Neural data compression optimization project

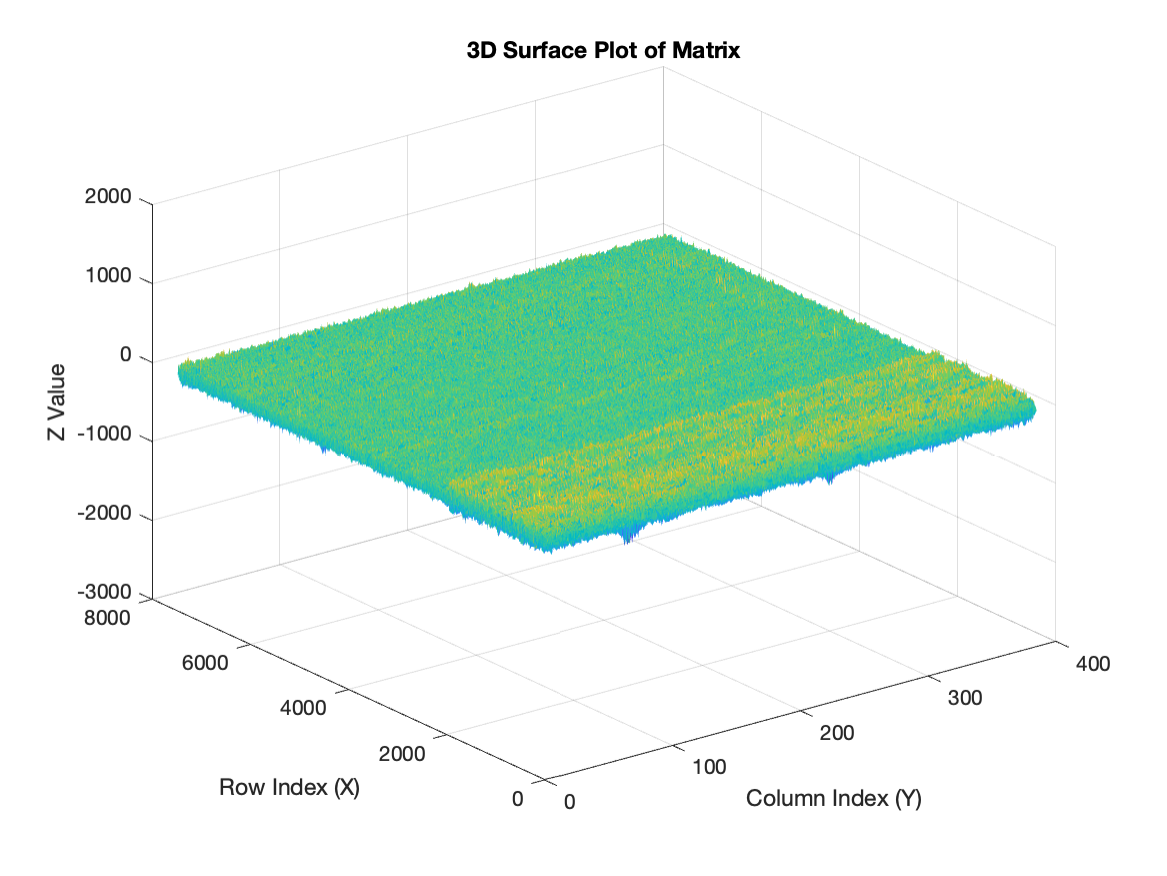
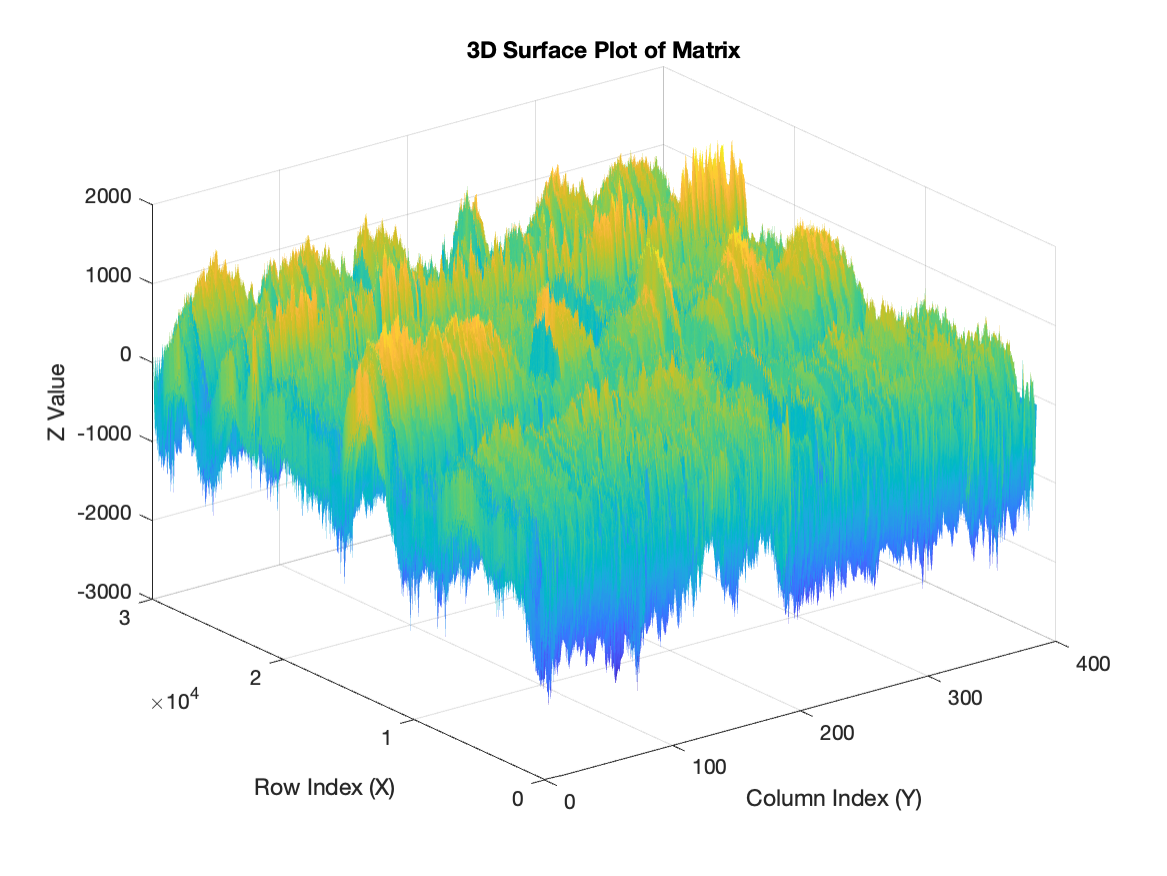
## Motivation:

With recent innovation in neural recording devices, a single Neuropixel generates 80GB of neural data per hour. It presents a challenge to store these large amount of data with a low carbon footprint. The high data rate (144Mbps for Neuropixel) also made it unsuitable for portable wireless transmission (Bluetooth: 2Mbps max, WiFi is too power consuming). Therefore lossless compression is key in addressing these challenges.

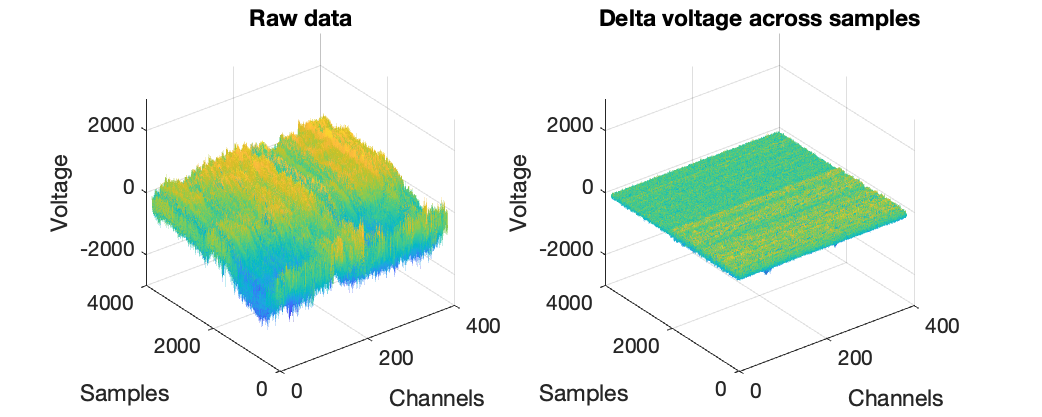
pasted-image.tiffpasted-image.tiff

## Overview:

Raw recording data from Neuropixel 2.0 (high density neural recording probe) is used in this optimization project. The compression pipeline consists of two stages, delta encoding and followed by auto-encoding. First stage is lossless, meaning the process can be reverted and obtained the exact original data, however the second stage is lossy, therefore we need to optimize the auto-encoding neural network to minimize the loss (quantified by MSE between the recovered data and the original data).

Stage 1: Delta encoding (pre-processing)

Entropy = 10.35 => Entropy = 7.6



Stage 2: Auto-encoder

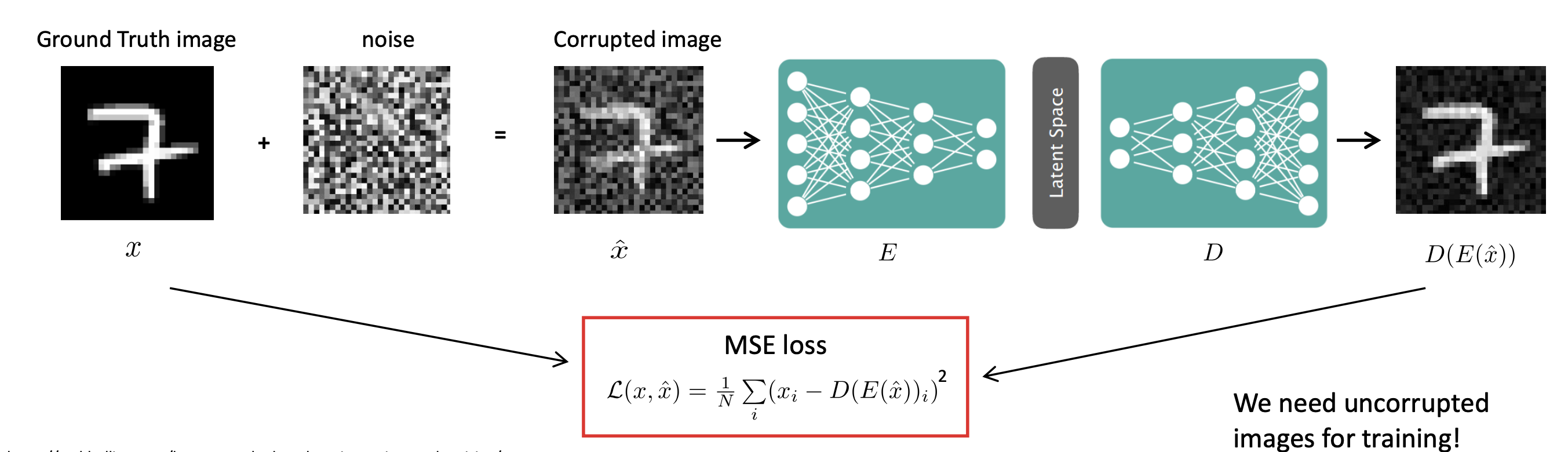
**Input:** delta-encoded neural activity from Neuralpixels 2.0 (chunk size = 384(channel)\*384(time))

**Encoder:** compresses the input data into lower dimensional representation, and captures the most important information.

**Latent space:** captured the compressed and essential neural data for storage/transmission

**Decoder:** mirror image of encoder, takes the compressed representation and reconstruct close to original input data, by increase the dimensionality (lossy)

**Output:** decoded data

**Goal:** Working the loss between output and ground truth, and then optimizing the weights of both the encoder and decoder networks through gradient decent and back propagation algorithm, to minimize the reconstruction error, thereby forcing the autoencoder to learn an efficient and meaningful representation in the latent space.

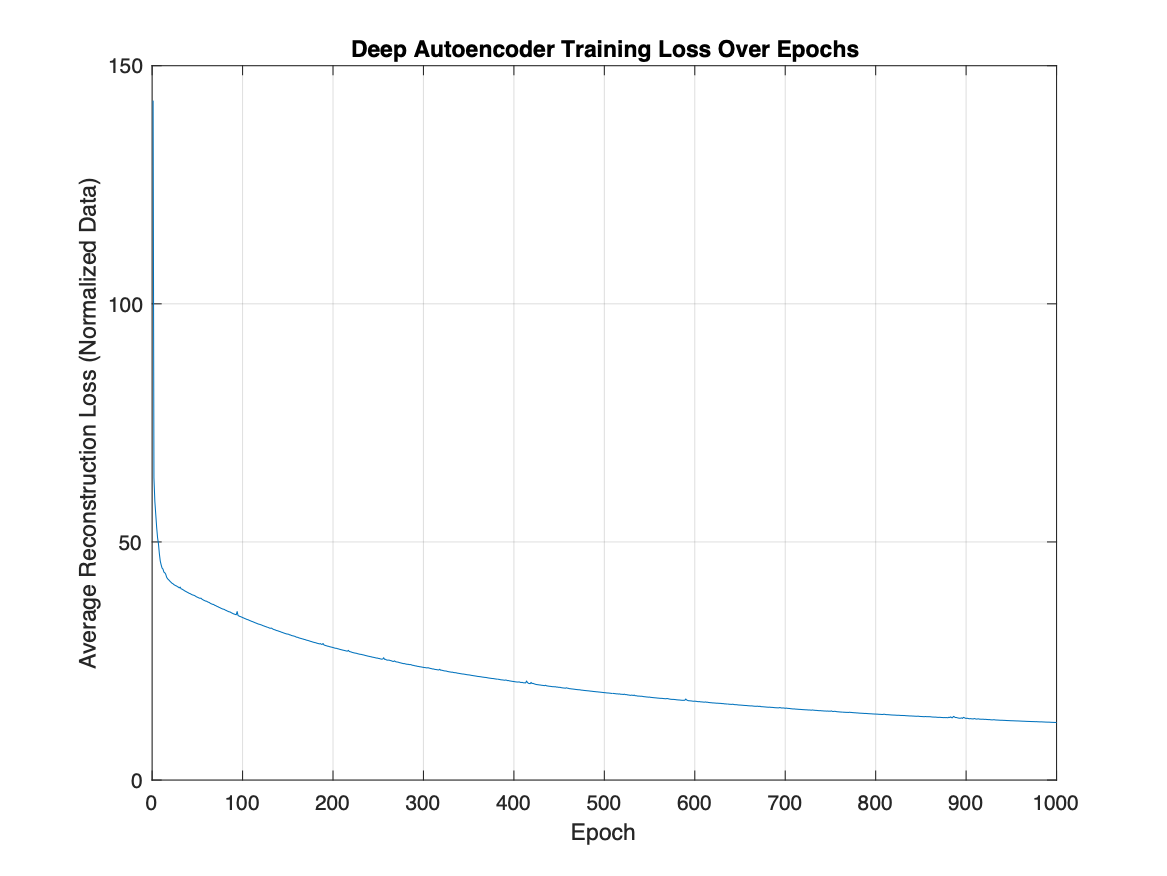
Input: 384 channels

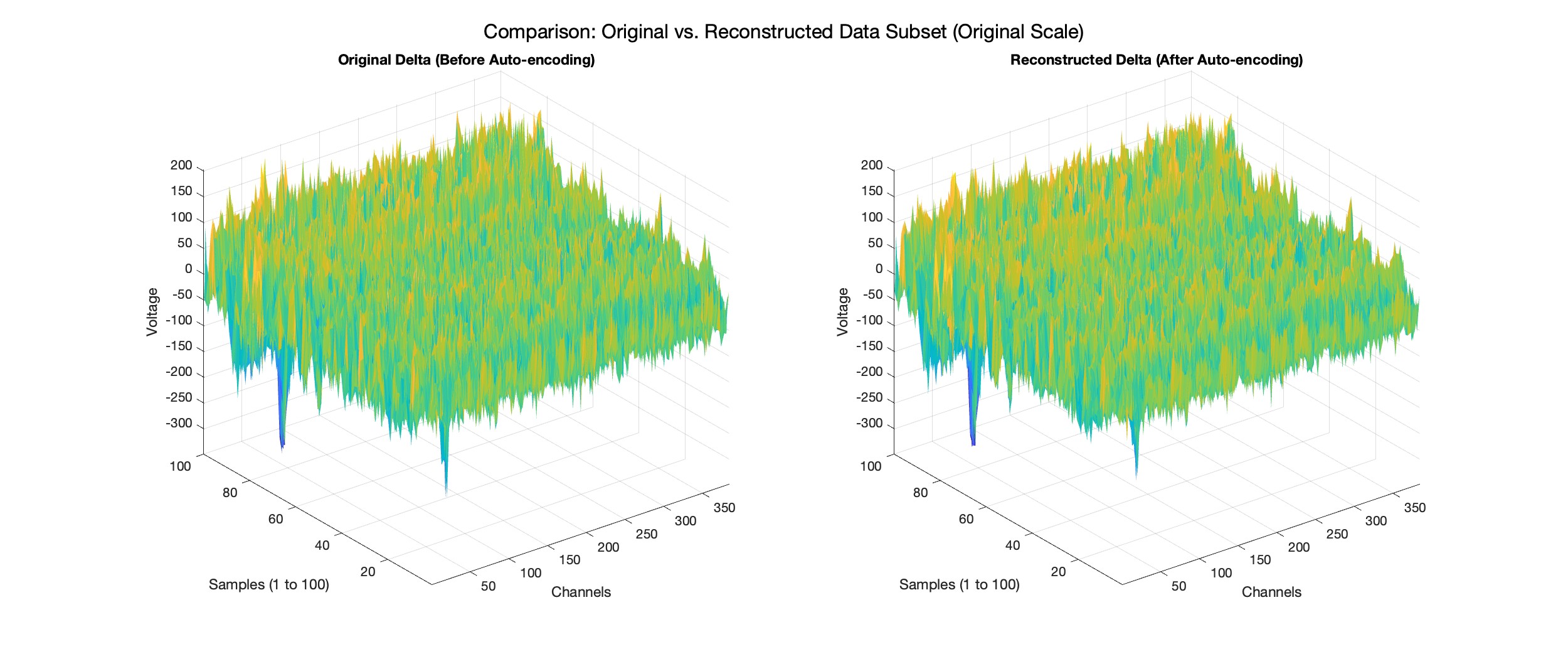
Encoder: 384 → 200 → 200 (latent space)

Decoder: 200 → 200 → 384

ReLU activations throughout

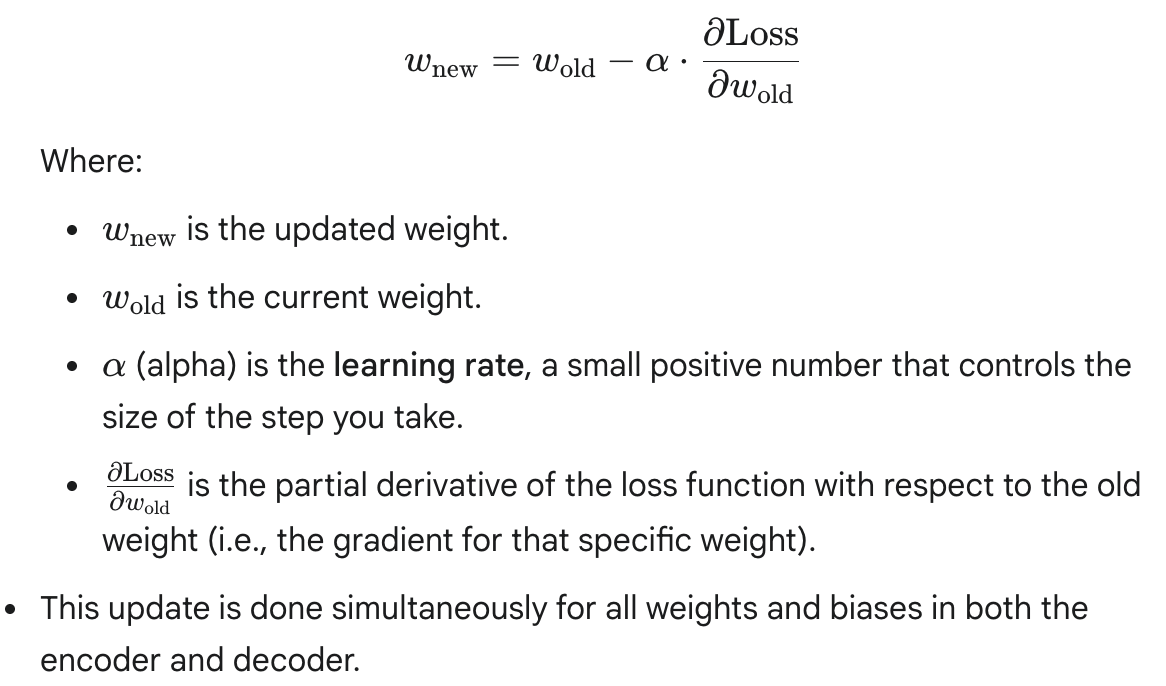
Adam optimizer

Auto-encoder preliminary result (~2x compression ratio):



## Optimization method:

Gradient Descent

(The gradient is a vector of partial derivatives of the loss function with respect to *each* parameter (all the weights and biases in both the encoder and the decoder)where the loss increases most rapidly)

Epoch

1. Take a chunk (or a "mini-batch" of chunks) of your delta-encoded Neuropixels data.
2. Perform the forward pass to get the reconstructed output.
3. Calculate the loss (e.g., MSE).
4. Perform backpropagation to compute the gradients.
5. Update all weights and biases using the gradient descent rule.
6. Repeat this process for many mini-batches.

## Parameters to optimize:

Encoder neural network’s weight matrix

(Weights determines the strength of the connections between neurons in successive layers, biases can shift the output of neurons)

## Evaluation matrix:

Minimize loss from of data

Total Loss = mean squared error loss

Constraint:

Can not use optimization toolbox, but is deep learning toolbox allowed? ( can not download for matlab)

Problem

only training on the first chunk (384 samples) of a larger dataset

Using Z-score normalization on that specific chunk

No validation set or cross-validation

No regularization techniques

The key issue: when you try to use the model on different chunks, the performance drops significantly because the model has learned features specific to the first chunk rather than general patterns in your neural data.

Solution

**1. Use Cross-Validation with Multiple Chunks**

Instead of training on just one chunk, implement k-fold cross-validation across multiple chunks of your neural data

**2. Add Regularization to Prevent Overfitting**

Add regularization techniques to your network to prevent overfitting

**3. Use Data Augmentation for Neural Data**

Data augmentation can make your model more robust by introducing variations to your training data