Autoencoders, Unsupervised Learning, and Deep Architectures

Pierre Baldi
Department of Computer Science
University of California, Irvine
pfbaldi@uci.edu

Abstract

To better understand deep architectures and unsupervised learning, uncluttered by hardware details, we develop a general autoencoder framework for the comparative study of autoencoders, including Boolean autoencoders. We derive several results regarding autoencoders and autoencoder learning, including results on learning complexity, vertical and horizontal composition, and fundamental connections between critical points and clustering. Possible implications for the theory of deep architectures are discussed.

1 Introduction

Autoencoders are simple learning circuits which aim to transform inputs into outputs with the least possible amount of deformation. While conceptually simple, they play an important role in machine learning. Autoencoders were first introduced in the 1980s by Hinton and the PDP group [18] to address the problem of "backpropagation without a teacher", by using the input data as the teacher. Together with Hebbian learning rules, autoencoders provide one of the fundamental paradigms for unsupervised learning and for addressing the mystery of how synaptic changes induced by local biochemical events can be coordinated in a self-organized manner to produce global learning and intelligent behavior. More recently, autoencoders have taken center stage again in the "deep architecture" approach [11, 12, 3, 4], where autoencoders in the form of Restricted Boltzmann Machines (RBMS) are stacked and trained bottom up in unsupervised fashion, followed by a supervised learning phase to train the top layer and fine-tune the entire architecture. This largely unsupervised approach has been shown to lead to state-of-the-art results on a number of challenging classification and regression problems.

In spite of the interest they have generated, and with a few exceptions [2, 20], little theoretical understanding of autoencoders and deep architectures has been obtained to this date. Additional confusion may have been created by the use of the term "deep". A deep architecture from a computer science perspective should have N^α layers, for some small $\alpha > 0$ (N being the size of the input vectors). But that is not the case in the architectures described in [11, 12], which seem to have constant or at best logarithmic depth, the distinction between finite and logarithmic depth being almost impossible for the typical values of N used in computer vision, speech recognition, and other typical problems. Thus the main motivation behind this work is to derive a better theoretical understanding of autorencoders, with the hope of gaining better insights into the nature of unsupervised learning and deep architectures. If general theoretical results about deep architectures exist, these are unlikely to depend on a particular hardware realization, such as RBMs. Similar results ought to be true for alternative, or more general, forms of computation. Thus the strategy proposed here is to introduce and study other autoencoder circuits, in particular Boolean autoencoders which can be viewed as the most primitive form of non-linear autoencoders. The expectation is that certain properties of autoencoders and deep architectures may be easier to identify and understand mathematically in simpler hardware embodiments, and that the study of different kinds of autoencoders may facilitate abstractions and identifications of common properties.

For this purpose, we begin in Section 2 by providing a fairly general framework for studying autoencoders. In Section 3, we review and extend the results of [2] on linear autoencoders. In the light of deep architectures, we look at novel properties such a vertical composition (stacking) and connection of critical points to stability under recycling (feeding outputs back to the input layer). In Section 4, we study Boolean autoencoders, and prove several properties including their fundamental connection to clustering. In Section 5, we address the complexity of Boolean autoencoder learning. In Section 6, we study autoencoders with large hidden layers, and introduce the notion of horizontal composition of autoencoders. In Section 7, we address other classes of autoencoders and generalizations. Finally, in Section 8, we summarize the results and their possible consequences for the theory of deep architectures.

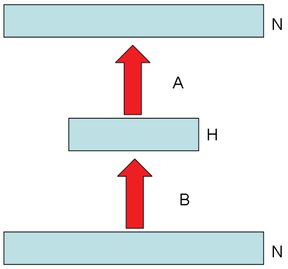

Figure 1: NHN Autoencoder Architecture.

2 A General Autoencoder Framework

A fairly general framework for differen kinds of autoencoders can be derived by considering an architecture with an input layer of size N, a hidden layer of size H, and an output layer of size N (Figure 1), together with a set of input training vectors $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$, a function $\mathbf{B}$ from the input layer to the hidden layer, and a function $\mathbf{A}$ from the hidden layer to the output layer. When presented with an input vector $\mathbf{x}$, the circuit produces a hidden vector $\mathbf{h} = \mathbf{B}(\mathbf{x})$ and an output vector $\mathbf{y} = \mathbf{AB}(\mathbf{x})$. The goal of learning is to minimize an error or energy function E in the form

$$\min E(\mathbf{A}, \mathbf{B}) = \min_{\mathbf{A},\mathbf{B}} \sum_{i=1}^{M} \Delta(\mathbf{y}_i, \mathbf{x}_i) = \min_{\mathbf{A},\mathbf{B}} \sum_{i=1}^{M} \Delta\big(\mathbf{AB}(\mathbf{x}_i), \mathbf{x}_i\big) \tag{1}$$

where Δ is a distance or dissimilarity measure. Examples of useful measures are the squared Euclidean distance $\Delta = L_2^2$ and the Hamming distance $\Delta = H$. Other common measures are the L_p measures. We assume that the transformations $\mathbf{A}$ and $\mathbf{B}$ belong respectively to two classes $\mathcal{A}$ and $\mathcal{B}$ of transformations. The components of the $\mathbf{x}$ vectors are in a set $\mathbb{F}$, and the components of the $\mathbf{h}$ vectors are in a set $\mathbb{G}$. In most cases, $\mathbb{F} = \mathbb{G}$ and in most cases $\mathbb{F}$ and $\mathbb{G}$ are fields. When $\mathbb{F}$ and $\mathbb{G}$ are fields, then one can consider linear autoencoders where the transformations $\mathbf{A}$ and $\mathbf{B}$ are defined by their matrices. For brevity, we refer to these architectures as NHN architectures. Initially we also assume that $H < N$, corresponding to the regime where the autoencoder tries to implement some form of compression. But the case of $N \leq H$ is also of interest and will be considered later in the paper together with more complex architectures. Obviously, from this general framework, different kinds of autoencoders can be derived depending, for instance, on the choice of sets $\mathbb{F}$ and $\mathbb{G}$, transformation classes $\mathcal{A}$ and $\mathcal{B}$, error function E, as well as the presence of additional constraints, such as regularization. To the best of our knowledge, neural network autoencoders were first introduced by the PDP group as a special case of this definition, with all vectors components in $\mathbb{F} = \mathbb{G} = \mathbb{R}$ and $\mathbf{A}$ and $\mathbf{B}$ corresponding to matrix multiplications followed by non-linear sigmoidal transformations with an L_2^2 error function. As an approximation to this case, in the next section, we study the linear case with $\mathbb{F} = \mathbb{G} = \mathbb{R}$.

3 The Linear Autoencoder

We partly restate without proof the results derived in Baldi and Hornik [2], but in a way that will highlight the connections to other kinds of autoencoders, and extend their results from a deep architecture perspective. We use $\mathbf{A}^t$ to denote the transpose of any matrix $\mathbf{A}$ and assume that the data $\mathcal{X}$ is centered.

1) Group Invariance. Every solution is defined up to multiplication by an invertible $H \times H$ matrix $\mathbf{C}$, or equivalently up to a change of coordinates in the hidden layer, since $\mathbf{A}\mathbf{C}^{-1}\mathbf{C}\mathbf{B} = \mathbf{A}\mathbf{B}$.
2) Problem Complexity. While the cost function is quadratic and all the operations are linear, the overall problem is not convex because the hidden layer limits the rank of the overall transformation to be at most H, and the set of matrices of rank H or less is *not* convex. However the problem can be solved analytically.
3) Fixed Layer Solution. The problem becomes convex if $\mathbf{A}$ is fixed, or if $\mathbf{B}$ is fixed. When $\mathbf{A}$ is fixed, assuming $\mathbf{A}$ has rank H and that the data covariance matrix $\mathbf{\Sigma}_{XX}$ is invertible, then $\mathbf{B}^* = \mathbf{B}(\mathbf{A}) = (\mathbf{A}^t\mathbf{A})^{-1}\mathbf{A}^t$. When $\mathbf{B}$ is fixed, assuming $\mathbf{B}$ has rank H and that $\mathbf{\Sigma}_{XX}$ is invertible, then $\mathbf{A}^* = \mathbf{A}(\mathbf{B}) = \mathbf{\Sigma}_{XX}\mathbf{B}^t(\mathbf{B}\mathbf{\Sigma}_{XX}\mathbf{B}^t)^{-1}$.
4) The Landscape of E. The overall landscape of E has no local minima. All the critical points where the gradient of E is zero, correspond to projections onto subspaces associated with H eigenvectors of the covariance matrix $\mathbf{\Sigma}_{XX}$. Projections onto the subspace associated with the H largest eigenvalues correspond to the global minimum and Principal Component Analysis. All other critical point, corresponding to projections onto subspaces associated with other set of eigenvalues, are saddle points. More precisely, if $\mathcal{I} = i_1, \ldots, i_p$ $(1 \leq i_i < \ldots < i_H \leq N)$ is any ordered list of indices, let $\mathbf{U}_\mathcal{I} = [\mathbf{u}_1, \ldots, \mathbf{u}_H]$ denote the matrix formed by the orthonormal eigenvectors of $\mathbf{\Sigma}_{XX}$ associated with the eigenvalues $\lambda_{i_1}, \ldots, \lambda_{i_H}$. Then two matrices $\mathbf{A}$ and $\mathbf{B}$ of rank H define a critical point if and only if there is a set $\mathcal{I}$ and an invertible $H \times H$ matrix $\mathbf{C}$ such that $\mathbf{A} = \mathbf{U}_\mathcal{I}\mathbf{C}$, $\mathbf{B} = \mathbf{C}^{-1}\mathbf{U}_\mathcal{I}^t$, and $\mathbf{W} = \mathbf{A}\mathbf{B} = \mathbf{P}_{\mathbf{U}_\mathcal{I}}$, where $\mathbf{P}_{\mathbf{U}_\mathcal{I}}$ is the orthogonal projection onto the subspace spanned by the columns of $\mathbf{U}_\mathcal{I}$. At the global minimum, assuming that $\mathbf{C} = \mathbf{I}$, the activities in the hidden layer are given by the dot products $\mathbf{u}_1^t\mathbf{x} \ldots \mathbf{u}_H^t\mathbf{x}$ and correspond to the coordinates of $\mathbf{x}$ along the first H eigenvectors of $\mathbf{\Sigma}_{XX}$.
5) Clustering. Thus the global minimum performs a form of clustering by hyperplane: for any given vector $\mathbf{x}$, all the vectors of the form $\mathbf{x}+Ker(\mathbf{B})$ are mapped onto the same vector $\mathbf{y} = \mathbf{A}\mathbf{B}(\mathbf{x}) = \mathbf{A}\mathbf{B}(\mathbf{x}+Ker\mathbf{B})$.
6) Recycling Stability. At any critical point, recycling outputs is stable at the first pass: $(\mathbf{A}\mathbf{B})^n)(\mathbf{x}) = \mathbf{A}\mathbf{B}(\mathbf{x}) = \mathbf{U}_\mathcal{I}\mathbf{U}_\mathcal{I}^t(\mathbf{x})$ for any $n \geq 1$.
7) Generalization. At any critical point, for any $\mathbf{x}$, $\mathbf{A}\mathbf{B}(\mathbf{x})$ is equal to the projection of $\mathbf{x}$ onto the corresponding subspace and the corresponding error can be expressed easily.
8) Vertical Composition. The global minimum of E remains the same if additional matrices of rank greater or equal to H are introduced between the input layer and the hidden layer and/or the hidden layer and the output layer. Thus there is no reduction in overall distortion by introducing such matrices. However, if such matrices are introduced for other reasons, there is a composition law so that the optimum solution for a deep autoencoder with a stack of matrices, can be obtained by combining the optimal solutions of shallow autoencoders. More precisely, consider an autoencoder network with layers of size $N, H1, H, H1, N$ (Figure

2) with $N > H1 > H$. Then the optimal solution of this network can be obtained by first computing the optimal solution for an $N, H1, N$ autoencoder network, and combining it with the optimal solution of an $H1, H, H1$ autoencoder network using the activity in the hidden layer of the first network as the training set for the second network, exactly as in the case of stacked RBMs [11, 12]. This is because the projection onto the subspace spanned by the top H eigenvectors can be composed by a projection onto the subspace spanned by the top $H1$ vectors, followed by a projection onto the subspace spanned by the top H vectors.
9) External Targets. With the proper adjustments [2], the results above remain essentially the same if a set of target output vectors $\mathbf{y}_1, \ldots, \mathbf{y}_M$ is provided, instead of $\mathbf{x}_1, \ldots, \mathbf{x}_M$ serving as the targets.
10) Symmetries and Hebbian Rules. At the global minimum, for $\mathbf{C} = \mathbf{I}$, $\mathbf{A} = \mathbf{B}^t$. The constraint $\mathbf{A} = \mathbf{B}^t$ can be imposed during learning by "weight sharing" and is consistent with a Hebbian rule that is symmetric between the pre- and post synaptic neurons and is applied to the network by clamping the output units to be equal to the input units (or having a folded autoencoder).

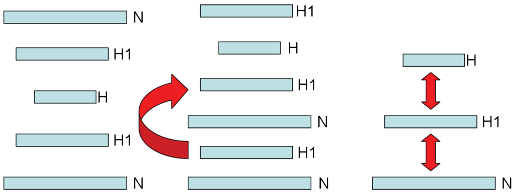

Figure 2: Vertical Composition of Autoencoders.

4 The Boolean Autoencoder

The Boolean autoencoder is perhaps the simplest form of non-linear autoencoder. In the purely Boolean case, we have $\mathbb{F} = \mathbb{G} = \{0, 1\}$, $\mathbf{A}$ and $\mathbf{B}$ are unrestricted Boolean function, and Δ is the Hamming distance. Many variants of this problem can be obtained by restricting the classes $\mathcal{A}$ and $\mathcal{B}$ of Boolean functions, for instance by bounding the connectivity. The linear case with $\mathbb{F} = \mathbb{G} = \{0, 1\} = \mathbb{F}_2$, where $\mathbb{F}_2$ is the Galois field with two elements, is a special case of the Boolean case and will be discussed later. For lack of space, proofs can only be sketched here.

1) Group Invariance. Every solution is defined up to a permutation of the 2^H points of the hypercube $\mathbb{H}^H$. This is because the Boolean function are unrestricted and therefore their lookup tables can accommodate any such permutation, or relabeling of the hidden states.
2) Problem Complexity. In general, the overall optimization problem is NP-hard. To be more precise, one must specify the regime of interest characterized by which variables (out of N, M, and H) are going to infinity. Obviously one must have $N \to \infty$. If H does not go to infinity, then the problem can be polynomial, for instance when the centroids must belong to the training set. If $H \to \infty$ and M is a polynomial in N, which is the case of interest in machine learning where typically M is a low degree polynomial in N, then the problem of finding the best boolean mapping is NP hard (or the corresponding decision problem is NP-complete). A proof of this is given in the next section.
3) Fixed Layer Solution. If the $\mathbf{A}$ mapping is fixed, then it is easy to find the optimal $\mathbf{B}$ mapping. Conversely if the $\mathbf{B}$ mapping is fixed, it is easy to find the optimal $\mathbf{A}$ mapping. Assume that $\mathbf{A}$ is fixed. Then for each of the 2^H possible Boolean vectors $\mathbf{h}_1, \ldots, \mathbf{h}_{2^H}$ of the hidden layer, $\mathbf{A}(\mathbf{h}_1) \ldots, \mathbf{A}(\mathbf{h}_{2^H})$ provide 2^H points in the hypercube $\mathbb{H}^N$. One can build the corresponding Voronoi partition by assigning each point to its closest centroid, breaking ties arbitrarily, thus forming a partition of $\mathbb{H}^N$ into 2^H corresponding clusters $\mathcal{C}_1, \ldots, \mathcal{C}_{2^H}$, with $\mathcal{C}_i = \mathcal{C}^{Vor}(\mathbf{A}(h_i))$. The optimal mapping $\mathbf{B}^*$ is then easily defined by setting $\mathbf{B}^*(\mathbf{x}) = \mathbf{h}_i$ for any $\mathbf{x}$ in $\mathcal{C}_i = \mathcal{C}^{Vor}(\mathbf{A}(\mathbf{h}_i))$. Conversely, assume that $\mathbf{B}$ is fixed. Then for each of the 2^H possible Boolean vectors $\mathbf{h}_1, \ldots, \mathbf{h}_{2^H}$ of the hidden layer, let $\mathcal{C}^{\mathbf{B}}(\mathbf{h}_i) = \{\mathbf{x} \in \mathbb{H}^N : \mathbf{B}(\mathbf{x}) = \mathbf{h}_i\}$. To minimize the reconstruction error, $\mathbf{A}^*$ must map $\mathbf{h}_i$ onto a point $\mathbf{y}$ of $\mathbb{H}^N$ minimizing the sum of Hamming

distances to points in $\mathcal{X} \cap \mathcal{C}^{\mathbf{B}}(\mathbf{h}_i)$. It is easy to see that the minimum is realized by the component-wise majority vector $\mathbf{A}^*(\mathbf{h}_i) = Majority[\mathcal{X} \cap \mathcal{C}^{\mathbf{B}}(\mathbf{h}_i)]$ (breaking ties arbitrarily).
4) The Landscape of E. In general E has many local minima (e.g with respect to the Hamming distance applied to the lookup tables of $\mathbf{A}$ and $\mathbf{B}$). Critical points are defined to be the points satisfying simultaneously the equations above for $\mathbf{A}^*$ and $\mathbf{B}^*$.
5) Clustering. The overall optimization problem is a problem of optimal clustering. The clustering is defined by the transformation $\mathbf{B}$. Approximate solutions can be sought by many algorithms, such as k-means, belief propagation [6], minimum spanning paths and trees [19], and hierarchical clustering.
6) Recycling Stability. At any critical point, recycling outputs is stable at the first pass: for any $\mathbf{x}$ $(\mathbf{AB})^n)(\mathbf{x}) = \mathbf{AB}(\mathbf{x})$ and equal to the majority vector of the corresponding Voronoi cluster.
7) Generalization. At any critical point, for any $\mathbf{x}$, $\mathbf{AB}(\mathbf{x})$ is equal to the centroid of the corresponding Voronoi cluster and the corresponding error can be expressed easily.
8) Vertical Composition. The global optimum remains the same if additional Boolean layers of size equal or greater to H are introduced between the input layer and the hidden layer and/or the hidden layer and the output layer. Thus there is no reduction in overall distortion E by adding such layers. Cconsider a Boolean autoencoder network with layers of size $N, H1, H, H1, N$ (Figure 2) with $N > H1 > H$. Then the optimal solution of this network can be obtained by first computing the optimal solution for an $N, H1, N$ autoencoder network, and combining it with the optimal solution of an $H1, H, H1$ autoencoder network using the activity in the hidden layer of the first network as the training set, exactly as in the case of stacked RBMs. The reason for this is that the global optimum correspond to clustering into 2^H clusters, and this can be obtained by first clustering into 2^{H_1} clusters, and then clustering these clusters into 2^H clusters. The stack of Boolean functions performs hierarchical clustering with respect to the input space.
9) External Targets. With the proper adjustments, the results above remain essentially the same if a set of target output vectors $\mathbf{y}_1, \ldots, \mathbf{y}_M$ is provided, instead of $\mathbf{x}_1, \ldots, \mathbf{x}_M$ serving as the targets. To see this, consider a deep architecture consisting of a stack of autoencoders along the lines of [11]. For any activity vector $\mathbf{h}$ in the last hidden layer before the output layer, compute the set of points $\mathcal{C}(\mathbf{h})$ in the training set that are mapped to $\mathbf{h}$ by the stacked architecture. Assume, without any loss of generality, that $\mathcal{C}(\mathbf{h}) = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ with corresponding targets $\{\mathbf{y}_1, \ldots, \mathbf{y}_k\}$. Then it is easy to see that the final output for $\mathbf{h}$ produced by the top layer ought to be the centroid of the targets given by $Majority(\mathbf{y}_1, \ldots, \mathbf{y}_k)$

5 Clustering Complexity on the Hypercube

The complexity of various clustering problems, in different spaces, or with different objective functions, has been studied in the literature. There are primarily two kind of results: (1) graphical results derived on graphs $G = (V, E, \Delta)$ where the dissimilarity Δ is not necessarily a distance; and (2) geometric results derived in the Euclidean space $\mathbb{R}^d$ where $\Delta = L_2^2$, L_2, or L_1. In general, the clustering decision problem is NP-complete and the clustering optimization problem is NP-hard, except in some simple cases involving either a constant number K of clusters or clustering in the 1-dimensional Euclidean space. In general, the results in Euclidean spaces are harder to derive than the results on graphs. When polynomial time algorithms exist, geometric problems tend to have faster solutions taking advantage of the geometric properties. None of the existing complexity theorems directly addresses the problem of clustering on the hypercube with the Hamming distance.

To deal with the hypercube clustering problem one must first understand which quantities are allowed to go to infinity. If N is not allowed to go to infinity, then the number M of training examples is also bounded by 2^N and, since we are assuming $H < N$, there is no quantity that can scale. Thus by necessity we must have $N \to \infty$. We must also have $M \to \infty$. The case of interest for machine learning is when M is a low degree polynomial of N. Obviously the hypercube clustering problem is in NP, and it is a special case of clustering in $\mathbb{R}^N$. Thus the only important problem to be addressed is the reduction of a known NP-complete problem to a hypercube clustering problem.

For the reduction, it is natural to start from a known NP-complete graphical or geometric clustering problem. In both case, one must find ways to embed the original problem with its original metric into the hypercube with the Hamming distance. There are theorems for homeomorphic or squashed-embedding of graphs into the hypercube [9, 22], however these emebeddings do not map the original dissimilarity function onto the the Hamming metric. Thus here we prefer to start from some of the known geometric results and use a strict cubical graph embedding. A graph is cubical if it is the subgraph of some hypercube $\mathbb{H}^d$ for some d [8, 13]. Although deciding whether a graph is cubical is NP-complete [1], there is a theorem [10] providing a necessary and sufficient condition for a graph to be cubical. A graph$G(V, E)$ is cubical and embeddable in $\mathbb{H}^d$ if and only if it is possible to color the edges of G with d colors such that: (1) All edges incident with a common vertex are of different color; (2) In each path of G, there is some color that appears an odd number of times; and (3) In each cycle of G, no color appears an odd number of times.

To sketch the final reduction, we start from the problem of clustering M points in the plane $\mathbb{R}^2$ using cluster centroids and the L_1 distance, which is NP-complete [17] by reduction from 3-SAT [7] when $K \sim M^\epsilon$ ($\epsilon > 0$) (see, also related results in [14, 21]). Without any loss of generality, we can assume that the points in these problems are on the vertices of a square lattice. Using the theorem in [10], one can show that a $n \times m$ square lattice in the plane can be embedded into $\mathbb{H}^{n+m}$. In fact, an explicit embedding is given in Figure 3. It is easy to check that the L_1 or Manhattan distance between any two points on the square lattice is equal to the corresponding Hamming distance in $\mathbb{H}^{n+m}$. This polynomial reduction completes the proof that if the number of cluster is $M^\epsilon = 2^H$ (equivalent to $H = \epsilon \log_2 M \approx C \log N$) then the hypercube clustering problem associated with the Boolean autoencoder is NP-hard. If the numbers K of clusters is fixed and the centroids must belong to the training set, there are only $\binom{M}{K} \sim M^K$ possible choices for the centroids inducing the corresponding Voronoi clusters. This yields a trivial, albeit not efficient, polynomial time algorithm. When the centroids are not required to be in the training set, the same result should hold by adapting the corresponding theorems in Euclidean space.

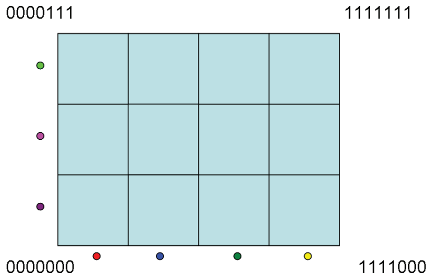

Figure 3: Embedding of a 3×4 Square Lattice onto $\mathbb{H}^7$ by Edge Coloring. All edges in the same row or column have the same color. Each color correspond to one hypercube dimension.

6 The Case $H \geq N$

When the hidden layer is larger than the input layer and $\mathbb{F} = \mathbb{G}$, there is an optimal 0-distortion solution involving the identity function. Thus this case is interesting only if additional constraints are added to the problem. These can come in the form of regularization, for instance to ensure sparsity of the hidden-layer representation, or restrictions on the classes of functions $\mathcal{A}$ and $\mathcal{B}$, or noise in the hidden layer (see next section). When these constraints force the hidden layer to assume only K different values and $K < M$, for instance in the case of a sparse Boolean hidden layer, then the previous analyses hold and the problem reduces to a K clustering problem.

In this context, in addition to vertical composition, there is also a natural horizontal composition for autoencoders that can be used to create large hidden layer representations (Figure 4) simply by horizontally combining autoencoders. Two (or more) autoencoders with architecture $NH1N$ and $NH2N$ can be trained and the hidden layers can be combined to yield an expanded hidden representation of size $H1 + H2$ that can then be fed to the subsequent layers of the overall architecture. Differences in the $H1$ and $H2$ hidden representations could be introduced by many different mechanisms, for instance using different learning algorithms, different initializations, different training samples, different learning rates, etc. It is also possible to envision algorithms that incrementally add hidden units to the hidden layer. In the linear case over $\mathbb{R}$, for instance, a first hidden unit can be trained to extract the first principal component, a second hidden unit can then be added to extract the second principal component, and so forth.

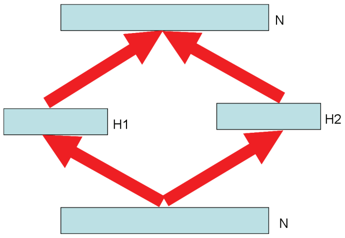

Figure 4: Horizontal Composition of Autoencoders to Expand the Hidden Layer Representation.

7 Other Generalizations

Within the general framework introduced here, other kinds of autoencoders can be considered. First, one can consider mixed autoencoders with different constraints on $\mathbb{F}$ and $\mathbb{G}$, or different constraints on $\mathcal{A}$ and $\mathcal{B}$. A simple example is when the input and output layers are real $\mathbb{F} = \mathbb{R}$ and the hidden layer is binary $\mathbb{G} = \{0, 1\}$ (and $\Delta = L_2^2$). It is easy to check that in this case, as long as $2^H = K < M$, the autoencoder aims at clustering the real data into K clusters and all the results obtained in the Boolean case are applicable with the proper adjustments. For instance, the centroid associated with a hidden state $\mathbf{h}$ should be the center of mass of the input vectors mapped onto $\mathbf{h}$. In general, this mixed autoencoder is also NP hard and, from a probabilistic view point, it corresponds to a mixture of K Gaussians model.

A second natural direction is to consider autoencoders that are linear but over fields other than the real numbers, for instance over the field $\mathbb{C}$ of complex numbers, or over finite fields. For all these linear autoencoders, the Kernel of $\mathbf{B}$ plays an important role since inputs vectors are basically clustered modulo this kernel. These autoencoders are not without theoretical and practical interests. Consider the linear autoencoder over the Galois field with two elements $GF(2) = \mathbb{F}_2$. It is easy to see that this is a special case of the Boolean autoencoder, where the Boolean functions are restricted to parity functions. This autoencoder can also be seen as implementing a linear code [16]. When there is noise in the 'transmission" of the hidden layer and $H > N$, one can consider solutions where N units in the hidden layer correspond to the identity function and the remaining $H - N$ units implement additional parity check bits that are linearly computed from the input and used for error correction. Thus all well known linear codes, such as Hamming or Reed-Solomon codes, can be viewed within this linear autoencoder framework. While the linear autoencoder over $\mathbb{F}_2$ will be discussed elsewhere, it is worth noting that it also yields an NP-hard problem. This can be seen by considering that finding the minimum (non-zero) weight vector in the kernel of a binary matrix, or the radius of a code, are NP-complete problems [15, 5]. A simple classification of autoencoders is given in Figure 5.

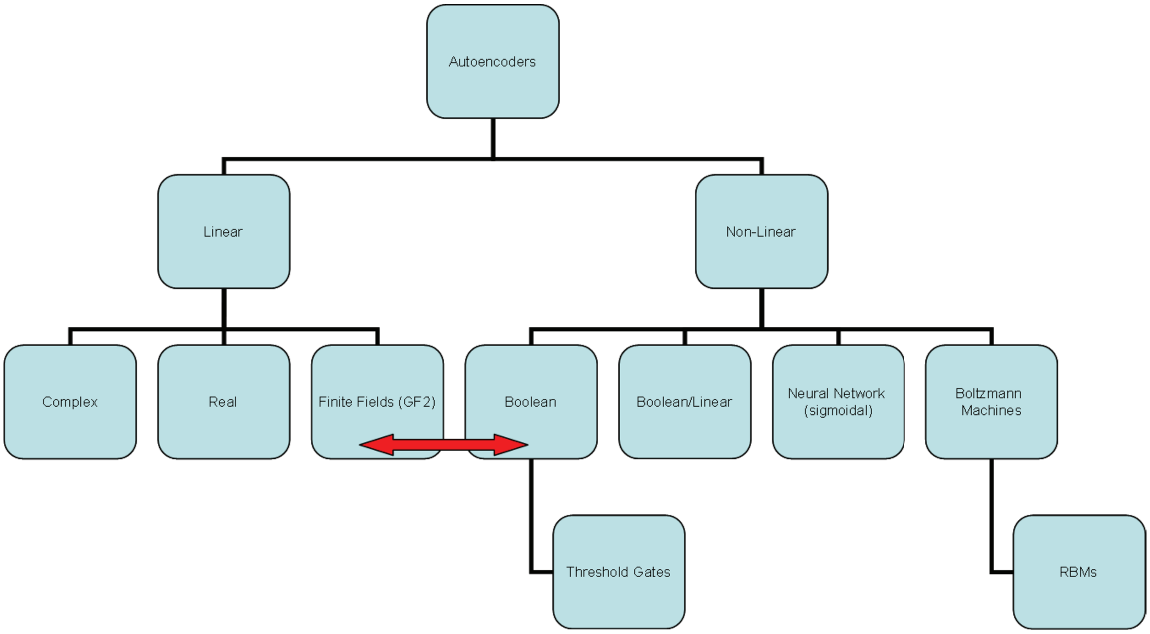

Figure 5: Simple Autoencoder Classification.

8 Discussion

Studying the linear and Boolean autoencoder in detail enables one to gain a general perspective on autoencoders, define key properties that are shared by different autoencoders and that ought to be checked systematically in any new kind of autoencoder (e.g. group invariances, clustering, recycling stability). The general emerging picture is that autoencoders learning is in general NP-complete [1] except in simple but important cases (eg. linear over $\mathbb{R}$, Boolean with fixed K) and that in essence all autoencoders are performing some form of clustering ($N < H$). While autoencoders and Hebbian rules provide unsupervised learning implementations, it is clustering that provides the basic conceptual operation that underlies them.

RBMs and their efficient contrastive learning algorithm may provide an elegant and efficient form of autoencoder and autoencoder learning, but it is doubtful that there is anything special about RBMs at a deeper conceptual level. Thus it ought to be possible to derive results comparable to those described in [11, 12] by stacking other kinds of autoencoders, and more generally by hierarchically stacking a series of clustering algorithms using vertical composition, perhaps also in combination with horizontal composition. Simulations along these lines are in progress. As pointed out in the previous sections, it is easy to add a top layer for supervised regression or classification tasks on top of the hierarchical clustering stack. In aggregate, these results suggest that: (1) the so-called deep architectures may in fact have a non-trivial but constant (or logarithmic) depth, which is also consistent with what is observed in sensory neuronal circuits; (2) the fundamental unsupervised operation behind deep architectures, in one form or the other, is clustering, which is composable both horizontally and vertically; and (3) the generalization properties of deep architectures may be easier to understand when ignoring many of the hardware details, in terms of the most simple forms of autoencoders (e.g Boolean), or in terms of the more fundamental underlying clustering operations.

[1] RBM learning is NP-complete by similarity with minimizing a quadratic form over the hypercube.

References

[1] F. Afrati, C. Papadimitriou, and G. Papageorgiou. The complexity of cubical graphs. *Automata, Languages and Programming*, pages 51–57.

[2] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1988.

[3] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*. MIT Press, 2007.

[4] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, February 2010.

[5] M. Frances and A. Litman. On covering problems of codes. *Theory of Computing Systems*, 30(2):113–119, 1997.

[6] B.J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972, 2007.

[7] M.R. Garey and D.S. Johnson. *Computers and Intractability*. Freeman San Francisco, 1979.

[8] F. Harary. Cubical graphs and cubical dimensions. *Computers & Mathematics with Applications*, 15(4):271–275, 1988.

[9] J. Hartman. The homeomorphic embedding of Kn in the m-cube* 1. *Discrete Mathematics*, 16(2):157–160, 1976.

[10] I. Havel and J. Morávek. B-valuations of graphs. *Czechoslovak Mathematical Journal*, 22(2):338–351, 1972.

[11] G.E. Hinton, S. Osindero, and Y.W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[12] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504, 2006.

[13] M. Livingston and Q.F. Stout. Embeddings in hypercubes. *Mathematical and Computer Modelling*, 11:222–227, 1988.

[14] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is NP-hard. *WALCOM: Algorithms and Computation*, pages 274–285.

[15] R. McEliece and H. van Tilborg. On the inherent intractability of certain coding problems(Corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[16] R. J. McEliece. *The Theory of Information and Coding*. Addison-Wesley Publishing Company, Reading, MA, 1977.

[17] N. Megiddo and K.J. Supowit. On the complexity of some common geometric location problems. *SIAM J. COMPUT.*, 13(1):182–196, 1984.

[18] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing. Vol 1: Foundations*. MIT Press, Cambridge, MA, 1986.

[19] JL Slagle, CL Chang, and SR Heller. A clustering and data reorganization algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, 5:121–128, 1975.

[20] I. Sutskever and G.E. Hinton. Deep, narrow sigmoid belief networks are universal approximators. *Neural Computation*, 20(11):2629–2636, 2008.

[21] A. Vattani. A simpler proof of the hardness of k-means clustering in the plane. *UCSD Technical Report*.

[22] P.M. Winkler. Proof of the squashed cube conjecture. *Combinatorica*, 3(1):135–139, 1983.