# COSC179 - Coding Assignment III Writeup

Eric Brown

Feburary 2026

# Problem 3A - Sparse Autoencoder

## Design/Approach

The implemented model is a sparse autoencoder, an unsupervised neural network derived from an assignment originally created by Andrew Ng at Stanford. The model is trained on 10,000 randomly sampled 8×8 patches from a set of natural images, and its goal is to learn a compressed, sparse representation of those patches using only 25 hidden units.

The key components of the sparse autoencoder are:

- **Image Sampling:** 10,000 random 8×8 patches are sampled from a set of 10 natural images and flattened into 64-dimensional vectors to form the training set.
- **Encoder/Decoder:** A three-layer network with 64 input units, 25 hidden units, and 64 output units using sigmoid activations. The encoder compresses the input into a sparse hidden representation, and the decoder attempts to reconstruct the original patch from it.
- **Cost Function:** The training objective combines three terms — a reconstruction error, an L2 weight decay penalty to prevent weights from growing too large, and a KL divergence sparsity penalty that penalizes hidden units whose average activation deviates from the target $\rho = 0.01$.
- **Gradient Checking:** A numerical gradient check using $\epsilon = 10^{-4}$ is used to verify that the analytically computed gradients from backpropagation are correct before full training begins.
- **Training:** The model is optimized using scipy.optimize.fmin_l_bfgs_b

## Results

The following figure depicts the results from training. They are similar to the results in the README.md, but seem less defined. Further parameter tuning may lead to more efficient models.

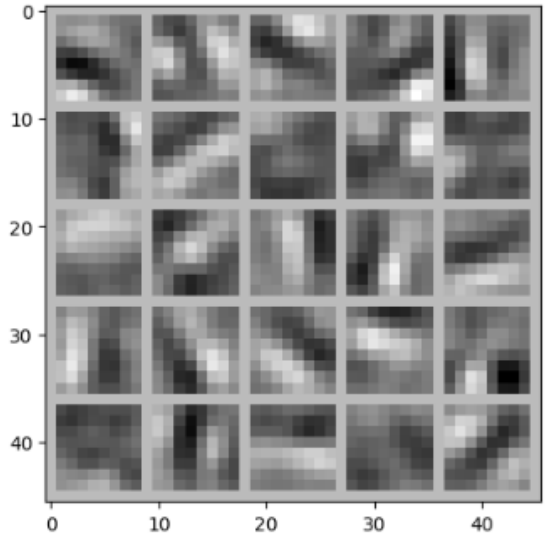


Figure 1: The figure depicts the 25 learned weights

## Evaluation Analysis

The autoencoder successfully trained and converged. The untrained model scored **48.85** on the reconstruction metric, while the trained model reduced this to **0.481**, demonstrating effective learning. The learned weight visualizations show clear edge detectors at various orientations, consistent with the expected result described in the assignment.

## Reflection

This lab was a good introduction to how diffusion models, such as DALLE, work in practice. Interestingly, the model is really just learning to predict noise, yet this is enough to generate some real digits. It was also interesting to see how quickly the loss dropped in the first two epochs, going from 0.0795 down to 0.0372, suggesting the model grasped the broad structure of the task early on. This could have been negative, though, if the model is overfitting on a minority class. The unclear digits being generated could be why.

# Problem 3B - Training a Transformer

## Design/Approach

The implemented model is based on the "nanoGPT" [1]. The model is trained on the Shakespeare Work Dataset [2]. It learned to predict the next word/token in a sequence.

- **Context Length (256):** The number of previous tokens the model can see when making a prediction. 256 was chosen as a balance between capturing enough context to produce coherent sentences without requiring excessive memory
- **Embedding Dimension (384):** The size of the vector representing each token. 384 provides enough capacity to capture meaningful word relationships while keeping the model a manageable size
- **Transformer Blocks (6):** The depth of the model. 6 layers allows the model to learn progressively more abstract patterns in the text without being too slow to train
- **Attention Heads (6):** The number of parallel attention mechanisms. 6 heads let the model attend to different parts of the context simultaneously, with each head operating on a dimension of $384/6 = 64$
- **Dropout (0.2):** A regularization rate of 20% was used to prevent overfitting on the relatively small Shakespeare dataset
- **Learning Rate (1e-3):** The learning rate helps the model converge faster, which is needed on a limited dataset

## Evaluation Analysis

The model parameters were adopted from Andrej Karpathy's nanoGPT [3] project, which has proven to be a good baseline. Reasoning is then explained.

The total model size is 10,690,625 parameters, consistent with the README.md findings.

[1] (https://github.com/karpathy/nanoGPT)
[2] (https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinyshakespeare/input.txt)
[3] (https://github.com/karpathy/nanoGPT/blob/master/config/train$_s hakespeare_c har.py$)

## Reflection

This lab made the inner workings of large language models much more concrete. Implementing the attention mechanism in code made the mechanism clearer. I have more experience implementing Machine Learning software than the actual underlying math. Although the math may be confusing at times, it is really interesting that the research community figured out how to build these statistical methods. In recent years, it has been fascinating to see the growth in some areas, like image generation, are seeing.

# Problem 3C - Denoising Diffusion Probabilistic Models

## Design/Approach

The implemented model is based on the paper "Denoising Diffusion Probabilistic Models"[4]. The model is trained on the MNIST dataset, 28x28 grayscale handwritten digits. The model's intended purpose is to generate new digit images by injecting noise during training.

The key components of the diffusion model are defined in the README.md as:

- **Noise Scheduler:** Controls how much noise is added at each step, following a linear schedule that starts subtle and grows stronger
- **Sinusoidal Time Embedding:** Tells the network which step of the denoising process it is currently on, so it knows how much noise to expect and remove
- **U-Net:** The neural network responsible for predicting the noise. It compresses the image down to understand the overall noise pattern, then expands back up using skip connections to restore fine detail
- **Training:** At each step, the model is shown a noisy image and asked to predict the noise that was added. The difference between its prediction and the actual noise is used to improve the model
- **Sampling:** To generate a new image, the model starts from pure random noise and repeatedly removes small amounts of predicted noise until a clean digit emerges

[4](https://arxiv.org/abs/2006.11239)

## Results

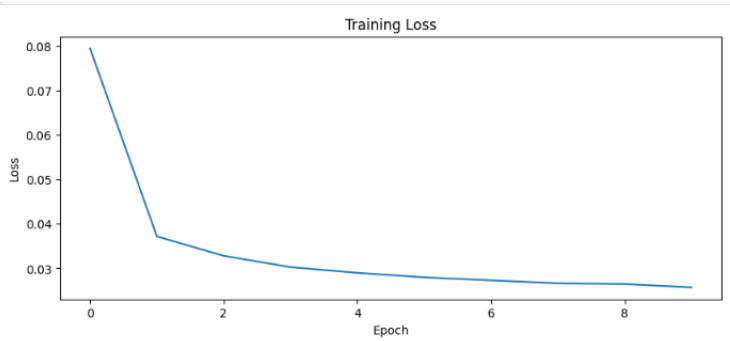


Figure 2: The figure depicts the loss curve over training epochs

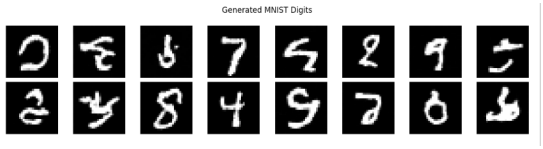


Figure 3: The figure shown is the images generated by our model

## Evaluation Analysis

The generated samples show semi-recognizable digit shapes after only 10 epochs, confirming the model successfully learned the MNIST distribution. The loss curve follows the expected pattern. A sharp initial drop as the model learns to predict large-scale noise, followed by slower convergence as finer detail is refined and more accurate images are generated. With more robust methods and a larger dataset, the model accuracy has the potential to improve.

## Reflection

This lab was a good introduction to how diffusion models, such as DALLE, work in practice. Interestingly, the model is really just learning to predict noise, yet this is enough to generate some real digits. It was also interesting to see how quickly the loss dropped in the first two epochs, going from 0.0795 down to 0.0372, suggesting the model grasped the broad structure of the task early on. This could have been negative, though, if the model is overfitting on a minority class. The unclear digits being generated could be why.