This document is to serve as a starting guide for AutoEncoders, their development, usage, and mathematical foundations. For any questions, contact misaelmorales@utexas.edu.

Overview of AutoEncoders

An AutoEncoder, in general, is any architecture that seeks to compress data (encoding) and then reconstruct the encoded data back to the original domain (decoding). The encoded data is often referred to as the latent representation, z. The encoder and decoder portions of the architecture are mirror images of each other, and the goal is to obtain lossless compression, or more practically to minimize the error in the lossy compression. Figure 1 shows the simplified schematic of an AE architecture.

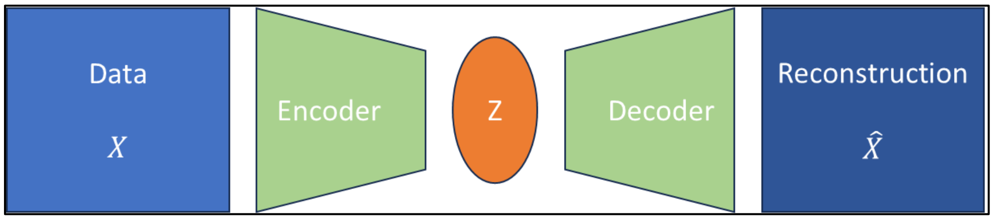


Figure 1: Simplified AutoEncoder architecture

Dimensionality Reduction

To understand AEs, let's dive first into dimensionality reduction. In general, dimensionality reduction is the process of taking a multivariate or multidimensional dataset, X, and compressing it into a latent representation, z. We do these in an attempt to remove redundancies, noise, dependent features, and to extract only the meaningful patterns hidden in the data.

Typical examples of dimensionality reduction techniques include principal component analysis (PCA), singular value decomposition (SVD), discrete/fast Fourier transform (FFT), discrete wavelet transform (DWT), dictionary learning, proper orthogonal decomposition (POD), dynamic mode decomposition (DMD), and deep learning-based AutoEncoders. All of these techniques aim to represent the data in a lower-dimensional state using a certain compression technique.

There also exist other dimensionality reduction techniques that aim to project the data into a lower-dimensional representation, but do not have a back-transformation to reconstruct the data in the original domain. Most of these techniques fall under the umbrella term of Manifold Learning, and are based on the Manifold Hypothesis, which states that all high-dimensional data can exist on a lower-dimensional manifold. Examples include isomaps, locally linear embedding (LLE), spectral embedding, multi-dimensional scaling (MDS), and t-distributed stochastic neighbor embedding (tSNE).

For instance, FFT transform our data from space-time domain to the frequency-wavenumber domain. Then, we can truncate the frequencies that have low amplitudes, and utilize only the dominant frequencies to reconstruct our data. This truncation is the dimensionality reduction step. Dictionary learning is based on the idea from natural language processing that every single sentence, song, book, etc., is composed of a finite number of words. For example, using the Merriam-Webster dictionary, we can reconstruct any sentence, song, book, in the history of humanity. Therefore, dictionary learning aims to construct a dictionary of words (sometimes referred to as atoms) that can then be sparsely sampled to reconstruct our data. For a dimensionality reduction architecture, we construct an under-complete dictionary, meaning that the constructed dictionary is lower in dimensionality than the original data.

Subsurface Example:

We have 300 2D realizations of permeability for a 60×60 reservoir, as shown in Figure 2.

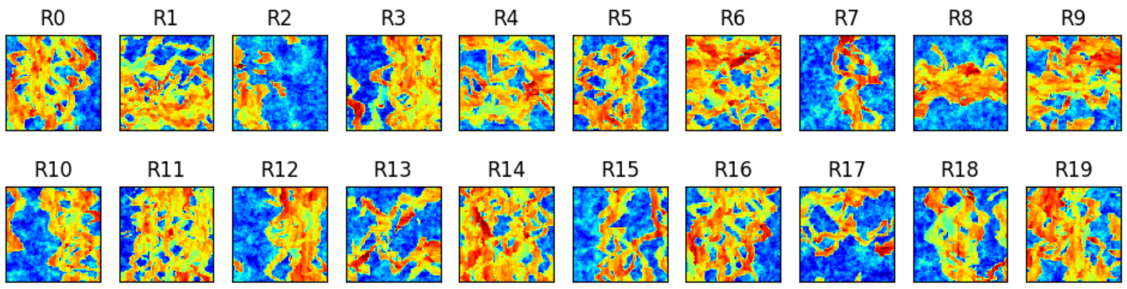
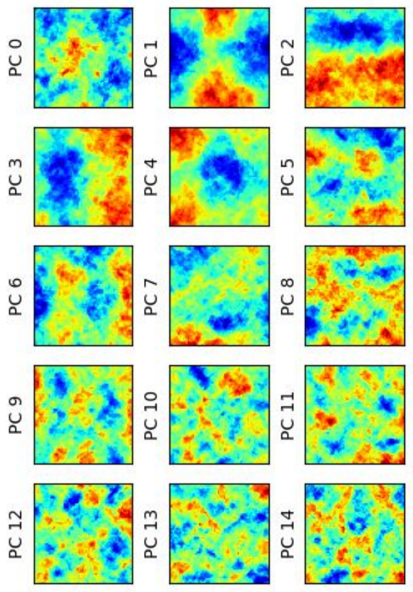


Figure 2: 60x60 permeability realizations.

Let’s say we want to do optimization or history matching. These are expensive routines that require a large number of forward simulations runs for each realization. What if we can compress the realizations into some latent space, z, and then do the optimization or history matching with the latent variables instead of the full-domain data? This would significantly accelerate the optimization or history matching routines. However, we have to ensure that the latent space, z, is truly representative of the data; that it contains all the important information and that it is informative enough to accurately reconstruct our data. Therefore, we build an AE that will compress the data, X, into a latent space, z, and the reconstruct efficiently into $\hat{X}$, so that our z is truly representative and useful.



Recall we have 300 realizations of 60×60. Therefore, the data dimensionality is $(300,60,60)$. If vectorized, this becomes $(300,3600)$. We use PCA to compute the 60 leading eigenvectors-eigenvalues and obtain something like Figure 3. We observe that the leading PCs capture the large-scale features of the data, while the trailing PCs capture the more small-scale details and even the noise in the data. Therefore, we can just use the leading PCs to reconstruct. This is the decoding. Figure 4 shows the reconstructed permeability realizations using only the 60 leading PCs. The results have an MSE of 0.009 and SSIM of 92.1%, pretty good!

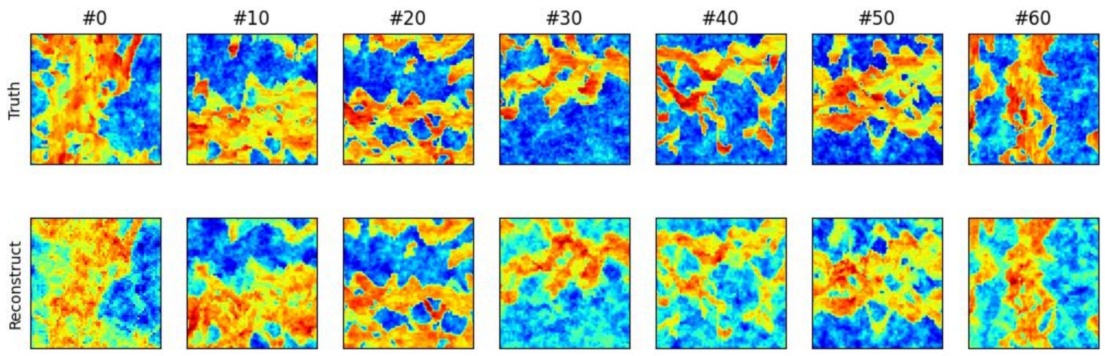


Figure 3: True and Reconstructed permeability realizations using PCA.

AutoEncoders

One-way dimensionality reduction will sometimes minimize the variance, entropy, Euclidean distance or some other calculated property of the data and latent representation. However, for two-way dimensionality reduction, or AutoEncoders, we want to have a decoder that is the mirror image of the encoder so that we can compress the data into a latent representation and then reconstruct back to the original data domain. The goal is that the reconstruction is as similar as possible to the original data. Therefore, to formulate it simply as a minimization problem, we have:

$$\mathcal{L} = \min\left\|X - \hat{X}\right\|$$

where X is the original data, and $\hat{X}$ is the reconstructed. We can minimize in terms of the ℓ_1-norm, ℓ_2-norm, or any other loss function $(\mathcal{L})$ that we choose. For example, in computer vision, we often choose the Structural Similarity Index Measure (SSIM) in conjunction with the Mean Squared Error (MSE) or ℓ_2-norm to minimize the perceptual (whole image) loss and the pixel-wise loss.

Taking this one step further into AutoEncoders, we can formulate this ideally-lossless compression problem as follows:

$$\mathcal{L} = \min\left\|X - Dec\big(Enc(X)\big)\right\|$$

equivalently:

$$\mathcal{L} = \min\|X - Dec(z)\|$$

This means that we are trying to minimize the difference between the original data and the decoded data. Where the decoder is a mirror of the encoder; the encoder compresses the data, X, into the latent representation, z, and the decoder reconstruct $\hat{X}$ from z.

Encoder-Decoder architectures

Recall that an AutoEncoder is when we try to reconstruct the original data by compressing it and then using a mirror image of the encoder as the decoder. This means that we want $\hat{X}$ as similar as possible to X, and our technique is to compress X into z and make it such that z has sufficient information about X to reconstruct it. Namely

$$X \approx \hat{X} \Longrightarrow \hat{X} = Dec\big(Enc(X)\big) = Dec(z)$$

We can also exploit the concept of dimensionality reduction for prediction or inversion problems, where the inputs and outputs are not the same anymore. Namely, we try to predict y from X by compressing X into a latent space, z, and then predicting y from z. These are known as Encoder-Decoder architectures, where the Encoder and Decoder portions are not mirror images of each other, and the compression and reconstruction steps have different requirements. The idea is no longer that z contains sufficient information about X to reconstruct it, but rather that z contains sufficient information about X to predict y. Mathematically, we now have:

$$\mathcal{L} = \min\|y - \hat{y}\|$$

$$\hat{y} = Dec\big(Enc(X)\big) = Dec(z)$$

Deep Learning-based AutoEncoders

Deep learning is a powerful tool to build AutoEncoders. Based on the Universal Approximation Theorem, it is believed that there must always exist some deep learning architecture that is capable of predicting anything from anything. Therefore, we can build AutoEncoders with neural networks!

In general, a neural network is simply a nested function, or a composition. In each layer, we repeat a set of operations and flow data from one hidden layer to the next. For a fully-connected neural network, with linear activation functions, we can write it mathematically as follows:

$$f(x) = g(a(\dots(a(w_2 a(w_1 x + b_1) + b_2) \dots + b_k))$$

$$f(x) = g \circ f_k \circ \dots \circ f_2 \circ f_1(x)$$

where

$$f_i(x) = a(w_i x + b_i)$$

Our datum, x_i, is a linear function of the weights, w_i, and biases, b_i, in each layer. We then "activate" this product using some nonlinear (or sometimes linear) activation function. Typical examples of these are sigmoid, hyperbolic tangent, and rectified linear units (ReLU).

Posed as an optimization problem, we want to find the optimal weights and biases in each layer so that the reconstruction, $\hat{X}$, matches the original data, X. Namely:

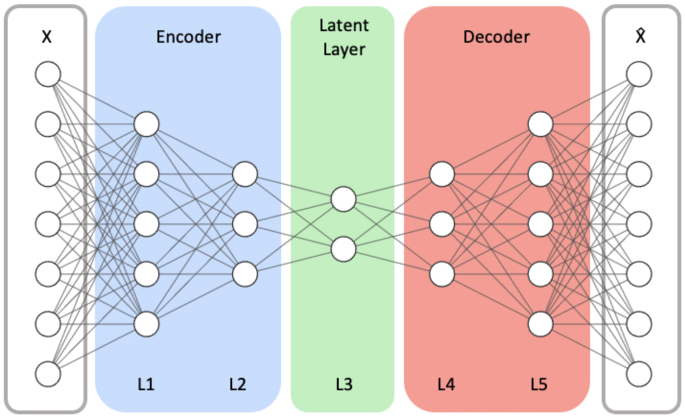
$$\mathcal{L} = \min_{w,b} \|X - \hat{X}\|$$

$$\mathcal{L} = \min_{w,b} \|X - Dec(Enc(X))\| \quad = \quad \min_{w,b} \|X - Dec(z)\|$$

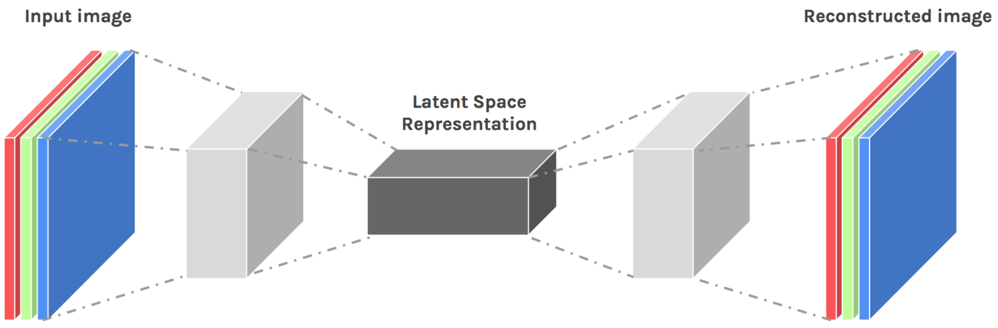
Recall, that the encoder and decoders are neural network architectures where each layer has their weights and biases. Therefore, we have:

$$\mathcal{L} = \min_{w,b} \left\|X - \left(D_i \circ E_j(x)\right)\right\|$$

This means we optimize the weights and biases in each layer of the Encoder and Decoder so that the output matches the input (original data). Figure 5 shows a simple schematic of a Fully-Connected (also known as Artificial) Neural Network (NN) AutoEncoder.



AutoEncoders come in all shapes and forms. We just explained how a Fully-Connected AutoEncoder looks like mathematically and visually. However, we can construct our deep learning-based AE with any deep learning architecture, not just Fully-Connected NN. For computer vision problems, we most likely will use Convolutional AutoEncoders, and for time-series or natural language processing we will use Recurrent AutoEncoders. The difference is that the convolution or recurrent operators are no longer the linear function $f_i(x) = a(w_i x + b_i)$ in each layer, but rather a different operation, f, on the data x. However, we still aim to optimize the weights and biases associated with the operations to minimize our loss, $\mathcal{L}$. Figure 6 shows a schematic of a Convolutional AE architecture.



Many other variations, improvements, and tuning can be done to deep learning-based AEs. Examples include:

- Variational AutoEncoders – we make z a distribution instead of a discrete vector and sample from the distribution to attempt to reconstruct $\hat{X} \approx X$.
- U-Net AutoEncoders – we connect corresponding layers between the Encoder and Decoder. This aims to help with multi-resolution reconstruction and helps flow data between different compressions in the hidden layers.

These, and many other examples, are expert-designed AE architectures that build upon traditional deep learning-based AEs with special tricks or designs that improve the accuracy for specific tasks.

Conclusions

AutoEncoders are simply an architecture or concept. An AutoEncoder is any function that compresses data, X, into a latent space, z, and then reconstructs back into $\hat{X}$, with the goal of minimizing the difference between X and $\hat{X}$. Not to be confused with Encoder-Decoder structures, that compress X into z and use z to predict y. AutoEncoders are mostly used for dimensionality reduction to (1) accelerate workflows by only using z instead of X in subsequent calculations, (2) reduce the number of features in the data if the data is extremely large or correlated, (3) denoise data by removing trailing components that do not hold salient information, and (4) to make inferential analysis (e.g., latent clustering) on the latent representation of the data.