# Computational Statistics Literature Review

Liam Dillingham

May 4, 2019

## 1 Introduction: *Autoencoders, Unsupervised Learning, and Deep Architectures*

This paper discusses a few different cases (namely linear and nonlinear) of a learning device called an *autoencoder*. An *autoencoder* is a type of *artificial neural network*. Typically an ANN is a supervised learner, i.e. given a set of inputs, we try to predict an output(s) label. An autoencoder instead connects its outputs back to its inputs, and the goal of the network is to compress and reconstruct its inputs by learning the most significant signals in the data, and ignore the noise. Once instance of use of autoencoders is dimensionality reduction, where the data is compressed into the desired number of dimensions, and then upon reconstruction, the quality of the compression can be measured by the difference in original inputs and predicted inputs. Because of the construction of the autoencoder, and it's goal, it is an unsupervised learning method.

The paper goes on to describe the difficulty in deriving a theoretical understanding of autoencoders, trying to solve their different cases analytically, and the results. We will review the paper in three parts: The general case of the autoencoder, as described by the paper, a summary of the linear case of the autoencoder, and a summary of (what the paper calls the "most non-linear case") the Boolean autoencoder.

## 2 General Autoencoder Framework

This section defines the components of an autoencoder, and how the device operates to achieve its intended goal (i.e. optimization). It describes an autoencder as an $n/p/n$ *autoencoder*, where $n$ is the size of the input/output vector, and $p$ is the number of dimensions in which we are trying to compress the data into. The componenets of an autoencoder can be contained within a tuple $(n, p, m, \mathbb{F}, \mathbb{G}, \mathcal{A}, \mathcal{B}, \mathcal{X}, \Delta)$, where:

1. $\mathbb{F}, \mathbb{G}$ are sets
2. $n, p$ are the positive integers which we defined earlier, with the case $0 < p < n$
3. $\mathcal{A}$ is the class of function which map $\mathbb{G}^p \to \mathbb{F}^n$, i.e. which *decode* the data
4. $\mathcal{B}$ is the class of function which map $\mathbb{F}^n \to \mathbb{G}^p$, i.e. which *encode* the data
5. $\mathcal{X}$ is a set of size $m$ of training vectors $x_t$ such that $x_t \in \mathbb{F}^n$
6. And $\Delta$ is our dissimilarity or distortion function defined over $\mathbb{F}^n$.

The goal of the autoencoder is to take any input vector $x_t \in \mathbb{F}^n$ and transform it into an output vector. Given $A \in \mathbb{A}$ and $B \in \mathbb{B}$, we have $A(B(x_t)) \in \mathbb{F}^n$. The *autoencoder problem* is to find $A, B$ which minimize the distortion function $\Delta$:

$$\min E(A,B) = \min_{A,B} \sum_{t=1}^{m} E(x_t) = \min_{A,B} \sum_{t=1}^{m} \Delta(A(B(x_t), x_t)$$

The paper never really defines what the function $E(\cdot,\cdot)$, but based on the rest of the paper, it seems as though it is the error function.

## 3 The Case of the Linear Autoencoder

The paper states that for the linear case, the *autoencoder problem* can actually be solved analytically over $\mathbb{R}$. While the problem is not convex, the landscape of $E$ has no local minima. Each location of $E$ which has gradient zero corresponds to projections onto subspaces. These projections corresponds to the global minimum and the results of Principal component analysis. That is, the $p$ value of the autoencoder would correspond to the first $p$ principal components.

## 4 The Case of the Boolean Autoencoder

The optimization problem for the non-linear (in this case the "most" non-linear) is NP-hard. That is one must specify the regime of interested characterized by our variables $n, m, p$, and which of these are diverging towards infinity. If $p$ does not goto infinity, then the problem can be polynomial, for instance, when the centroids belong to the training set. It is also worth noting that unlike the linear case, this non-linear case may have many local minima. However, we can approximate the solutions (and perhaps verify that we are not in a local minima), by using a clustering algorithm such as K-Means. By computing K-Means, and picking the optimal $k$, we can check how closely our autoencoder matches the approximate solution. If there is a large enough difference, we may be in a local minima.

## 5 Clustering Complexity on the Hypercube

This section describes the clustering complexity for the Boolean Autoencoder, and proves that optimizing a boolean autoencoder is NP-hard.
**Theorem.** Consider the following hypercube clustering problem:
Input: $m$ binary vectors $x_1, ..., x_m$ of length $n$ and integer $k$.
Output: $k$ binary vectors $c_1, ..., c_k$ of length $n$ (the centroids) and a function $f$ from $\{x_1, ..., x_m\} \rightarrow \{c_1, ..., c_k\}$ that minimizes the distortion $E = \sum_{t=1}^{m} \Delta(x_t, f(x_t))$ where $\Delta$ is the Hamming distance. The hypercube clustering problem is NP hard when $k \sim m^{\epsilon} (\epsilon > 0)$.
**Proof.** I'll remove the proof for the summary of this paper.

when $p \geq n$, then the function can be rationalized as simply the identity function unless addition constraints are added. These additional nodes in the hidden layer can also add noise to the data.

## 6 Discussion

The paper, in its full length, goes to show the connection between autoencoders and information and coding theory, with two points:

1. When $n < p$, this case corresponds to the classical noisy channel transmission and coding problem
2. When $p < n$, We get the expected compression which we outlined in the beginning of this review. The compression can be lossy when the number of states/neurons in the hidden layer is less than the number of training examples, and lossless otherwise