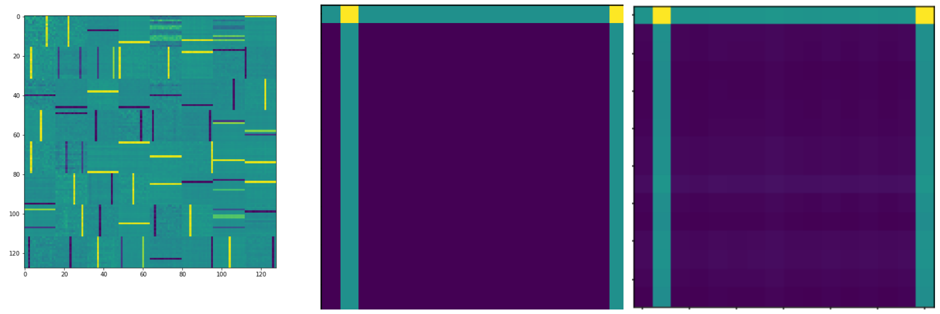

Figure 6: Sparse Features and reconstruction of random bars

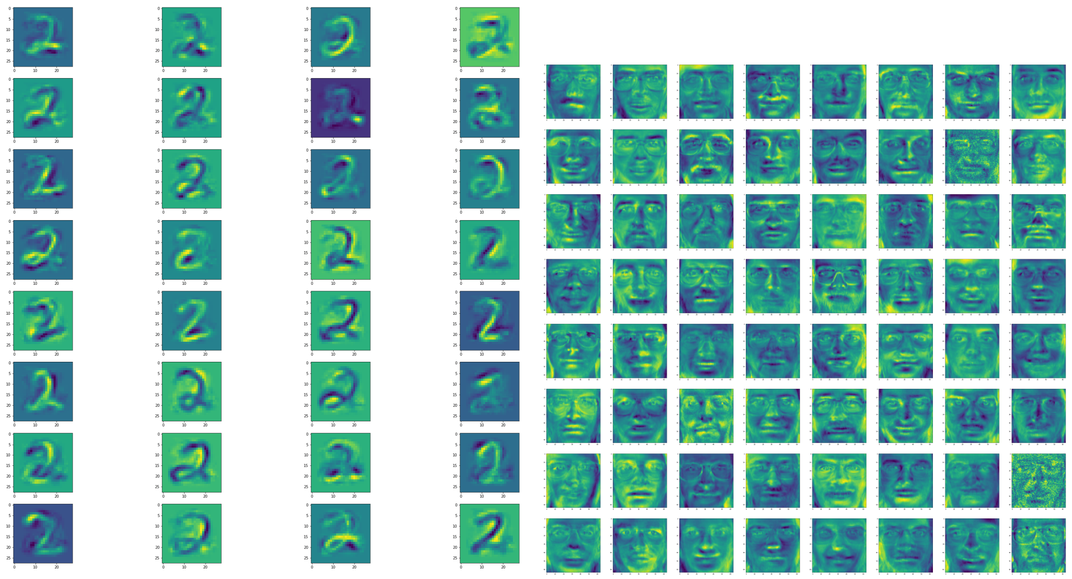

Figure 7: Learned features for mnist ”2” and faces

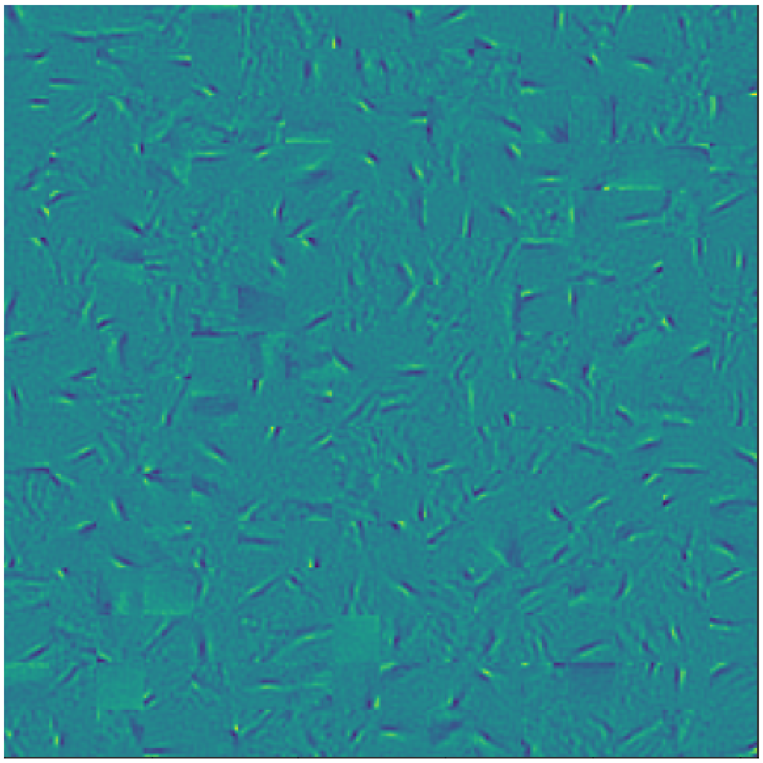

Figure 8: 16x16 Natural Image Features

3.3 Performance Comparison of Naive and Sparse Coding Helmholtz Models

We compared the performance of the sparse input helmholtz model to the pixel input model in two metrics: Mean Squared Error (MSE) and Peak Signal-to-noise Ratio (PSNR). PSNR is derived from MSE, but it provides an image processing interpretation of our results, whereas MSE provides a statistics perspective. We also included our experimental Lasso Helmholtz machine in these results, but this model is still being developed, so take any of these results with a grain of salt. We first compared the model performances in generating an mnist ”2”. We can see from figure 10 that the naive model actually achieves the lowest MSE and highest PSNR, meaning that it is closest to the original image. This was initially disappointing, but it seems that the naive model actually produces very clear results, perhaps due to the simplicity and large quantity of zero values in the number input. The sparse coding had a lot of noise, perhaps because we did not train the sparse coding model long enough to get truly distinct features. The final images that our Helmholtz machine produced can be seen in figure 9.

We then moved on to comparing the performance of the Helmholtz machines on natural image inputs, where we hoped that the sparse coding model would have an edge due to the complexity of the input. Indeed, we found that the sparse coding model gave much lower error and a much higher PSNR, as seen in figure 12. This implies that our sparse helmholtz model is much more likely to be successful on more ”realistic” imagery, as found in our natural image inputs than the naive pixel model; however, our PSNR values are still relatively low, at an average of 16 dB. This PSNR value wouldn't be acceptable for video compression, which typically requires values around 30-50 dB. The results can be seen in figure 11.

These results are encouraging, as we were able to use an unsupervised learning algorithm to recognizably recreate an input that we passed in. It's likely that we could get more intelligible results if we tuned our hyperparameters and had more time to train our models. The results also show the strengths and weaknesses of our model. For recreations of simple inputs, like numbers, the training time of the sparse coding model adds on unnecessary computation time and even seems to add noise (though this latter issue could be fixed with a more overcomplete feature set and longer training times). For recreations of more complex inputs, the naive model fails, while our sparse coding helmholtz model is relatively successful, making the training time tradeoff worthwhile.

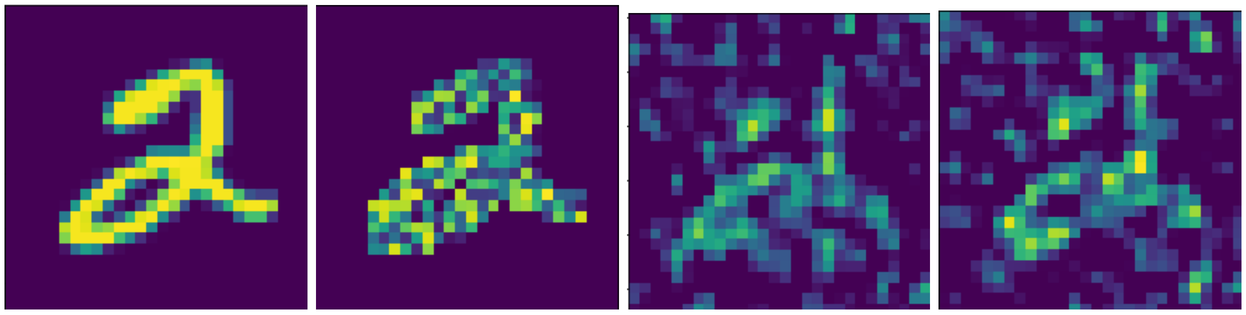

Figure 9: Helmholtz results for MNIST 2. Left to Right: Test, Naive Model, Sparse Coding, Sparse Coding + Lasso

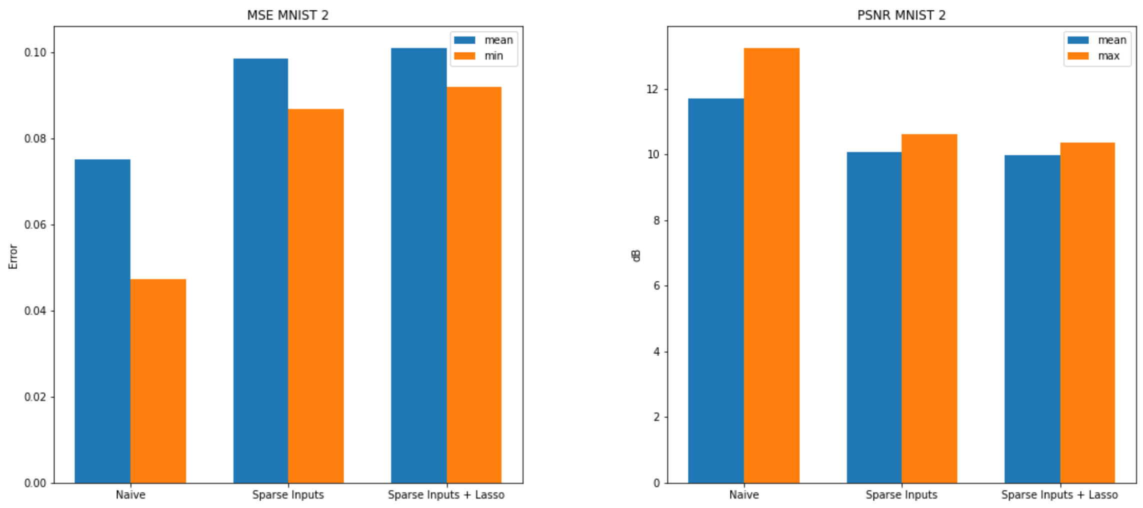

Figure 10: Comparing Sparse Code and Pixel Helmholtz Machines in MNIST

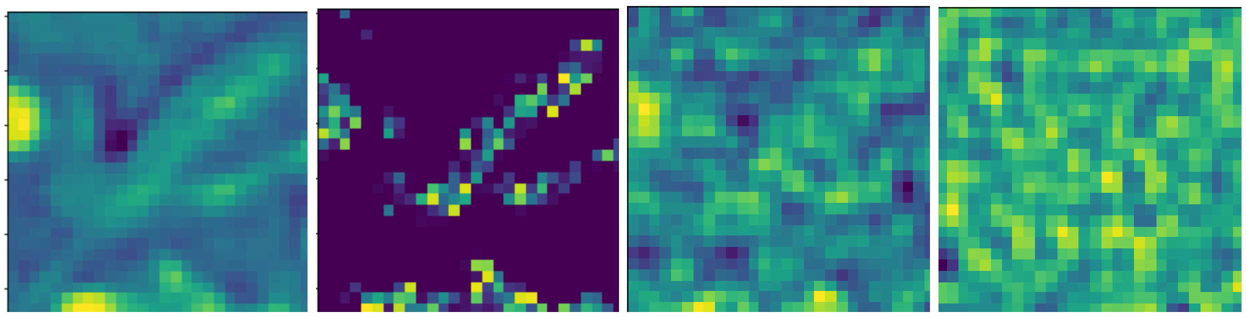

Figure 11: Helmholtz results for Natural Images. Left to Right: Test, Naive Model, Sparse Coding, Sparse Coding + Lasso

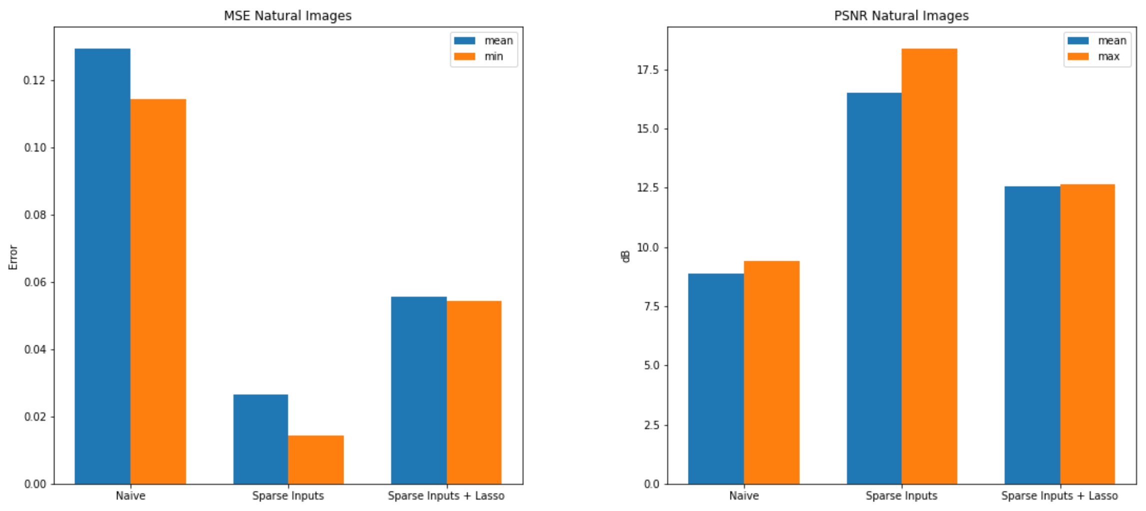

Figure 12: Comparing Sparse Code and Pixel Helmholtz Machines in Natural Images

3.4 Investigation of Network Self-Organization

We also sought to investigate how our Helmholtz model might organize itself, in hopes that this process might give us insight into how the mid/high-level visual cortex organizes itself, given inputs from V1 cells. To gain this insight, we looked at the weights of our sparse coding helmholtz machine on mnist inputs, with the sparse coding features trained on natural images. This should give us our best chance at interpretability, as numbers are composed of various simple line segments, which could easily be reconstructed from our natural image features. Our helmholtz model was composed of two layers, with feed-forward recognition weights, and backwards generative weights. The first layer had $n = image_size$ neurons, and we hard-coded the second layer to have 4 neurons. In our experiment, we hoped to find that each of these second-layer neurons would generate unique components of the output image, which could then be combined to form our desired output, a ”2” in this case.

We isolated each of the neurons by creating a new Helmholtz machine with all hidden weights zeroed out, except for those corresponding to the neuron we wanted to test. The weights for the visible layer were unchanged. This ensured that we only got the hidden layer activations for the neuron we wanted to examine. We then generated an image with these new weights, for each of the 4 neurons in the hidden layer, which can be seen in figure 13. From this figure, we do not get indivudal components, but different versions of the ”2” that we passed in as input. There are slight individual differences between the neuron activations, but it seems that they all converge towards the same representation of the ”2” that we passed in as an input.

We then tried examined how our Helmholtz model might learn features from inputs that were randomly sampled from a set of twos. For two images, as seen in figure 14, we could not find any interpretable features when isolating our hidden layer neurons. However, we did find that these isolated neuron dreams were more distinct than our dreams from the single input test. More investigation is needed here, as it’s possible that the addition of another hidden layer and the usage of more sample images could reveal the component features that we expect. Further, our model may require more training time to reach an equilibrium state.

Ultimately, our attempt to tease out any meaning from isolated model weights was a failure. For our experiment with a single image input, all of our neurons learned the same activation pattern, rather than learning distinct ones. This could mean that our model just converges to one representation, and that multiple neurons in our hidden layer are redundant. For our second experiment, we investigated if training the Helmholtz model on multiple inputs could utilize more of the hidden layer neurons. We found some differentiation between our hidden layer neurons, but the dreams they created were unintelligible.

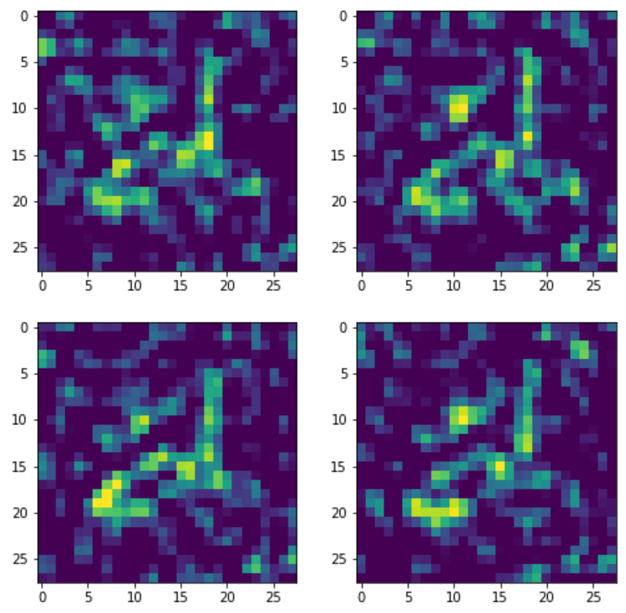

Figure 13: Dreams for 4 Isolated Neurons

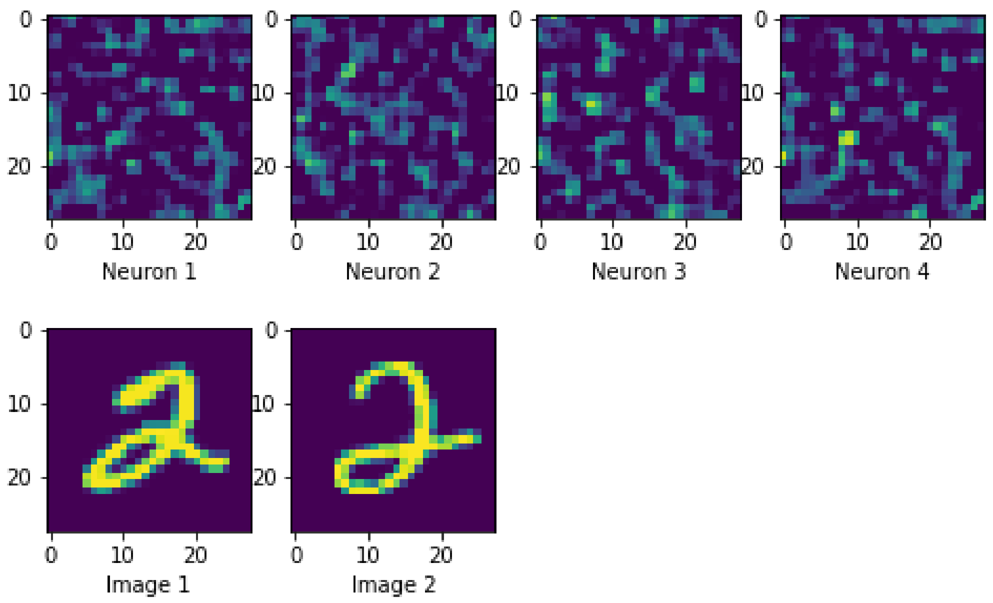

Figure 14: Isolated Dreams Generated for 4 Neurons with 2 Images

4 Discussion

We set out to investigate whether the integration of sparse coding activations into a Helmholtz machine could improve performance over a naive implementation, and whether this integration could tell us anything about how the visual cortex might organize itself. We had hypothesized that the features learned from sparse coding would mimic the receptive fields of simple cells in V1, and that our Helmholtz machine would mirror the learning and pruning of neurons in the visual cortex and learn mid-level features of an image input. To test this, we trained a sparse coding model on natural image patches. We then trained a Helmholtz machine on activations of these features corresponding to different image inputs, and compared the results to a naive model that used only pixel inputs. Finally, we isolated the hidden layer neurons in our Helmholtz machine to learn what components of our inputs they represented, if any. This method shows promise, as we were ultimately able to improve performance over the naive pixel model for natural image inputs, but we did not have as much success in interpreting our model's weights as we had hoped. Our hidden layer seems to converge to a single representation of our input, rather than component parts. We also noticed that the sparse model is more sensitive to small perturbations than the naive model, as a small change in sparse coding activations has a much larger effect on the output image than a change in a single pixel value. This could perhaps be solved with a more overcomplete feature set, which would give us more resilience to these small changes via more redundancy.

This lack of interpretability could be a result of a lack of hyperparameter tuning in the Helmholtz machine or the relatively short training time of our sparse coding model. With more investigation, we might be able to find our desired results. Due to the short time frame for this paper, we jumped straight into building and combining our two models, but our research likely would have benefited from a deeper study of unsupervised methods of learning. It may be the case that another model, like a Restricted Boltzmann machine, may have suited our needs better than the Helmholtz machine. We also made some false assumptions about the sparse coding model, assuming that training on MNIST images would give us component parts of the numbers, rather than the different versions of the numbers that we actually got. In hindsight, this result is obvious. We ultimately were unable to properly examine how the model evolved over time, and unable to interpret the significance of our model's weights after training. Nevertheless, there may be some promise in deriving sparse coding features prior to model learning, as we were able to successfully reconstruct natural images and MNIST numbers with our Helmholtz machine. This method could be used particularly well in conjunction with supervised learning techniques for object recognition, or perhaps in supervised generative networks. The features generated by sparse coding models are much more interpretable than raw pix values, potentially allowing these models to mimic the visual cortex.

Unsupervised learning algorithms like Sparse Coding and the Helmholtz machine offer a really interesting area for future study. The model we created here was just a "toy" model, and was very removed from any biological processes. First of all, our model was really composed of two separate learning stages; a sparse coding stage and a Helmholtz stage. In a more accurate biological model, we would seek to integrate these two stages, such that the model learns sparse coding features at the same time as the network updates its connections. This could help us with our interpretability, as mid-level components may be learned via co-activations from the sparse coding model. Further, we would also like to apply sparsity to the Helmholtz machine itself. We began this process by creating a Lasso regularized version of our Helmholtz machine, but we didn't have too much time to investigate it. We hoped that this model would cause many of the weights to become zeroes, resulting in a single hidden layer neuron strongly activating a few lower layer neurons, while completely ignoring all other neurons. However, it's also possible that this regularization could just act like a dropout layer, and turn off some neurons that are unnecessary. This isn't a bad thing, but it's not quite what we're looking for. Finally, while building the sparse coding model, we noticed that our model had difficulty learning the component parts of certain image sets that we passed in, due to the similarity of the inputs. We hypothesize that we could do a better job of learning a sparse feature set of these similar inputs by randomly "masking" portions of our inputs, artificially creating differences between our similar inputs. This masking could be done by choosing a 2D Gaussian window with a random mean sampled from within the size of the image and a random standard deviation. This could almost act like foveation of a certain part of the image.

Finally, there is still a lot of interesting work to be done in understanding why we dream. Here, we built a toy model based on Crick and Mitchison's Wake-Sleep algorithm [1], and we were able to generate some pretty cool pictures. I believe that this computational modeling of dreams can help us begin to understand the meaning of our dreams, and can be used in concert with neuroimaging and psychology studies to build a better picture of why and how we dream. There are many parts of dreams that still need to be examined computationally, like the motion and audition of our dreams. Our model only takes in simple cell inputs, but the brain obviously has a wealth of sensory inputs to utilize in dreaming. A similar experiment might be conducted with audio samples, to see how sound can be recreated in dreams. There are also parts of dreams that we can't really understand through computational modeling, particularly the emotional and narrative aspects of the dreams. It doesn't seem possible that these stories we tell ourselves can just be the product of random neural activations being pruned away. We look forward to investigating dreams in the future.

References

[1] Crick, F., Mitchison, G. The function of dream sleep. Nature 304, 111–114 (1983). https://doi.org/10.1038/304111a0

[2] Földiák, P. Forming sparse representations by local anti-Hebbian learning. Biological Cybernetics, 64(2), 165–170 (1990).
`https://doi.org/10.1007/bf02331346`

[3] Olshausen, B., Field, D. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature 381, 607–609 (1996). `https://doi.org/10.1038/381607a0`

[4] Hinton, G.E., P. Dayan, B.J. Frey and R.M. Neal. 1995. The wake-sleep algorithm for unsupervised neural networks. Science 268: 1158-1161. `https://doi.org/10.1126/science.7761831`

[5] sklearn: The Olivetti faces dataset
`https://scikit-learn.org/0.19/datasets/olivetti_faces.html`

[6] GitHub: VS265 Final Project - Dream Machines
`https://github.com/saarangp/dreammachines`

[7] Saarang Panchavati: Sparse Coding Dream Machines
`https://colab.research.google.com/github/saarangp/dreammachines/blob/main/Sparse_Coding_Dream_Machines.ipynb`

[8] Henry Allen: Sparse Coding Demo
`https://colab.research.google.com/github/saarangp/dreammachines/blob/main/Sparse_Coding_Demo.ipynb`