Chapter 5

Autoencoders

So far, we have acquired a good understanding of $A(n, m)$ and $A(n, \infty, m)$ architectures. Thus we now begin to turn towards the study of $A(n, m, p)$ architectures. We can remove some degrees of freedom from the problem by having $p = n$ and letting the input data also be the target data. This is the case of autoencoder networks.

Thus in this chapter we focus on autoencoder networks. The basic original idea behind autoencoders is to use the input data as the target, i.e. to try to reconstruct the input data in the output layer. As described in [11], the idea was originally suggested by Sanjaya Addanki and further developed by the PDP group [552, 441]. The basic idea of training a network to reproduce an input that is already available may seem silly at first sight, until one realizes that what is of interest in such a network is not so much its output, but rather the representations that emerge in the hidden layers. Autoencoders can have recurrent connections and multiple hidden layers, but in this chapter we will focus primarily on feedforward autoencoders with a single hidden layer, and thus on $A(n, m, n)$ architectures. This is for simplicity, but also because we want to further the study of $A(n, m, p)$ networks with a single hidden layer started in the previous chapter. Furthermore, in certain important cases such as the linear case, the solution of the single-hidden layer regime is sufficient to understand what happens in the multiple-hidden layer regime. An example of recurrent autoencoder will be studied in a later chapter in the section on recirculation algorithms.

To be more precise, there are several important reasons to spend an entire chapter on autoencoders. Among the main ones:

1. Autoencoders provide an essential bridge between supervised and unsupervised learning, basically by turning an unsupervised problem into a supervised problem. As such, they are the canonical example of what is called self-supervised learning.

2. Autoencoders are useful in practice and can be used in different tasks ranging from dimensionality reduction, to denoising, to deep architecture initialization.

3. Autoencoders provide a natural bridge to several other areas, such as data compression, coding and information theory, principal component analysis and generative and latent-variable models in statistics, to name a few.

4. Autoencoders building blocks can be composed in many ways in order to design more complex networks. For instance, they can be stacked vertically to create certain kinds of deep architectures.

5. Finally, they are essential for the theory of deep learning. After understanding shallow networks and the universal properties of deep networks with a single

hidden layer, it is natural to look at specific networks $A(n, m, p)$ with a single hidden layer. Using the input data as the target is a natural way to reduce the degrees of freedom of the problem and results in $A(n, m, n)$. As we shall see, this reduction is not very limiting: in all the cases where one can obtain a full understanding of the corresponding autoencoder, it is relatively easy to move from the auto association case to the general case of hetero-association. Moreover, in a similar way, we shall see that in all the cases where one can obtain a full understanding of the corresponding autoencoder, it is relatively easy to move from the single hidden layer to the multiple hidden layer case and thereby derive important insights about deeper architectures. We will present a complete treatment of autoencoders in the linear case and the unrestricted Boolean case. This will provide a complete solution to the problem of understanding the functional capacity of these models, and will also lead to key insights on issues of local minima and complexity in more general cases.

We begin by describing a general framework for organizing the study of various autoencoders.

5.1 A General Autoencoder Framework

To derive a fairly general framework, an autoencoder with architecture $A(n, m, n)$ (Figure 5.1) is defined by a tuple $n, m, K, \mathbb{F}, \mathbb{G}, \mathcal{A}, \mathcal{B}, \mathcal{X}, \Delta$ where:

1. n, m and K are positive integers.
2. $\mathbb{F}$ and $\mathbb{G}$ are sets.
3. $\mathcal{A}$ is a class of functions from $\mathbb{G}^m$ to $\mathbb{F}^n$.
4. $\mathcal{B}$ is a class of functions from $\mathbb{F}^n$ to $\mathbb{G}^m$.
5. $\mathcal{X} = \{x_1, \ldots, x_K\}$ is a set of K training vectors in $\mathbb{F}^n$. When external targets are present (hetero-association), we let $\mathcal{Y} = \{y_1, \ldots, y_K\}$ denote the corresponding set of target vectors in $\mathbb{F}^n$. In the hetero-associative case, it is also possible for the targets to be in $\mathbb{F}^p$, for some $p \neq n$.
6. Δ is a distance or distortion function (e.g. L_p norm, Hamming distance) defined over $\mathbb{F}^n$.

For any $A \in \mathcal{A}$ and $B \in \mathcal{B}$, the autoencoder transforms an input vector $x \in \mathbb{F}^n$ into an output vector $A \circ B(x) \in \mathbb{F}^n$ (Figure 5.1). The corresponding *autoencoder*

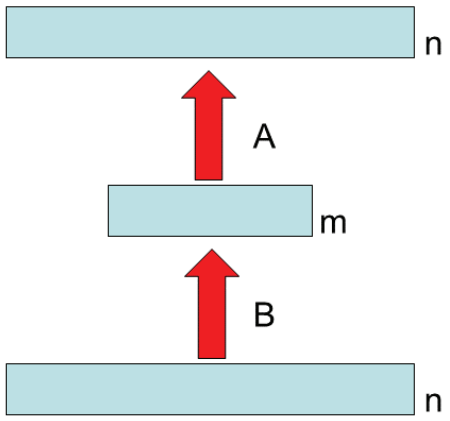

Figure 5.1: An $A(n,m,n)$ Autoencoder Architecture.

problem is to find $A \in \mathcal{A}$ and $B \in \mathcal{B}$ that minimize the overall error or distortion function:

$$\min_{A,B} \mathcal{E}(A,B) = \min_{A,B} \sum_{k=1}^{K} \mathcal{E}(x_k) = \min_{A,B} \sum_{k=1}^{K} \Delta\big(A \circ B(x_k), x_k\big) \tag{5.1}$$

In the hetero-associative case, when external targets y_k are provided, the minimization problem becomes:

$$\min_{A,B} \mathcal{E}(A,B) = \min_{A,B} \sum_{k=1}^{K} \mathcal{E}(x_k, y_k) = \min_{A,B} \sum_{k=1}^{K} \Delta\big(A \circ B(x_k), y_k\big) \tag{5.2}$$

The framework above is not meant to capture all possible kinds of autoencoders, but only a sufficiently large subset, in order to get us started. Within this framework, many different kinds of autoencoders can be considered. For instance, in the linear case, the functions A and B are represented by matrices with entries over a field. In this case, the most relevant autoencoders for this book are the linear autoencoders over the real numbers [79] with the usual quadratic distance. But one can also consider linear autoencoders over the complex numbers [83], or linear autoencoder over finite fields [68, 1]. The theory of linear autoencoder over the complex numbers is very similar to the theory of linear autoencoders over the real numbers (simply by replacing "transpose" by "conjugate transpose" everywhere). The theory of autoencoders over finite fields is closely related to coding theory and beyond the scope of this book. Similarly, in the non-linear case, one can consider different kinds of non-linear functions, such as Boolean functions (e.g. unrestricted or linear threshold), or differentiable functions (e.g. sigmoidal). Here we will focus primarily on the Boolean unrestricted case, which is the only non-linear case that has received a complete treatment so far.

When $n > m$ the autoencoder is called compressive, and when $n \leq m$ the autoencoder is called expansive. Expansive autoencoders are of interest, for instance in a biological context [57], but in general require additional assumptions in order to avoid trivial solutions based on the identity function and their theory is under-developed. Thus, in the next sections, we are going to focus on compressive autoencoders, and specifically on the linear autoencoder over the real numbers and the unrestricted Boolean autoencoder with binary $\{0, 1\}$ variables.

5.2 General Autoencoder Properties

One of the main benefits of studying different classes of autoencoders within this general framework is the identification of a list of common properties that may be investigated in each specific case. This general list is thus a *result* of the analysis which emerges, by generalization, only after careful consideration of several individual cases. With the benefit of hindsight, however, it is given here upfront before delving into the details of each specific case.

1) Invariances. What are the relevant group actions for the problem? What are the transformations of $\mathbb{F}^n$ and $\mathbb{G}^p$, or A and B, that leave the problem or the learning algorithm invariant?

2) Fixed Layer Solutions. Is it possible to optimize A (resp. B), fully or partially, while B (resp. A) is held constant?

3) Problem Complexity. How complex is the autoencoder optimization problem? Is there an overall analytical solution? Is the corresponding decision problem NP-complete?

4) Landscape of E. What is the landscape of the overall error $\mathcal{E}$? Are there any symmetries and critical points (maxima, minima, saddle points)? How can they be characterized?

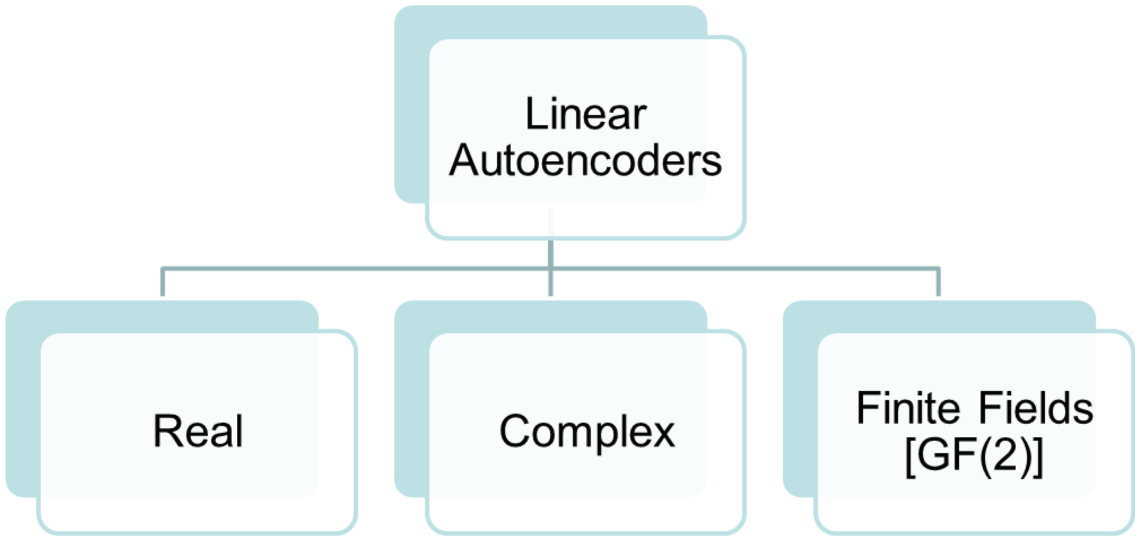

Figure 5.2: Classification of Linear Autoencoders. Linear autoencoders can be defined over different fields, in particular infinite fields such as $\mathbb{R}$ or $\mathbb{C}$, or finite fields such as the Galois Field with two elements GF(2) ($\mathbb{F}_2 = \{0, 1\}$).

5) Clustering. Especially in the compressive case where $m < n$, what is the relationship to clustering?

6) Transposition. Is there a notion of symmetry or transposition between the decoding and coding transformations A and B, in particular around critical points?

7) Recirculating. What happens if the values from the output layer are recycled into the input layer, in particular around critical points?

8) Learning Algorithms. What are the learning algorithms and their properties? In particular, can A and B be fully, or partially, optimized in alternation? And if so, is the algorithm convergent? And if so, at what speed and what are the properties of the corresponding limit points?

9) Generalization. What are the generalization properties of the autoencoder after learning?

10) External Targets. How does the problem change if external targets are provided?

11) Composition. What are the effects of composing autoencoders horizontally or vertically?

Not all these properties can be addressed analytically for all types of autoencoders. But they can be addressed for several important classes of autoencoders, including the linear autoencoders over the real and complex numbers, the linear autoencoders over finite fields, and the non-linear unrestricted Boolean or unrestricted probabilistic autoencoders.

5.3 Linear Autoencoders

Here we consider the compressive, real-valued, linear case where $\mathbb{F} = \mathbb{G} = \mathbb{R}$ and the goal is the minimization of the squared Euclidean distance:

$$\min \mathcal{E}(A,B) = \min_{A,B} \sum_{t=k}^{K} ||x_k - ABx_k||^2 = \sum_{k=1}^{K} (x_k - ABx_k)^t (x_k - ABx_k) \tag{5.3}$$

Unless otherwise specified, all vectors are column vectors and we use x^t (resp. X^t) to denote the transpose (or conjugate transpose in order to deal with the complex case) of a vector x (resp. of a matrix X). As we shall see, one can also address the hetero-associative case where external targets are available, in which case the goal is the minimization of the distance:

$$\min \mathcal{E}(A,B) = \min_{A,B} \sum_{k=1}^{K} ||y_k - ABx_k||^2 = \sum_{k=1}^{K} (y_k - ABx_k)^t (y_k - ABx_k) \tag{5.4}$$

The following covariance matrices will be useful. In general, we define:

$$\Sigma_{XY} = \sum_k x_k y_k^t \tag{5.5}$$

Using this definition, Σ_{XX}, Σ_{YY} are symmetric (Hermitian in the complex case) matrices $(\Sigma_{XX})^t = \Sigma_{XX}$ and $(\Sigma_{YY})^t = \Sigma_{YY}$, and $(\Sigma_{XY})^t = \Sigma_{YX}$. We also define:

$$\Sigma = \Sigma_{YX} \Sigma_{XX}^{-1} \Sigma_{XY} \tag{5.6}$$

Σ is also symmetric. In the *auto-associative* case, $x_k = y_k$ for all k resulting in $\Sigma = \Sigma_{XX}$. Note that any symmetric matrix admits a set of orthonormal eigenvectors and all its eigenvalues are real. Finally, we let I_m denote the $m \times m$ identity matrix.

For several results, we will make the assumption that Σ is invertible. This is not a very restrictive assumption for several reasons. First, by adding a small amount of noise to the data, a non-invertible Σ could be converted to an invertible Σ. More importantly, in many settings one can expect the training vectors to span the entire input space and thus Σ to be invertible. If the training vectors span a smaller subspace, then the original problem can be transformed to an equivalent problem defined on the smaller subspace.

5.3.1 Useful Reminders

Here we provide a few reminders that will be useful to solve the linear autoencoder from first principles in a general, coordinate-invariant, way.

Standard Linear Regression. Consider the standard linear regression problem of minimizing $\mathcal{E}(B) = \sum_k ||y_k - Bx_k||^2$, where B is an $m \times n$ matrix, corresponding to a linear neural network without any hidden layers. Then we can write:

$$\mathcal{E}(B) = \sum_k x_k^t B^t B x_k - 2y_k^t B x_k + ||y_k||^2 \tag{5.7}$$

Thus $\mathcal{E}$ is a convex function in B because the associated quadratic form is equal to:

$$\sum_k x_k^t C^t C x_k = \sum_k ||Cx_k||^2 \geq 0 \tag{5.8}$$

Let B be a critical point. Then by definition for any $m \times n$ matrix C we must have $lim_{\epsilon \to 0}\ [\mathcal{E}(B + \epsilon C) - \mathcal{E}(B)]/\epsilon = 0$. Expanding and simplifying this expression gives:

$$\sum_k x_k^t B^t C x_k - y_k^t B C x_k = 0 \tag{5.9}$$

for all $m \times n$ matrices C. Using the linearity of the trace operator and its invariance under circular permutation of its arguments[1], this is equivalent to:

[1]It is easy to show directly that for any matrices A and B of the proper size, $\mathrm{Tr}(AB) = \mathrm{Tr}(BA)$ [389]. Therefore for any matrices A, B, and C of the proper size, we have $\mathrm{Tr}(ABC) = \mathrm{Tr}(CAB) = \mathrm{Tr}(BCA)$.

$$\mathrm{Tr}\left((\Sigma_{XX}B^t - \Sigma_{XY})C\right) = 0 \tag{5.10}$$

for any C. Thus we have $\Sigma_{XX}B^t - \Sigma_{XY} = 0$ and therefore:

$$B\Sigma_{XX} = \Sigma_{YX} \tag{5.11}$$

If Σ_{XX} is invertible, then $Cx_k = 0$ for any t is equivalent to $C = 0$, and thus the function $\mathcal{E}(B)$ is strictly convex in B. The unique critical point is the global minimum given by $B = \Sigma_{YX}\Sigma_{XX}^{-1}$. This is just another way of deriving the linear regression result obtained by looking at gradient descent in shallow networks in Chapter 3. As we shall see, the solution to the standard linear regression problem, together with the general approach given here to solve it, is also key for solving the more general linear autoencoder problem. The solution will also involve projection matrices.

Projection Matrices. For any $n \times m$ matrix A with $m \leq n$, let P_A denote the orthogonal projection onto the subspace generated by the columns of A. Then P_A is a symmetric matrix and $P_A^2 = P_A$, $P_AA = A$ since the image of P_A is spanned by the columns of A and these are invariant under P_A. The kernel of P_A is the space $A^\perp$ orthogonal to the space spanned by the columns of A. Obviously, we have $P_AA^\perp = 0$ and $A^tP_A = A^t$. The projection onto the space orthogonal to the space spanned by the columns of A is given by $I_n - P_A$. In addition, if the columns of A are independent (i.e. A has full rank m), then the matrix of the orthogonal projection is given by $P_A = A(A^tA)^{-1}A^t$ [449] and $P_A^t = P_A$. Note that all these relationships are true even when the columns of A are not orthonormal.

Some Misconceptions. As we shall see, the global minimum of the linear autoencoder corresponds to Principal Component Analysis. While the global minimum solution of linear autoencoders over $\mathbb{R}$ and $\mathbb{C}$ can be expressed analytically, it is often not well appreciated that there is more to be understood about linear autoencoders. In particular, if one is interested in learning algorithms that proceed through incremental and somewhat "blind" weight adjustments, then one must study the entire landscape of $\mathcal{E}$, including all the critical points of $\mathcal{E}$, and derive and compare different learning algorithms. A second misconception is to believe that the problem is a convex optimization problem, hence somewhat trivial, since after all the error function is quadratic and the transformation $W = AB$ is linear. As previously mentioned, the problem with this argument is that the bottleneck layer forces W to be of rank m or less, and the set of matrices of rank at most m is *not* convex. What is true, and crucial for solving the linear autoencoders over infinite fields, is that the problem becomes convex when A or B is fixed.

5.3.2 Group Invariances

For any autoencoder, it is important to investigate whether there are any group of transformations on the data or on the weights that leave its properties invariant.

Change of Coordinates in the Hidden Layer. Note that for any invertible $m \times m$ matrix C, we have $W = AB = ACC^{-1}B$ and $\mathcal{E}(A, B) = \mathcal{E}(AC, C^{-1}B)$. Thus all the properties of the linear autoencoder are invariant with respect to any change of coordinates in the hidden layer.

Change of Coordinates in the Input/Output Spaces. Consider an orthonormal change of coordinates in the output space defined by an orthogonal (or unitary) $n \times n$ matrix D, and any change of coordinates in the input space defined by an invertible $N \times N$ matrix C. This leads to a new autoencoder problem with input vectors $Cx_1, \dots, Cx_K$ and target output vectors of the form $Dy_1, \dots, Dy_K$ with reconstruction error of the form:

$$\mathcal{E}(A', B') = \sum_k ||Dy_k - A'B'Cx_k||^2 \tag{5.12}$$

If we use the one-to-one mapping between pairs of matrices (A, B) and (A', B') defined by $A' = DA$ and $B' = BC^{-1}$, we have:

$$\mathcal{E}(A', B') = \sum_k ||Dy_k - A'B'Cx_k||^2 = \sum_k ||Dy_k - DABx_k||^2 = \mathcal{E}(A, B) \tag{5.13}$$

the last equality using the fact that D is an isometry which preserves distances. Thus, using the transformation $A' = DA$ and $B' = BC^{-1}$ the original problem and the transformed problem are equivalent and the function $\mathcal{E}(A, B)$ and $\mathcal{E}(A', B')$ have the same landscape. In particular, in the auto-associative case, we can take $C = D$ to be a unitary matrix. This leads to an equivalent autoencoder problem with input vectors Cx_k and covariance matrix $C\Sigma C^{-1}$. For the proper choice of C there is an equivalent problem where the basis of the space is provided by the eigenvectors of the covariance matrix and the covariance matrix is a diagonal matrix with diagonal entries equal to the eigenvalues of the original covariance matrix Σ (see exercises).

5.3.3 Fixed-Layer and Convexity Results

A key technique for studying any autoencoder, is to simplify the problem by fixing all its transformations but one. Thus in this section we study what happens to the linear autoencoder problem when either A or B is fixed, essentially reducing the problem to standard linear regression.

Theorem 9. (Fixed A) *For any fixed $n \times m$ matrix A, the function $\mathcal{E}(A, B)$ is convex in the coefficients of B and attains its minimum for any B satisfying the equation:*

$$A^t AB\Sigma_{XX} = A^t \Sigma_{YX} \tag{5.14}$$

If Σ_{XX} is invertible and A is of full rank m, then $\mathcal{E}$ is strictly convex and has a unique minimum reached when:

$$B = (A^t A)^{-1} A^t \Sigma_{YX} \Sigma_{XX}^{-1} \tag{5.15}$$

In the auto-associative case, if Σ_{XX} is invertible and A is of full rank m, then the optimal B has full rank M and does not depend on the data. It is given by:

$$B = (A^t A)^{-1} A^t \tag{5.16}$$

and in this case, $W = AB = A(A^t A)^{-1} A^t = P_A$ and $BA = I_m$.

Proof. We write:

$$\mathcal{E}(A, B) = \sum_k x_k^t B^t A^t AB x_k - 2(y_k^t AB x_k) + ||y_k||^2 \tag{5.17}$$

Then for fixed A, $\mathcal{E}$ is a convex function because the associated quadratic form is equal to:

$$\sum_k x_k^t C^t A^t AC x_k = \sum_k ||ACx_k||^2 \geq 0 \tag{5.18}$$

for any $m \times n$ matrix C. Let B be a critical point. Then by definition for any $m \times n$ matrix C we must have $lim_{\epsilon \to 0}$ $[\mathcal{E}(A, B + \epsilon C) - \mathcal{E}(A, B)]/\epsilon = 0$. Expanding and simplifying this expression gives:

$$\sum_k x_k^t B^t A^t AC x_k - y_k^t AC x_k = 0 \tag{5.19}$$

for all $m \times n$ matrices C. Using the linearity of the trace operator and its invariance under circular permutation of its arguments, this is equivalent to:

$$\operatorname{Tr}\left((\Sigma_{XX}B^tA^tA - \Sigma_{XY}A)C\right) = 0 \tag{5.20}$$

for any C. Thus we have $\Sigma_{XX}B^tA^tA - \Sigma_{XY}A = 0$ and therefore:

$$A^tAB\Sigma_{XX} = A^t\Sigma_{YX} \tag{5.21}$$

Finally, if Σ_{XX} is invertible and if A is of full rank, then $ACx_k = 0$ for any k is equivalent to $C = 0$, and thus the function $\mathcal{E}(A, B)$ is strictly convex in B. Since A^tA is invertible, the unique critical point is obtained by solving Equation 5.14.

In similar fashion, we have the following theorem.

Theorem 10 (Fixed B). *For any fixed $m \times n$ matrix B, the function $\mathcal{E}(A, B)$ is convex in the coefficients of A and attains its minimum for any A satisfying the equation:*

$$AB\Sigma_{XX}B^t = \Sigma_{YX}B^t \tag{5.22}$$

If Σ_{XX} is invertible and B is of full rank, then $\mathcal{E}$ is strictly convex and has a unique minimum reached when:

$$A = \Sigma_{YX}B^t(B\Sigma_{XX}B^t)^{-1} \tag{5.23}$$

In the auto-associative case, if Σ_{XX} is invertible and B is of full rank, then the optimal A has full rank m and depends on the data. It is given by:

$$A = \Sigma_{XX}B^t(B\Sigma_{XX}B^t)^{-1} \tag{5.24}$$

and $BA = I_m$.

Proof. From Equation 5.17, the function $\mathcal{E}(A, B)$ is a convex function in A. The condition for A to be a critical point is:

$$\sum_k x_k^tB^tA^tCBx_k - y_k^tCBx_k = 0 \tag{5.25}$$

for any $m \times n$ matrix C, which is equivalent to:

$$\operatorname{Tr}\left((B\Sigma_{XX}B^tA^t - B\Sigma_{XY})C\right) = 0 \tag{5.26}$$

for any matrix C. Thus $B\Sigma_{XX}B^tA^t - B\Sigma_{XY} = 0$ which implies Equation 5.22. The other statements in the theorem follow immediately.

Remark 1. *Note that from Theorems 9 and 10 and their proofs, we have that (A, B) is a critical point of $\mathcal{E}(A, B)$ if and only if Equation 5.14 and Equation 5.22 are simultaneously satisfied, that is if and only if $A^tAB\Sigma_{XX} = A^t\Sigma_{YX}$ and $AB\Sigma_{XX}B^t = \Sigma_{YX}B^t$.*

5.3.4 Critical Points and the Landscape of $\mathcal{E}$

In this section we further study the landscape of $\mathcal{E}$, its critical points, and the properties of $W = AB$ at those critical points.

Theorem 11. (Critical Points) *Assume that Σ_{XX} is invertible. Then two matrices (A, B) define a critical point of $\mathcal{E}$, if and only if the global map $W = AB$ is of the form:*

$$W = P_A\Sigma_{YX}\Sigma_{XX}^{-1} \tag{5.27}$$

with A satisfying:

$$P_A\Sigma = P_A\Sigma P_A = \Sigma P_A \tag{5.28}$$

In the auto-associative case, this becomes:

$$W = AB = P_A \tag{5.29}$$

and:

$$P_A\Sigma_{XX} = P_A\Sigma_{XX}P_A = \Sigma_{XX}P_A \tag{5.30}$$

If A is of full rank, then the pair (A, B) defines a critical point of $\mathcal{E}$ if and only if A satisfies Equation 5.28 and B satisfies Equation 5.16. Hence B must also be of full rank.

Proof. If (A, B) is a critical point of $\mathcal{E}$, then from Equation 5.14, we must have:

$$A^t(AB - \Sigma_{YX}\Sigma_{XX}^{-1}) = 0 \tag{5.31}$$

Let:

$$S = AB - P_A\Sigma_{YX}\Sigma_{XX}^{-1} \tag{5.32}$$

Then since $A^tP_A = A^t$, we have $A^tS = 0$. Thus the space spanned by the columns of S is a subset of the space orthogonal to the space spanned by the columns of A (i.e. $S \in A^\perp$). On the other hand, since:

$$P_AS = S \tag{5.33}$$

S is also in the space spanned by the columns of A (i.e. $S \in Span(A)$). Taken together, these two facts imply that $S = 0$, resulting in $W = AB = P_A\Sigma_{YX}\Sigma_{XX}^{-1}$, which proves Equation 5.27. Note that for this result, we need only B to be critical (i.e. optimized with respect to A). Using the definition of Σ, we have:

$$P_A\Sigma P_A = P_A\Sigma_{YX}\Sigma_{XX}^{-1}\Sigma_{XX}\Sigma_{XX}^{-1}\Sigma_{XY}P_A \tag{5.34}$$

Since $S = 0$, we have $AB = P_A\Sigma_{YX}\Sigma_{XX}^{-1}$ and thus:

$$P_A\Sigma P_A = P_A\Sigma_{YX}\Sigma_{XX}^{-1}\Sigma_{XX}\Sigma_{XX}^{-1}\Sigma_{XY}P_A = AB\Sigma_{XX}B^tA^t \tag{5.35}$$

Similarly, we have:

$$P_A\Sigma = AB\Sigma_{XY} \tag{5.36}$$

and:

$$\Sigma P_A = \Sigma_{YX}B^tA^t \tag{5.37}$$

Then Equation 5.28 result immediately by combining Equations 5.35, 5.36, and 5.37 using Equation 5.22. The rest of the theorem follows easily.

Remark 2. *The above proof unifies the cases when AB is of rank m and strictly less than m.*

Theorem 12. (Critical Points of Full Rank) *Assume that Σ is of full rank with n distinct eigenvalues $\lambda_1 > \cdots > \lambda_n$ and let $u_1, \ldots, u_n$ denote a corresponding basis of orthonormal eigenvectors. If $\mathcal{I} = \{i_1, \ldots, i_m\}$ $(1 \leq i_1 < \ldots < i_m \leq n)$ is any ordered set of indices of size m, let $U_{\mathcal{I}} = (u_{i_1}, \ldots, u_{i_m})$ denote the matrix formed using the corresponding column eigenvectors. Then two full rank matrices A, B define a critical point of $\mathcal{E}$ if and only if there exists an ordered m-index set $\mathcal{I}$ and an invertible $m \times m$ matrix C such that:*

$$A = U_{\mathcal{I}} C \quad \text{and} \quad B = C^{-1} U_{\mathcal{I}}^t \Sigma_{YX} \Sigma_{XX}^{-1} \tag{5.38}$$

For such critical point, we have:

$$W = AB = P_{U_{\mathcal{I}}} \Sigma_{YX} \Sigma_{XX}^{-1} \tag{5.39}$$

and:

$$\mathcal{E}(A, B) = \operatorname{Tr} \Sigma_{YY} - \sum_{i \in \mathcal{I}} \lambda_i \tag{5.40}$$

In the auto-associative case, these equations reduce to:

$$A = U_{\mathcal{I}} C \quad \text{and} \quad B = C^{-1} U_{\mathcal{I}}^t \tag{5.41}$$

$$W = AB = P_{U_{\mathcal{I}}} \tag{5.42}$$

and:

$$\mathcal{E}(A, B) = \operatorname{Tr} \Sigma - \sum_{i \in \mathcal{I}} \lambda_i = \sum_{i \in \bar{\mathcal{I}}} \lambda_i \tag{5.43}$$

where $\bar{\mathcal{I}} = \{1, \ldots, N\} \backslash \mathcal{I}$ is the complement of $\mathcal{I}$.

Proof. Since $P_A\Sigma = \Sigma P_A$, we have

$$P_A\Sigma A = \Sigma P_A A = \Sigma A \tag{5.44}$$

Thus the columns of A form an invariant space of Σ. Thus A is of the form $U_{\mathcal{I}}C$. The conclusion for B follows from Equation 5.27 and the rest is easily derived. Equation 5.43 can be derived easily by using the reminders given at the beginning of this section on linear autoencoders and using the unitary change of coordinates under which Σ_{XX} becomes a diagonal matrix. In this system of coordinates, we have:

$$\mathcal{E}(A,B) = \sum_k ||y_k||^2 + \sum_k \mathrm{Tr}\,(x_k^t(AB)^tABx_k) - 2\sum_k \mathrm{Tr}\,(y_k^tABx_k)$$

Therefore, using the invariance property of the trace under circular permutations, we have:

$$\mathcal{E}(A,B) = \mathrm{Tr}\,(\Sigma) + \mathrm{Tr}\,((AB)^2\Sigma) - 2\mathrm{Tr}\,(AB\Sigma)$$

Since AB is a projection operator, this yields Equation 5.43. In the auto-associative case with these coordinates it is easy to see that Wx_k and $\mathcal{E}(A,B) = \sum_k \mathcal{E}(x_k)$ are easily computed from the values of Wu_i. In particular, $\mathcal{E}(A,B) = \sum_{i=1}^n \lambda_i(u_i - Wu_i)^2$. In addition, at the critical points, we have $Wu_i = u_i$ if $i \in I$, and $Wu_i = 0$ otherwise.

Remark 3. *All the previous theorems are true in the hetero-associative case with targets y_k. Thus they can readily be applied to address the linear denoising autoencoder [660, 659] over $\mathbb{R}$ or $\mathbb{C}$. The linear denoising autoencoder is an autoencoder trained to remove noise by having to associate noisy versions of the inputs with the correct inputs. In other words, using the current notation, it is an autoencoder where the inputs x_k are replaced by $x_k + n_k$ where n_k is the noise vector and the target outputs y_k are of the form $y_k = x_k$. Thus the previous theorems can be applied using the following replacements: $\Sigma_{XX} = \Sigma_{XX} + \Sigma_{NN} + \Sigma_{NX} + \Sigma_{XN}$, $\Sigma_{XY} = \Sigma_{XX} + \Sigma_{NX}$, $\Sigma_{YX} = \Sigma_{XX} + \Sigma_{XN}$. Further simplifications can be obtained using particular assumptions on the noise, such as $\Sigma_{NX} = \Sigma_{XN} = 0$.*

Theorem 13. (Absence of Local Minima) *The global minimum of the linear autoencoder is achieved by full rank matrices A and B associated with the index set $1,\ldots,m$ of the m largest eigenvalues of Σ with $A = U_{\mathcal{I}}C$ and $B = C^{-1}U_{\mathcal{I}}^t$ (and where C is any invertible $m \times m$ matrix). When $C = I$, $A = B^t$. All other critical points are saddle points associated with corresponding projections onto non-optimal sets of eigenvectors of Σ of size m or less.*

Proof. The proof is by a perturbation argument showing that, for the critical points that are not associated with the global minimum, there is always a direction of escape that can be derived using unused eigenvectors associated with higher eigenvalues in order to lower the error $\mathcal{E}$ (see [79] for more details). The proof can be simplified by using the group invariance properties under transformation of the coordinates by a unitary matrix. With such a transformation, it is sufficient to study the landscape of $\mathcal{E}$ when Σ is a diagonal matrix and $A = B^t = U_{\mathcal{I}}$.

Remark 4. *At the global minimum, if C is the $m \times m$ identity matrix ($C = I$), in the auto-associative case then the activities in the hidden layer are given by $u_1^t x, \ldots, u_p^t x$, corresponding to the coordinates of x along the first m eigenvectors of Σ_{XX}. These are the so called principal components of x and the autoencoder implements Principal Component Analysis (PCA), also closely related to the Singular Value Decomposition (SVD), establishing another bridge between neural networks and statistics.*

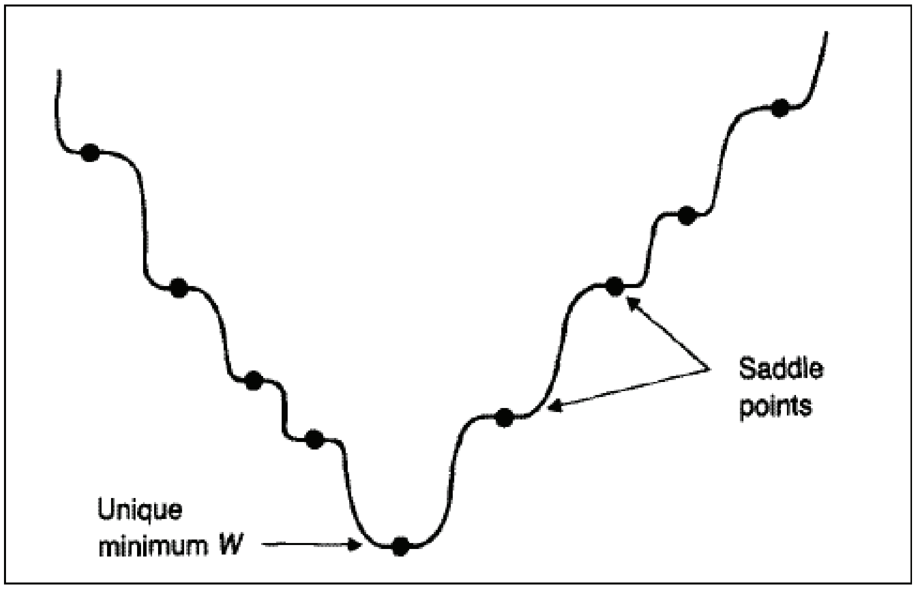

Figure 5.3: Landscape of $\mathcal{E}$.

The theorem above shows that when Σ is full rank, there is a special class of critical points associated with $C = I$. In the auto-associative case, this class is characterized by the fact that A and B are transpose of each other ($A = B^t$).

Theorem 14. (Conjugate Transposition) *Assume Σ_{XX} is of full rank in the auto-associative case. Consider any point (A, B) where B has been optimized with respect to A, including all critical points. Then:*

$$W = AB = B^t A^t AB = B^t A^t = W^t \quad \text{and} \quad \mathcal{E}(A, B) = \mathcal{E}(B^t, A^t) \tag{5.45}$$

Furthermore, when A is full rank:

$$W = P_A = P_A^t = W^t \tag{5.46}$$

Proof. By Theorem 1, in the auto-associate case, we have:

$$A^t AB = A^t$$

Thus, by taking the transpose of each side, we have:

$$B^t A^t A = A$$

It follows that:

$$B^t A^t = B^t A^t AB = AB$$

which proves Equation 5.45. If in addition A is full rank, then by Theorem 9 $W = AB = P_A$ and the rest follows immediately.

Remark 5. *Starting from a pair (A, B) with $W = AB$ and where B has been optimized with respect to A, let $A' = B^t$ and optimize B again so that $B' = (A'A'^t)^{-1}A'^t$. Then we also have:*

$$W' = A'B' = W^t = W = P_A \quad \text{and} \quad \mathcal{E}(A, B) = \mathcal{E}(A', B') \tag{5.47}$$

5.3.5 Learning Algorithms

Although mathematical formula for the global minimum solution of the linear autoencoder have been derived, it is still useful to study alternative incremental learning algorithms for several reasons. First, the global solution may not be available immediately to a physical, self-adjusting, learning circuit capable of making only small adjustments at each learning step. In addition, small adjustments may also be preferable in a non-stationary environment where the set $\mathcal{X}$ of training vectors changes with time, or in memory-limited situations where the entire training set cannot be stored. Finally, the study of incremental learning algorithms in linear circuits may shed some light on similar incremental algorithms applied to non-linear circuits where the global optimum cannot be derived analytically.

For the linear autoencoder, several such algorithms can be conceived, including: (stochastic) gradient descent studied in the exercises and in the next chapter; recirculation (studied in a later chapter); and alternative partial or full optimization of A and B. Here we provide one theorem on the latter, leaving the proof as an exercise (see, also [83] for additional details).

Theorem 15. (Alternate Minimization) *Consider the algorithm where A and B are optimized in alternation (starting from A or B), holding the other one fixed. This algorithm will converge to a critical point of $\mathcal{E}$. Furthermore, if the starting value of A or B is initialized randomly, then with probability one the algorithm will converge to a critical point where both A and B are full rank.*

5.3.6 Generalization Properties

One of the most fundamental problems in machine learning is to understand the generalization properties of a learning system. Although in general this is not a simple problem, in the case of the autoencoder the generalization properties can easily be understood. After learning, A and B must be at a critical point. Assuming without much loss of generality that A is also full rank and Σ_{XX} is invertible, then from Theorem 1 we know in the auto-associative case that $W = P_A$. Thus we have the following result.

Theorem 16. (Generalization Properties) *Assume in the auto-associative case that Σ_{XX} is invertible. For any learning algorithm that converges to a point where B is optimized with respect to A and A is full rank (including all full rank critical points), then for any vector x we have $Wx = ABx = P_Ax$ and:*

$$\mathcal{E}(x) = ||x - ABx||^2 = ||x - P_Ax||^2 \tag{5.48}$$

Remark 6. *Thus the reconstruction error of any vector is equal to the square of its distance to the subspace spanned by the columns of A, or the square of the norm of its projection onto the orthogonal subspace. The general hetero-associative case can also be treated using Theorem 1. In this case, under the same assumptions, we have: $W = P_A\Sigma_{YX}\Sigma_{XX}^{-1}$.*

5.4 Non-Linear Autoencoders: Unrestricted Boolean Case

5.4.1 Analysis of the Unrestricted Boolean Autoencoder

The unrestricted Boolean autoencoder where the classes $\mathcal{A}$ and $\mathcal{B}$ correspond to unrestricted Boolean transformations is the most extreme form of non-linear autoencoder. In the purely Boolean case, we have $\mathbb{F} = \mathbb{G} = \{0, 1\}$, A and B are unrestricted Boolean functions, and Δ is the Hamming distance. The only case of interest is the compressive case when $m < n$ (to avoid the identity with zero training error) and $2^m < K$. If $2^m \geq K$, it is easy to see that the training data can be implemented with zero error by associating each training vector with a unique hidden representation. 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 of the hidden units, or restricting them to particular classes of Boolean functions such as threshold gates or monotone Boolean functions. The linear case discussed in the previous section, where $\mathbb{F} = \mathbb{G} = \{0, 1\} = \mathbb{F}_2$ is the Galois field with two elements, is also a special case of the general Boolean case.

1) Invariances.

Permutations in the Hidden Layer. Every solution is defined up to arbitrary permutations of the 2^m points of the hypercube $\mathbb{H}^m$. This is because the Boolean functions are unrestricted and therefore their lookup tables can accommodate any such permutation, or relabeling of the hidden states.

Change of Coordinates in the Input/Output Layers. Likewise, any change of coordinates in $\mathbb{H}^n$ that preserves Hamming distances (these are generated by coordinate permutations and bit flips) leads to an equivalent problem.

2) Fixed Layer Solution. The best way to understand this autoencoder is to look at what happens when one of the two transformations A or B is fixed. In each case, one can assume that the transformations are defined on the training set alone, or on the entire input space. Here we will choose the latter, although the reader should also examine the other cases and the small adjustments they require.

First, note that since $2^m < K$, the mapping B clusters the K training points in at most 2^m clusters. The hidden vectors $h_i = B(x_i)$ are the labels for each cluster and the outputs $A(h_i)$ ought to be cluster centroids to minimize the reconstruction error. It is easy to see that in order to be optimal B ought to be surjective, i.e. use all the 2^m possible hidden values. If a hidden value h is not used, A and B can be improved by taking any training point x_i with non-zero distortion and setting $B(x_i) = h$ and $A(h) = x_i$. By isolating x_i into its own cluster, $\mathcal{E}(x_i)$ decreases to 0, while $\mathcal{E}(x)$ remains unchanged for all the other training points, thus the overall reconstruction error decreases. Similarly, one can see that in order to be optimal A ought to be injective. Under the safe assumption that A is injective and B surjective, we can now show that if A is fixed, then it is easy to find the optimal B. Conversely, if B is fixed, it is easy to find the optimal A.

To see this more precisely, assume first that A is fixed and defined over the entire space $\mathbb{H}^m$. Then for each of the 2^m Boolean vectors $h_1, \ldots, h_{2^M}$ of the hidden layer, $A(h_1) \ldots, A(h_{2^m})$ provide 2^m distinct points (centroids) in the hypercube $\mathbb{H}^n$. One can build the corresponding Voronoi partition by assigning each point of $\mathbb{H}^n$ to its closest centroid, breaking ties arbitrarily, thus forming a partition of $\mathbb{H}^n$ into 2^m

corresponding clusters $\mathcal{C}_1, \ldots, \mathcal{C}_{2^m}$, with $\mathcal{C}_i = \mathcal{C}^{Vor}(A(h_i))$. The optimal mapping B^* is then easily defined by setting $B^*(x) = h_i$ for any x in $\mathcal{C}_i = \mathcal{C}^{Vor}(A(h_i))$ (Figure 5.4). Note that this provides not only a definition of $B*$ over $\mathcal{X}$, but also a unique and optimal generalization for $B*$ and hence $W = A \circ B*$ over the entire input space $\mathbb{H}^n$. The hetero-associative case can be handled similarly.

Conversely, assume that B is fixed and defined over the entire space $\mathbb{H}^n$. Then for each of the 2^m possible Boolean vectors $h_1, \ldots, h_{2^m}$ of the hidden layer, let $\mathcal{C}^B(h_i) = \{x \in \mathbb{H}^n : B(x) = h_i\} = B^{-1}(h_i)$. To minimize the reconstruction error over the training set, the optimal A^* must map h_i onto a point y of $\mathbb{H}^n$ minimizing the sum of Hamming distances to points in $\mathcal{X} \cap \mathcal{C}^B(h_i)$. It is easy to see that the minimum over the training set is realized by the component-wise majority vector $A^*(h_i) = Majority[\mathcal{X} \cap \mathcal{C}^B(h_i)]$, breaking ties arbitrarily (e.g. by a coin flip) (Figure 5.5). Note that it is also possible to take the vector in the training set that is closest to the centroid if it is required that the output vector be in the training set. The optimal generalization over the entire hypercube, however, is achieved by taking the centroid of the entire cluster $\mathcal{C}^B(h_i)$, rather than over the training vectors in the cluster. Note again that this provides a definition of $A*$ over the entire space $\mathbb{H}^m$ with an option for optimizing the training error or the generalization error of W over $\mathbb{H}^n$. The centroid of the training vectors in a Voronoi cluster provides the optimum for training whereas the centroid of the entire cluster provides the optimum for generalization. The hetero-associative case can be handled similarly.

3) 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 among n, m, and K are going to infinity. Obviously one must have $n \to \infty$. If m does not go to infinity, then the problem can be polynomial, for instance when the centroids must belong to the training set. If $m \to \infty$ and K is a polynomial in n, which is the case of interest in machine learning where typically K is a low degree polynomial in n, then the problem of finding the best Boolean mapping (i.e. the Boolean mapping that minimizes the distortion $\mathcal{E}$ associated with the Hamming distance on the training set) is NP-hard, or the corresponding decision problem is NP-complete. More precisely the optimisation problem is NP-hard in the regime where $m \sim \epsilon \log_2 K$ with $\epsilon > 0$. A proof of this result is given below.

4) The Landscape of E. In general $\mathcal{E}$ has many local minima (e.g with respect to the Hamming distance applied to the lookup tables of A and B). Critical points are defined to be the points satisfying simultaneously the equations above for A^* and B^*.

5) Clustering. The overall optimization problem is a problem of optimal clustering. The clustering is defined by the transformation B. Approximate solutions can be sought by many algorithms, such as k-means, belief propagation [262], minimum spanning paths and trees [595], and hierarchical clustering.

6) Transposition. In the case of the Boolean autoencoder, it is not entirely clear what a notion of transposition ought to be around optimal points. On the other hand, consider an optimal point where B maps several points $x_1, \ldots, x_q$ onto a vector h in the hidden layer. Then $A(h)$ must be the majority vector of $x_1, \ldots, x_q$ and one should have $B(A(h)) = h$. Thus $A(h)$ must be an element of $B^{-1}(h)$ and this could be used to define some notion of "transposition" around optimal points.
7) Recycling. At any critical point, recycling outputs is stable at the first pass so that for any x $(AB)^n(x) = AB(x)$ (and is equal to the majority vector of the corresponding Voronoi cluster).
8) Learning Algorithms. A possible learning algorithm is again to alternate between the optimization of A and B, while holding the other one fixed.
9) Generalization. At any critical point, for any x, $AB(x)$ is equal to the centroid of the corresponding Voronoi cluster and the corresponding error can be expressed easily.
10) External Targets. With the proper adjustments, the results above remain essentially the same if a set of target output vectors $y_1, \ldots, y_m$ is provided, instead of $x_1, \ldots, x_m$ serving as the targets. To see this, consider a deep architecture consisting of a stack of autoencoders along the lines of [310]. For any activity vector h in the last hidden layer before the output layer, compute the set of points $\mathcal{C}(h)$ in the training set that are mapped to h by the stacked architecture. Assume, without any loss of generality, that $\mathcal{C}(h) = \{x_1, \ldots, x_k\}$ with corresponding targets $\{y_1, \ldots, y_k\}$. Then it is easy to see that the final output for h produced by the top layer ought to be the centroid of the targets given by $Majority(y_1, \ldots, y_k)$.
11) Composition. The global optimum remains the same if additional Boolean layers of size equal or greater to p 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 $\mathcal{E}$ by adding such layers. (see also Section 6.3). For instance, consider a Boolean autoencoder network with layers of size n, p_1, p, p_1, n (Figure 5.7) with $n > p_1 > p$. Then any optimal solution of this network induces an optimal solution for the corresponding n, p, n autoencoder and conversely, any optimal solution of the n, p, n autoencoder induces (usually in many ways) an optimal solution of the n, p_1, p, p_1, n autoencoder. Finding an optimal solution for the n, p_1, n autoencoder network, and combining it with the optimal solution of the p_1, p, p_1 autoencoder network derived using the activity in the hidden layer of the first network as the training set, exactly as in the case of stacked RBMs, is a reasonable strategy but it is not clear that it is guaranteed to provide the global optimum of the n, p, n problem in all cases. In other words, clustering the data first into 2^{p_1} clusters, and then clustering these clusters into 2^p clusters, may not always provide the best clustering of the data into 2^p clusters. However, in practice, each clustering problem is NP-complete and thus combining clustering algorithms in a hierarchical fashion, or equivalently stacking autoencoders, should provide a

reasonable approximate algorithm.

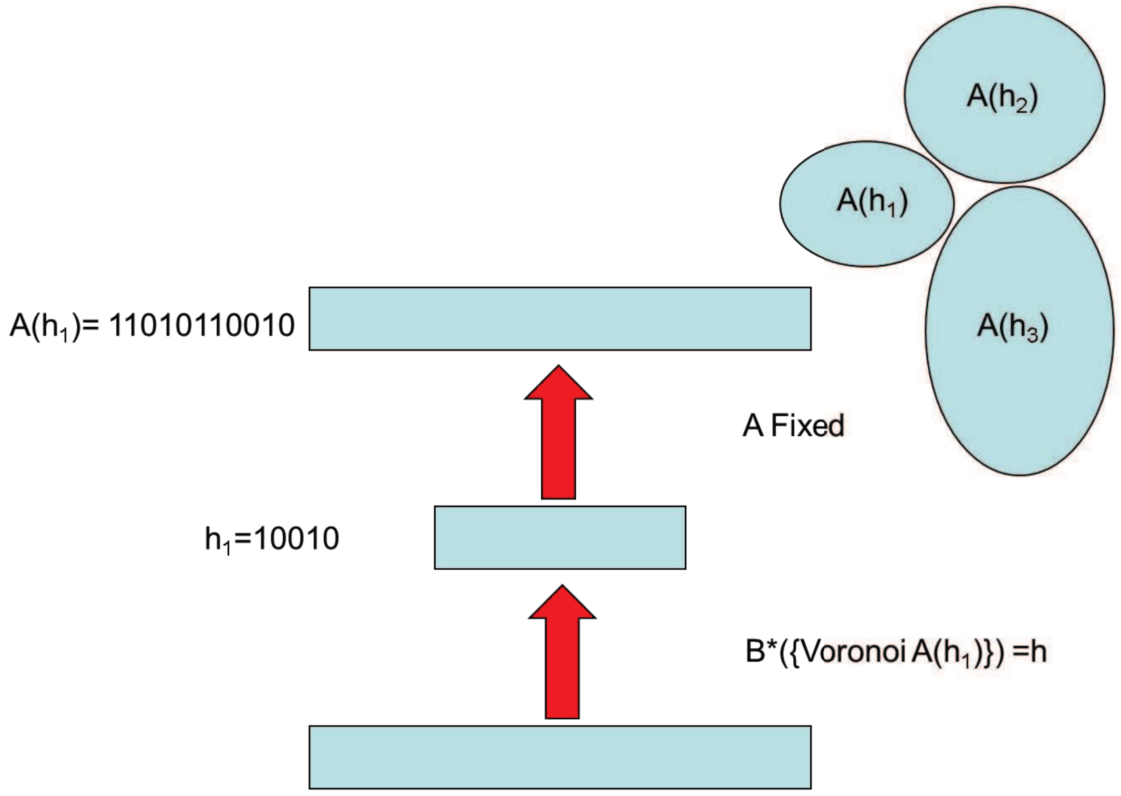

Figure 5.4: $A(n, m, n)$ Unrestricted Boolean Autoencoder with Fixed Transformation A. For each vector h, in the hidden layer, A produces an output vector $A(h)$. The vectors $A(h)$ induce a Voronoi partition of the output space, hence of the input space. Any vector $x \in \mathbb{F}^n$ is in a partition of the form $A(h)$, for some h. In order to minimize the final distortion, one must have $B^*(x) = h$.

5.4.2 Boolean Autoencoder Problem Complexity

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 $p < n$, 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 in general 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

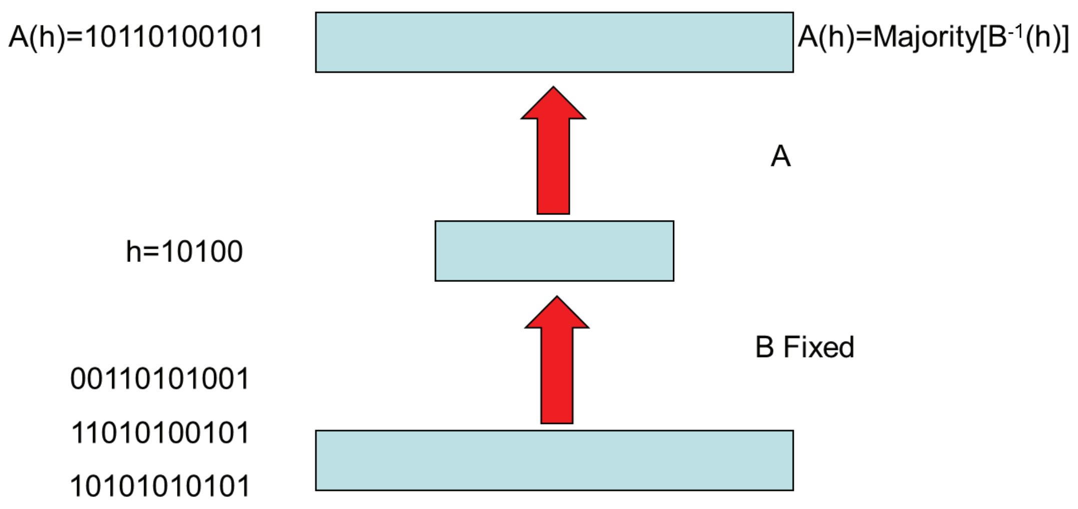

Figure 5.5: $A(n, m, n)$ Unrestricted Boolean Autoencoder with Fixed Transformation B. As shown, consider the set of vectors $B^{-1}(h)$ in the input space that are mapped to the same vector vector h in the hidden layer. Then the vector $A(h)$ must have minimal average Hamming distance to all the vectors in $B^{-1}(h)$. This is achieved if $A^*(h)$ is the Majority vector of $B^{-1}(h)$.

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 cases, one must find ways to embed the original problem with its original metric into the hypercube with the Hamming distance.

Problem: HYPERCUBE CLUSTERING

Instance: K binary vectors $x_1, \ldots, x_K$ of length n and an integer k.

Question: Can we identify k binary vectors $c_1, \ldots, c_k$ of length n (the centroids) and a function f from $\{x_1, \ldots, x_K\}$ to $\{c_1, \ldots, c_K\}$ that minimizes the distortion $E = \sum_{t=1}^{K} \Delta(x_t, f(x_t))$ where Δ is the Hamming distance?

The hypercube clustering problem is NP hard when $k \sim K^{\epsilon}$ ($\epsilon > 0$).

Proof. To sketch the reduction, we start from the optimization problem of clustering K points in the plane $\mathbb{R}^2$ using cluster centroids and the L_1 distance, which is NP-hard [447] by reduction from 3-SAT [271] when $k \sim K^\epsilon$ ($\epsilon > 0$) (see also related results in [430] and [651]). Without any loss of generality, we can assume that the points in these problems lie on the vertices of a square lattice. Using the theorem in [301], one can show that a $n_1 \times n_2$ square lattice in the plane can be embedded into the hypercube $\mathbb{H}^{n_1+n_2}$ (i.e. $n = n_1 + n_2$). More precisely, an explicit embedding is given in Figure 5.6 associating one distinct hypercube component to each horizontal step and each vertical step in the lattice. 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 $\mathbb{H}^{n_1+n_2}$. Thus a solution of the clustering problem on the hypercube would yield a solution of the clustering problem on the 2D lattice. This polynomial reduction completes the proof that if the number of cluster satisfies $k = 2^m \sim K^\epsilon$, or equivalently $m \sim \epsilon \log_2 K \sim C \log n$, then the hypercube clustering problem associated with the Boolean autoencoder is NP-hard. If the number k of clusters is fixed and the centroids must belong to the training set, there are only $\binom{K}{k} \sim K^k$ possible choices for the centroids inducing the corresponding Voronoi clusters. This yields a trivial, albeit not efficient, polynomial time algorithm.

There is another significant result on the NP-completeness of learning in neural networks. In [129], it is shown that the problem of training an $A(n, 2, 1)$ architecture of threshold gates, to achieve zero-error on a training set of size $O(n)$ is NP-complete. This remains true in the case of number of variations, for instance even if the weights are restricted to be binary. None of these NP-completeness results should worry practitioners, however, for several well-known reasons. In particular, for many applications, zero-error is not achievable, but also not necessary .

5.5 Other Autoencoders and Autoencoder Properties

5.5.1 Threshold Gate, Sigmoidal, and Mixed Autoencoders

Within the general framework described above, a number of different autoencoders can be considered with different constraints on $\mathbb{F}$ and $\mathbb{G}$, or different constraints on $\mathcal{A}$ and $\mathcal{B}$, for instance by varying the transfer functions in the hidden layer (e.g. Boolean unrestricted, Boolean linear threshold, linear, sigmoidal), the transfer functions in the output layer (e.g. Boolean unrestricted, Boolean linear threshold, linear, sigmoidal), the kinds of training data (binary versus real-valued), and the error function (e.g. Hamming, relative entropy, quadratic). We leave it as an exercise to examine some of the main combinations, however many of these can be under-

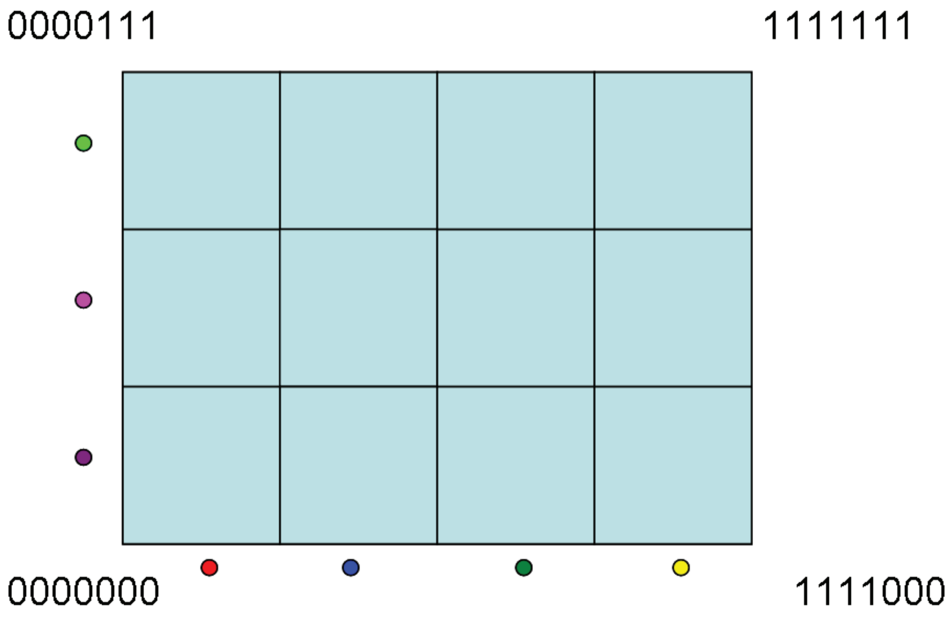

Figure 5.6: Embedding of a 3×4 Square Lattice onto $\mathbb{H}^7$ by Edge Coloring. All edges in the same row or column are given the same color. Each color corresponds to one of the dimensions of the 7-dimensional hypercube. For any pair of points, their Manhattan distance on the lattice is equal to the Hamming distance between their images in the 7-dimensional hypercube.

stood, at least at an intuitive level, using the results derived above in the linear and unrestricted Boolean case. 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 $2^m < K$, the autoencoder aims at clustering the real data into $k = 2^m$ 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 h should be the center of mass of the input vectors mapped onto h. In general, the optimization decision problem for these autoencoders is also NP-complete and, more importantly, from a probabilistic view point, they correspond exactly to a mixture of k-Gaussians model with hard cluster assignments in the deterministic case, and soft cluster assignments in the probabilistic case. More generally, as pointed out in [159], a number of classical problems and algorithms can be recovered by forcing one of the two classes $\mathcal{A}$ or $\mathcal{B}$ to be the class of linear functions, leaving the other one fully unrestricted (or with other non-linear constraints).

Within the general class of Boolean autoencoders, one can study different subclasses by restricting the sets of allowed Boolean functions. We have seen one such example in the case of the linear autoencoders over GF(2). Another interesting example corresponds to the Boolean autoencoder made exclusively of threshold gates [68]. These can be viewed as the limiting case of standard sigmoidal neural network autoencoders when the gain of the sigmoidal functions is taken to infinity. Another interesting restricted class of Boolean functions is the class of monotone functions,i.e. Boolean functions that can be built using only the AND and OR operators (these are called monotone because if the number of ones in the input is increased, the output can only remain identical, or go from 0 to 1.)

More importantly for this book, the solution of the real-valued linear autoencoder and the Boolean unrestricted autoencoder provide at least an intuitive picture of how learning may proceed in a compressive neural network autoencoder with sigmoidal transfer functions in the hidden layer. In the early stages of training, when the weights are small, the hidden neurons operate in their linear regime and learning may proceed toward PCA. As the weights increase and the hidden neurons begin to operate in the saturated portions of the sigmoidal transfer functions, learning may proceed towards optimal clustering in, or near, the PCA space.

5.5.2 Inverting, Redressing, and De-Noising Autoencoders

As a starting point, consider having a data set $D = \{x_1, \ldots, x_K\}$ where the points x_i are in a set A, and $A \subset \mathbb{R}^n$ or $A \subset H^n$ ($H = \{0, 1\}$). Assume that there is a (easily computable) function f from A to B, where $B \subset \mathbb{R}^p$ or $B \subset H^p$. We can use an $A(n, m, p)$ architecture to try to learn f using the pairs $(x, f(x))$ and, more importantly, an $A(p, m, n)$ architecture to learn f^{-1} when f can be inverted. If $p = n$, this leads to several different kinds of $A(n, m, n)$ autoencoders which somehow interpolate between the auto-associative and the hetero-associative cases. In these inverting autoencoders, the input $f(x)$ can often be viewed as a perturbed version of the data and the network is trained to restore or redress the input towards the unperturbed version $x = f^{-1}f(x)$. For instance if f is a rotation by α degrees, the corresponding redressing autoencoder would learn to rotate by $-\alpha$ degrees. Note that f does not need to be a function, it can also be a relation or process, as long as it is injective (i.e. $f(x)$ can take more than one value, but $f(x) = f(y)$ must imply $x = y$). When the function or process f corresponds to the addition of some kind of noise (e.g. Gaussian noise, random erasures) to the data, this approach is often called denoising autoencoder [660, 661].

Thus denoising autoencoders can be trained with the purpose of cleaning up noisy data, or reconstructing data (e.g. images or text) with missing or occluded entries (pattern completion). Denoising autoencoders can also be viewed in the context of regularization or data augmentation in order to prevent overfitting. This is particularly relevant if the autoencoder has a large number of parameters relative to the training data available initially. The denoising procedure can be viewed as a form of data augmentation by adding new training data, or as a form of dropout applied to the input layer [605, 91] (see next chapter). In the linear and unrestricted Boolean cases with $m < n$ the denoising autoencoder can be analyzed with the techniques described above, since these techniques work in the hetero-associative case, where the inputs and the targets differ.

Related to denoising autoencoders is also the idea of deblurring or training autoencoder architectures to produce high-resolution version of low resolution inputs. It is also possible to train autoencoder architectures in the reverse direction, to produce low resolution versions of the input in the output layer through a bottle neck. This is the basic idea behind the U-Net architectures for image segmentation [544] where images are presented in the input and segmented images are used as the targets for the output layer.

Along these lines, there are other ways for using an $A(n, m, n)$ architecture with supervised learning, where the input and the output target are slightly different and related through a simple process. For example, when dealing with sequences or time series, one could learn to predictx_{t+1} as a function of x_t. More generally, for a given fixed window of size k, one could learn to predict: $x_{t+1}, x_{t+2}, \ldots, x_{t+k}, x_{t+k+1}$ as a function of the input $x_t, x_{t+1}, \ldots, x_{t+k}$ where the size of the input and output layers is now kn(e.g. [498] and references therein). All the variations in this subsection are examples of self-supervised learning, where supervised learning is used with inputs and targets that are both derived from the original data through some operations.

5.5.3 Probabilistic Autoencoders and Variational Autoencoders

Probabilistic autoencoders with shape $A(n, m, n)$ (auto-associative), or even $A(n, m, p)$ (hetero-associative) can be incorporated into the above framework by considering that the input I, hidden H, or output O vectors are random variables and $\mathcal{A}$ and $\mathcal{B}$ represent classes of possible conditional distributions $P(O|H)$ and $P(H|I)$ over the right spaces. For instance, in the unrestricted probabilistic case, $\mathcal{A}$ and $\mathcal{B}$ are the classes of all possible conditional distributions over the right spaces. If all the units are binary and $m < n$, one obtains the binary compressive unrestricted probabilistic autoencoder where each A consists of a table of conditional probability distributions $P(O = y|H = h)$, where O denotes the binary output random variable and H the binary hidden layer random variable, and each B consists of a table of conditional probability distributions $P(H = h|I = x)$. The most relevant case here is when neural networks are used to parameterize one, or both, of the conditional distributions above. As described in Chapter 3, this can easily be done via sampling. A special case of this approach is the variational autoencoder described below. Different error functions can be used in the output layer. For instance, in the auto-associative case, one can consider the average error:

$$\mathcal{E} = \sum_x \sum_y P(I = x)P(O = y|I = x)\Delta(x, y) \tag{5.49}$$

where $\Delta(x, y)$ measures the degree of distortion between x and y (e.g. Hamming distance, quadratic error) and $P(O = y|I = x) = \sum_h P(H = h|I = x)(P(O = y|H = h)$, or the average relative entropy:

$$\mathcal{E} = -\sum_x P(I = x) \log P(O = x|I = x) \tag{5.50}$$

or the relative entropy between $P(I = x)$ and $P(O = y)$.

For brevity we will not further discuss these autoencoders here but it should be clear that some of the previous results can be extended to this case. For instance, the binary compressive unrestricted probabilistic autoencoder has the same invariances as the unrestricted Boolean autoencoder and similar considerations can be made for the unrestricted real-valued probabilistic autoencoder. The binary compressive unrestricted probabilistic autoencoder implements probabilistic clustering over the 2^m clusters indexed by the hidden layer activities in the sense that each input x has some probability $P(H = h|I = x)$ of being associated with the cluster labeled by h.

With a compressive layer, in both the auto-and hetero-associative case, it is also useful to consider the error function:

$$\mathcal{E} = I(X, H) - \beta I(H, T)$$

to be minimized over the free, or constrained, or parameterized, set of conditional distributions $P(H|I)$. Here β is positive weighting parameter which controls the relative role of the mutual information I between the input, renamed X to avoid any confusion, and the hidden variable H, and between the H and a target distribution variable T. When $T = X$, one simply seeks to maximize the mutual information between X and H. These probabilistc autoencoders correspond to the bottle-neck method [629] and the Blahut-Arimoto alternate optimization algorithm [125, 43] to try to find the optimal solution.

Note that since deterministic autoencoders can be viewed as special cases of probabilistic autoencoders, learning in probabilistic autoencoders is in general also NP-complete.

Variational Autoencoder

The variational autoencoder [368] is a special case of probabilistic autoencoder where the hidden layer is probabilistic. As we have seen in Chapter 3, the hidden layer of a feedforward neural network can be made stochastic by sampling. Thus in the variational autoencoder, the hidden layer vector H is a deterministic function of the input vector I. This function is parameterized by an encoding neural network. This vector H represents the parameters of a latent distribution over a latent space. During the forward propagation H is used to produce a sample $\tilde{H}$. The final output O is a deterministic function of $\tilde{H}$, which is also parameterized by a generative neural network. Once trained, the generative portion of the variational autoencoder can be used to provide a generative model of the data. Variational autoencoders can be trained by gradient descent using the methods described in the next chapter. As we shall see, the propagation of gradients through stochastic layers is facilitated if one can provide a simple parameterization of the relationship between the hidden activities and the corresponding sample. As a simple example, imagine that H is a two-dimensional vector encoding the mean μ and standard deviation σ of a normal distribution $\mathcal{N}(\mu, \sigma)$ over the latent space. Then, the sample $\tilde{H}$ can be parameterized in the form: $\tilde{H} = \mu + \epsilon\sigma$, where the sampling variable ϵ follows a standard normal $\mathcal{N}(0, 1)$ distribution.

5.5.4 Expansive Autoencoders

When the hidden layer is larger than the input layer ($m > n$)and $\mathbb{F} = \mathbb{G}$, there is in general an optimal 0-distortion solution to the autoencoder problem based on using the identity function between the layers. Thus in general this case is interesting only if additional constraints are added to the problem to prevent trivial solutions. These can come in many forms, for instance in terms of restrictions on the classes of functions $\mathcal{A}$ and $\mathcal{B}$,in terms of noise and robustness, or in terms of regularization. Restrictions could exclude the identity function and any other closely related functions. Noise could lead to the need for additional "parity check" bits, as in coding and communication theory. Regularization could be used, for instance to ensure sparsity of the hidden-layer representation, or to constrain the Jacobian or the Hessian of the data representation [435, 194, 539]. When these constraints force the hidden layer to assume only k different values then some of the previous analyses hold and the problem may reduce to clustering into k clusters. Another possibility for creating large hidden layer is the horizontal composition of autoencoders (see below). It is worth mentioning that expansive layers with sparse encoding are frequently found in the early stages of biological sensory systems [57]. While sparsity can be enforced artificially by regularization (e.g. L1 regularization), it is more interesting to explore biophysical or computational reasons for why layer size expansion, coupled with sparse encoding, could be advantageous in a physical neural system.

5.5.5 Composition of Autoencoders

Autoencoders can be composed both vertically (Figure 5.7) and horizontally. When autoencoders are stacked vertically, the hidden layer at one level of the stack can be used as the input layer for the next level of the stack. Thus the stack can be trained in an unsupervised layer from bottom to top. This approach can be used to initialize the layers of a deep architecture in an unsupervised way. When the hidden layers of the autoencoders are compressive, the results in this section suggest that the stack approximates some kind of hierarchical clustering of the data (at least when the hidden layers are binary, or close to binary when sigmoidal functions are in their saturated regimes). This can be viewed as a way of extracting increasingly more abstract representations of the data, as one goes up the hierarchy.

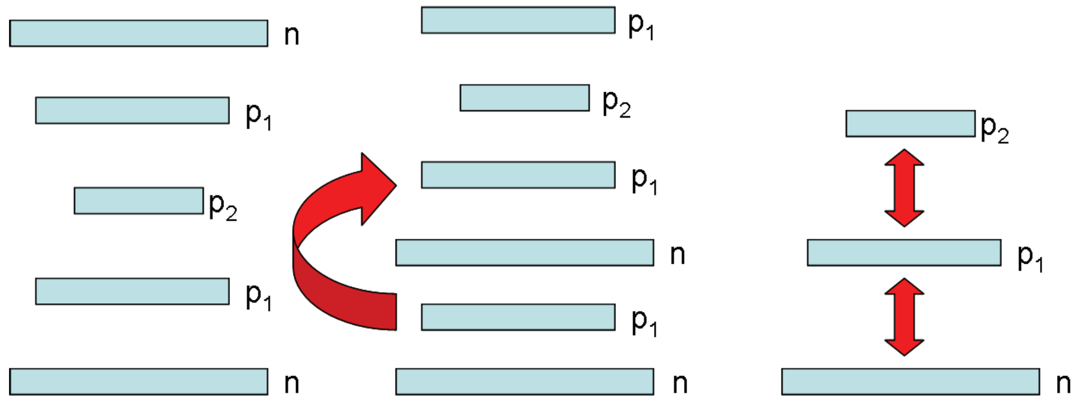

Figure 5.7: Left: Compressive autoencoder with multiple hidden layers. Middle and right: Vertical Composition of Autoencoders.

In addition to vertical composition, autoencoders can also be composed horizontally in various ways. For instance, two autoencoders with architectures $A(n_1, m_1, n_1)$ and $A(n_2, m_2, n_2)$ can be trained and combined into an $A(n_1 + n_2, m_1 + m_2, n_1 + n_2)$ architecture. It is also possible to share the same input and output layer and combine an $A(n, m_1, n)$ autoencoder with an $A(n, m_2, n)$ autoencoder in order to create an $A(n, m_1 + m_2, n)$ autoencoder with an expanded hidden layer representation (Figure 5.8), which in turn can be fed to other subsequent layer of the overall architecture. If $m_1 + m_2 < n$, the expanded hidden layer representation can still be compressive. Differences in the two hidden representations associated with the layers of size m_1 and m_2 can be introduced by many different mechanisms, for instance by using different learning algorithms, different initializations, different training samples, different learning rates, or different distortion measures. It is also possible to envision algorithms that incrementally add (or remove) hidden units to the hidden layer [535, 387]. 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.

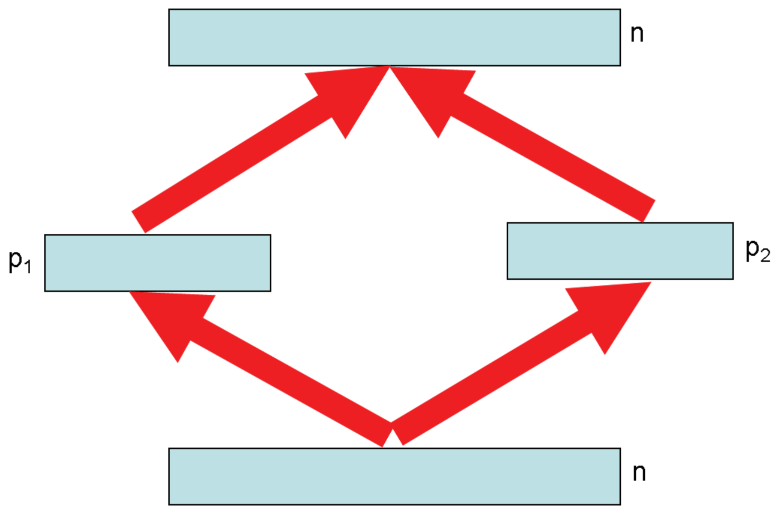

Figure 5.8: Horizontal Composition of Autoencoders to Expand the Hidden Layer Representation.

5.6 Exercises

5.1 Consider the linear autoencoder over the real numbers. Show that all the information about the data is contained in the mean and covariance matrix of the data. Show that the standard least square error function is a quadratic function (parabola) in each individual weight, if all the other weights are assumed to be constant.

5.2 Consider the $A(n, m, n)$ linear autoencoder over the real numbers with matrices A and B. Derive the gradient descent equation for the top matrix A using the results of the chapter on shallow learning. Derive the gradient descent equations for the matrix B (this becomes easy after seeing the chapter on backpropagation).

5.3 Consider the $A(n, m, n)$ linear autoencoder over the real numbers with matrices A and B. Show directly that if B is fixed, the problem is convex in A and similarly if

A is fixed the problem is convex in B. Derive the gradient descent learning equations for A and B in matrix form. Convert the corresponding difference equations into a system of differential equations. In general, does the system have a closed form solution? If not, can you find a closed form solution in some special cases?

5.4 In this chapter, we solved the linear autoencoder in a coordinate-independent way. An alternative approach is to solve the problem in a specific coordinate system, and then generalize the approach. Solve the linear autoencoder problem when the data mean is zero and the covariance matrix is diagonal. Solve the linear autoencoder problem when the data mean is not zero and the covariance matrix is diagonal. Generalize these results in order to solve the linear autoencoder problem when the data mean is zero, or non-zero, and the covariance matrix is not diagonal.

5.5 Consider a linear $A(n, m, n)$ autoencoder over the real numbers with partial connectivity, i.e. some of the entries of the matrices A or B are forced to be 0 at all times. Show that the total number of connectivity patterns is given by: $2^{nm}2^{nm}$. Is there a choice of a partial connectivity pattern for which the corresponding linear autoencoder can exhibit true spurious local minima (i.e. local minima where the quadratic error is strictly larger than the error achieved at the global minima)?

5.6 Consider a linear $A(n, m, n)$ autoencoder over the real numbers with a training set of the form $I(k)$ for $k = 1, \dots, K$ and matrices A and B. Thus the output has the form: $O = ABI$, and the quadratic error function is given by: $\mathcal{E} = \sum_k ||I(k) - ABI(k)||^2/2$. Write down, in matrix form, the learning equations for A and B under simple Hebbian learning. Write down in matrix form the learning equations for A and B under gradient descent learning. Convert all the previous learning equations into differential equations and compare the two forms of learning.

5.7 Study the linear $A(n, m, n)$ autoencoder problem over the complex field $\mathbb{C}$.

5.8 Study the linear $A(n, m)$ regression problem over any finite field, for instance over the finite field $\mathbb{F}_2$ with two elements. Study the linear $A(n, m, n)$ autoencoder problem over any finite field, for instance over $\mathbb{F}_2$.

5.9 Consider the $A(n, m, n)$ autoencoder problem with linear threshold gates in the hidden and output layers, the Hamming distance as the error function $\mathcal{E}$, and a training set of polynomial size in n and m. Is the problem of minimizing $\mathcal{E}$ NP-hard?

5.10 Consider the $A(n, m, n)$ autoencoder problem with monotone Boolean gates in the hidden and output layers, the Hamming distance as the error function $\mathcal{E}$, and a training set of polynomial size in n and m. Is the problem of minimizing $\mathcal{E}$ NP-hard?

5.11 Consider an unrestricted Boolean autoencoder $A(n, m, n)$ where the K binary training examples are generated as follows. First, k_1 centroids are generated using independent Bernoulli coin flips with probability $p_1 = 0.5$. Then, starting from each centroid, k_2 examples are generated by perturbing the centroid using independent Bernoulli coin flips with small probability $p_2 = \epsilon$ of changing the corresponding bit, so that $K = k_1 k_2$. Examine different regimes for N, M, K, k_1 and ϵ and estimate the expected training and generalization errors.

5.12 Consider an unrestricted Boolean autoencoder $A(n, m, n)$ where the connectivity is local. In general, compared to the fully connected case, is it easier or harder to solve it? Why? Which algorithm would you use to try to minimize the Hamming error?

5.13 Consider different compressive autoencoders by varying the type of units (e.g. unrestricted Boolean, threshold, sigmoidal, linear) in the hidden layer or the output layer, the kind of training data (binary or real-valued), and the error function (e.g. Hamming distance, relative entropy, quadratic). Using the results in the chapter, for each autoencoder type develop an intuitive understanding of its behavior and confirm or disprove your intuition by mathematical analyses or simulations.

5.14 Formalize a notion of binary, compressive, unrestricted, probabilistic $A(n, m, n)$ autoencoder and analyze its properties. Here unrestricted means that any probabilistic table between input and hidden vectors, or hidden and output vectors, can be realized.

5.15 Assume that the n-dimensional input vectors x of a linear channel have a normal distribution with mean zero and covariance matrix Σ_{xx}. The output of the channel are constrained to be p-dimensional vectors y of the form $y = Bx$ for some $p \times n$ matrix B with $p < n$. The differential entropy of x is defined by:

$$H(x) - \int P(x) \log P(x) dx \tag{5.51}$$

Prove that:
1) y has a normal distribution with mean 0 and covariance matrix $\Sigma_{yy} = B\Sigma_{xx}B^t$.
2)

$$H(x) = \frac{1}{2} \log \left[(2\pi e)^n \det(\Sigma_{xx})\right] \tag{5.52}$$

and

$$H(y)\frac{1}{2}\log\left[(2\pi e)^p \det(B\Sigma_{xx}B^t)\right] \tag{5.53}$$

3) The conditional distribution of x given y is normal with mean: $\Sigma_{xy}\Sigma yy^{-1}y$, and covariance matrix: $\Sigma = \Sigma_{xx} - \Sigma_{xy}\Sigma yy^{-1}\Sigma_{yx}$.
4) The conditional entropy of x given y is given by:

$$H(x|y) = \frac{1}{2}\log\left[(2\pi e)^{n-p}\gamma_1\gamma_2\dots\gamma_{n-p}\right] \tag{5.54}$$

where $\gamma_1 \geq \gamma_2 \geq \dots \geq \gamma_{n-p} > 0$ are the non-zero eigenvalues of Σ.
5) Maximizing $I(x,y)$ is equivalent to minimizing $H(x|y)$ and this is achieved by PCA, i.e. by having $B = CU_p^t$ where C is an invertible $p\times p$ matrix and U is a matrix of column vectors $U_p = [u_1,\dots,u_p]$. Here $u_1,\dots,u_p$ denote p principal normalized eigenvectors of Σ_{xx} with eigenvalues $\lambda_1,\dots,\lambda_p$.
6) At this maximum:

$$I_{PCA}(x,y) = \frac{1}{2}\log\left[(2\pi e)^p\lambda_1\lambda_2\dots\lambda_p\right] \tag{5.55}$$

5.16 Derive critical or consistency relations for the bottleneck method in simple cases, for instance, when the variables have Gaussian distributions.

5.17 In a variational autoencoder, assume that the hidden variable H encodes the parameters of a Beta distribution

$$B(a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}x^{a-1}(1-x)^{b-1}$$

How can one parameterize a sample $\tilde{H} \in [0,1]$?

5.18 Prove a variational bound for the variational autoencoder.